Alen Jakupović
Sabrina Šuman

OSNOVE PROGRAMIRANJA

veleri

Alen Jakupović
Sabrina Šuman

OSNOVE PROGRAMIRANJA

DRUGO IZDANJE

VELEUČILIŠTE U RIJECI
RIJEKA, 2020.

Dr.sc. Alen Jakupović, izvanredni profesor i prof. visoke škole
Dr. sc. Sabrina Šuman, viši predavač

OSNOVE PROGRAMIRANJA

Nakladnik
Veleučilište u Rijeci
www.veleri.hr

Za nakladnika
Dr. sc. Saša Hirnig, prof. visoke škole

Recenzenti
Dr. sc. Mile Pavlić, redoviti profesor
Mr. sc. Maja Gligora Marković, viši predavač

Lektorica
Marijana Trinajstić, prof.

Slog i prijelom
Dr. sc. Alen Jakupović, izvanredni profesor

Dizajn ovitka
Dr. sc. Sabrina Šuman, viši predavač

Veleučilište u Rijeci uvrstilo je ovaj udžbenik u veleučilišne udžbenike
(Klasa: 602-09/14-01/03, Ur.broj: 2170-57-02-14-2)

Povjerenstvo za izdavačku djelatnost Veleučilišta u Rijeci odobrilo je objavljivanje ovog veleučilišnog udžbenika kao elektroničke publikacije Odlukom:
KLASA: 003-11/20-01/10, URBROJ: 2170-57-01-20-11/MJG

UDK 004.42(075.8)
004.43 C(075.8)

ISBN 978-953-8286-01-8

Svojoj djeci, Loredani, Damianu i Benjaminu.

Alen

Svojim sinovima, Orlandu i Riccardu.

Sabrina

PREDGOVOR DRUGOM IZDANJU

Iskustvo stečeno u šest godina primjene udžbenika "Osnove programiranja" u nastavi Preddiplomskih stručnih studija Informatika i Telematika Veleučilišta u Rijeci dokazuje opravdanost njegova izdavanja. Velika potražnja ovoga udžbenika među redovnim i izvanrednim studentima nagnala je autore da pripreme njegovo drugo izdanje ali u elektroničkom obliku tako da udžbenik bude slobodno dostupan svima zainteresiranima za stjecanje osnovnih znanja iz područja proceduralnog programiranja.

Strukturalno i sadržajno se drugo izdanje udžbenika "Osnove programiranja" ne razlikuje od njezina prvog izdanja. Autori udžbenika su od prvoga izdanja stekli nova akademska i nastavno-znanstvena zvanja te je to zahtijevalo izmjenu u autorskom potpisu.

Zahvaljujemo se svim dosadašnjim čitateljima i molimo ih da nam jave svoja iskustva u učenju primjenom ovog udžbenika. Svim budućim čitateljima želimo puno uspjeha u učenju – javite nam i vaša iskustva. Naša elektronička pošta je alen.jakupovic@veleri.hr i sabrina.suman@veleri.hr.

U Rijeci, ožujak 2020.

izv. prof. dr. sc. Alen Jakupović, prof. v. š.

dr. sc. Sabrina Šuman, viši predavač

PREDGOVOR PRVOM IZDANJU

Udžbenik **Osnove programiranja** namijenjen je studentima prve godine Stručnoga studija informatike Veleučilišta u Rijeci ali i ostalima koji žele steći osnovna znanja iz proceduralnoga programiranja.

Temeljni je cilj udžbenika čitatelja upoznati s osnovama proceduralnoga programiranja uz primjenu programskoga jezika C. Nakon usvajanja izloženoga gradiva, čitatelj bi trebao biti sposoban: izraditi algoritam u skladu s proceduralnim pristupom (*top-down*), izrađeni algoritam oblikovati u računalni program primjenom programskoga jezika C, testirati računalni program te primjenjivati osnovne tehnike pronalaska i ispravljanja pogrešaka.

Za praćenje izloženoga sadržaja nisu potrebna prethodna programerska iskustva. Svi prikazani primjeri računalnih programa izrađeni su u besplatnom alatu Bloodshed Dev-C++ koji se može pronaći na internetskoj stranici http://bloodshed-dev-c.en.softonic.com/.

Udžbenik je podijeljen u devet međusobno povezanih poglavlja. Svako poglavlje započinje popisom ishoda učenja koji se mogu dostići njegovim sadržajem. Poglavlja koja opisuju i praktični rad u programskome jeziku C završavaju nizom izrađenih i objašnjenih praktičnih primjera te popisom zadataka za daljnju vježbu. Programski su kodovi, korišteni u knjizi, prikazani tako da je svaka njihova linija koda numerirana čime je olakšano praćenje njihova referenciranja iz teksta. Pojedini prikazani primjeri programskoga koda namjerno odstupaju od dobre prakse (npr. u nazivanju varijabli i procedura i sl.). To je učinjeno zato što se u praksi mogu naći i takvi programski kodovi koje je onda i teže za razumjeti i u konačnici za održavati – time se čitatelju nastojalo zorno predočiti potreba pisanja "samodokumentirajućega" programskog koda.

Poglavlje **1. Programiranje** opisuje temeljne pojmove koji se odnose na algoritam, načine njegova prikazivanja, vrste algoritamskih struktura, programiranje, programski jezik i faze njegova razvoja, računalni program, njegove vrste, strategiju izvođenja i životni ciklus. Ovo poglavlje prikazuje širi kontekst postupka programiranja. Poglavlje završava nizom primjera algoritama.

Poglavlje **2. Varijable** predstavlja temeljne informacije o varijablama – jednom od najvažnijih pojmova u programiranju. Shvaćanje pojma varijable ključno je u razumijevanju procesa programiranja. Poglavlje opisuje memoriju računala i zapisa vrijednosti, pojam varijable i njezinih osnovnih svojstava te prikaz primjene varijable u programskome jeziku C.

Poglavlje **3. Linijska algoritamska struktura u C** opisuje implementaciju ove strukture u programskome jeziku C. Prikazan je osnovni kostur računalnoga programa pisanoga u C-u te njegovi glavni dijelovi (deklaracija varijabli, unos podataka, izračun, ispis podataka). Prikazane informacije zorno su objašnjene kroz velik broj praktičnih primjera. Na kraju poglavlja nalaze se zadaci za vježbu.

Poglavlje **4. Razgranata algoritamska struktura u C** prikazuje implementaciju ove strukture primjenom naredbi if–else if–else i switch–case. Primjena je naredbi pokazana nizom različitih primjera računalnih programa detaljno opisanih tekstom. Poglavlje završava zadacima za vježbu.

Poglavlje **5. Ciklička algoritamska struktura u C** opisuje naredbe for, while i do–while koje implementiraju ovu strukturu. Nizom je opisanih primjera računalnih programa zorno prikazana praktična primjena navedenih naredbi. Zadaci za vježbu nalaze se na kraju poglavlja.

Poglavlje **6. Algoritamska struktura bezuvjetnoga skoka u C** prikazuje implementaciju strukture kroz naredbe exit, break, continue, return i goto. Naredbe su praktično primijenjene u nizu detaljno opisanih računalnih programa.

Poglavlje **7. Procedure u računalnome programu** opisuje svrhu izrade procedura, način njihova definiranja i komunikacije s mjestom poziva. Praktično, nizom opisanih primjera računalnih programa, prikazana je izrada procedura u programskome jeziku C. Na kraju je poglavlja niz zadataka za vježbu.

Poglavlje **8. Rad s datotekama** opisuje tekstualne i binarne datoteke. Prikazane su naredbe fopen, fclose, fprintf, fscanf, fread, fwrite njihovom praktičnom primjenom u računalnim programima.

Poglavlje **9. Pronalaženje i ispravljanje pogrešaka u računalnome programu** općenito prikazuje pojmove kao što su alat za pronalaženje pogrešaka (*engl. debugger*), točka prekida izvođenja programa (*engl. break point*), izvedi naredni korak (*engl. next step*), izvedi unutrašnje korake (*engl. step into*), nastavi izvođenje (*engl. continue*) i dodaj varijablu na promatranje (*engl. add watch*). Opisana je primjena alata za pronalaženje i ispravljanje pogrešaka koji je integralni dio računalnoga programa Bloodshed Dev-C++.

Prilog **Podsjetnik**, koji se nalazi na kraju udžbenika, može se koristiti kao podsjetnik na naredbe koje su se koristile u primjerima računalnih programa. Studenti mogu prilog koristiti tijekom praktičnoga dijela ispita.

Na kraju, zahvaljujemo recenzentima, prof.dr.sc. Mili Pavliću i mr.sc. Maji Gligora Marković te kolegi Marinu Franušiću, na ustupljenim primjerima praktičnih zadataka. Posebnu zahvalu upućujemo svim čitateljima koji će za stjecanje programerskoga znanja izabrati i ovaj udžbenik. Njima upućujemo i zamolbu da sve primjedbe, sugestije, komentare i uočene pogreške dojave elektroničkom poštom – alen.jakupovic@veleri.hr i sabrina.suman@veleri.hr, biti ćemo im veoma zahvalni.

U Rijeci, travanj 2014.

doc. dr.sc. Alen Jakupović, prof. v. š.

Sabrina Šuman, predavač

.

Sadržaj

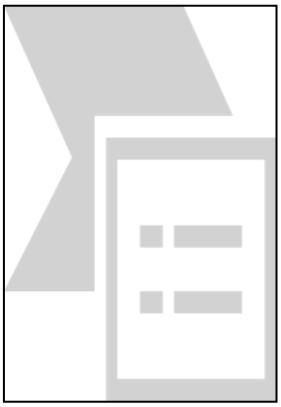
1. PROGRAMIRANJE

Poglavlje **Programiranje** odnosni se na pojam algoritma, način njegova prikaza, osnovne elemente iz kojih je izgrađen, definiciju programiranja, programskoga jezika i računalnoga programa, strategije izvođenja računalnoga programa, vrste računalnih programa i programiranja, faze razvoja programskih jezika te životni ciklus računalnoga programa.

Po završetku ovoga poglavlja čitatelj će moći:

- definirati pojam algoritma
- nabrojati načine prikaza algoritma
- primijeniti jedan način prikaza algoritma koji je dan u drugom prikazu
- izraditi algoritam za dani problem
- nabrojati osnovne algoritamske strukture
- prepoznati osnovne algoritamske strukture u danom algoritmu
- izračunati izlaz iz danog algoritma uz poznate ulaze
- identificirati grešku u danom algoritmu
- provjeriti dani algoritam uz dane ulaze i izlaze
- preispitati mogućnost postojanja nekog drugog algoritma za rješavanje istog problema
- definirati pojam programiranje
- definirati pojam programski jezik
- definirati pojam računalni program
- navesti vrste računalnih programa
- objasniti razliku između različitih vrsta računalnih programa
- navesti vrste programiranja
- objasniti razliku između različitih vrsta programiranja
- navesti generacije programskih jezika
- objasniti razliku između pojedinih generacija programskih jezika
- navesti strategije izvođenja računalnoga programa
- objasniti razliku između strategija izvođenja računalnoga programa
- objasniti razliku između strojnog i izvornog koda računalnoga programa
- navesti faze životnoga ciklusa računalnoga programa
- objasniti pojedinu fazu životnoga ciklusa računalnoga programa.

1.1. Pojam algoritma



Riječ **algoritam** dolazi od latinskoga prijevoda imena arapskoga matematičara *Abu Ja'far* Muhammad *ibn Musa Al-Khwarizmi* (Muhamed, otac Jafarov, sin Muse iz Khwarizma). Rođen je 780. godine na području današnjeg Uzbekistana, a umro je 850. godine u Bagdadu. Smatra se ocem algebre - grane matematike koja izučava matematičke operacije i relacije te strukture i koncepte koji iz njih proizlaze.

Pojmovno **algoritam** označava **precizan** i **jednoznačan** opis rješenja nekoga problema. Sastoji se iz **konačnoga** skupa uputa koje **potpuno** i **nedvosmisleno** definiraju korake koji se trebaju učiniti, kao i njihov **redoslijed**, s ciljem rješavanja danoga problema. Svaki korak mora biti definiran **dopuštenim** (**izvedivim**) instrukcijama. Isti problem može imati više različitih rješenja, pa time i više različitih algoritama.

Primjer nepreciznih instrukcija:	**Primjer dvosmislenih instrukcija:**
Podijeli neki broj s 20. Čekaj par minuta. Nazovi profesora. Pričekaj me u Rijeci. Otvori vrata.	Podijeli broj 10 s 2 ili 4. Čekaj me 10 minuta ili 2 sata. Očisti kolegije s prve godine. Zagrij stolicu. Budi glasan.
Primjer nepotpunih instrukcija:	**Primjer nedopuštenih (neizvedivih) instrukcija:**
Podijeli broj 20 s Čekaj me u Nazovi me preko Otvori vrata u Pročitaj knjigu o	Podijeli broj 2 s 0. Pomnoži broj 10 s beskonačno. Trčeći prođi kroz zid. Nauči programirati u 5 minuta. Uberi cvijet na Marsu.

Tablica 1.1. Primjeri loših algoritamskih instrukcija

Do 20. se stoljeća pojam algoritam koristio kao sredstvo opisa postupka rješavanja matematičkih problema.

Primjer postupka (algoritma) rješavanja jednadžbe ax=b:	
1.	Ako su **a** i **b** jednaki **0**, jednadžba ima beskonačno mnogo rješenja (**x** može biti bilo koji broj).
2.	Ako je **a** jednak **0**, a **b** različit od **0**, jednadžba nema rješenja (ne postoji broj kojim se može zamijeniti **x**).
3.	Ako su **a** i **b** različiti od **0**, jednadžba ima jedno rješenje **x=b/a** (**x** može biti samo jedan broj).

Algoritam 1.1.

Od 20. se stoljeća, pojavom računala, pojam algoritam počinje koristiti i u računarstvu, a zatim i na drugim područjima. Mnogi istraživači vjeruju da je bilo koja aktivnost ljudskog uma ustvari rezultat izvođenja algoritma - posljedica toga je razvoj znanosti koja se bavi umjetnom inteligencijom.

Algoritam (rješenje nekoga problema), se može opisati na sljedeće načine:

1. **tekstualno**
2. **grafički**
3. **pseudo kodom**
4. **kombinacijom grafičkog opisa i pseudo koda**
5. **programskim jezikom.**

1.1.1. Tekstualni opis algoritma

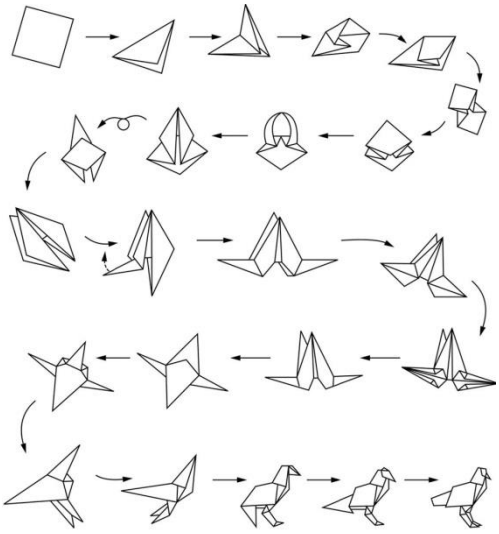
U tekstualnom se opisu, koraci algoritma prikazuju rečenicama govornoga jezika. Prednost je ovakvoga opisa u tome što ga može razumjeti širi krug ljudi. Veliki je nedostatak nepreciznost opisa koja proizlazi iz nepreciznosti samoga govornog jezika.

Primjer tekstualnog opisa algoritma - recept za domaći kruh:	
1.	Kvasac razmrvite, dodajte malo šećera i 50 ml mlake vode pa ostavite na toplom mjestu oko 10 minuta.
2.	Prosijanom brašnu dodajte sol i kvasac pa pomiješajte s preostalom vodom.
3.	Umiješajte rastopljeni maslac i zamijesite glatko tijesto. Ostavite neka se diže na toplom mjestu oko 30 minuta.
4.	Tijesto premijesite, oblikujte štrucu i stavite u namašćeni kalup.
5.	Prije pečenja tijesto stavite još jedanput dizati.
6.	Zarežite ga nožem i poprskajte vodom, stavite u zagrijanu pećnicu na 225°C i pecite oko 10 minuta, a zatim smanjite na 170°C i pecite još 35 minuta.
Primjer tekstualnog opisa algoritma - provjera je li godina prijestupna:	
1.	Ako je godina djeljiva s 400 ona je prijestupna godina.
2.	Ako je godina djeljiva s 4 ali ne i sa 100 ona je prijestupna godina.

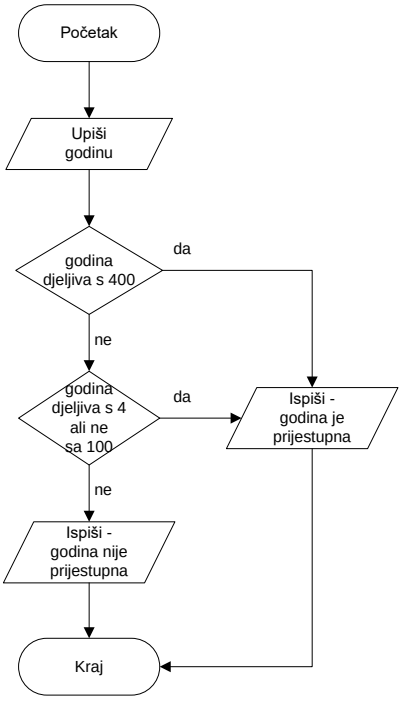
Algoritam 1.2.

1.1.2. Grafički opis algoritma

U grafičkom se opisu, koraci algoritma predstavljaju grafičkim simbolima. Grafički su simboli međusobno povezani strelicama koje upućuju na redoslijed grafičkih simbola, odnosno redoslijed izvođenja koraka algoritma. Ovaj način opisa algoritma precizniji je od tekstualnoga opisa, ali je za njegovo shvaćanje potrebno poznavanje značenja grafičkih simbola iz kojih se sastoji opis, te je stoga razumljiv užemu krugu ljudi.



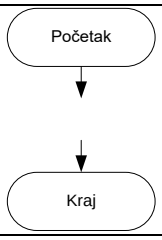
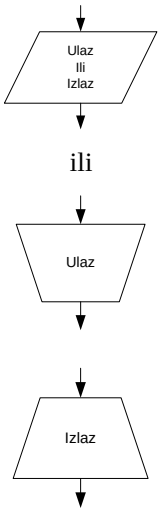
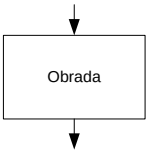
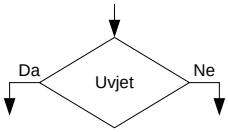
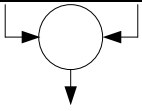
Dijagram 1.1. Primjer grafičkoga opisa algoritma – origami ptica od papira

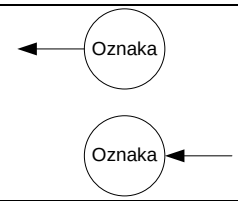


Dijagram 1.2. Primjer grafičkoga opisa algoritma – provjera je li godina prijestupna

Posljednji grafički opis algoritma (dijagram 1.2.) zove se **dijagram toka** ili **blok dijagram**. Blok dijagram je vrsta grafičkoga opisa algoritma primjenom unaprijed definiranih grafičkih simbola. Grafički simboli predstavljaju određeni algoritamski korak, a strelice koje ih povezuju, sljedeći algoritamski korak koji se treba izvesti.

Blok dijagram se koristi tijekom oblikovanja algoritma. Tablica 1.2. prikazuje grafičke simbole koji se koriste pri izradi bloka dijagrama.

Grafički simbol	Značenje
	Početak i kraj algoritma. Početak nema ulaznih tokova i ima jedan izlazni tok. Kraj ima jedan izlazni tok i nema izlaznih tokova.
	Ulaz u algoritam ili izlaz iz algoritma. Upis podataka potrebnih za algoritam ili ispis rezultata algoritma. Imaju jedan ulazni i jedan izlazni tok.
	Obrada (procesiranje) u algoritmu, sadrži jedan ulazni i jedan izlazni tok.
	Odluka u algoritmu (grananje), sadrži jedan ulazni tok i dva izlazna.
	Spajanje grana, sadrži dva ulazna toka i jedan izlazni.

	Priključna točka, sadrži jedan tok, ulazni ili izlazni.
	Tok izvođenja koraka algoritma

Tablica 1.2. Grafički simboli bloka dijagrama

1.1.3. Algoritam opisan pseudo kodom

Pseudo kod predstavlja formalizirani tekstualni opis algoritma kojim se postiže veća preciznost koja nedostaje klasičnome govornom jeziku. Formaliziranost se ogleda u unaprijed definiranim načinima prikaza osnovnih struktura algoritma - linijska, razgranata, ciklička i skok (o ovim strukturama bit će više riječi u posebnome poglavlju). Obično se svaka linija pseudo koda označava rednim brojem. Zbog svoga tekstualnog opisa, pseudo kod je razumljiv širemu krugu ljudi.

Pseudo kod predstavlja detaljni opis algoritma primjenom formaliziranoga prirodnog jezika. Zbog toga je on čitljiv ljudima, a ne računalima. Koristi se prilikom oblikovanja algoritma, a prije njegove same implementacije u nekome programskom jeziku koji je razumljiv računalima. Pseudo kod bi trebao biti potpuno nezavisan o bilo kojemu programskom jeziku, odnosno riječi koje se koriste u pseudo kodu ne bi trebale biti vezane za neki specifični programski jezik.

Opći oblik pseudo koda za algoritam:

```
1.  Doznaj Vrijednost
2.  Izvedi obradu Vrijednosti
3.  Reci koliko iznose Rezultati
4.  Doznaj želi li se završiti algoritam (Da/Ne)
5.  Ako je odgovor = Ne skoči na algoritamski korak 1
```

Algoritam 1.3.

Primjer algoritma opisanoga pseudo kodom - rješenje jednadžbe ax=b:

```
 1.  Unesi koeficijent A
 2.  Unesi koeficijent B
 3.  Ako je A=0 i B=0 onda
 4.     Ispiši "X može biti bilo koji broj."
 5.  Inače ako je A=0 i B<>0 onda
 6.     Ispiši "Jednadžba nema rješenja."
 7.  Inače
 8.     X=B/A
 9.     Ispiši "Rješenje jednadžbe glasi: X."
10.  Kraj Ako je
```

Algoritam 1.4.

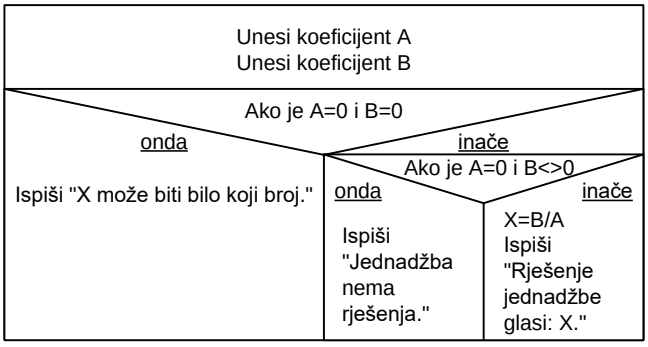
Primjer algoritma opisanoga pseudo kodom - provjera je li godina prijestupna:

```
1.  Unesi godinu G
2.  Ako je G djeljivo s 400 onda
3.     Ispiši "Godina G je prijestupna."
4.  Inače ako je G djeljivo s 4 i G nije djeljivo sa 100 onda
5.     Ispiši "Godina G je prijestupna."
6.  Inače
7.     Ispiši "Godina G nije prijestupna."
8.  Kraj Ako je
```

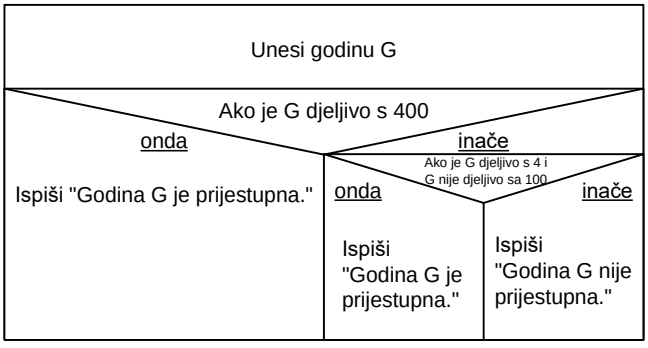
Algoritam 1.5.

1.1.4. Algoritam opisan kombinacijom grafičkoga opisa i pseudo kodom

Ovaj opis kombinira značajke grafičkoga opisa i pseudo koda. Slično kao i u grafičkom opisu, postoje posebni grafički simboli kojima se predstavljaju osnovne algoritamske strukture. U te se simbole upisuje pseudo kod kojim se detaljnije opisuju koraci algoritma. Ovime se dodatno povećava preciznost opisa algoritma, ali se sužuje krug ljudi koji ga razumiju, budući je potrebno poznavati značenje pojedinoga grafičkog simbola. Prikazani se opis algoritama naziva **strukturogram**.



Dijagram 1.3. Primjer algoritma opisanoga kombinacijom grafičkoga opisa i pseudo koda – rješenje jednadžbe ax=b



Dijagram 1.4. Primjer algoritma opisanoga kombinacijom grafičkoga opisa i pseudo koda – provjera je li godina prijestupna

1.1.5. Algoritam opisan programskim jezikom

Algoritam opisan nekim programskim jezikom postaje računalni program koji izvodi računalo. Programski je jezik posebna vrsta jezika kojim se računalu izdaju naredbe (instrukcije) koje treba izvesti. Njihovim izvođenjem računalo izvodi opisani algoritam te time rješava neki problem. Programskim se jezikom algoritam najpreciznije opisuje. Osim računalu, ovaj je opis razumljiv i uskome krugu ljudi - programerima.

Primjer algoritma opisanoga programskim jezikom - rješenje jednadžbe ax=b:

```
 1. input "Unesite koeficijent a: "; a
 2. input "Unesite koeficijent b: "; b
 3. if a=0 and b=0 then
 4.   print "X može biti bilo koji broj."
 5. else
 6.   if a=0 and b<>0 then
 7.     print "Jednadžba nema rješenja."
 8.   else
 9.     let x=b/a
10.     print "Rješenje jednadžbe glasi: "; x
11.   end if
12. end if
```

Algoritam 1.6.

Primjer algoritma opisanoga programskim jezikom - provjera je li godina prijestupna:

```
 1. #include <conio.h>
 2. #include <stdio.h>
 3. main(){
 4.   unsigned int G;
 5.   printf ("Unesite godinu: ");
 6.   scanf("%d", &G);
 7.   if (G%400==0)
 8.     printf("Godina %d je prijestupna.", G);
 9.   else if (G%4==0 && G%100!=0)
10.     printf("Godina %d je prijestupna.", G);
11.   else
12.     printf("Godina %d nije prijestupna.", G);
13.   getch();}
```

Algoritam 1.7.

Prvi je algoritam opisan programskim jezikom **Basic** (jedan besplatni alat za programiranje u Basicu je *Just BASIC*).

Drugi je algoritam opisan programskim jezikom **C**. Samo on će se dalje koristiti, a besplatni alat u kome se može programirati u C-u je *Dev-C++* (u njemu se može programirati i u programskome jeziku C++). U narednim će poglavljima biti više riječi o programskome jeziku C.

1.2. Osnovne algoritamske strukture

Algoritam predstavlja opis dolaska do rješenja nekoga problema. Sastoji se iz algoritamskih koraka čijim izvođenjem izvršitelj (čovjek, računalo itd.) dolazi do rješenja. Svaki algoritamski korak pripada nekoj osnovnoj algoritamskoj strukturi.

Osnovne algoritamske strukture su:

1. **linijska (slijed, sekvencija)**
2. **razgranata (grananje, selekcija)**
3. **struktura bezuvjetnog skoka**
4. **ciklička (ponavljanje, iteracija).**

Osnovnim se algoritamskim strukturama može prikazati bilo koji algoritam.

U narednim će se poglavljima osnovne algoritamske strukture prikazati uz primjenu pseudo koda i bloka dijagrama.

1.2.1. Linijska algoritamska struktura prikazana pseudo kodom

Linijska algoritamska struktura (slijed, sekvencija) je osnovna struktura u kojoj se algoritamski koraci izvode linijski (slijedno, sekvencijalno). To znači da se ne može izvesti naredni algoritamski korak ako nije izveden onaj koji mu prethodi. Dakle, nema preskakanja i proizvoljnoga reda izvođenja algoritamskih koraka, već je redoslijed njihova izvođenja strogo određen.

Pseudo kodom se općenito ova struktura može prikazati na sljedeći način:

Opći primjer algoritma s linijskom algoritamskom strukturom:

```
 1.  Algoritamski korak 1
 2.  Algoritamski korak 2
...  ...
 n.  Algoritamski korak n
```

Algoritam 1.8.

Izvršitelj gore opisanoga algoritma će prvo izvesti `Algoritamski korak 1`. Nakon što ga izvede krenut će u izvođenje `Algoritamskog koraka 2` itd. dok na kraju ne izvede i zadnji `Algoritamski korak n`.

U pseudo se kodu algoritamski koraci označavaju rednim brojem čime se jasno pokazuje redoslijed njihova izvođenja.

Primjer linijske algoritamske strukture - izračun indeksa tjelesne mase prema formuli BMI = m/v² (m - masa; v - visina):

```
 1.  Doznaj masu m u kilogramima.
 2.  Doznaj visinu v u centimetrima.
 3.  Izračunaj BMI = m/(v/100)².
 4.  Reci da za masu m i visinu v indeks tjelesne mase iznosi
     BMI.
```

Algoritam 1.9.

Primjer linijske algoritamske strukture - izračun iznosa PDVa iz maloprodajne cijene artikla prema formuli IznosBezPDV = IznosSPDV·100/(100+PDV):

```
 1. Doznaj maloprodajnu cijenu artikla IznosSPDV.
 2. Doznaj iznos PDV u %.
 3. Izračunaj IznosBezPDV = IznosSPDV*100/(100+PDV).
 4. Reci da za maloprodajnu cijenu artikla IznosSPDV i iznos
    poreza PDV cijena bez poreza iznosi IznosBezPDV.
```

Algoritam 1.10.

Iz prikazanih je primjera vidljivo da linijska algoritamska struktura omogućava prihvaćanje ulaznih vrijednosti, izvođenja aritmetičko/logičkih operacija nad njima i stvaranje novih vrijednosti te davanje izlaznih vrijednosti. Podebljano tiskane i podcrtane riječi unutar algoritamskih koraka su nazivi pod kojima se čuvaju vrijednosti (u programiranju se oni zovu **varijable** - o njima će biti više riječi u narednim poglavljima). Linijska algoritamska struktura se može sastojati iz bilo koje druge algoritamske strukture.

1.2.2. Linijska algoritamska struktura prikazana blok dijagramom

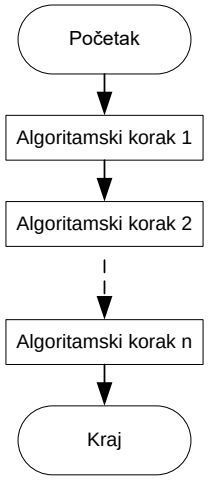
U prethodnome poglavlju je prikazan opći oblik linijske algoritamske struktura iskazane pseudo kodom.

Opći primjer algoritma s linijskom algoritamskom strukturom opisan pseudo kodom:

```
 1. Algoritamski korak 1
 2. Algoritamski korak 2
... ...
 n. Algoritamski korak n
```

Algoritam 1.11

Isti bi se opći oblik linijske algoritamske strukture u obliku blok dijagrama mogao prikazati na sljedeći način:



Dijagram 1.5. Opći primjer algoritma s linijskom algoritamskom strukturom opisan blok dijagramom

Algoritamski koraci se izvode redom, jedan za drugim. `Algoritamski korak i` (i=2, ..., n) ne može započeti s izvođenjem dok se `Algoritamski korak i-1` ne završi. Vidljivo je da blok dijagram započinje i završava sa simbolima za početak i kraj algoritma.

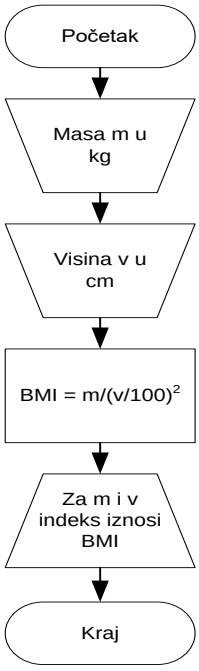
Primjer linijske algoritamske strukture - izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina):	
1.	Doznaj masu **m** u kilogramima.
2.	Doznaj visinu **v** u centimetrima.
3.	Izračunaj **BMI** = **m**/(**v**/100)^2.
4.	Reci da za masu **m** i visinu **v** indeks tjelesne mase iznosi **BMI**.

Algoritam 1.12.

Primjer linijske algoritamske strukture - izračun iznosa PDVa iz maloprodajne cijene artikla prema formuli IznosBezPDV = IznosSPDV·100/(100+PDV):	
1.	Doznaj maloprodajnu cijenu artikla **IznosSPDV**.
2.	Doznaj iznos **PDV** u %.
3.	Izračunaj **IznosBezPDV** = **IznosSPDV***100/(100+**PDV**).
4.	Reci da za maloprodajnu cijenu artikla **IznosSPDV** i iznos poreza **PDV** cijena bez poreza iznosi **IznosBezPDV**.

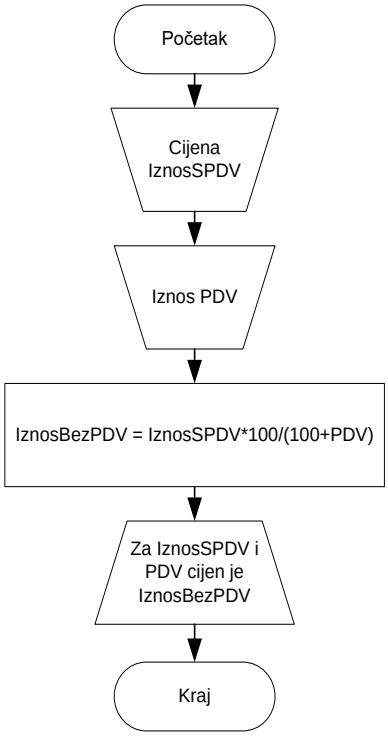
Algoritam 1.13.

Prikazani primjeri opisani pseudo kodom su već prethodno opisani u Algoritam 1.9. i 1.10. Isti se ovi primjeri preko blok dijagrama mogu prikazati na sljedeći način.



Dijagram 1.6. Primjer linijske algoritamske strukture – izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina)

Blok dijagram započinje sa simbolom za početak algoritma. Potom se prema toku algoritma dolazi do simbola za unos podatka u algoritam (*Masa m u kg*) - ovdje se mogao koristiti i grafički simbol koji predstavlja ulaz ili izlaz. Tok algoritma zatim vodi ponovno do simbola za unos (*Visina v u cm*) te do simbola za obradu gdje se izračunava indeks tjelesne mase (*BMI*). Zatim tok vodi do simbola za izlaz podatka iz algoritma (*Za m i v indeks iznosi BMI*). Blok dijagram završava sa simbolom za kraj algoritma.



Dijagram 1.7. Primjer linijske algoritamske strukture – izračun iznosa PDVa iz maloprodajne cijene artikla prema formuli IznosBezPDV = IznosSPDV·100/(100+PDV)

Opis je ovoga bloka dijagrama analogan prethodnom opisu.

1.2.3. Razgranata algoritamska struktura prikazana pseudo kodom

Razgranata algoritamska struktura (grananje, selekcija) je osnovna struktura koja omogućava uvjetno izvođenje niza algoritamskih koraka. To znači da se posebnim algoritamskim korakom opisuje koji logički uvjet mora biti ispunjen kako bi se izvela neka skupina algoritamskih koraka. Logički uvjet može imati dvije vrijednosti - istina (*engl. True*) i neistina (*engl. False*).

Pseudo kodom se općenito ova struktura može prikazati na sljedeći način:

Opći primjer algoritma s razgranatom algoritamskom strukturom:

```
  1.  Ako je uvjet 1 onda
  2.     Algoritamski korak 11
  3.     Algoritamski korak 12
 ...     ...
  i.     Algoritamski korak 1x
i+1.  inače ako je uvjet 2 onda
i+2.     Algoritamski korak 21
i+3.     Algoritamski korak 22
 ...     ...
  j.  inače ako je uvjet m onda
j+1.     Algoritamski korak m1
j+2.     Algoritamski korak m2
 ...     ...
  k.  Inače
k+1.     Algoritamski korak n1
k+2.     Algoritamski korak n2
 ...     ...
  p.  Kraj Ako je
p+1.  Algoritamski korak z
 ...  ...
```

Algoritam 1.14.

Izvršitelj algoritma će prilikom izvođenja 1. algoritamskog koraka provjeriti je li zadovoljen `uvjet 1`. Ako `uvjet 1` ima vrijednost istina, tada će krenuti izvoditi `algoritamske korake 11, 12... 1x`. Ovaj niz algoritamskih koraka tvori blok koraka koji će se izvesti uz ispunjenje `uvjeta 1`. Nakon izvođenja bloka koraka izvršitelj preskače sve ostale algoritamske korake (linije od i+1 do p) i kreće u izvođenje `algoritamskog koraka z`.

Ako u 1. algoritamskom koraku nije ispunjen `uvjet 1` (dakle ima vrijednost neistina), izvršitelj kreće u izvođenje i+1. algoritamskog koraka u kojemu provjerava je li ispunjen `uvjet 2`. Ako je ispunjen, izvodi njegov blok koraka (analogno kako je to već objašnjeno kod ispunjenja `uvjeta 1`).

Ako izvršitelj otkrije da niti jedan od uvjeta (od `uvjeta 1` do `uvjeta m`) nije ispunjen, onda se izvodi blok koraka koji se nalazi iza k+1. algoritamskog koraka (`Algoritamski korak n1, n2, ...`).

Bez obzira je li neki uvjet bio ispunjen ili su svi bili neispunjeni, `algoritamski korak z` će uvijek biti izveden.

Primjer razgranate algoritamske strukture - izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina):

```
 1.  Doznaj masu m u kilogramima.
 2.  Doznaj visinu v u centimetrima.
 3.  Izračunaj BMI = m/(v/100)^2.
 4.  Reci da za masu m i visinu v indeks tjelesne mase iznosi
     BMI.
```

```
 5.  Ako je BMI<18.5 onda
 6.    Reci da indeks tjelesne mase BMI znači Pothranjenost.
 7.  inače ako je BMI<25 onda
 8.    Reci da indeks tjelesne mase BMI znači Normalna
       tjelesna težina.
 9.  inače ako je BMI<30 onda
10.    Reci da indeks tjelesne mase BMI znači Prekomjerna
       tjelesna težina.
11.  inače ako je BMI<40 onda
12.    Reci da indeks tjelesne mase BMI znači Pretilost.
13.  inače
14.    Reci da indeks tjelesne mase BMI znači Izrazita
       pretilost.
15.  Kraj Ako je
```

Algoritam 1.15.

Primjer razgranate algoritamske strukture - koliko kalendarski mjesec ima dana:

```
 1.  Doznaj broj mjeseca M.
 2.  Ako je M=1 ili M=3 ili M=5 ili M=7 ili M=8 ili M=10 ili
     M=12 onda
 3.    Reci da M. mjesec ima 31 dan.
 4.  inače ako je M=4 ili M=6 ili M=9 ili M=11 onda
 5.    Reci da M. mjesec ima 30 dana.
 6.  inače ako je M=2 onda
 7.    Reci da M. mjesec ima 28 ili 29 dana, ovisno je li
       godina prijestupna ili ne.
 8.  inače
 9.    Reci da M. mjesec ne postoji.
10.  Kraj Ako je
```

Algoritam 1.16.

Podebljane i podcrtane riječi predstavljaju nazive (varijable) pod kojima se čuvaju vrijednosti nad kojima će se izvesti aritmetičke i logičke operacije. Pravokutnikom su označeni logički uvjeti. Može se uočiti da logički uvjeti koji se nalaze u razgranatoj algoritamskoj strukturi mogu biti složeni što oni često i jesu. Također se može uočiti da se sastoje iz znaka < i = (ovo su relacijski operatori o kojima će biti više riječi u narednim poglavljima - svi relacijski operatori su >, <, >=, <=, <> i =) i riječi `ili` (on spada u skupinu logičkih operatora – osnovni su logički operatori `I`, `ILI` i `NEGACIJA` - o njima će biti više riječi u narednim poglavljima). Unutar bloka koraka koji se uvjetno izvode mogu biti bilo koje druge algoritamske strukture.

1.2.4. Razgranata algoritamska struktura prikazana blok dijagramom

Već je prikazano da se pseudo kodom općenito razgranata algoritamska struktura može prikazati na sljedeći način:

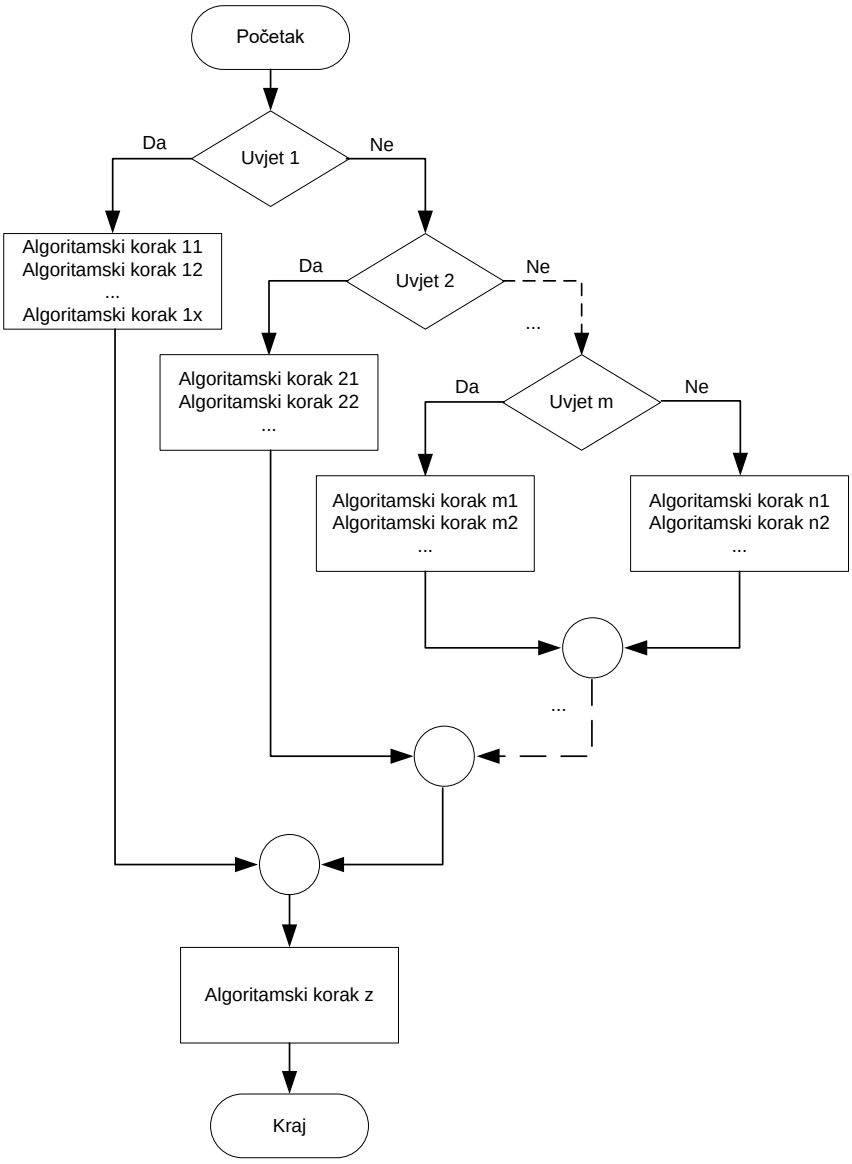
Opći primjer algoritma s razgranatom algoritamskom strukturom:

```
  1.  Ako je uvjet 1 onda
  2.     Algoritamski korak 11
  3.     Algoritamski korak 12
 ...     ...
  i.     Algoritamski korak 1x
i+1.  inače ako je uvjet 2 onda
i+2.     Algoritamski korak 21
i+3.     Algoritamski korak 22
 ...     ...
  j.  inače ako je uvjet m onda
j+1.     Algoritamski korak m1
j+2.     Algoritamski korak m2
 ...     ...
  k.  Inače
k+1.     Algoritamski korak n1
k+2.     Algoritamski korak n2
 ...     ...
  p.  Kraj Ako je
p+1.  Algoritamski korak z
 ...  ...
```

Algoritam 1.17.

Isti se opći oblik razgranate algoritamske strukture blok dijagramom može prikazati kao na dijagramu 1.8.

Blok dijagram započinje i završava sa simbolima za početak i kraj algoritma. Nakon simbola za početak slijedi simbol za uvjet koji jedan tok razdvaja u dva toka u ovisnosti je li uvjet zadovoljen ili ne. Tokovi dalje vode ili do simbola za novi uvjet ili do simbola za obradu. Grane se tokova trebaju spojiti simbolom za spajanje grana iz koje izlazi samo jedan tok. Tok iz zadnjeg simbola za spajanje vodi do simbola za obradu (`Algoritamski korak z`) i na kraju do simbola za završetak algoritma.



Dijagram 1.8. Opći primjer algoritma s razgranatom algoritamskom strukturom opisan blok dijagramom

Primjer razgranate algoritamske strukture - izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina):	
1.	Doznaj masu **m** u kilogramima.
2.	Doznaj visinu **v** u centimetrima.
3.	Izračunaj **BMI** = **m**/(**v**/100)^2.
4.	Reci da za masu **m** i visinu **v** indeks tjelesne mase iznosi

```
     BMI.
 5.  Ako je BMI<18.5 onda
 6.    Reci da indeks tjelesne mase BMI znači Pothranjenost.
 7.  inače ako je BMI<25 onda
 8.    Reci da indeks tjelesne mase BMI znači Normalna tjelesna
       težina.
 9.  inače ako je BMI<30 onda
10.    Reci da indeks tjelesne mase BMI znači Prekomjerna
       tjelesna težina.
11.  inače ako je BMI<40 onda
12.    Reci da indeks tjelesne mase BMI znači Pretilost.
13.  inače
14.    Reci da indeks tjelesne mase BMI znači Izrazita
       pretilost.
15.  Kraj Ako je
```

Algoritam 1.18.

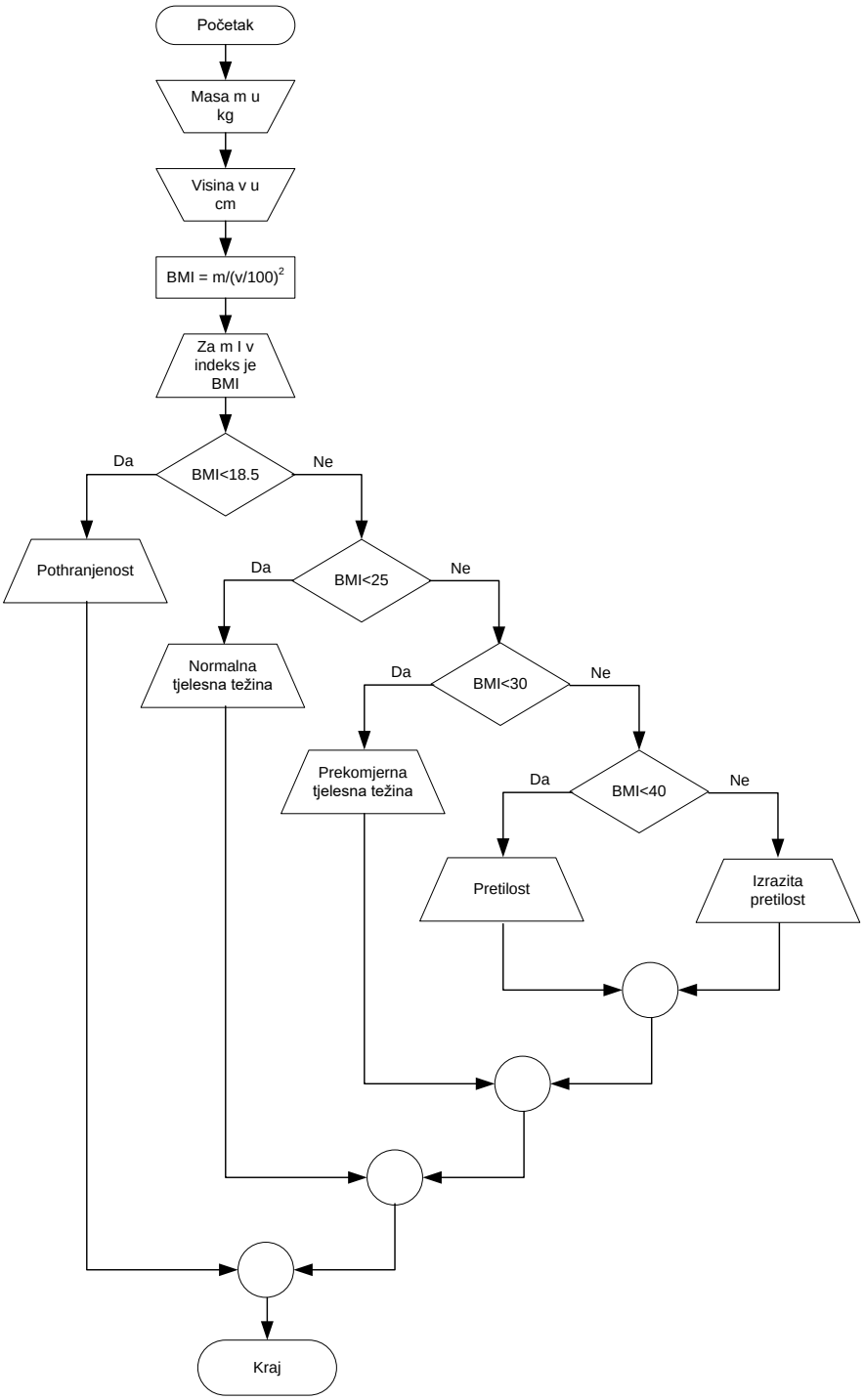
Primjer razgranate algoritamske strukture - koliko kalendarski mjesec ima dana:

```
 1.  Doznaj broj mjeseca M.
 2.  Ako je M=1 ili M=3 ili M=5 ili M=7 ili M=8 ili M=10 ili
     M=12 onda
 3.    Reci da M. mjesec ima 31 dan.
 4.  inače ako je M=4 ili M=6 ili M=9 ili M=11 onda
 5.    Reci da M. mjesec ima 30 dana.
 6.  inače ako je M=2 onda
 7.    Reci da M. mjesec ima 28 ili 29 dana, ovisno je li
       godina prijestupna ili ne.
 8.  inače
 9.    Reci da M. mjesec ne postoji.
10.  Kraj Ako je
```

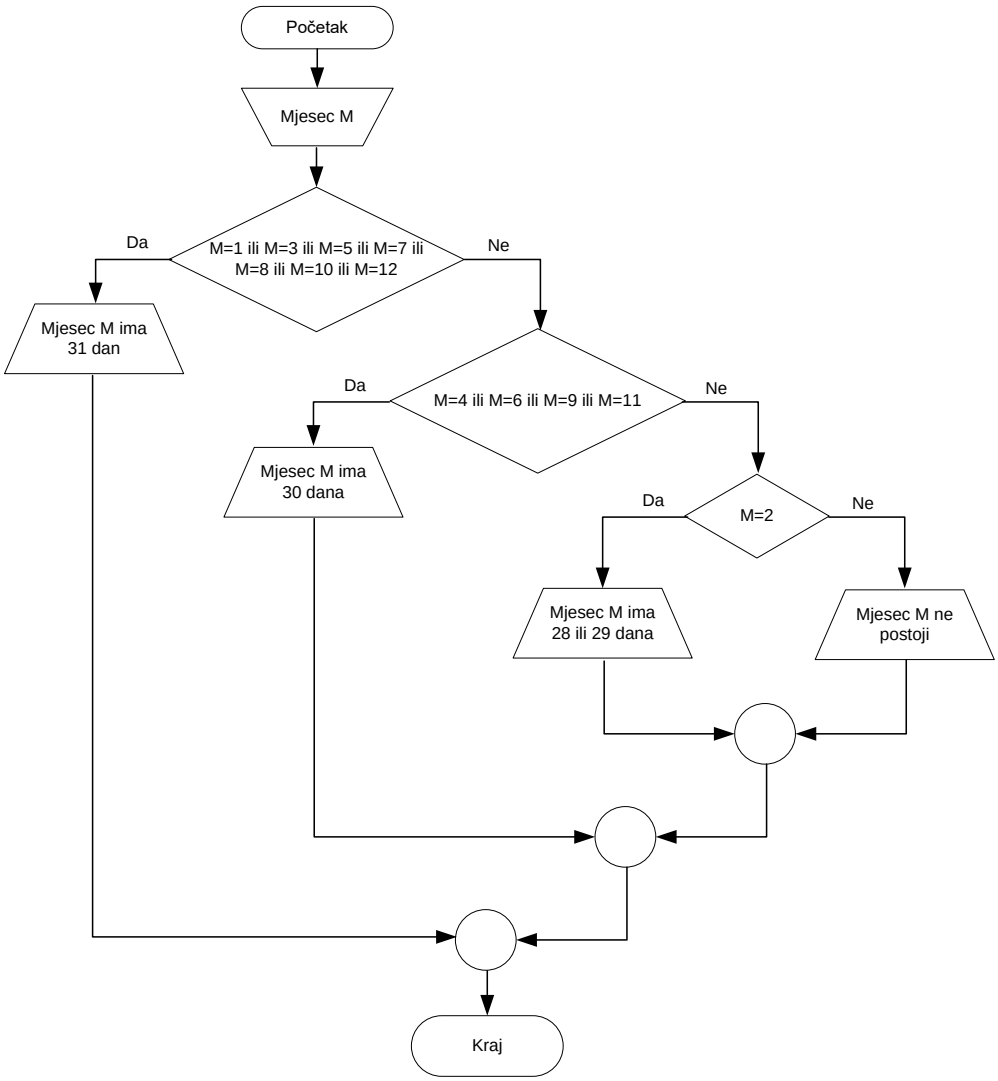
Algoritam 1.19.

Prikazani se primjeri algoritama u pseudo kodu u blok dijagramu mogu prikazati kao na dijagramima 1.9. i 1.10.

Nakon početka, u blok dijagramu (dijagram 1.9.) se nalaze dva simbola za ulaz (*za masu m i visinu v*), zatim simbol za obradu (*izračun BMI*) i nakon toga simbol za izlaz (*za BMI*). Potom slijedi prva odluka s uvjetom *BMI<18.5*. Ako je uvjet zadovoljen (*tok Da*), onda se ide na simbol za izlaz, nakon toga na simbol za spajanje i zatim na kraj. Ako uvjet nije zadovoljen (*tok Ne*), ide se na novi simbol odluke koji ponovno jedan tok grana u dva itd.



Dijagram 1.9. Primjer razgranate algoritamske strukture – izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina)



Dijagram 1.10. Primjer razgranate algoritamske strukture – koliko kalendarski mjesec ima dana

Opis je ovoga dijagrama analogan opisu prethodnoga.

1.2.5. Algoritamska struktura bezuvjetnoga skoka prikazana pseudo kodom

Algoritamska struktura bezuvjetnoga skoka je osnovna algoritamska struktura koja omogućava skok na neki algoritamski korak nakon čijega izvođenja se dalje izvodi algoritamski korak koji ga slijedi (nema povratka na algoritamski korak s koga je izvršen skok). Postoji poseban algoritamski korak kojim se navodi oznaka algoritamskoga koraka na koji se skače.

Pseudo kodom se općenito ova algoritamska struktura može prikazati na sljedeći način:

Opći primjer algoritma sa strukturom bezuvjetnoga skoka:

```
 1.  oznaka: Algoritamski korak 1
 2.  Algoritamski korak 2
 3.  Skoči na oznaka.
...  ...
 n.  Algoritamski korak n
```

Algoritam 1.20.

Izvršitelj izvodi 1. algoritamski korak, zatim 2. algoritamski korak, te 3. algoritamski korak u komu se nalazi korak `Skoči na oznaka`. Zbog toga izvršitelj ponovno izvodi 1. algoritamski korak, zatim 2. i ponovno 3. koji mu opet kaže skoči na 1. itd. Lako se može zaključiti da ovakav algoritam ne završava u konačnome broju koraka (stalno izvodi prva tri algoritamska koraka) i nikada se neće izvesti n. algoritamski korak.

Primjer algoritma sa strukturom bezuvjetnoga skoka - izračun indeksa tjelesne mase prema formuli BMI = m/v² (m - masa; v - visina):

```
 1.  oznaka: Doznaj masu m u kilogramima.
 2.  Doznaj visinu v u centimetrima.
 3.  Izračunaj BMI = m/(v/100)^2.
 4.  Reci da za masu m i visinu v indeks tjelesne mase iznosi
     BMI.
 5.  Ako je BMI<18.5 onda
 6.    Reci da indeks tjelesne mase BMI znači Pothranjenost.
 7.  inače ako je BMI<25 onda
 8.    Reci da indeks tjelesne mase BMI znači Normalna
       tjelesna težina.
 9.  inače ako je BMI<30 onda
10.    Reci da indeks tjelesne mase BMI znači Prekomjerna
       tjelesna težina.
11.  inače ako je BMI<40 onda
12.    Reci da indeks tjelesne mase BMI znači Pretilost.
13.  Inače
14.    Reci da indeks tjelesne mase BMI znači Izrazita
       pretilost.
15.  Kraj Ako je
16.  Doznaj želi li se računati novi indeks tjelesne mase
     (Da/Ne).
17.  Ako je odgovor = Da onda
18.    Skoči na oznaka.
19.  Kraj Ako je
```

Algoritam 1.21.

Primjer algoritma sa strukturom bezuvjetnog skoka - koliko kalendarski mjesec ima dana:

```
 1.  oznaka: Doznaj broj mjeseca M.
 2.  Ako je M<1 ili M >12 onda
```

```
 3.    Reci da M. mjesec ne postoji.
 4.    Skoči na oznaka.
 5.  Ako je M=1 ili M=3 ili M=5 ili M=7 ili M=8 ili M=10 ili
     M=12 onda
 6.    Reci da M. mjesec ima 31 dan.
 7.  inače ako je M=4 ili M=6 ili M=9 ili M=11 onda
 8.    Reci da M. mjesec ima 30 dana.
 9.  inače
10.    Reci da M. mjesec ima 28 ili 29 dana, ovisno je li
       godina prijestupna ili ne.
11.  Kraj Ako je
```

Algoritam 1.22.

Podebljana i podcrtana riječ predstavlja simboličko ime linije algoritamskoga koraka na koji izvršitelj treba skočiti i od koga dalje treba izvoditi korake.

Prikazani primjeri primjene strukture bezuvjetnoga skoka nisu poželjne pojave u algoritmima (poželjno ih je zamijeniti cikličkom algoritamskom strukturom). Ipak, neki se oblici bezuvjetnoga skoka redovito koriste u računalnim programima (npr. za prekid programa) - o njima će biti više riječi u narednim poglavljima.

1.2.6. Algoritamska struktura bezuvjetnoga skoka prikazana blok dijagramom

Opći se oblik algoritamske strukture bezuvjetnoga skoka pseudo kodom može opisati na sljedeći način:

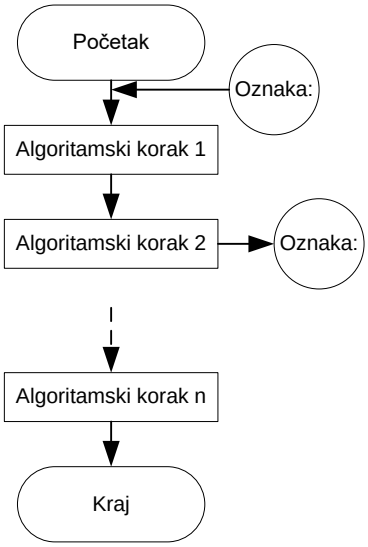
Opći primjer algoritma sa strukturom bezuvjetnoga skoka opisan pseudo kodom:

```
  1.  oznaka: Algoritamski korak 1
  2.  Algoritamski korak 2
  3.  Skoči na oznaka.
....  ...
  n.  Algoritamski korak n
```

Algoritam 1.23.

Ista algoritamska struktura se blok dijagramom može prikazati kao na dijagramu 1.11.

Iz prikaza bloka dijagrama za algoritamsku strukturu bezuvjetnoga skoka vidljivo je da se za skok koristi simbol priključne točke. Ista takva priključna točka treba postojati negdje u blok dijagramu i ona pokazuje mjesto početka izvođenja algoritamskih koraka.



Dijagram 1.11 Opći primjer algoritma sa strukturom bezuvjetnoga skoka opisan blok dijagramom

Primjer algoritma sa strukturom bezuvjetnoga skoka - izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina):

```
 1. A: Doznaj masu m u kilogramima.
 2. Doznaj visinu v u centimetrima.
 3. Izračunaj BMI = m/(v/100)^2.
 4. Reci da za masu m i visinu v indeks tjelesne mase iznosi
    BMI.
 5. Ako je BMI<18.5 onda
 6.    Reci da indeks tjelesne mase BMI znači Pothranjenost.
 7. inače ako je BMI<25 onda
 8.    Reci da indeks tjelesne mase BMI znači Normalna
       tjelesna težina.
 9. inače ako je BMI<30 onda
10.    Reci da indeks tjelesne mase BMI znači Prekomjerna
       tjelesna težina.
11. inače ako je BMI<40 onda
12.    Reci da indeks tjelesne mase BMI znači Pretilost.
13. inače
14.    Reci da indeks tjelesne mase BMI znači Izrazita
       pretilost.
15. Kraj Ako je
16. Doznaj želi li se računati novi indeks tjelesne mase
    (Da/Ne).
17. Ako je odgovor = Da onda
18.    Skoči na A.
19. Kraj Ako je
```

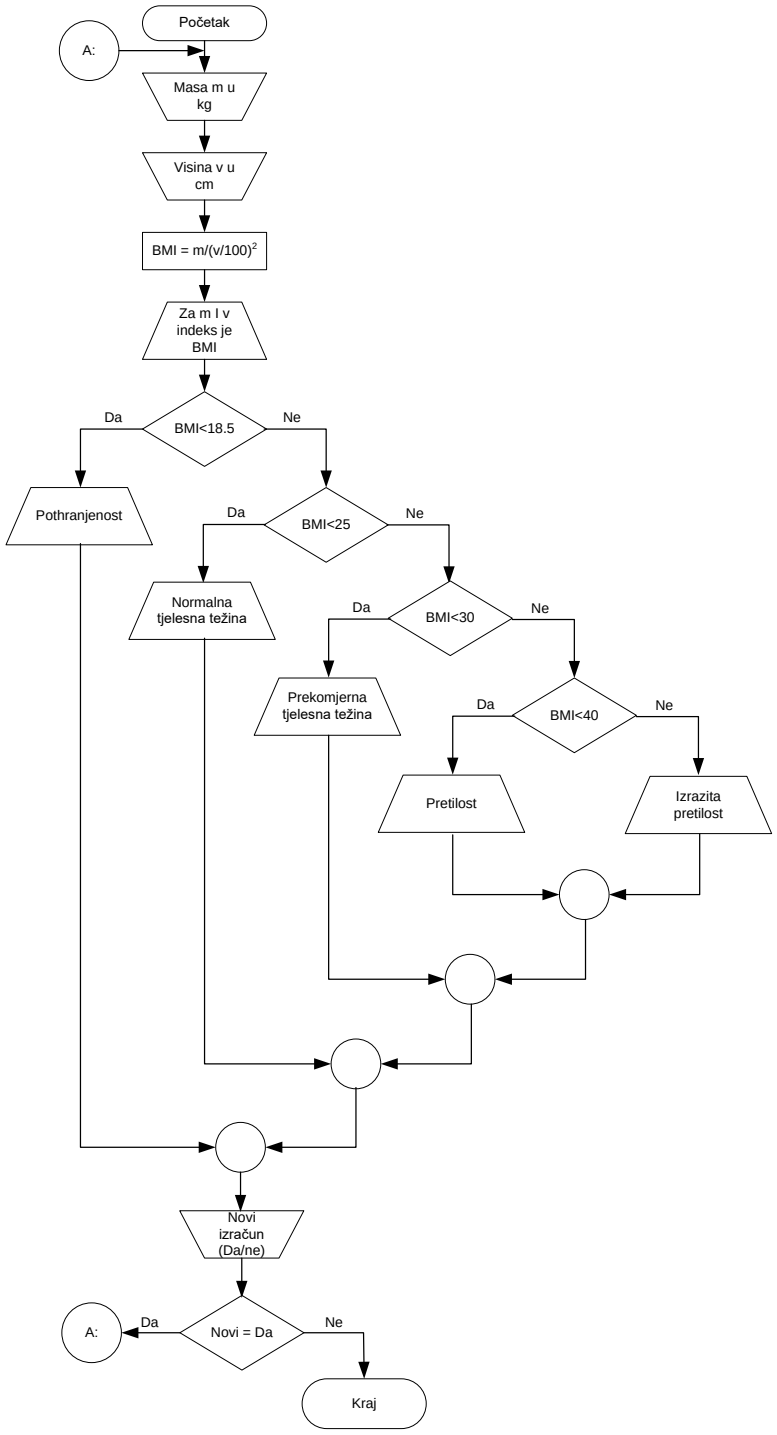
Algoritam 1.24.

Primjer algoritma sa strukturom bezuvjetnoga skoka - koliko kalendarski mjesec ima dana:

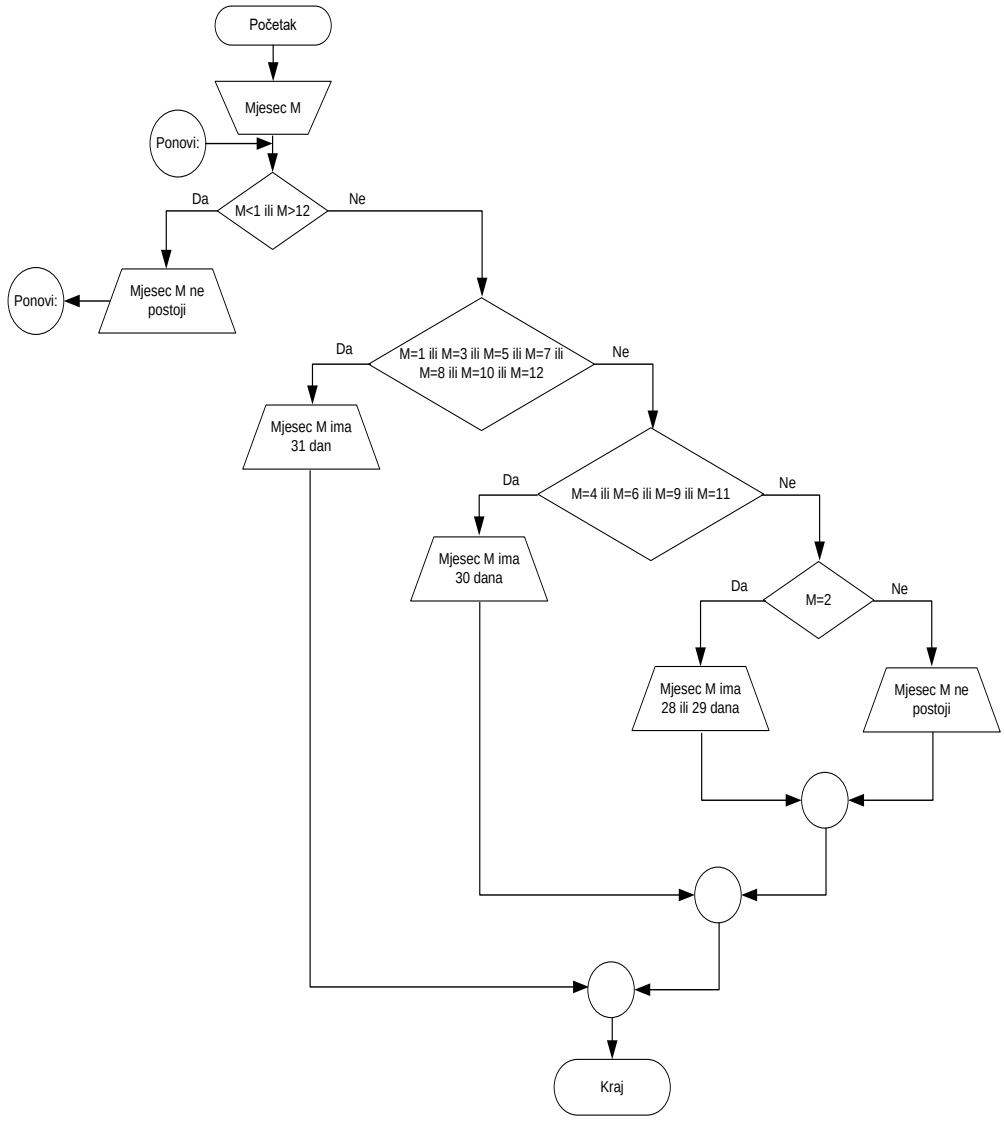
```
 1. ponovi: Doznaj broj mjeseca M.
 2. Ako je M<1 ili M >12 onda
 3.    Reci da M. mjesec ne postoji.
 4.    Skoči na ponovi.
 5. Ako je M=1 ili M=3 ili M=5 ili M=7 ili M=8 ili M=10 ili
    M=12 onda
 6.    Reci da M. mjesec ima 31 dan.
 7. inače ako je M=4 ili M=6 ili M=9 ili M=11 onda
 8.    Reci da M. mjesec ima 30 dana.
 9. inače
10.    Reci da M. mjesec ima 28 ili 29 dana, ovisno je li
       godina prijestupna ili ne.
11. Kraj Ako je
```

Algoritam 1.25.

Na kraju je bloka dijagrama (dijagram 1.12.) vidljiv simbol odluke (*Novi = Da*) koji u jednom slučaju ima tok prema simbolu priključna točka s nazivom *A:*. Druga se priključna točka naziva *A:* nalazi neposredno iza simbola za početak budući da se cjelokupni prikazani algoritam treba ponovno izvesti (dakle svi njegovi koraci).



Dijagram 1.12. Primjer algoritma sa strukturom bezuvjetnoga skoka – izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina)



Dijagram 1.13. Primjer algoritma sa strukturom bezuvjetnoga skoka - koliko kalendarski mjesec ima dana

1.2.7. Ciklička algoritamska struktura prikazana pseudo kodom

Ciklička algoritamska struktura (ponavljanje, iteracija) je izvedena algoritamska struktura koja omogućava uvjetno ponavljanje izvođenja niza algoritamskih koraka. Izvedena je jer se može realizirati s dvjema osnovnim algoritamskim strukturama - linijskom i strukturom bezuvjetnoga skoka. Postoji poseban algoritamski korak kojim se navodi logički uvjet koji mora biti ispunjen kako bi se blok koraka izvodio i dok god je taj logički uvjet ispunjen, izvođenje bloka koraka se ponavlja. Sve što je rečeno

o logičkom uvjetu u razgranatoj algoritamskoj strukturi vrijedi i ovdje. Postoje tri oblika cikličkih algoritamskih struktura:

1. **ponavljanje s izlazom na vrhu**
2. **ponavljanje s izlazom na dnu**
3. **ponavljanje s eksplicitnim brojačem.**

Slijedi opći prikaz navedenih oblika cikličke algoritamske strukture u pseudo kodu.

1.2.7.1. Ponavljanje s izlazom na vrhu

Opći se oblik cikličke algoritamske strukture s izlazom na vrhu u pseudo kodu može prikazati kako slijedi.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na vrhu:

```
  1.  Dok vrijedi uvjet ponavljaj
  2.     Algoritamski korak 1
  3.     Algoritamski korak 2
 ...     ...
  i.     Algoritamski korak x
i+1.  Kraj Dok vrijedi
i+2.  Algoritamski korak n
```

Algoritam 1.26.

Izvršitelj u 1. algoritamskom koraku provjerava je li zadovoljen `uvjet`. Ako je `uvjet` zadovoljen, izvode se `algoritamski koraci 1, 2, ...x` (oni tvore ponavljajući blok koraka). Nakon `algoritamskog koraka x`, izvršitelj ponovno provjerava je li zadovoljen `uvjet` (ponavljajući blok koraka može izmijeniti zadovoljenost `uvjeta`). Ako je `uvjet` zadovoljen, ponovno se izvode algoritamski koraci u ponavljajućem bloku koraka. Ako `uvjet` nije zadovoljen, izvršitelj više ne izvodi blok koraka, već se izvodi `algoritamski korak n` i to je kraj ponavljanja. Kako bi se blok koraka izveo barem jednom, `uvjet` mora biti zadovoljen prilikom prvog izvođenja 1. algoritamskog koraka. Kako se prije izvođenja bloka koraka prvo provjerava zadovoljenje `uvjeta`, ovo ponavljanje se naziva ponavljanje s izlazom na vrhu.

Ponavljanje s izlazom na vrhu se može realizirati i preko linijske i strukture bezuvjetnoga skoka.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na vrhu (realizacija preko linijske i strukture bezuvjetnoga skoka):

```
  1.  oznaka: Ako je uvjet onda
  2.     Algoritamski korak 1
  3.     Algoritamski korak 2
 ...     ...
  i.     Algoritamski korak x
i+1.     Skoči na oznaka.
i+2.  Kraj Ako je
i+3.  Algoritamski korak n
```

Algoritam 1.27.

1.2.7.2. Ponavljanje s izlazom na dnu

Opći se oblik cikličke algoritamske strukture s izlazom na dnu u pseudo kodu može prikazati kako slijedi.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na dnu:

```
  1.  Ponavljaj
  2.      Algoritamski korak 1
  3.      Algoritamski korak 2
 ...      ...
  i.      Algoritamski korak x
i+1.  Dok vrijedi uvjet
i+2.  Algoritamski korak n
```

Algoritam 1.28.

Izvršitelj u 1. algoritamskom koraku otkriva početak bloka koraka koji se treba ponavljati. Izvodi `algoritamske korake 1, 2, ...x`. Tek nakon izvođenja `algoritamskog koraka x` izvodi algoritamski korak u liniji i+1. – korak u komu provjerava zadovoljenost `uvjeta` (i ovdje ponavljajući blok koraka može izmijeniti zadovoljenost `uvjeta`). Ako je `uvjet` zadovoljen, ponovno izvodi `algoritamske korake 1, 2, ...x`, te ponovno dolazi do i+1. koraka u kojemu provjerava `uvjet`. Ako `uvjet` nije zadovoljen, izvršitelj više ne izvodi blok koraka, već se izvodi `algoritamski korak n` i to je kraj ponavljanja. Kako bi se blok koraka izveo barem jednom, `uvjet` ne mora biti zadovoljen prilikom prvog izvođenja 1. algoritamskog koraka (što je razlika u odnosu na ponavljanje s izlazom na vrhu). To znači da će se blok koraka u ovoj vrsti cikličke algoritamske strukture sigurno izvesti barem jednom. Kako se zadovoljenje `uvjeta` provjerava nakon izvođenja bloka koraka, ovo ponavljanje se naziva ponavljanje s izlazom na dnu.

Ponavljanje s izlazom na dnu se može realizirati i preko linijske i strukture bezuvjetnoga skoka.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na dnu (realizacija preko linijske i strukture bezuvjetnoga skoka):

```
  1.  oznaka: Algoritamski korak 1
  2.  Algoritamski korak 2
 ...  ...
  i.  Algoritamski korak x
i+1.  Ako je uvjet onda
i+2.      Skoči na oznaka.
i+3.  Kraj Ako je
i+4.  Algoritamski korak n
```

Algoritam 1.29.

1.2.7.3. Ponavljanje s eksplicitnim brojačem

Opći se oblik cikličke algoritamske strukture s izlazom na dnu u pseudo kodu može prikazati kako slijedi.

Opći primjer algoritma s cikličkom algoritamskom strukturom - eksplicitni brojač:

```
  1.  Ponavljaj za svaki i=i1 do i2 korak i3
  2.     Algoritamski korak 1
  3.     Algoritamski korak 2
 ...     ...
  j.     Algoritamski korak x
j+1.  Kraj Ponavljaj
j+2.  Algoritamski korak n
```

Algoritam 1.30.

Izvršitelj izvodi blok koraka koji se sastoji iz `algoritamskih koraka 1, 2, ...x` za svaki izračunati `i` koji ide od broja i_1 do broja i_2 uz korak i_3 (zato naziv ponavljanje s eksplicitnim brojačem). Nakon toga izvodi `algoritamski korak n`. "`Za svaki i = i1 do i2 korak i3`" predstavlja uvjet koji mora biti zadovoljen kako bi se izvodio blok koraka. Npr. ako se navede uvjet - "`za svaki i = 1 do 10 korak 2`" - vrijednost od `i` tijekom ponavljanja će biti redom brojevi 1, 3, 5, 7 i 9 što će uzrokovati ponavljanje bloka koraka od ukupno 5 puta. Općenito se broj ponavljanja može izračunati prema formuli: cijeli broj od $\{[(i_2-i_1)/i_3]\}+1$

Za prikazani primjer uvjeta formula daje sljedeći rezultat: cijeli broj od $\{[(10-1)/2]\}+1$ = cijeli broj od $\{4.5\}+1 = 4+1 = 5$

Ponavljanje s eksplicitnim brojačem se može realizirati i preko linijske i strukture bezuvjetnoga skoka.

Opći primjer algoritma s cikličkom algoritamskom strukturom - eksplicitni brojač (realizacija preko linijske i strukture bezuvjetnoga skoka):

```
  1.  i=i1
  2.  oznaka: Ako je i<=i2 onda
  3.     Algoritamski korak 1
  4.     Algoritamski korak 2
 ...     ...
  j.     Algoritamski korak x
j+1.     i=i+i3
j+2.     Skoči na oznaka.
j+3.  Kraj Ako je
j+4.  Algoritamski korak n
```

Algoritam 1.31.

Slijedi nekoliko primjera u kojima se pojavljuju opisane tri vrste cikličkih algoritamskih struktura.

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu) - izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina):

```
 1.  Ponavljaj
 2.    Doznaj masu m u kilogramima.
 3.    Doznaj visinu v u centimetrima.
 4.    Izračunaj BMI = m/(v/100)^2.
 5.    Reci da za masu m i visinu v indeks tjelesne mase
       iznosi BMI.
```

```
 6.    Ako je BMI<18.5 onda
 7.       Reci da indeks tjelesne mase BMI znači
          Pothranjenost.
 8.    inače ako je BMI<25 onda
 9.       Reci da indeks tjelesne mase BMI znači
          Normalna tjelesna težina.
10.    inače ako je BMI<30 onda
11.       Reci da indeks tjelesne mase BMI znači
          Prekomjerna tjelesna težina.
12.    inače ako je BMI<40 onda
13.       Reci da indeks tjelesne mase BMI znači
          Pretilost.
14.    inače
15.       Reci da indeks tjelesne mase BMI znači
          Izrazita pretilost.
16.    Kraj Ako je
17.    Doznaj želi li se računati novi indeks tjelesne mase
       (Da/Ne).
18. Dok vrijedi odgovor = Da
```

Algoritam 1.32.

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu) - koliko kalendarski mjesec ima dana:

```
 1. Ponavljaj
 2.    Doznaj broj mjeseca M.
 3.    Ako je M<1 ili M>12 onda
 4.       Reci da M. mjesec ne postoji.
 5. Dok vrijedi M<1 ili M>12
 6. Ako je M=1 ili M=3 ili M=5 ili M=7 ili M=8 ili M=10 ili
    M=12 onda
 7.    Reci da M. mjesec ima 31 dan.
 8. inače ako je M=4 ili M=6 ili M=9 ili M=11 onda
 9.    Reci da M. mjesec ima 30 dana.
10. inače
11.    Reci da M. mjesec ima 28 ili 29 dana, ovisno je li
       godina prijestupna ili ne.
12. Kraj Ako je
```

Algoritam 1.33.

U prvome primjeru (Algoritam 1.32.), izračun se indeksa tjelesne mase ponavlja dok god vrijedi uvjet `odgovor=Da` (tj. dok se želi računati novi indeks tjelesne mase). U drugome se primjeru (Algoritam 1.33.) doznaje se broj mjeseca sve dok taj broj nije ispravan, tj. dok vrijedi `M<1` ili `M>12`. Tek kad je broj mjeseca ispravan, izvode se sljedeći algoritamski koraci (otkriva se broj dana u mjesecu).

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na vrhu) - izračun sume prirodnih brojeva do nekoga broja (npr. za 3 treba izračunati 1+2+3):

```
 1. Broj=0.
 2. Dok vrijedi Broj<1 Ponavljaj
 3.    Doznaj neki prirodni broj Broj.
```

```
 4.  Kraj Dok vrijedi
 5.  Suma=0.
 6.  Brojac=1.
 7.  Dok vrijedi Brojac<=Broj
 8.     Suma=Suma+Brojac.
 9.     Brojac=Brojac+1.
10.  Kraj Dok vrijedi
11.  Reci suma prirodnih brojeva do broja Broj iznosi Suma.
```

Algoritam 1.34.

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na vrhu) - izračun prosječne temperature u 10 očitanja:

```
1.  SumaTemp=0
2.  Brojac=1
3.  Dok vrijedi Brojac<=10
4.     Doznaj iznos temperature Temp.
5.     SumaTemp=SumaTemp+Temp.
6.     Brojac=Brojac+1
7.  Kraj Dok vrijedi
8.  ProsjekTemp=SumaTemp/10.
9.  Reci prosječna temperatura u 10 očitanja iznosi
    ProsjekTemp.
```

Algoritam 1.35.

U algoritmu 1.34. doznaje se `broj` dok god on nije valjan prirodni broj (uvjet je `Broj<1`). Nakon toga se u nazivima `Suma` i `Brojac` (varijablama) stavljaju vrijednosti 0 odnosno 1. Zatim se dolazi do novog uvjeta za ponavljanje gdje se provjerava uvjet `Brojac<=Broj`. Npr. ako se u `Broj` nalazi vrijednost 3, tada je uvjet zadovoljen, tj. ima vrijednost istina (jer je 1<=3). To znači da se izvodi blok koraka. Računa se nova `Suma` tako da se trenutnoj vrijednosti (a to je 0) doda vrijednost koju ima `Brojac` (a to je 1). Zbog toga sada `Suma` ima vrijednost 1. Nakon toga se vrijednost u `Brojac` povećava za 1 te se sada u `Brojac` nalazi vrijednost 2.

Ponovno se provjerava uvjet ponavljanja `Brojac<=Broj` (tj. 2<=3). Kako je uvjet ispunjen ponavlja se blok koraka u kome se računa nova `Suma` tako da se trenutnoj vrijednosti (a to je sada 1) doda vrijednost `Brojaca` (a to je 2) pa se sada u `Suma` nalazi vrijednost 3. Potom se `Brojac` povećava za 1. Kako je on imao vrijednost 2, sada je njegova vrijednost 3.

Ponovno se provjerava uvjet ponavljanja `Brojac<=Broj` (tj. 3<=3). Ponovno je uvjet ispunjen i ponovno se izvodi blok koraka. `Suma` se računa kao trenutna vrijednost (a to je 3) zbrojena s `Brojac` (a to je 3). Sada `Suma` iznosi 6. `Brojac` se ponovno povećava za 1 i iznosi 4.

Ponovno se provjerava uvjet ponavljanja `Brojac<=Broj` (tj. 4<=3). Sada uvjet ponavljanja nije ispunjen (nema vrijednost istina), te se blok naredbi više ne ponavlja. Skače se na korak 11. tj., izvođač treba reći da je suma prirodnih brojeva do broja 3 (to

je vrijednost od `Broj`) iznosi 6 (to je vrijednost od `Suma`). Analogno se može objasniti što se događa u algoritmu 1.35.

Isti se primjeri za ponavljanje s izlazom na vrhu mogu prikazati i s ponavljanjem s eksplicitnim brojačem.

Primjer algoritma s cikličkom algoritamskom strukturom (eksplicitni brojač) - izračun sume prirodnih brojeva do nekoga broja (npr. za 3 treba izračunati 1+2+3):

```
1.  Ponavljaj
2.     Doznaj neki prirodni broj Broj.
3.  Dok vrijedi Broj<1
4.  Suma=0.
5.  Ponavljaj za svaki Brojac=1 do Broj korak 1
6.     Suma=Suma+Brojac.
7.  Kraj Ponavljaj
8.  Reci suma prirodnih brojeva do broja Broj iznosi Suma.
```

Algoritam 1.36.

Primjer algoritma s cikličkom algoritamskom strukturom (eksplicitni brojač) - izračun prosječne temperature u 10 očitanja:

```
1.  SumaTemp=0
2.  Ponavljaj za svaki Brojac=1 do 10 korak 1
3.     Doznaj iznos temperature Temp.
4.     SumaTemp=SumaTemp+Temp.
5.  Kraj Ponavljaj.
6.  ProsjekTemp=SumaTemp/10.
7.  Reci prosječna temperatura u 10 očitanja iznosi
    ProsjekTemp.
```

Algoritam 1.37.

1.2.8. Ciklička algoritamska struktura prikazana blok dijagramom

Kako je već rečeno, postoje tri oblika cikličke algoritamske strukture: ponavljanje s izlazom na vrhu, ponavljanje s izlazom na dnu i ponavljanje s eksplicitnim brojačem. Sve se one mogu realizirati preko razgranate algoritamske strukture i algoritamske strukture bezuvjetnoga skoka.

1.2.8.1. Ponavljanje s izlazom na vrhu

Općenito se u pseudo kodu ponavljanje s izlazom na vrhu može prikazati kako slijedi:

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na vrhu:

```
  1.  Dok vrijedi uvjet ponavljaj
  2.      Algoritamski korak 1
  3.      Algoritamski korak 2
 ...      ...
  i.      Algoritamski korak x
i+1.  Kraj Dok vrijedi
i+2.  Algoritamski korak n
```

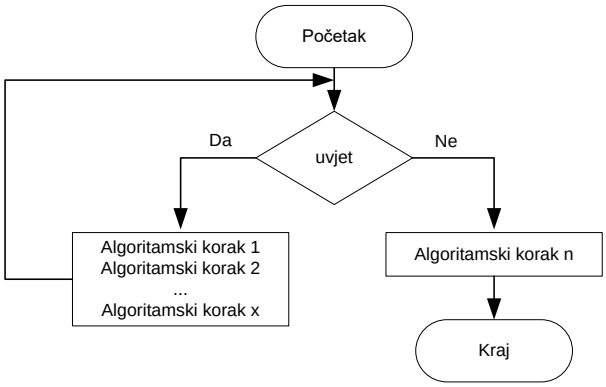
Algoritam 1.38.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na vrhu (realizacija preko linijske i strukture bezuvjetnoga skoka):

```
  1.  A: Ako je uvjet onda
  2.      Algoritamski korak 1
  3.      Algoritamski korak 2
 ...      ...
  i.      Algoritamski korak x
i+1.      Skoči na A.
i+2.  Kraj Ako je
i+3.  Algoritamski korak n
```

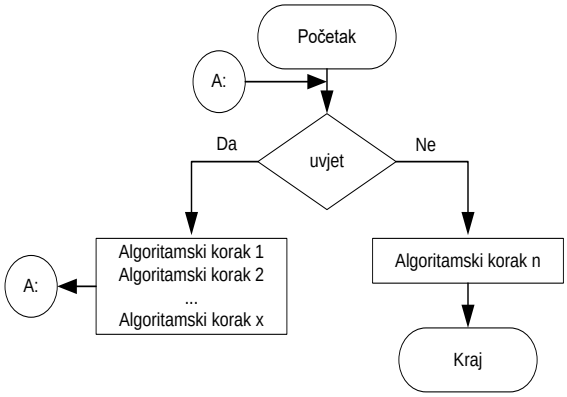
Algoritam 1.39.

Ponavljanje se s izlazom na vrhu blok dijagramom prikazuje na sljedeći način:



Dijagram 1.14. Opći primjer algoritma s cikličkom algoritamskom strukturom – izlaz na vrhu

U blok dijagramu, nakon simbola odluke postoje dva toka. U jednome se toku nalazi izvođenje algoritamskih koraka iza kojih tok ponovno vodi do provjere je li zadovoljen uvjet, dok se u drugome toku izvode drugi algoritamski koraci te se završava s algoritmom.



Dijagram 1.15. Opći primjer algoritma s cikličkom algoritamskom strukturom – izlaz na vrhu (realizacija preko linijske i strukture bezuvjetnoga skoka)

1.2.8.2. Ponavljanje s izlazom na dnu

Ponavljanje s izlazom na dnu se u pseudo kodu prikazuje kako slijedi.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na dnu:

```
  1.  Ponavljaj
  2.     Algoritamski korak 1
  3.     Algoritamski korak 2
 ...     ...
  i.     Algoritamski korak x
i+1.  Dok vrijedi uvjet
i+2.  Algoritamski korak n
```

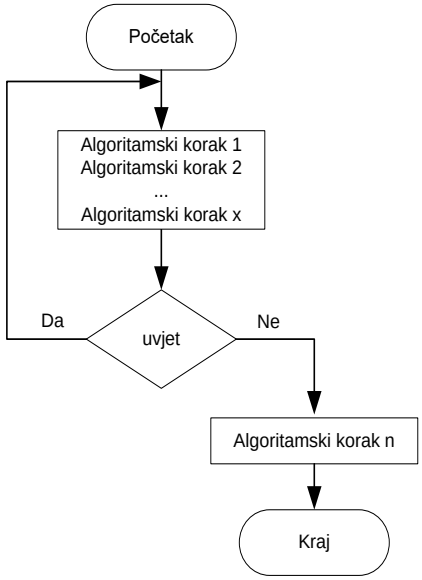
Algoritam 1.40.

Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na dnu (realizacija preko linijske i strukture bezuvjetnoga skoka):

```
  1.  A: Algoritamski korak 1
  2.  Algoritamski korak 2
 ...  ...
  i.  Algoritamski korak x
i+1.  Ako je uvjet onda
i+2.     Skoči na A.
i+3.  Kraj Ako je
i+4.  Algoritamski korak n
```

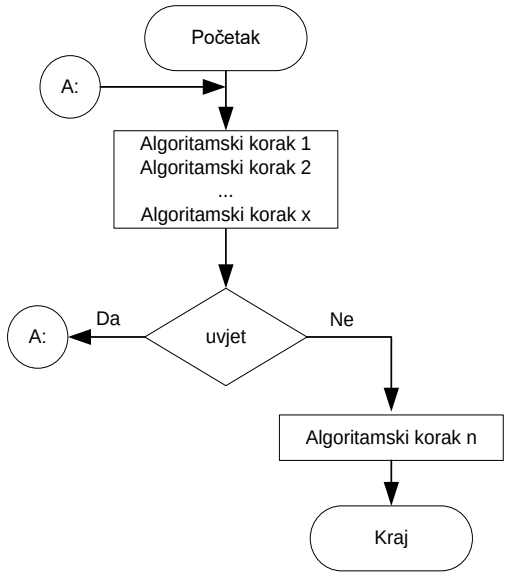
Algoritam 1.41.

Algoritamska se struktura ponavljanja s izlazom na dnu blok dijagramom prikazuje na sljedeći način.



Dijagram 1.16. Opći primjer algoritma s cikličkom algoritamskom strukturom - izlaz na dnu

Nakon simbola za obradu, slijedi tok i simbol za odluku iza koga idu dva toka. Ako je uvjet u odluci zadovoljen, tada tok izvođenje algoritma ponovno ide na simbol za obradu.



Dijagram 1.17. Opći primjer algoritma s cikličkom algoritamskom strukturom – izlaz na dnu (realizacija preko linijske i strukture bezuvjetnoga skoka)

1.2.8.3. Ponavljanje s eksplicitnim brojačem

Ponavljanje s eksplicitnim brojačem se pseudo kodom prikazuje na sljedeći način.

Opći primjer algoritma s cikličkom algoritamskom strukturom - eksplicitni brojač:	
1.	Ponavljaj **za svaki i=i_1 do i_2 korak i_3**
2.	Algoritamski korak 1
3.	Algoritamski korak 2
...	...
j.	Algoritamski korak x
j+1.	Kraj Ponavljaj
j+2.	Algoritamski korak n

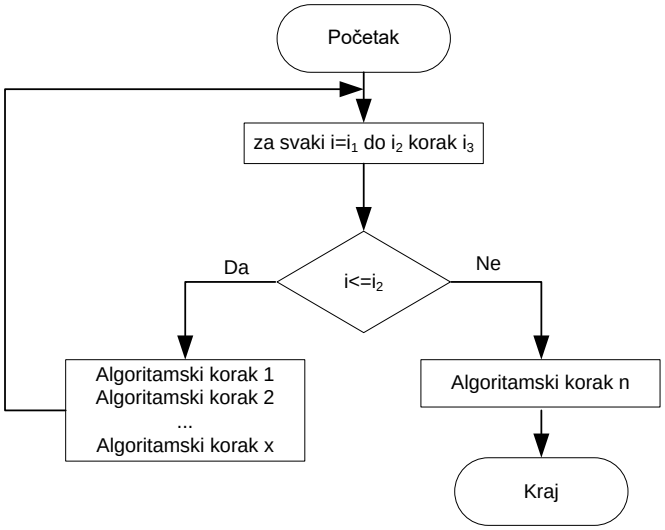
Algoritam 1.42.

Opći primjer algoritma s cikličkom algoritamskom strukturom - eksplicitni brojač (realizacija preko linijske i strukture bezuvjetnog skoka):	
1.	**i**=i_1
2.	**A:** Ako je **i**<=i_2 onda
3.	Algoritamski korak 1
...	Algoritamski korak 2
j.	...
j+1.	Algoritamski korak x
j+2.	**i**=**i**+i_3

```
j+3.    Skoči na A.
j+4.  Kraj Ako je
j+5.  Algoritamski korak n
```

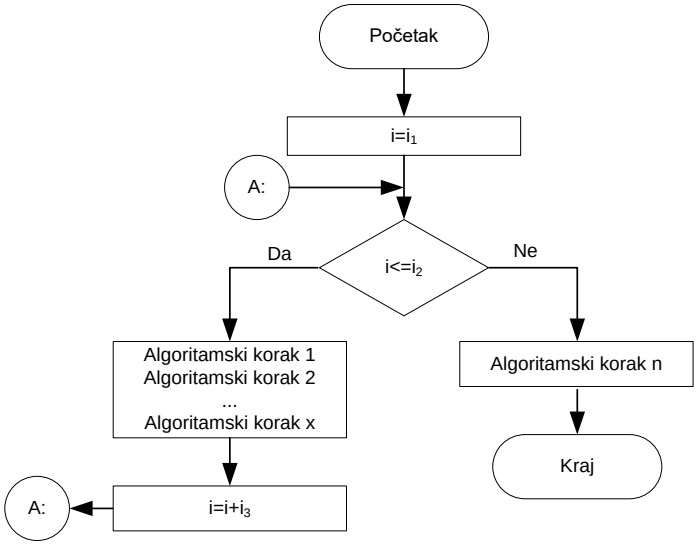
Algoritam 1.43.

Slijedi prikaz ove algoritamske strukture blok dijagramom.



Dijagram 1.18. Opći primjer algoritma s cikličkom algoritamskom strukturom – eksplicitni brojač

Blok dijagram ima simbol za obradu u kome stoji da se u obzir treba uzeti *svaki i u intervalu od i_1 do i_2 ali s korakom i_3*. Nakon toga slijedi provjera je li *i manji od i_3* (kraja intervala). Ako jest, izvode se algoritamski koraci te se ponovno izvodi izračun novoga *i*.



Dijagram 1.19. Opći primjer algoritma s cikličkom algoritamskom strukturom – eksplicitni brojač (realizacija preko linijske i strukture bezuvjetnoga skoka)

1.2.9. Primjeri prikaza cikličkih algoritamskih struktura blok dijagramom

Slijedi niz primjera cikličkih algoritamskih struktura prije prikazanih u pseudo kodu.

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu) - izračun indeksa tjelesne mase prema formuli BMI = m/v^2 (m - masa; v - visina):

```
 1. Ponavljaj
 2.   Doznaj masu m u kilogramima.
 3.   Doznaj visinu v u centimetrima.
 4.   Izračunaj BMI = m/(v/100)^2.
 5.   Reci da za masu m i visinu v indeks tjelesne mase
      iznosi BMI.
 6.   Ako je BMI<18.5 onda
 7.     Reci da indeks tjelesne mase BMI znači
        Pothranjenost.
 8.   inače ako je BMI<25 onda
 9.     Reci da indeks tjelesne mase BMI znači
        Normalna tjelesna težina.
10.   inače ako je BMI<30 onda
11.     Reci da indeks tjelesne mase BMI znači
        Prekomjerna tjelesna težina.
12.   inače ako je BMI<40 onda
13.     Reci da indeks tjelesne mase BMI znači
        Pretilost.
14.   inače
15.     Reci da indeks tjelesne mase BMI znači
```

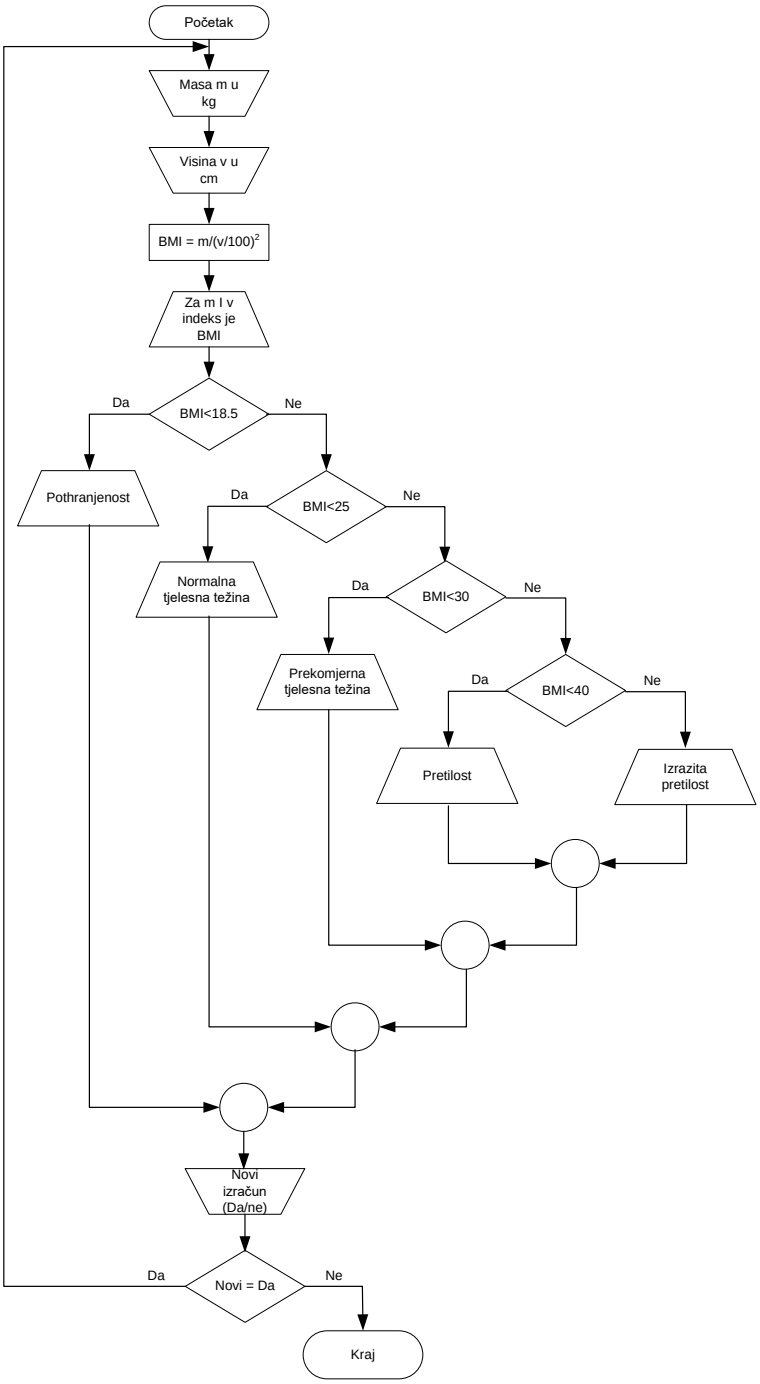
```
      Izrazita pretilost.
16.   Kraj Ako je
17.   Doznaj želi li se računati novi indeks tjelesne mase
      (Da/Ne).
18. Dok vrijedi odgovor = Da
```

Algoritam 1.44.

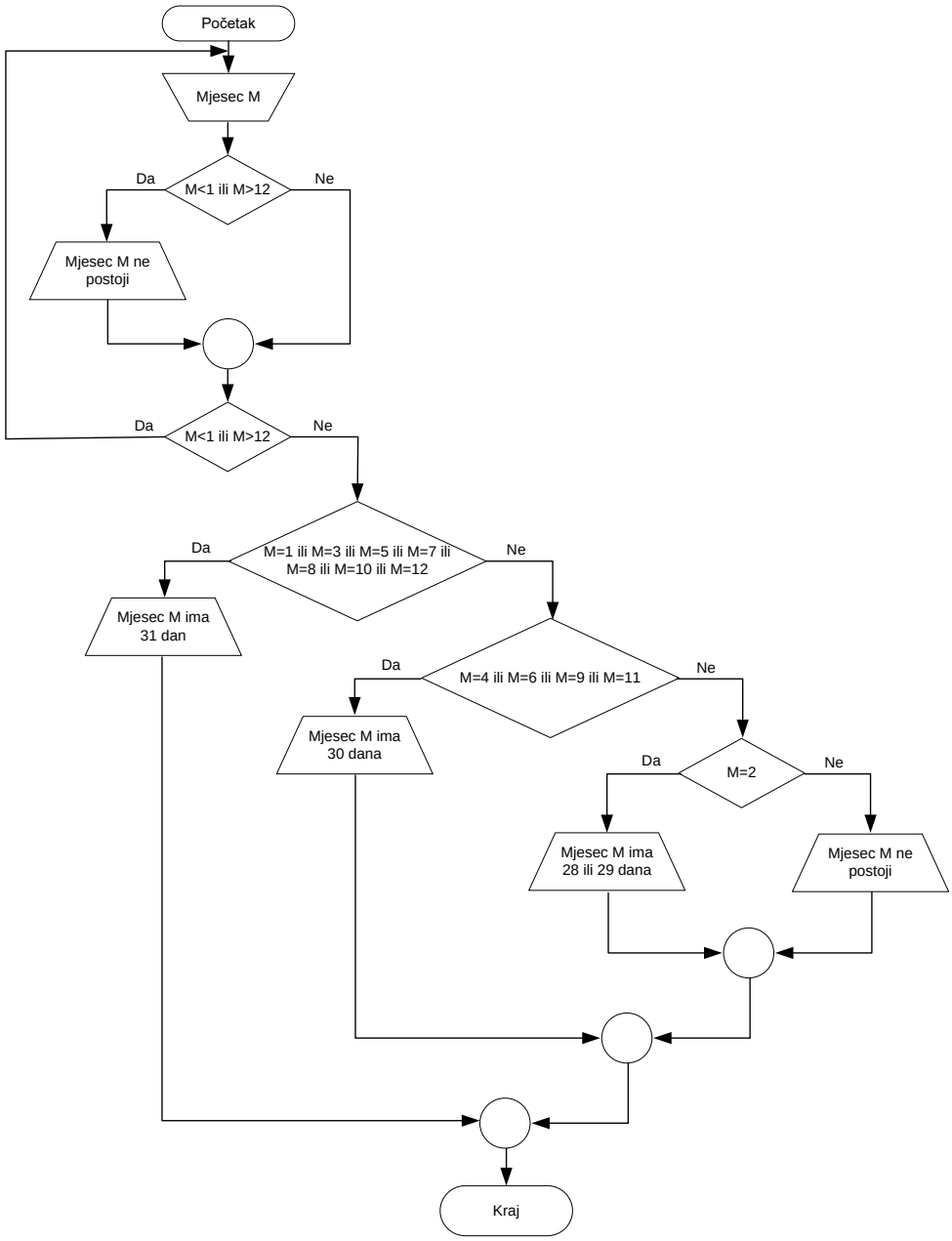
Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu) - koliko kalendarski mjesec ima dana:

```
 1. Ponavljaj
 2.   Doznaj broj mjeseca M.
 3.   Ako je M<1 ili M>12 onda
 4.     Reci da M. mjesec ne postoji.
 5. Dok vrijedi M<1 ili M>12
 6. Ako je M=1 ili M=3 ili M=5 ili M=7 ili M=8 ili M=10 ili
 7. M=12 onda
 8.   Reci da M. mjesec ima 31 dan.
 9. inače ako je M=4 ili M=6 ili M=9 ili M=11 onda
10.   Reci da M. mjesec ima 30 dana.
11. inače
12.   Reci da M. mjesec ima 28 ili 29 dana, ovisno je li
      godina prijestupna ili ne.
13. Kraj Ako je
```

Algoritam 1.45.



Dijagram 1.20. Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu) – izračun indeksa tjelesne mase prema formuli BMI = m/v2 (m - masa; v - visina)

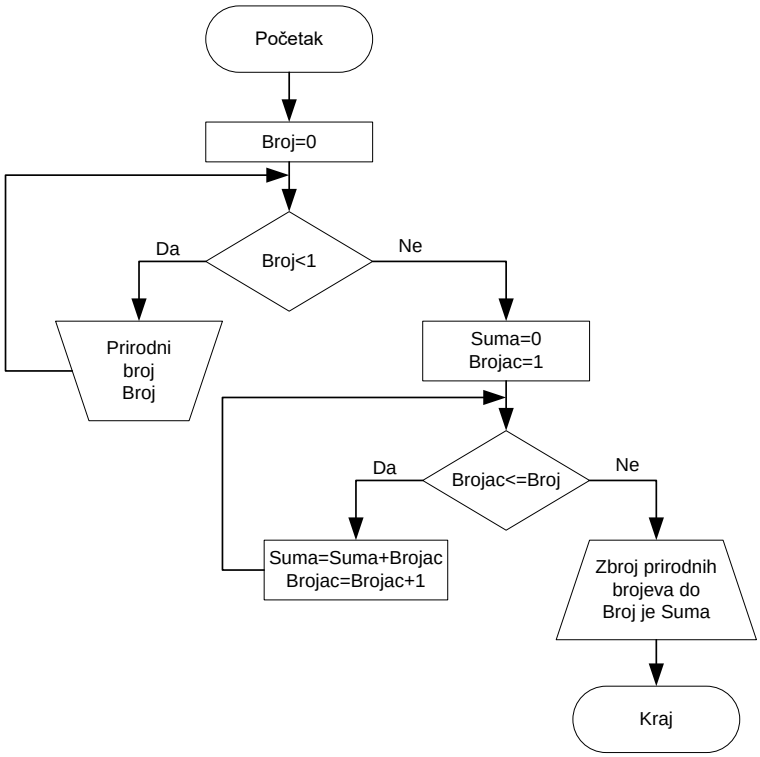


Dijagram 1.21. Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu) – koliko kalendarski mjesec ima dana

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na vrhu) - izračun sume prirodnih brojeva do nekoga broja (npr. za 3 treba izračunati 1+2+3):

```
1.  Broj=0.
2.  Dok vrijedi Broj<1 Ponavljaj
3.    Doznaj neki prirodni broj Broj.
4.  Kraj Dok vrijedi
5.  Suma=0.
6.  Brojac=1.
7.  Dok vrijedi Brojac<=Broj
8.    Suma=Suma+Brojac.
9.    Brojac=Brojac+1.
10. Kraj Dok vrijedi
11. Reci suma prirodnih brojeva do broja Broj iznosi Suma.
```

Algoritam 1.46.



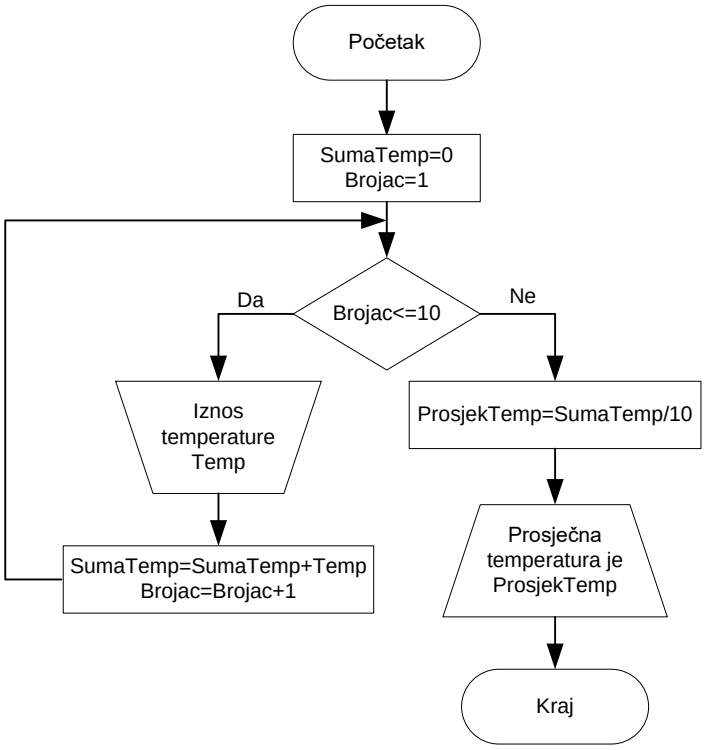
Dijagram 1.22. Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na vrhu) – izračun sume prirodnih brojeva do nekoga broja (npr. za 3 treba izračunati 1+2+3)

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na vrhu) - izračun prosječne temperature u 10 očitanja:

```
1.  SumaTemp=0
2.  Brojac=1
3.  Dok vrijedi Brojac<=10
4.     Doznaj iznos temperature Temp.
5.     SumaTemp=SumaTemp+Temp.
```

```
6.      Brojac=Brojac+1
7.  Kraj Dok vrijedi
8.  ProsjekTemp=SumaTemp/10.
9.  Reci prosječna temperatura u 10 očitanja iznosi
    ProsjekTemp.
```

Algoritam 1.47.

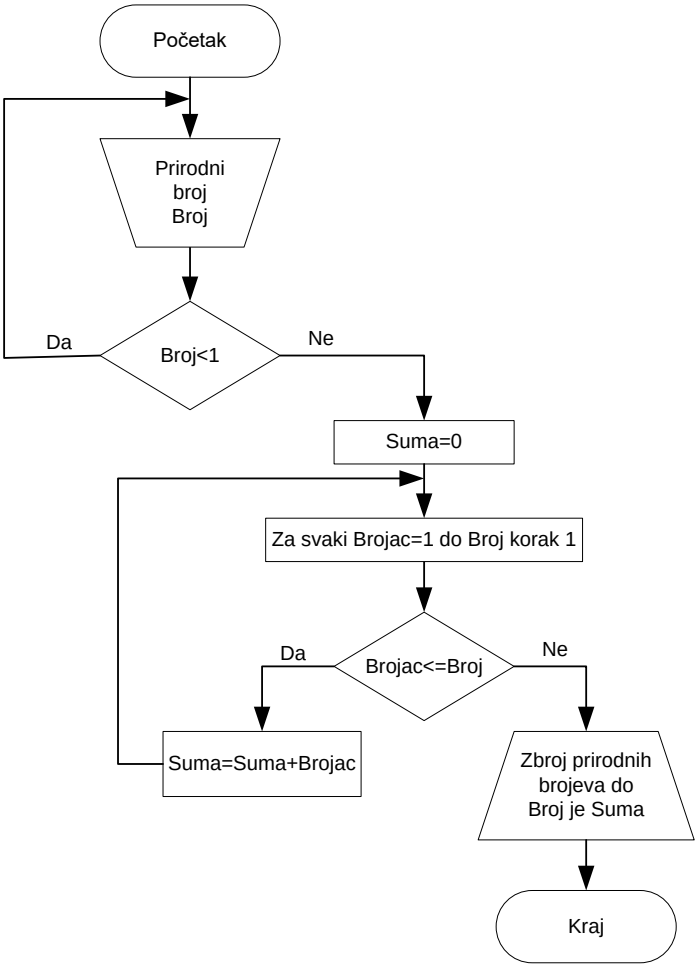


Dijagram 1.23. Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na vrhu) – izračun prosječne temperature u 10 očitanja

Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu i eksplicitni brojač) - izračun sume prirodnih brojeva do nekoga broja (npr. za 3 treba izračunati 1+2+3):

```
1.  Ponavljaj
2.      Doznaj neki prirodni broj Broj.
3.  Dok vrijedi Broj<1
4.  Suma=0.
5.  Ponavljaj za svaki Brojac=1 do Broj korak 1
6.      Suma=Suma+Brojac.
7.  Kraj Ponavljaj
8.  Reci suma prirodnih brojeva do broja Broj iznosi Suma.
```

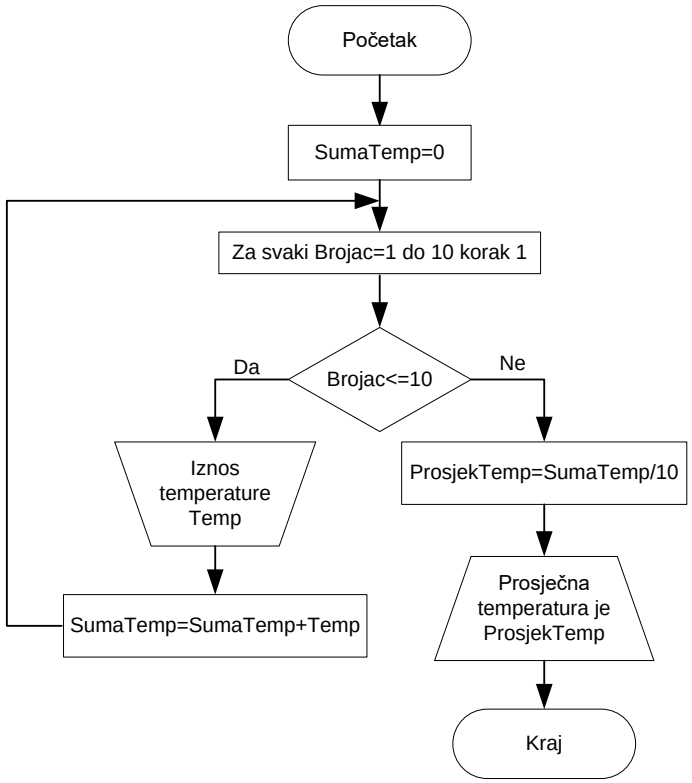
Algoritam 1.48.



Dijagram 1.24. Primjer algoritma s cikličkom algoritamskom strukturom (izlaz na dnu i eksplicitni brojač) – izračun sume prirodnih brojeva do nekoga broja (npr. za 3 treba izračunati 1+2+3)

Primjer algoritma s cikličkom algoritamskom strukturom (eksplicitni brojač) - izračun prosječne temperature u 10 očitanja:	
1.	**SumaTemp**=0
2.	Ponavljaj za svaki **Brojac**=1 do 10 korak 1
3.	Doznaj iznos temperature **Temp.**
4.	**SumaTemp**=**SumaTemp**+**Temp.**
5.	Kraj Ponavljaj.
6.	**ProsjekTemp**=**SumaTemp**/10.
7.	Reci prosječna temperatura u 10 očitanja iznosi **ProsjekTemp.**

Algoritam 1.49.



Dijagram 1.25. Primjer algoritma s cikličkom algoritamskom strukturom (eksplicitni brojač) – izračun prosječne temperature u 10 očitanja

1.3. Pojam računalni program, programski jezik, programiranje

Algoritam, dakle, predstavlja opis rješenja nekoga problema. Rješenje se nekoga problema (algoritam) može opisati tekstualno, grafički, pseudo kodom, kombinacijom grafičkog opisa i pseudo koda te programskim jezikom. Opisom algoritma nekim programskim jezikom nastaje računalni program. Računalni program predstavlja algoritam opisan nekim programskim jezikom i time prilagođen izvođenju na računalu. Računala ne mogu funkcionirati bez računalnoga programa koji izvode.

Programski je jezik umjetno stvoren jezik koji je namijenjen davanju instrukcija računalima. Ovim se jezikom algoritam može precizno opisati kao niz instrukcija koje računalo izvodi. Programski jezik se sastoji iz dvije osnove komponente - **sintakse** (forme) i **semantike** (značenja). Sintaksa programskoga jezika kaže kakva forma instrukcije mora biti, a kako bi je računalo razumjelo (npr. u govornome jeziku je jasno da je riječ "Prgrm" pogrešna – greška sintakse – jer je ispravno "Program"). Spomenute se greške lako otkrivaju (samo računalo ih otkriva) i lako ispravljaju. Semantika je značenje onoga što se programskim jezikom opisalo. Primjera radi,

programskim se jezikom opisao izračun prosječne ocjene iz pet kolegija (ocjene su *k1, k2, k3, k4* i *k5*). U jednom se dijelu algoritma nalazi formula za izračun prosječne ocjene koja glasi *Prosjek=(k1+k2+k3+k4+k5)/2*. Računalo će prema navedenoj formuli izračunati prosjek ocjena. Jasno je da formula nije ispravna (dijeli se s dva umjesto s pet). No, računalo ovu semantičku pogrešku (pogrešno značenje) neće primijetiti te će dati pogrešan rezultat. Otkrivanje semantičkih pogrešaka u računalnome programu nije jednostavno i postoji posebni alat (*engl. debugger*) koji ovaj postupak olakšava.

Nekim je programskim jezikom algoritam prilagođen izvođenju na računalu i dobiven je računalni program koji je u formi izvornoga koda, tj. u formi čitljivoj ljudima. Postoje dva osnovna načina na koja računalo izvodi računalni program - interpretiranjem instrukcija u računalnome programu i izvođenjem izvršne verzije računalnoga programa.

Pri interpretiranju instrukcija, na računalu mora postojati posebni računalni program - **interpreter**. Interpreter učitava računalni program, zatim čita instrukciju po instrukciju, tumači ju i izvršava. Budući da tumačenje instrukcija troši vrijeme, interpreteri su spori. Računalni program nije u izvršnome obliku (strojnome kodu). On može biti u izvornome obliku (izvornome kodu) ili u nekome posebnom obliku potrebnom interpreteru (npr. *Bytecode* kod *JVM – Java Virtual Machine*).

Pri izvođenju izvršne verzije računalnoga programa (ekstenzija *EXE*), on se treba iz izvornoga koda preoblikovati u strojni kod. Ovo preoblikovanje izvodi posebni računalni program - **kompajler**. On učita cijeli računalni program te ga pretvori u poseban oblik koji računalo može izravno izvesti, a da ne treba dodatni računalni program. Računalni program u strojnome kodu nije čitljiv ljudima, već računalima.

Programiranje (računalno programiranje) je proces oblikovanja, pisanja, testiranja, pronalaženja i uklanjanja greški te održavanja izvornoga (ne strojnoga) koda računalnoga programa. Osoba koja stvara računalni program programiranjem naziva se programer. Programer polazi od zadanoga problema, oblikuje algoritam koji problem rješava te ga opisuje nekim programskim jezikom. Učenje programiranja uključuje učenje sintakse (forme) nekoga programskog jezika i stjecanje osnovnih intuitivnih znanja glede algoritmizacije problema opisanoga riječima.

Postoje četiri osnovna pristupa u programiranju:

1. **proceduralno programiranje**
2. **objektno orijentirano programiranje**
3. **funkcionalno programiranje**
4. **logičko programiranje.**

Proceduralno programiranje se još naziva i algoritamsko programiranje jer se kod njega rješenje problema opisuje tako da se navode koraci (instrukcije) koje računalo treba izvesti. Računalnim programom se daje odgovor na pitanje kako se dolazi do rješenja (imperativno programiranje). Ako je problem složen, onda se on razbija na manje probleme koji se zatim rješavaju. Iz rješenja manjih problema proizlaze

procedure. Pozivajući procedure (dakle rješavajući manje probleme) računalo rješava glavni problem. Koriste se proceduralni programski jezici kao što su C, Pascal, Basic itd.

Pri **objektno orijentiranome** programiranju, glavni se problem razbija u manje probleme čija su rješenja predstavljena objektima. Objekti imaju stanja i metode. Računalo mijenja stanja objekata i poziva njegove metode te na taj način rješava glavni problem. Glavna poteškoća ovoga pristupa je uočavanje objekata koji su potrebni za rješavanje glavnoga problema. I ovdje se računalnim programom daje odgovor na pitanje kako se dolazi do rješenja (imperativno programiranje). Koriste se objektno orijentirani programski jezici kao što su C++, Java, C# itd.

Funkcionalno programiranje spada u skupinu deklarativnoga programiranja u kojem se računalu kaže što treba napraviti (a ne kako, kao kod imperativnog programiranja). Za rješavanje problema izrađuju se funkcije. Koriste se funkcionalni programski jezici kao što su Lisp, Haskell, ML.

Logičko programiranje također spada u skupinu deklarativnoga programiranja. Temelji se na matematičkoj logici, odnosno predikatnome računu (npr. ako je broj > 0 onda je broj pozitivan). Koriste se logički programski jezici (npr. Prolog).

Računalni programi se mogu podijeliti na sistemske računalne programe, računalne programe za programiranje i aplikativne računalne programe.

Sistemski računalni programi su računalni programi koji neposredno upravljaju računalnim sklopovljem, a kako bi podržali osnovne funkcionalnosti (npr. prikaz na monitoru i sl.) te računalni programi koji su platforma za pokretanje aplikativnih računalnih programa. Sistemski računalni programi uključuju različite pokretačke programe (*engl. drivers*), operacijske sustave (Linux, Windows itd.), poslužitelje (*engl. server* - npr. Web, baze podataka itd.), pomoćne programe (npr. antivirusni programi itd.) itd.

Računalni programi za programiranje su računalni programi koje koriste programeri kako bi u nekome programskom jeziku opisali algoritam i time izradili novi računalni program. Obično se sastoji iz kompajlera, interpretera, alata za otkrivanje greški (*engl. debugger*), alata za pisanje izvornoga koda (pisanje teksta) i alata za povezivanje programskih modula (*engl. linkers*).

Aplikativni računalni programi su računalni programi koji omogućavaju praktičnu primjenu računala. Primjer aplikativnih računalnih programa su: tekstualni procesori (npr. Word, Writer, Google Docs itd.), tablični kalkulatori (npr. Excel, Calc itd.), poslovni računalni programi (npr. ERP rješenja itd.), internet preglednici (npr. Mozzila Firefox, Google Chrome, Internet Explorer itd.) itd.

1.4. Razvoj programskih jezika

Na prvim elektroničkim računalima, programiranje se nije provodilo programskim jezikom već fizičkim lemljenjem pojedinih elektroničkih dijelova (projekt Eniac - prvo elektroničko računalo 1946. godine).

Prvi je programski jezik je razvio Nijemac Konrad Zuse 1946., a zvao se Plankalkul i on nikada nije primijenjen na elektroničkome računalu. Prvi jezik koji je primijenjen na elektroničkim računalima bio je Short-code1949. godine.

Programski jezici se klasificiraju u pet generacija.

Programski jezici prve generacije (1GL) su prvi programski jezici koji predstavljaju pisanje programa direktno u strojnome (binarnom) kodu. Za izvođenje ovih programa nije potreban interpreter niti kompajler, već ih je računalo odmah izvelo. Programeri su računalni program unosili u računalo preko bušenih kartica, traka i prekidača. Računalni se programi pisani u ostalim generacijama programskih jezika prije izvođenja na računalu trebaju prevesti u strojni kod. Svaka vrsta procesora ima svoju formu strojnoga koda.

Programiranje u strojnome kodu je izrazito teško, te se ubrzo razvijaju **programski jezici druge generacije** (2GL) - asemblerski jezik. Naime, programerima je puno lakše pamtiti i razumjeti kratke riječi nego binarni broj. Zbog toga se naredbe računalu umjesto binarno, predstavljaju simboličkim kodovima (npr. save, move, load, store, add ...). Svaki simbolički kod se prije izvođenja na računalu pretvara u jedan odgovarajući strojni kod. Slično kao i pri strojnome kodu, asemblerski kod je vezan za određenu vrstu procesora.

Iako je asemblerski jezik donekle olakšao pisanje računalnih programa, on je nepraktičan i težak pri pisanju složenijih računalnih programa. Također je i veliki problem specifičnosti asemblerskoga koda za pojedinu vrstu procesora (da je npr. MS Word pisan u asembleru, morala bi postojati posebna verzija za Pentium 4 i Dual core procesore, a da se ne govori o procesorima različitih proizvođača - npr. Intel i AMD procesori). Spomenute probleme rješavaju **programski jezici treće generacije** (3GL). Računalni se program napisan u programskome jeziku treće generacije konvertira u strojni kod pri čemu se jedna linija koda često pretvara u stotine linija strojnoga koda. Budući da posebni programi (kompajleri) konvertiraju napisani računalni program u strojni kod, to oni više nisu ovisni o vrsti procesora. Programskim se jezicima treće generacije računalu naređuje što treba napraviti kako bi se došlo do rješenja. Ovaj način programiranja je imperativan - napisane naredbe se izvode jedna za drugom. Primjeri jezika treće generacije su Java, PHP, C, C++, C#, Pascal, Visual Basic, Fortran, Cobol itd. Programski jezici treće generacije isprva omogućavaju samo proceduralno, a nakon toga (negdje od 1980. godine) i objektno orijentirano programiranje.

Programski jezici četvrte generacije (4GL) jesu programski jezici koji su orijentirani na ono što se treba riješiti (deklarativno programiranje), a ne kako (imperativno programiranje). Najčešće se odnose na pristup bazama podataka. Od svih

generacija najbliži su ljudskome jeziku. Primjer programskoga jezika ove generacije je SQL, ali i razni jezici za izradu izvještaja (npr. Oracle Reports) i grafičkih korisničkih sučelja (npr. XUL). Programski jezici četvrte generacije imaju tendenciju neovisnosti o računalnim platformama i vrstama procesora.

Posljednja generacija programskih jezika je **peta generacija** (5GL). U petu generaciju spadaju programski jezici namijenjeni programiranju umjetne inteligencije (ekspertni sustavi, neuronske mreže itd.). Primjer jezika pete generacije su Prolog, OPS5 i Mercury.

1.5. Životni ciklus razvoja računalnoga programa

Programeri, prilikom razvoja računalnoga programa ne kreću odmah u pisanje računalnoga koda. Oni prate plan, metodiku koja cjelokupni proces razvoja računalnoga programa razbija u manje međusobno povezane korake. Postoje različite metodike, tj. različiti popisi i definicije ovih koraka, ali svi su oni izvedeni iz jednoga općeg životnog ciklusa razvoja računalnoga programa (vodiča za programere) koji se sastoji iz šest koraka:

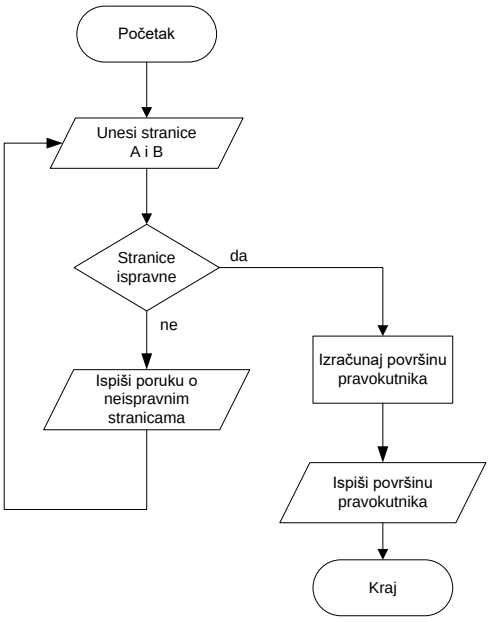
1. **analiza problema**
2. **dizajn računalnoga programa**
3. **kodiranje računalnoga programa**
4. **testiranje računalnoga programa i ispravljanje pogrešaka**
5. **formalizacija i dokumentiranje računalnoga programa**
6. **održavanje računalnoga programa.**

Analiza problema se sastoji iz preciznoga definiranja problema koji se rješava te pisanja specifikacije računalnoga programa, odnosno opisa njegovih ulaza, obrade i izlaza te korisničkoga sučelja. Programeru od pomoći može biti dijagram definiranja, odnosno tablica od tri stupca (Ulaz, Obrada i Izlaz) u koju se unose svi ulazi u računalni program, opisuju se sve obrade koje računalni program treba obaviti te se popisuju svi izlazi koje računalni program treba dati. Npr. za računalni program koji treba izračunati površinu pravokutnika uz dane stranice, dijagram definiranja bi izgledao kako slijedi.

Ulaz	Obrada	Izlaz
1. Stranica a 2. Stranica b	1. Provjeriti jesu li stranice ispravne (veće od 0) 2. Izračunati površinu pravokutnika	1. Poruka o neispravnim stranicama 2. Površina pravokutnika

Dijagram 1.26. Dijagram definiranja za računalni program izračuna površine pravokutnika

Dizajn računalnoga programa se sastoji iz detaljnog opisa aktivnosti računalnoga programa grupiranih u module, zatim od opisa rješenja, odnosno algoritma za svaki modul (pseudo kodom, blok dijagramom itd.) te testiranja algoritama. Veliki se problem treba razbiti u manje probleme. Potrebno je definirati sve korake koje računalni program treba izvesti. To se može prikazati blok dijagramom. Npr. za računalni program za izračun površine pravokutnika, opis koraka blok dijagramom bi izgledao kako slijedi.



Dijagram 1.27. Blok dijagram za računalni program izračuna površine pravokutnika

Nakon opisa koraka, slijedi opis algoritama pseudo kodom. Svaki korak algoritma opisanog pseudo kodom će postati jedna ili više linija koda koji će programer napisati u nekome programskom jeziku. Programer treba znati je li algoritam opisan pseudo kodom ispravan. Svaki ispravan algoritam u pseudo kodu programer može prevesti u računalnim program u bilo kojem programskome jeziku. Algoritam u pseudo kodu za računalni program izračuna površine pravokutnika glasi:

Algoritam za računalni program izračuna površine pravokutnika u pseudo kodu:

```
1.  Ponavljaj
2.     Doznaj stranicu A pravokutnika.
3.     Doznaj stranicu B pravokutnika.
4.     Ako je A<=0 ili B<=0 onda a
5.        Reci da su unesene stranice neispravne.
6.  Dok vrijedi A<=0 ili B<=0
7.  Povrsina=A*B.
8.  Reci površina pravokutnika sa stranicama A i B iznosi
    Povrsina.
```

Algoritam 1.50.

Prije sljedećega koraka treba provjeriti ispravnost algoritma u pseudo kodu.

Kodiranje računalnoga programa predstavlja prevođenje dizajna računalnoga programa dobivenoga prethodnim korakom nekim programskim jezikom ili alatom za razvoj aplikacija. Kodiranje se odnosi na izradu korisničkoga sučelja i pisanje programskoga koda, ali i njegovu internu dokumentaciju kroz pisanje komentara koji pojašnjavaju svrhu neke naredbe. Kodiranjem se računalnoga programa dobiva izvorni kod računalnoga programa koji računalo izvodi interpreterom ili kompajlerom.

Primjer izvornoga koda u C-u za računalni program izračuna površine pravokutnika dan je u nastavku.

Izvorni kod u C-u za računalni program izračuna površine pravokutnika:

```
 1.  #include <conio.h>
 2.  #include <stdio.h>
 3.  main(){
 4.     int stranicaA, stranicaB, povrsina;
 5.     do{
 6.       printf ("Unesite duljinu stranice a pravokutnika: ");
 7.       scanf("%d", &stranicaA);
 8.       printf ("Unesite duljinu stranice b pravokutnika: ");
 9.       scanf("%d", &stranicaB);
10.       if (stranicaA<=0 || stranicaB<=0){
11.         printf ("Unesene stranice su neispravne!\n");
12.       }
13.    } while (stranicaA<=0 || stranicaB<=0);
14.    povrsina = stranicaA * stranicaB;
15.    printf("Povrsina pravokutnika sa stranicama %d i %d
             iznosi %d",stranicaA, stranicaB, povrsina);
16.    getch();}
```

Program 1.1.

Testiranje računalnoga programa i ispravljanje pogrešaka se sastoji u provjeravanju, pronalaženju i ispravljanju pogrešaka. Računalni program ima pogrešku ako se ne ponaša prema specifikacijama. Korak treba ponavljati sve dok se računalni program ne oslobodi grešaka. Kao pomoć u ovome koraku programeri se služe *debug* alatima - alatima za pronalazak pogreški. U osnovi postoje dvije vrste pogrešaka - sintaktičke i semantičke. Sintaktičke pogreške se lako otkrivaju (otkrivaju ih sama računala). Npr. ako se na kraju linije u programskome kodu nije stavila točka-zarez (;), prilikom pokretanja računalnoga programa sam kompajler će javiti pogrešku. Semantičku pogrešku računalo ne može otkriti već to mora učiniti programer. Npr. ako je programer naredio računalu da izračuna $2+3\cdot5$, a u stvari je mislio $(2+3)\cdot5$, računalo neće znati da je došlo do pogreške. Semantička se pogreška iskazuje tijekom izvođenja računalnoga programa.

Formalizacija i dokumentiranje računalnoga programa se sastoji u ponovnome pregledavanju i eventualnoj izmjeni interne dokumentacije (komentara u izvornome programskom kodu), njegove pretvorbe u oblik (paket) pogodan za dostavu korisniku te izradi korisničke dokumentacije (eksterne dokumentacije). Pokretanje računalnoga programa na računalu izaziva njegovo kompajliranje, eventualno povezivanje s drugim vanjskim datotekama itd. Računalni program namijenjen krajnjem korisniku mora sadržavati sve ono što je potrebno kako bi se pokrenuo na korisničkome računalu (bilo da se to nalazi u paketu zajedno s računalnim programom, bilo da postoji informacija kako doći do komponenti potrebnih za pokretanje računalnoga programa). I u ovome koraku treba nastaviti sa testiranjem računalnoga programa. Moguća je pojava greške prilikom izvođenja programa (*run-time error*) zbog nepostojanja odgovarajuće komponente koju računalni program zahtijeva.

Dokumentacija računalnoga programa se sastoji od svega što je izrađeno tijekom životnoga ciklusa razvoja računalnoga programa (specifikacije, blok dijagrami, pseudo kod, linije koda, korisnička dokumentacija itd.). Svrha je interne dokumentacije da olakša programerima shvaćanje vlastitoga ili tuđega izvornoga programskog koda. Eksterna dokumentacija je namijenjena krajnjim korisnicima i ona predstavlja uputu o primjeni računalnoga programa, ali bi trebala zadržavati i dio najčešćih pitanja i odgovora (*FAQ*), te potpunu pomoć (*Help*).

Posljednja faza životnoga ciklusa razvoja računalnoga programa je **održavanje računalnoga programa**. Ono uključuje svu potrebnu edukaciju i podršku krajnjim korisnicima, ispravljanje svih pogrešaka uočenih tijekom rada te identificiranje izmjena u korisničkim zahtjevima (poboljšanja). Jednom kad se greške ili poboljšanja identificiraju, životni ciklus razvoja računalnoga programa ponovno počinje s prvim korakom. Održavanje računalnoga programa je najduži korak u životnome ciklusu. Iako je obavljeno testiranje računalnoga programa često krajnji korisnici u radu otkrivaju nove greške. Računalni program treba održavati sve dok se ne dostigne točka u kojoj je on suvišan ili je jednostavno ostario. U toj točki održavanje treba zaustaviti te pokrenuti životni ciklus razvoja novoga računalnoga programa.

1.6. Primjeri algoritama

1. Potrebno je izraditi algoritam za pretvorbu mjernih jedinica za temperaturu (Celzijev stupanj - °C u stupanj Fahrenheita - °F i obrnuto). Formula za pretvorbu je $C = (F - 32) \cdot 5/9$, odnosno $F=(C \cdot 9/5) + 32$.

```
 1. Ponavljaj
 2.    Doznaj izbor (1 - pretvorba Celzij u Fahrenheit, 2 -
       pretvorba Fahrenheit u Celzij) izbor.
 3.    Ako je izbor <> 1 i izbor <> 2 onda
 4.      Reci da se za izbor mogu izabrati samo 1 ili 2.
 5.    Kraj Ako je
 6. Dok vrijedi izbor <> 1 i izbor <> 2
 7. Ako je izbor = 1 onda
 8.    Doznaj temperaturu u Celzijevim stupnjevima temperaturaC
 9.    temperaturaF = (temperaturaC · 9/5) + 32
10.    Reci da temperaturaC u Celzijevim stupnjevima iznosi
       temperaturaF.
11. inače Ako je izbor = 2 onda
12.    Doznaj temperaturu u Fahrenheit stupnjevima temperaturaF
13.    temperaturaC = (temperaturaF - 32) · 5/9
14.    Reci da temperaturaF u Fahrenheit stupnjevima iznosi
       temperaturaC.
15. Kraj Ako je
```

Algoritam 1.51.

2. Potrebno je izraditi algoritam za izračun površine trokuta koji je dan duljinama svojih stranica a, b i c. Duljine a, b i c tvore trokut ako vrijedi a+b>c i a+c>b i b+c>a.

Površina se računa prema formuli $P = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$, gdje je $s = \frac{a+b+c}{2}$.

```
 1.  Ponavljaj
 2.    Doznaj duljinu stranice A stranicaA
 3.    Doznaj duljinu stranice B stranicaB
 4.    Doznaj duljinu stranice C stranicaC
 5.    Ako je stranicaA <= 0 ili stranicaB <= 0 ili
       stranicaC <= 0 onda
 6.       Reci da se za duljinu stranice mogu unijeti samo
          brojevi veći od 0.
 7.    Kraj Ako je
 8.    Ako je stranicaA + stranicaB <= stranicaC ili
       stranicaA + stranicaC <= stranicaB ili
       stranicaB + stranicaC <= stranicaA onda
 9.       Reci da unesene duljine stranica ne tvore trokut.
10.    Kraj Ako je
11.  Dok vrijedi stranicaA <= 0 ili stranicaB <= 0 ili
     stranicaC <= 0 ili stranicaA + stranicaB <= stranicaC ili
     stranicaA + stranicaC <= stranicaB ili
     stranicaB + stranicaC <= stranicaA
12.  s = (stranicaA+stranicaB+stranicaC)/2
13.  povrsina=√s·(s−stranicaA·(s−stranicaB·(s−stranicaC)
15.  Reci da površina trokuta sa stranicama stranicaA,
     stranicaB i stranicaC iznosi povrsina.
```

Algoritam 1.52.

3. Potrebno je izraditi algoritam kojim se računa faktorijela unesenoga broja koji treba biti veći ili jednak 0. Faktorijel se računa kako slijedi: 0! = 1, 1! = 1, 2! = 1·2, 3! = 1·2·3, 4! = 1·2·3·4, 5! = 1·2·3·4·5, odnosno općenito n! = 1·2·3·...·(n-1)·n

```
 1.  Ponavljaj
 2.    Doznaj neki broj (veći ili jednak od 0) UneseniBroj.
 3.    Ako je UneseniBroj < 0 onda
 4.       Reci da se mogu unijeti samo brojevi veći ili jednaki
          od 0
 5.    Kraj Ako je
 6.  Dok vrijedi UneseniBroj < 0
 7.  Faktorijel = 1
 8.  Ponavljaj za svaki Broj=1 do UneseniBroj korak 1
 9.    Faktorijel = Faktorijel * Broj
10.  Kraj Ponavljaj
11.  Reci da faktorijel od UneseniBroj iznosi Faktorijel.
```

Algoritam 1.53.

4. Potrebno je izgraditi jednostavni kalkulator za računske operacije +, -, *, /.

```
 1. Doznaj prvi broj UneseniBroj1
 2. Ponavljaj
 3.   Doznaj neku računsku operaciju (+. -, *, /)
      RacunskaOperacija.
 4.   Ako je RacunskaOperacija <> "+" i
      RacunskaOperacija <> "-" i RacunskaOperacija <> "*" i
      RacunskaOperacija <> "/" onda
 5.     Reci da se mogu unijeti samo računske operacije +, -,
        *, /
 6.   Kraj Ako je
 7. Dok vrijedi RacunskaOperacija <> "+" i
    RacunskaOperacija <> "-" i RacunskaOperacija <> "*" i
    RacunskaOperacija <> "/"
 8. Doznaj drugi broj UneseniBroj2
 9. Ako je RacunskaOperacija <> "+" onda
10.   Rezultat = UneseniBroj1 + UneseniBroj2
11.   Reci da je rezultat zbroja brojeva UneseniBroj1 i
      UneseniBroj2 jednak Rezultat.
12. inače Ako je RacunskaOperacija <> "-" onda
13.   Rezultat = UneseniBroj1 - UneseniBroj2
14.   Reci da je rezultat razlike brojeva UneseniBroj1 i
      UneseniBroj2 jednak Rezultat.
15. inače Ako je RacunskaOperacija <> "*" onda
16.   Rezultat = UneseniBroj1 * UneseniBroj2
17.   Reci da je rezultat produkta brojeva UneseniBroj1 i
      UneseniBroj2 jednak Rezultat.
18. inače Ako je RacunskaOperacija <> "/" onda
19.   Ako je UneseniBroj2 = 0 onda
20.     Reci da dijeljenje s 0 nije moguće.
21.   inače
22.     Rezultat = UneseniBroj1 / UneseniBroj2
23.     Reci da je rezultat kvocjenta brojeva UneseniBroj1 i
        UneseniBroj2 jednak Rezultat.
24.   Kraj Ako je
25. Kraj Ako je
```

Algoritam 1.54.

5. Potrebno je izraditi algoritam za izračun telefonskih troškova na kraju mjeseca, ako su poznati količina telefonskih razgovora u minutama, cijena jedne minute te iznos telefonske pretplate. U iznos telefonske pretplate uračunato je 60 minuta razgovora.

```
 1. Ponavljaj
 2.   Doznaj količinu telefonskih razgovora u minutama
      Kolicina.
 3.   Ako je Kolicina < 0 onda
 4.     Reci da količina telefonskih razgovora može biti samo
        veća ili jednaka od 0.
 5.   Kraj Ako je
```

```
 6.  Dok vrijedi Kolicina < 0
 7.  Ponavljaj
 8.    Doznaj cijenu jedne minute razgovora Cijena.
 9.    Ako je Cijena <= 0 ili onda
10.      Reci da cijena jedne minute može biti samo veća od 0.
11.    Kraj Ako je
12.  Dok vrijedi Cijena < 0
13.  Ponavljaj
14.    Doznaj iznos telefonske pretplate Pretplata.
15.    Ako je Pretplata <= 0 ili onda
16.      Reci da iznos telefonske pretplate može biti samo veća
         od 0.
17.    Kraj Ako je
18.  Dok vrijedi Pretplata < 0
19.  Ako je Kolicina > 60 onda
20.    Visak = Kolicina - 60
21.  inače
22.    Visak = 0
23.  Kraj Ako je
24.  Trosak = Pretplata + Visak * Cijena
25.  Reci da za količinu telefonskog razgovora od Kolicina
     minuta, trošak iznosi Trosak.
```

Algoritam 1.55.

6. Potrebno je izraditi algoritam kojim se računa prosječna ocjena od unesenoga neograničenog broja ocjena (1, 2, 3, 4, 5). Unos 0 označava kraj unosa.

```
 1.  BrojUnesenihOcjena = 0
 2.  ZbrojUnesenihOcjena = 0
 3.  Ponavljaj
 4.    Doznaj ocjenu (1, 2, 3, 4, 5) ili 0 za kraj unosa Unos.
 5.    Ako je Unos < 0 ili Unos > 5 onda
 6.      Reci da unesena vrijednost može biti ocjena (1, 2, 3,
         4, 5) ili 0 za kraj unosa.
 7.    inače
 8.      ZbrojUnesenihOcjena = ZbrojUnesenihOcjena + Unos
 9.      Ako je Unos <> 0 onda
10.        BrojUnesenihOcjena = BrojUnesenihOcjena + 1
11.      Kraj Ako je
12.    Kraj Ako je
13.  Dok vrijedi Unos <> 0
14.  Ako je BrojUnesenihOcjena > 0 onda
15.    ProsjecnaOcjena = ZbrojUnesenihOcjena/BrojUnesenihOcjena
16.    Reci da prosječna ocjena iznosi ProsjecnaOcjena
17.  inače
18.    Reci da nije unesena niti jedna ocjena.
19.  Kraj Ako je
```

Algoritam 1.56.

7. Potrebno je izraditi algoritam za izdavanje računa. Unose se količina artikla i njegova jedinična cijena. Nakon svakoga unosa korisnika se pita za unos količine i jedinične cijene novoga artikla. Po završetku unosa artikala izračunava se ukupna cijena.

```
 1. CijenaStavkeRacuna = 0
 2. UkupnoRacun = 0
 3. Ponavljaj
 4.   Ponavljaj
 5.     Doznaj količinu artikla Kolicina.
 6.     Ako je Kolicina <= 0 onda
 7.       Reci da količina mora biti veća od 0.
 8.     Kraj Ako je
 9.   Dok vrijedi Kolicina <= 0
10.   Ponavljaj
11.     Doznaj jediničnu cijenu artikla JedinicnaCijena.
12.     Ako je JedinicnaCijena <= 0 onda
13.       Reci da jedinična cijena mora biti veća od 0.
14.     Kraj Ako je
15.   Dok vrijedi JedinicnaCijena <= 0
16.   CijenaStavkeRacuna = Kolicina * JedinicnaCijena
17.   UkupnoRacun = UkupnoRacun + CijenaStavkeRacuna
18.   Ponavljaj
19.     Doznaj želi li se ubaciti nova stavka računa (Da/Ne)
        Ponovi
20.     Ako je Ponovi <> "Da" i Ponovi <> "Ne" onda
21.       Reci da je moguće unijeti samo Da ili Ne.
22.     Kraj Ako je
23.   Dok vrijedi Ponovi <> "Da" i Ponovi <> "Ne"
24. Dok vrijedi Ponovi = "Da"
25. Reci da je ukupni iznos računa UkupnoRacun.
```

Algoritam 1.57.

8. Potrebno je izraditi algoritam koji pronalazi i ispisuje sve proste brojeve do nekoga unesenog prirodnog broja većeg od 1. Broj je prost (prim broj) ako je djeljiv s 1 i sa samim sobom. Djeljivost brojeva se provjerava pomoću operatora % (modulo) koji vraća ostatak pri cjelobrojnome dijeljenju. Npr. 5 % 2 = 1 jer je 5 / 2 = 2 i ostatak 1; 4 % 2 = 0 jer je 4 / 2 = 2 i ostatak 0.

```
 1. Ponavljaj
 2.   Doznaj neki prirodni broj veći od 1 PrirodniBroj.
 3.   Ako je PrirodniBroj <= 1 onda
 4.     Reci da unesena vrijednost može biti samo prirodni
        broj veći od 1.
 5. Dok vrijedi PrirodniBroj <= 1
 6. Ponavljaj za svaki Broj = 2 do PrirodniBroj korak 1
 7.   PrimBroj = "Da"
 8.   Ponavljaj za svaki Djelitelj = 2 do Broj/2 korak 1
 9.     Ako je Broj % Djelitelj = 0 onda
10.       PrimBroj = "Ne"
```

```
11.      Kraj Ako je
12.    Kraj Ponavljaj
13.    Ako je PrimBroj = "Da" onda
14.      Reci da je Broj prost broj.
15.    Kraj Ako je
16.  Kraj Ponavljaj
```

Algoritam 1.58.

9. Potrebno je izraditi algoritam kojim se računa niz Fibonaccijevih brojeva do nekoga zadanog rednog broja člana niza. To je sljedeći niz brojeva: 0, 1, 1, 2, 3, 5, 8, 13 itd. Općenito se Fibonaccijevi brojevi F računaju na sljedeći način:

F_i	je 1 ako je i = 0
	je 1 ako je i = 1
	je F_{i-1} - F_{i-2}

```
 1.  Ponavljaj
 2.    Doznaj redni broj člana niza (broj veći ili jednak 0)
       UneseniRedniBroj.
 3.    Ako je RedniBroj < 0 onda
 4.      Reci da se mogu unijeti samo brojevi veći ili jednaki
         od 0
 5.    Kraj Ako je
 6.  Dok vrijedi PrirodniBroj <> 0
 7.  BrojPrethodnika=0
 8.  Ponavljaj za svaki RedniBroj=0 do UneseniRedniBroj korak 1
 9.    Ako je RedniBroj = 0 ili RedniBroj = 1 onda
10.      FBroj = 1
11.    inače
12.      FBroj = FPrethodnik1 + FPrethodnik2
13.    Kraj Ako je
14.    BrojPrethodnika = BrojPrethodnika + 1
15.    Ako je BrojPrethodnika = 1 onda
16.      FPrethodnik1 = FBroj
17.    inače
18.      FPrethodnik2 = FBroj
19.      BrojPrethodnika = 0
20.    Kraj Ako je
21.    Reci da je RedniBroj. Fibonaccijev broj jednak FBroj
22.  Kraj Ponavljaj
```

Algoritam 1.59.

10. Potrebno je izraditi algoritam kojim će se pronaći najmanji i najveći broj između neograničenoga broja unesenih slučajnih prirodnih brojeva (1, 2, 3, ...). Unos broja 0 prekida unos prirodnih brojeva i daje rezultat.

```
 1.  PrviUnos = "Da"
 2.  Ponavljaj
 3.    Doznaj neki prirodni broj (0 za prekid unosa)
       PrirodniBroj.
```

```
 4.    Ako je PrirodniBroj < 0 onda
 5.      Reci da se mogu unijeti samo brojevi veći ili
         jednaki 1 ili 0 za kraj unosa
 6.    Inače Ako je PrirodniBroj >0 onda
 7.      Ako je PrviUnos = "Da" onda
 8.        PrviUnos = "Ne"
 9.        NajmanjiBroj = PrirodniBroj
10.        NajveciBroj = PrirodniBroj
11.      Inače
12.        Ako je NajmanjiBroj > PrirodniBroj onda
13.          NajmanjiBroj = PrirodniBroj
14.        Kraj Ako je
15.        Ako je NajveciBroj < PrirodniBroj onda
16.          NajveciBroj = PrirodniBroj
17.        Kraj Ako je
18.      Kraj Ako je
19.    Kraj Ako je
20.  Dok vrijedi PrirodniBroj <> 0
21.  Reci da je najmanji uneseni prirodni broj NajmanjiBroj, a
     najveći NajveciBroj.
```

Algoritam 1.60.

2. VARIJABLE

Poglavlje **Varijable** odnosi se na memoriju računala, zapis vrijednosti u memoriji računala, pojam varijable i njezinih osnovnih svojstava te prikaza primjene varijable u programskome jeziku C.

Po završetku ovoga poglavlja čitatelj će moći:

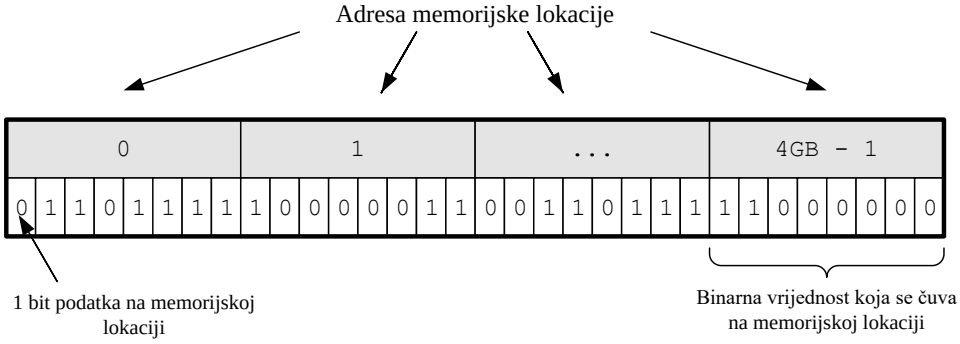
- objasniti strukturu memorije računala
- objasniti načine spremanja vrijednosti podatka u memoriju računala
- definirati pojam varijabla i objasniti vezu između varijable i memorije računala
- navesti, objasniti i razlikovati svojstva varijabli
- definirati pojam tip podatka
- razlikovati ispravne od neispravnih identifikatora varijable
- demonstrirati deklaraciju varijable u C
- definirati pojam polje
- navesti svojstva polja
- definirati pojam niz znakova
- navesti svojstva niza znakova
- demonstrirati deklaraciju polja u C
- demonstrirati deklaraciju niza znakova u C
- definirati pojam zapis
- navesti svojstva zapisa
- demonstrirati deklaraciju zapisa u C
- definirati pojam pokazivač
- navesti svojstva pokazivača
- demonstrirati deklaraciju pokazivača u C
- demonstrirati pridruživanje adrese varijable nekom pokazivaču
- demonstrirati dolaženje do vrijednosti varijable preko pokazivača
- navesti osnovne operacije s pokazivačima
- objasniti primjenu pojedine operacije s pokazivačima
- objasniti operator pridruživanja
- demonstrirati operator pridruživanja
- navesti algebarske operatore
- demonstrirati primjenu algebarskih operatora
- izračunati vrijednost varijabli
- navesti relacijske operatore
- demonstrirati primjenu relacijskih operatora
- navesti logičke operatore
- navesti tablicu istine pojedinome logičkom operatoru
- demonstrirati primjenu logičkih operatora
- izračunati vrijednost logičkog izraza
- sastaviti logički izraz prema zadanom problemu.

2.1. Memorija računala

Tijekom se osnovne škole (4. razred) naučilo da se površina bilo koga pravokutnika sa stranicama duljine a i b računa prema formuli P=a·b (s tim da su duljine izražene u istoj mjernoj jedinici – npr. u centimetrima; mala komplikacija nastaje ako su duljine izražene u različitim mjernim jedinicama). Dakle, ovaj algoritam bi se trebao nalaziti u našoj dugoročnoj memoriji. Kad bi smo se našli u situaciji da npr. nabavljamo tepison za našu radnu sobu – poslužili bi smo se navedenim algoritmom u izračunu površine sobe (jasno, ako je ona pravokutnog oblika). No, jedna je važna informacija potrebna kako bi se dobila tražena površina – dimenzija sobe. Neka se nakon mjerenja dobila dimenzija sobe 3 metra sa 4 metra. Sada se konačno može primijeniti algoritam (iz naše dugoročne memorije) i dobiti rezultat: 12 metara kvadratnih. No, pitanje koje se postavlja jest gdje su sačuvane vrijednosti 3 i 4 metra, te 12 metara kvadratnih. Odgovor je – u kratkoročnoj memoriji. Dakle, kad se izvelo mjerenje sobe, vrijednosti su unesene u našu kratkoročnu memoriju. S tim vrijednostima je pokrenut algoritam koji je vratio rezultat koji se ponovno smjestio u našu kratkoročnu memoriju.

Ako bi se isti ovaj scenarij htio provesti primjenom računala, tada bi i ono trebalo u svojoj memoriji (dugoročnoj – npr. tvrdi disk) imati pohranjen algoritam izračuna površine pravokutnika (u obliku računalnoga programa). Kad bi se taj algoritam pokrenuo, on bi zahtijevao dimenzije sobe koje bi računalo spremilo u svojoj kratkoročnoj memoriji (*RAM*), potom bi se izvelo izračunavanje, te bi računalo rezultat dostavilo (na zaslonu, štampano na papiru i sl.) i/ili bi ga spremilo u svoju kratkoročnu memoriju (*RAM*) i/ili dugoročnu memoriju.

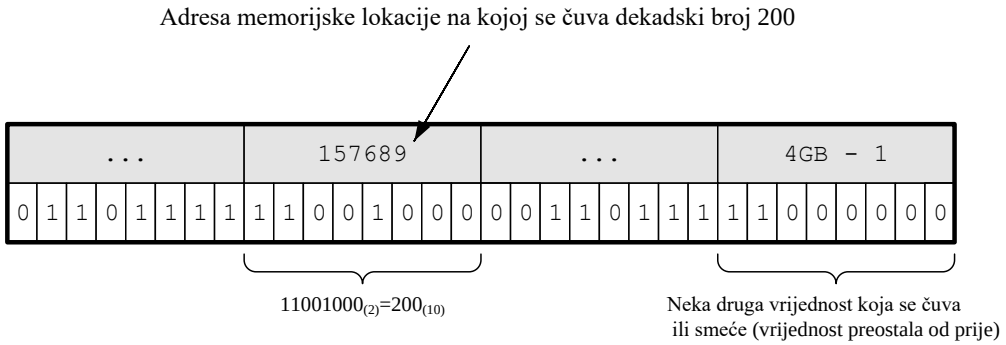
Važno za ovo poglavlje je RAM memorija računala budući da se uglavnom ona koristi kao radna memorija tijekom izvršavanja nekoga računalnog programa (u njoj se, pored ostaloga, nalazi sam računalni program te vrijednost njegovih ulaza, izlaza i ostalih pomoćnih međurezultata procesiranja). Računalo svoju RAM memoriju doživljava kao niz ćelija od 8 bitova (jedan bajt). Svaka takva ćelija ima jedinstvenu adresu (cijeli broj veći ili jednak 0) pomoću koje računalo dohvaća vrijednost koja se čuva unutar ćelije. Maksimalan broj ćelija koje računalo može adresirati (odnosno maksimalna veličina memorije) ovisi o broju bitova koje računalu stoji na raspolaganju za adresiranje ćelija. Kod 32-bitnih računala, za adresiranje memorije računalo ima na raspolaganju 32 bita, odnosno ukupno 2^{32} cijelih brojeva što omogućava adresiranje 4.294.967.296 ćelija (memorijski lokacija) od jednoga bajta, odnosno veličinu memorije od 4GB (gigabajta). Na sljedećoj slici je prikazana memorija kako je vidi računalo.



Dijagram 2.1. Prikaz memorije računala

2.2. Zapis vrijednosti u memoriji računala

Računalo u svojoj memoriji (točnije u adresiranim memorijskim lokacijama) čuva isključivo binarne vrijednosti (0 ili 1). U jednoj memorijskoj lokaciji se može sačuvati 256 različitih brojeva (od 0 do 255). Dakle, ako vrijednost ne izlazi iz ovoga intervala, tada je za njezino čuvanje u memoriji potrebna samo jedna memorijska lokacija. Npr. čuvanje bi broja 200 u memoriji računala zahtijevalo jednu memorijsku lokaciju na nekoj adresi na kojoj bi se nalazio binarni broj koji predstavlja dekadski broj 200. To prikazuje sljedeća slika.



Dijagram 2.2. Zapis broja 200 u memoriji računala

Što ako se želi čuvati neki negativni cijeli broj kao što je -100? Tada se ne koristi svih osam bita memorijske lokacije za čuvanje broja, već se jedan bit koristi za oznaku predznaka (0 ako je broj pozitivan, 1 ako je negativan). U tome je slučaju na raspolaganju sedam bitova za prikaz broja. Zbog toga je na ovakav način moguće u jednu memorijsku lokaciju zapisati brojeve u intervalu -127 do +128 (2^7).

Osim primjene prvoga bita za predznak, a ostalih bitova za prikaz pozitivnoga broja (što je moguće primijeniti za brojeve -127 do 128), prikaz se negativnih cijelih brojeva može dobiti primjenom komplementa binarnoga broja. I ovdje prvi bit predstavlja

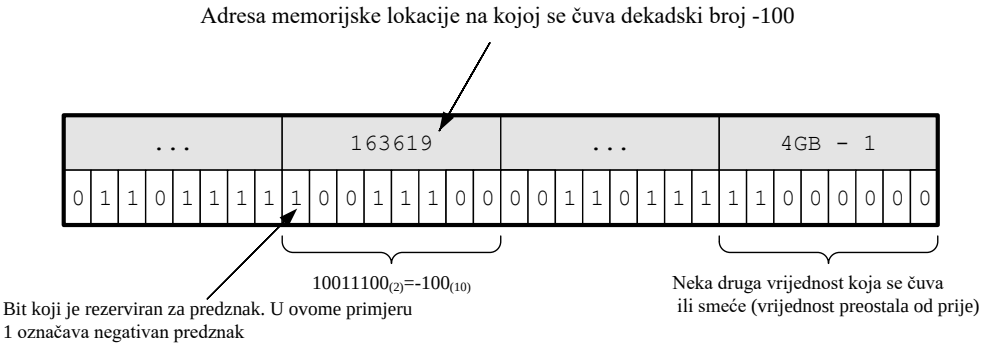
predznak broja. Slijedi algoritam za negativni broj (za pozitivni broj se ovaj algoritam ne provodi već se broj izravno prevodi u binarni broj, a na mjesto bita s najvećom važnosti se još dodaje 0):

1. kreće se s binarnim prikazom pozitivnoga broja
2. od binarnoga prikaza pozitivnoga broja izradi se komplement (jednostavno se 0 pretvori u 1 i obrnuto)
3. binarnom broju dobivenom u koraku 2. se dodaje dekadski broj 1
4. binarni se broj dobiven u koraku 3. sprema u memoriju računala kao prikaz negativnoga broja.

Prikaz dekadskoga broja -100 prema gore prikazanom algoritmu bi izgledao kako slijedi:

1. $100_{(10)} = 01100100_{(2)}$
2. komplement $(01100100_{(2)}) = 10011011_{(2)}$
3. $10011011_{(2)} + (00000001_{(2)}) = 10011100_{(2)} = -100_{(10)}$

Prema tome, u memoriji računala se čuva binarni broj kako je prikazano na sljedećoj slici.



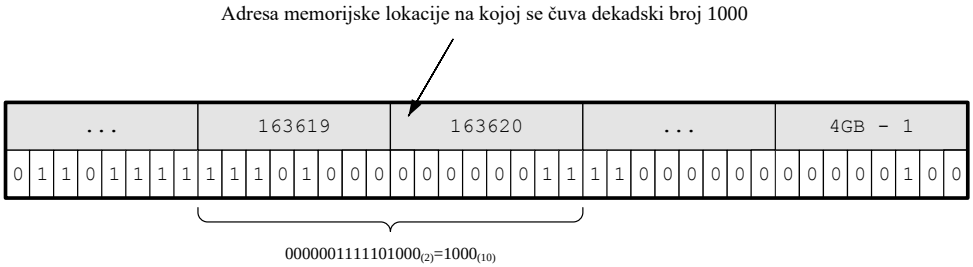
Dijagram 2.3. Zapisa broja -100 u memoriji računala

Prikazani se način zapisa negativnih brojeva naziva zapis pomoću dvojnoga komplementa. Ovakav zapis cijelih brojeva pojednostavljuje izvođenje računskih operacija s njihovim binarnim zapisima. Već je prikazano da broj -100 pomoću dvojnoga komplementa ima binarni zapis 10011100. Broj 100 pomoću istoga zapisa ima oblik 01100100. Slijedi prikaz kako se iz binarnih brojeva navedenoga prikaza pomoću dvojnoga komplementa dolazi do dekadskih brojeva.

$$10011100 = -1\cdot2^7 + 0\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 0\cdot2^0$$
$$= -128 + 0 + 0 + 16 + 8 + 4 + 0 + 0 = -100$$
$$01100100 = 0\cdot2^7 + 1\cdot2^6 + 1\cdot2^5 + 0\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 0\cdot2^0$$
$$= 0 + 64 + 32 + 0 + 0 + 4 + 0 + 0 = 100$$

Do sada je prikazano kako se u memoriji računala čuvaju brojevi koji mogu stati u jednu memorijsku lokaciju veličine 8 bita (jedan bajt). Što ako je potrebno čuvati veće brojeve koji ne mogu stati u jedan bajt (jednu memorijsku lokaciju)? Npr. kako će u

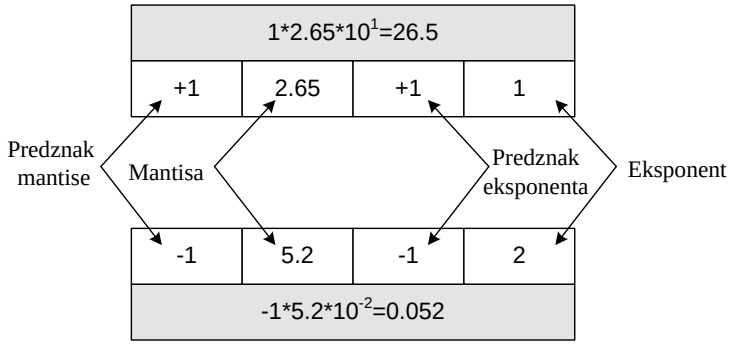
memoriji računala biti sačuvan dekadski broj 1000. Ako ga se pretvori u binarni broj, dobiva se 1111101000. Vidljivo je da ovaj broj ima ukupno 10 bita pa bi se za njegov zapis trebale iskoristiti dvije memorijske lokacije po 8 bita čime se dobiva prostor za zapis 16 bitnoga broja. I upravo to će računalo i napraviti. Dvije susjedne memorijske lokacije će iskoristiti za zapis brojeva za koje je potrebno 16 bita. Tako će prikaz broja 1000 u memoriji računala izgledati kako slijedi.



Dijagram 2.4. Zapis broja 1000 u memoriji računala

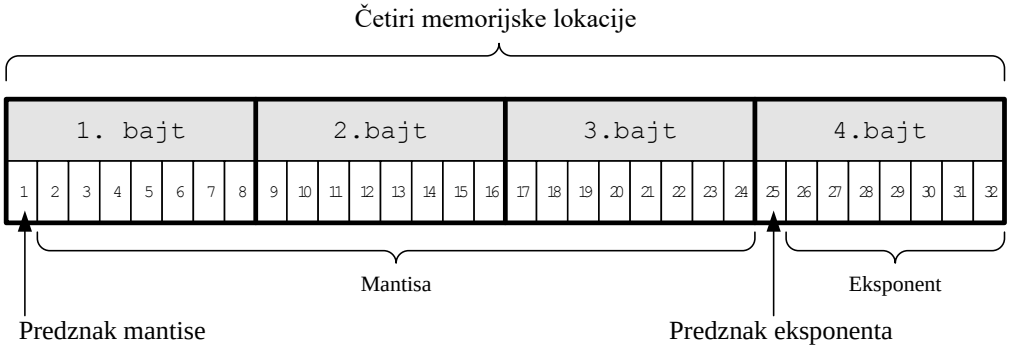
Dakle, da bi se dobila konačna vrijednost, bajtovi se čitaju tako da se prvo čita onaj bajt koji se čuva u memorijskoj lokaciji s većom adresom. U prikazanome primjeru, prvo se čita binarni broj 00000011 iz adrese 163620, a zatim binarni broj 11101000 iz manje adrese (163619) i tako se dobiva konačni binarni broj 0000001111101000, odnosno dekadski 1000. Zapis negativnoga broja je isti kako je već opisano.

Do sada su prikazani primjeri zapisa cijelih brojeva. Veličina vrijednosti koja se želi sačuvati u memoriji računala određuje koliko će se memorijskih lokacija koristiti, odnosno koliko bajtova. No, kako se u memoriji računala prikazuju decimalni brojevi kao što je npr. 26.5. U dekadskom zapisu, ovaj broj bi se mogao prikazati i na sljedeći način $26.5 = 1 \cdot 2.65 \cdot 10^{1} = 2.65E$. Slično bi se broj -0.052 mogao prikazati kako slijedi $-0.052 = -1 \cdot 5.2 \cdot 10^{-2} = -5.2E-2$. Dakle, dekadski se decimalni broj može zapisati tako da se posebno navede njegov predznak, potom jednoznamenkasti cijeli broj iza kojega slijede decimale i na kraju broj (u obliku baza brojevnoga sustava na eksponent), te se sve to međusobno pomnoži. Ova se notacija zapisa decimalnoga broja naziva E-notacija, znanstvena notacija ili notacija kliznoga zareza/točke. Dakle, da bi se prikazao decimalni broj, potrebno je sačuvati predznak decimalnoga broja, zatim decimalni broj koji ima samo jednu znamenku u svome cjelobrojnom dijelu, predznak eksponenta i vrijednost samoga eksponenta. Sljedeća slika prikazuje potrebne dijelove u prikazu decimalnoga broja i njihove nazive.



Dijagram 2.5. Zapis decimalnoga broja u notaciji kliznoga zareza

Opisani način prikaza dekadskih decimalnih brojeva koristi se i za prikaz binarnih decimalnih brojeva. Postoje dva standarda za prikaz binarnoga decimalnog broja - jednostruka i dvostruka preciznost. Standard za prikaz binarnoga decimalnog broja jednostruke preciznosti koristi ukupno četiri bajta (četiri memorijske lokacije) - 1 bit za predznak mantise, 23 bita za mantisu, 1 bit za predznak eksponenta i 7 bitova za eksponent. Sljedeća slika prikazuje koji bitovi se u navedena četiri bajta koriste za prikaz određenoga dijela decimalnog broja.



Dijagram 2.6. Zapis decimalnoga broja u memoriji računala

Standard za prikaz binarnoga decimalnog broja dvostruke preciznosti koristi osam bajtova (osam memorijskih lokacija).

Do sada je prikazano kako se u memoriji računala spremaju pozitivni i negativni cijeli i decimalni brojevi. Pitanje koje se postavlja je kako spremiti neki znak u memoriju računala (dakle ne broj). Znak može biti slovo (npr. A, a itd.), znamenka (0, 1, ..., 9) ili neki posebni znak (npr. ", #, %,), !, +, * itd.). Ideja je da se svaki znak kodira nekim dekadskim brojem. U svrhu kodiranja znakova koriste se ASCII kodovi (*engl. American Standard Code for Information Interchange*). Dio tih kodova prikazan je u sljedećoj tablici.

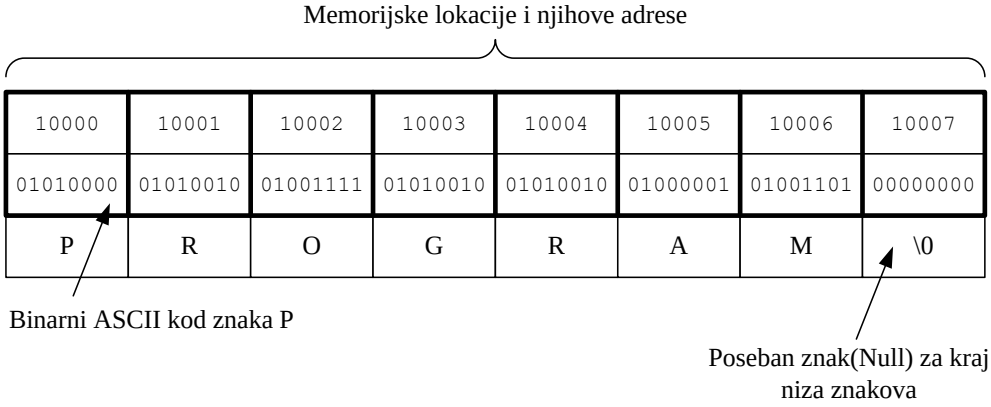
Binarno	Oktalno	Dekadski	Heksadecimalno	Znak
010 0000	040	32	20	
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M

100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|

111 1101	175	125	7D	}
111 1110	176	126	7E	~

Tablica 2.1. ASCII kodovi

Prema tome, želi li se u memoriju računala spremiti, npr. znak A, tada će se u jednoj memorijskoj lokaciji sačuvati binarni broj 01000001. No, želi li se sačuvati riječ "PROGRAM", tada će biti potrebno onoliko memorijskih lokacija u slijedu koliko ima znakova u promatranoj riječi (uz još jedan znak za kraj niza znakova, ali o tome u posebnome poglavlju). U svakoj od tih memorijskih lokacija će se čuvati ASCII kod jednoga znaka promatrane riječi. Sljedeća slika prikazuje sadržaj memorijskih lokacija u kojima se čuva riječ "PROGRAM".



Dijagram 2.7. Zapis riječi "PROGRAM" u memoriji računala

Sve do sada korištene vrijednosti spadaju u jednostavne vrste vrijednosti, odnosno u jednostavne tipove podataka. Zapis složenih vrijednosti (složenih tipova podataka) zahtijeva više memorijskih lokacija i drugačije načine zapisa samih vrijednosti (npr. vrijednost koja predstavlja datum - 01.01.2013.). Ipak, može se uočiti da se ti složeni tipovi podataka sastoje iz jednostavni pa se u njihovom prikazu u osnovi koriste prikazi jednostavnih tipova podataka.

2.3. Pojam varijable i njezina osnovna svojstva

U prethodnim je poglavljima prikazano kako računalo vidi memoriju te na koji način u nju sprema vrijednosti podataka. Pitanje koje se postavlja je kako se iz računalnoga programa pristupa memoriji računala u svrhu upisa vrijednosti nekoga podatka, odnosno dohvaćanja čuvanoga podataka. A prije toga, kako se uopće naređuje računalu da u svojoj memoriji pripremi prostor za čuvanje podataka.

Jedan od načina pristupanja memoriji računala je primjena memorijskih adresa. Dakle, programer bi trebao znati na kojoj memorijskoj adresi se nalazi neki podatak, odnosno na koju memorijsku adresu se podatak treba spremiti. Ovaj pristup je iz više razloga težak za programere (npr. pamćenje brojeva tipa 13.298.765). Puno je bolji pristup da se memorijskoj lokaciji dodijeli neko simboličko ime (npr. `stranicaA`) te joj se pristupa preko njega. I ovdje se dolazi do pojma varijable. **Varijabla** je imenovani dio

memorije računala. Preko varijable se izvodi ubacivanje vrijednosti u memoriju računala, odnosno njezino dohvaćanje.

U prethodnim se poglavljima može vidjeti da je za čuvanje vrijednosti nekog podatka u memoriji računala nužno znati što ta vrijednost predstavlja (npr. radi li se o cijelom broju, decimalnom broju, znaku itd.), te kakve vrijednosti podatak uopće može imati (npr. 0-255). Sve ovo je definirano pojmom **tip podatka**. Dakle, tip podatka definira koliko je potrebno memorijskih lokacija za čuvanje vrijednosti te na koji način se čuvane vrijednosti interpretiraju.

Za korištenje memorije računala nužno je računalu narediti da pripremi i rezervira određeni dio svoje memorije (određeni skup memorijskih lokacija) u kome će se čuvati vrijednosti podataka određenoga tipa. Ovo se radi u postupku **deklaracije varijable**. U osnovi, deklaracijom varijable u računalnome programu definira se tip podatka (opseg vrijednosti i njezinu interpretaciju) te identifikator varijable (njezin naziv) čime se računalu naređuje da u svojoj memoriji pripremi i rezervira dovoljan broj memorijskih lokacija za taj tip podatka, da tu skupinu memorijskih lokacija nazove simboličkim imenom (identifikatorom varijable) te da ono što će se u tom dijelu memorije čuvati interpretira u skladu sa željenim tipom podatka.

Varijable se uvijek trebaju deklarirati prije njihova prvoga korištenja u računalnome programu. Varijabli se u računalnome programu može dodijeliti neka vrijednost (dodjela znači naredba računalu da vrijednost spremi u dio memorije rezerviran za tu varijablu) ili se može dozvati vrijednost koja se čuva u memoriji koja je dodijeljena varijabli. Vrijednost varijabli se pridružuje primjenom operatora pridruživanja (=) na način da se ono što se nalazi na desnoj strani operatora pridružuje varijabli koja se nalazi na lijevoj strani.

Dio će se memorije računala koji je rezerviran za neku varijablu označavati na sljedeći način:

Memorijska adresa
Vrijednost varijable
Identifikator varijable

Dijagram 2.8. Prikaz rezervirane memorijske lokacije

Npr. ako je deklarirana varijabla `stranicaA` onda u memoriji računala postoji memorijska lokacija koja je rezervirana za tu varijablu te bi se to prikazalo na sljedeći način:

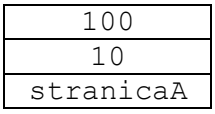
100
SMEĆE
stranicaA

Dijagram 2.9. Prikaz rezervirane memorijske lokacije za varijablu `stranicA`

Broj 100 je memorijska adresa (stavljeno proizvoljno). Pod vrijednost se nalazi *SMEĆE* – naime čim se izvede rezervacija dijela memorije, ona nije prazna – u njoj se

mogu nalaziti ostaci nekoga podatka koji je tu prije boravio. `stranicaA` označava identifikaciju prikazanoga dijela memorije.

Ako se sada napiše `stranicaA = 10` (dakle dodjela vrijednosti varijabli `stranicaA`) u memoriji će se nalaziti sljedeće.



Dijagram 2.10. Prikaz sadržaja rezervirane memorijske lokacije nakon dodjeljivanja vrijednosti varijabli `stranicA`

Dakle, nema više *SMEĆA*.

Svaka varijabla ima sljedeća svojstva:

1. **ime (identifikator)**
2. **adresa**
3. **vrijednost**
4. **tip (tip podatka)**
5. **trajanje**
6. **doseg.**

2.3.1. Ime (identifikator) varijable

Ime, odnosno identifikator varijable predstavlja simboličko ime koje će se dodijeliti nekome dijelu memorije računala. Dozvoljeni znakovi u sastavljanju imena (identifikatora) varijabli su:

1. **slova engleske abecede (A-Z, a-z)**
2. **brojevi (0-9)**
3. **podcrta (*underscore* '_').**

Prvi znak u imenu varijable ne smije biti broj. Ime varijable ne može biti isto kao neka ključna riječ u programskome jeziku (ključna riječ je riječ koja je rezervirana za neko drugo značenje u programskome jeziku – npr. naredbe programskoga jezika). U nekim programskim jezicima (npr. C, C++, Java) treba biti pažljiv s veličinom slova. Naime, u tim jezicima nije isto ako se varijabla deklarira imenom `stranicaA`, pa se onda poslije u računalnome programu koristi kao npr. `StranicaA` (slovo `S` nije malo slovo) – ime varijabli je osjetljivo na veličinu slova (*engl. Case-sensitive*). Sljedeća tablica prikazuje primjere ispravnih i neispravnih imena varijabli.

Ispravna imena varijabli	Neispravna imena varijabli	
`imePrezime`	`Popis Studenata`	(u imenu razmak)
`IMEPrezime`	`Popis-Studenata`	(u imenu znak -)
`Popis_Studenata`	`1godinaPopis`	(početak imena s brojem)
`popis1godina`	`51000`	(ime sami brojevi)
`Umnozak`	`Umnožak`	(u imenu znak ž)
	`switch`	(ključna riječ u C-u)

Tablica 2.2. Primjeri ispravnih i neispravnih imena varijabli

Prilikom definiranja naziva varijabli dobro je izabrati neko informativno ime koje će otkrivati semantiku vrijednosti koju varijabla čuva (dakle, svrhu varijable). Nepisano je pravilo da naziv varijable počinje malim početnim slovom. Npr. želi li se deklarirati varijabla koja će čuvati masu, onda je dobro da se ta varijabla nazove `masa`, a ne npr. `X`. Evo još nekih primjera:

Svrha varijable	Dobro ime	Loše ime
Stanje na računu	`stanje_na_racunu, stanje, saldo`	`SR, S, X, X1, X2`
Brzina vlaka	`brzina_vlaka, brzina, brzina_u_Kmh`	`Vlak, B, BV, X, X1, X2`
Tekući datum	`tekuci_datum`	`Datum, Tekuci, TD, X, X1, X2`
Broj linija po stranici	`broj_linija_po_stranici, br_lnij_po_strnc`	`BLS, Linija, L, X1, X2, X3`

Tablica 2.3. Primjer dobrih i loših imena varijabli

Ipak treba paziti da identifikator varijable ne bude predugačak.

2.3.2. Adresa varijable

Adresa varijable je adresa dijela memorije računala koji je rezerviran za varijablu. U nekim slučajevima u računalnome programu je važno vrijednosti varijable pristupati, odnosno zapisivati preko njezine adrese (u narednim poglavljima će se obraditi slučaj procedura).

2.3.3. Vrijednost varijable

Jednom deklariranoj varijabli u nekome računalnom programu, njezina vrijednost se postavlja ili se koristi. Vrijednost varijable predstavlja onu vrijednost koju u određenome trenutku izvođenja računalnoga programa varijabla ima, odnosno koja je zapisana u dijelu rezervirane memorije. Već je rečeno da se za dodjeljivanje vrijednosti varijable koristi operator pridruživanja (=). Varijabla se treba nalaziti s lijeve strane operatora pridruživanja, a vrijednost koja se dodjeljuje s desne.

Neka su deklarirane tri varijable `stranicaA`, `stranicaB` i `povrsina`. U memoriji računala su rezervirane tri memorijske lokacije (potencijalno različitih dimenzija što ovisi o tipu podatka koji će se čuvati) za promatrane varijable. Dakle, u memoriji računala postoji:

100	200	300
SMEĆE	SMEĆE	SMEĆE
stranicaA	stranicaB	povrsina

Dijagram 2.11. Prikaz sadržaja memorije računala

Neka je računalu naređeno da izvede sljedeće operacije pridruživanja.

```
1.  stranicaA = 5
2.  stranicaB = 2
3.  povrsina = stranicaA*stranicaB
```

Evo što se događa u memoriji računala.

- nakon izvođenja 1. koraka:

100	200	300
5	SMEĆE	SMEĆE
stranicaA	stranicaB	povrsina

- nakon izvođenja 2. koraka:

100	200	300
5	2	SMEĆE
stranicaA	stranicaB	povrsina

- nakon izvođenja 3. koraka:

100	200	300
5	2	10
stranicaA	stranicaB	povrsina

Dijagram 2.12. Prikaz sadržaja memorije računala nakon izvođenja niza pridruživanja

U 3. koraku, prije pridruživanja vrijednosti varijabli `povrsina`, dohvaćaju se vrijednosti varijabli `stranicaA` i `stranicaB`, izvodi se njihovo množenje, te se tako dobivena vrijednost sprema u varijablu `povrsina`. Varijabli se kao vrijednost može pridružiti neka konkretna vrijednost (npr. `stranicaA = 5`), vrijednost neke druge varijable (npr. `stranicaB = stranicaA`) te vrijednost koja se izračunava prema nekoj formuli (npr. `povrsina = stranicaA * stranicaB`).

Jedna posebna vrsta varijable kojoj je vrijednost moguće postaviti samo jednom u računalnome programu naziva se konstanta. Dakle, konstanta je varijabla s predefiniranom vrijednošću koja se u računalnome programu ne može mijenjati.

Neka su dana sljedeća dva jednostavna računalna programa:

1. program:

```
1.  maloprodajnaCijenaGoriva = 1.25 * veleprodajnaCijenaGoriva
2.  maloprodajnaCijenaMaterijala = 1.25*veleprodajnaCijenaMaterijala
3.  maloprodajnaCijenaUsluge = 1.25 * veleprodajnaCijenaUsluge
```

Algoritam 2.1.

2. program:

```
1.  PDV = 1.25
2.  maloprodajnaCijenaGoriva = PDV * veleprodajnaCijenaGoriva
3.  maloprodajnaCijenaMaterijala = PDV * veleprodajnaCijenaMaterijala
4.  maloprodajnaCijenaUsluge = PDV * veleprodajnaCijenaUsluge
```

Algoritam 2.2.

Oba programa ispravno računaju maloprodajnu cijenu. No, u drugome se programu koristi konstanta imena `PDV` kojoj je vrijednost 1.25 samo jednom dodijeljena. U slučaju da dođe do promjene iznosa `PDV`-a na 1.23, to bi značilo da bi se u prvome programu trebalo svaku pojavu vrijednosti 1.25 zamijeniti s vrijednošću 1.23. Pri ovome postupku lako je moguće pogriješiti (npr. zaboraviti zamijeniti neku vrijednost ili zamijeniti neku pogrešnu vrijednost i sl.). U drugome programu je dovoljno samo izmijeniti vrijednost konstante `PDV` (izmjena na samo jednom mjestu značajno smanjuje mogućnost pojave pogreške).

2.3.4. Tip podatka varijable

Tip podatka varijable definira koliko će računalo rezervirati prostora u svojoj memoriji za tu varijablu, te način na koji će se spremljena binarna vrijednost interpretira (npr. je li to cijeli broj, decimalni broj, znak i sl.). Sljedeća tablica prikazuje vrste podataka:

<table>
<tr><td rowspan="4">Interni</td><td rowspan="3">Statički</td><td rowspan="2">Jednostavni</td><td>Cjelobrojni (npr. -2)
Znakovni (npr. $)
Logički (npr. False)
Realni (npr. 0.5639)</td></tr>
<tr><td>Složeni</td><td>Polje (npr. {10, 2, 4})
Slog (npr. {Marko, 10, 2.65})</td></tr>
<tr><td colspan="2">Dinamički</td><td>Lista, Stog, Red, Stablo</td></tr>
<tr><td rowspan="3">Eksterni</td><td rowspan="2">Datoteke</td><td>Tekstualne</td><td>Niz znakova (ASCII kod)</td></tr>
<tr><td>Tipizirane</td><td>Niz slogova (sekvencijalne i indeksirane)</td></tr>
<tr><td colspan="2">Baze podataka</td><td>Niz tablica (SLQ jezik)</td></tr>
</table>

Tablica 2.4. Vrste podataka

Vrste podataka se dijele na interne i eksterne.

Interne vrste podataka su one koje se čuvaju u radnoj memoriji računala (RAM) i dijele se na **statičke** i **dinamičke**. Statička je vrsta podatka ona kojoj računalo za čuvanje dodjeljuje fiksnu veličinu memorijskoga prostora, dok dinamičkoj vrsti dodjeljuje memorijski prostor koji nije fiksne veličine već on može rasti ili padati (ovisno o vrijednosti podatka). Statička se vrsta podataka dalje dijeli na **jednostavne** i **složene**. Jednostavna vrsta podataka je ona koja je dalje nedjeljiva (cjelobrojni, znakovni, logički i realni tip podatka). Složena je vrsta podataka ona koja se sastoji iz jednostavnih (polje i slog). Kao primjer dinamičke vrste podataka može se navesti lista, stog, red, stablo i sl.

Eksterne vrste podataka su one koje se čuvaju na vanjskim nositeljima podataka. Dijele se na **datoteke** i **baze podataka**. Datoteka predstavlja imenovani skup znakova (tada se radi o **tekstualnoj** datoteci gdje su sačuvani ASCII kodovi znakova) ili imenovani skup slogova (tada se radi o **tipiziranoj** datoteci koja može biti sekvencijalna – pristup slogovima je u slijedu; ili indeksirana – pristup slogovima je

izravan preko indeksa). Baze podataka predstavljaju skup međusobno povezanih tablica s kojima se manipulira putem posebnoga jezika (SQL).

2.3.5. Trajanje varijable

Jednom deklarirana varijabla u nekome računalnom programu (čime se definira njezin tip podatka i identifikator) rezervira odgovarajući prostor u memoriji računala. Rezerviranjem memorijskoga prostora smanjuje se veličina slobodne memorije koja stoji na raspolaganju za rezervaciju drugim varijablama i drugim računalnim programima. Pitanje koje se postavlja je koliko dugo računalo treba držati neki memorijski prostor rezerviran. Jasno je da završetkom izvođenja nekoga računalnog programa, cjelokupan memorijski prostor koji je bio rezerviran za varijable unutar toga računalnog programa sada mora biti oslobođen. No, postoje slučajevi u kojima, tijekom izvođenja računalnoga programa, a prije njegova završetka, neke varijable više nisu potrebne (npr. tijekom izvođenja procedura - o njima u narednim poglavljima). Tada računalo može i prije završetka računalnoga programa izvesti oslobađanje memorijskoga prostora od onih varijabli koje više nisu potrebne. Trajanje varijable je vezano s mjestom i načinom deklariranja varijable i ono pokazuje koliko dugo računalo treba rezervirati neki memorijski prostor.

2.3.6. Doseg varijable

Varijabli koja je deklarirana i koristi se u jednome računalnom programu ne može se imenom pristupiti iz drugoga računalnog programa. Dakle, njezin doseg pristupa je računalni program u kome je deklarirana. No, doseg varijable je moguće ograničiti i u samom računalnom programu kojemu pripada. Naime, moguće je pristup varijabli ograničiti tako da je ona dostupna samo iz određenoga dijela računalnoga programa (npr. samo unutar procedure). Doseg varijable se definira mjestom i načinom njezine deklaracije i on pokazuje iz kojih je dijelova računalnoga programa moguć pristup varijabli.

2.4. Varijable u C

U nastavku će biti prikazano kako se u programskome jeziku C koriste varijable. Bit će prikazani način njezine deklaracije, jednostavni tipovi podataka kojima može biti deklarirana, dodjeljivanje vrijednosti i rada s memorijskom adresom. Bit će prikazani i složeni tipovi podataka - polje, niz znakova (posebna vrsta polja) i zapis. Prikazat će se i operacije koje se mogu izvesti nad varijablama.

2.4.1. Deklaracija varijabli i osnovni tipovi podataka

Već je rečeno da se prije uporabe varijable u računalnome programu ona mora deklarirati. Također je rečeno da se deklaracijom varijable definiraju njezin tip podatka i identifikator. Mjesto deklaracije kao i primjena posebnih ključnih riječi definira trajnost i opseg varijable (o tome više u narednim poglavljima).

Sljedeća tablica prikazuje jednostavne tipove podataka u C.

Tip	C jezik (ključna riječ)	Interval	Zauzeće memorije
znakovni tip	`[signed] char` `unsigned char`	-127 .. 128 0 .. 255	1 byte 1 byte

cjelobrojni tip	`[signed] int`
	`[signed] short`
	`[signed] long`
		-2147483648.. 2147483647
		-32768 .. 32767
		-2147483648.. 2147483647
			4 byte
			2 byte
			4 byte

kardinalni tip	`unsigned [int]`
	`unsigned short`
	`unsigned long`
		0 .. 4294967295
		0 .. 65535
		0 .. 4294967295
			4 byte
			2 byte
			4 byte

realni tip	`float`	min ± 1.175494351e-38 maks ± 3.402823466e+38	4 byte
realni tip dvostruke preciznosti	`double`	min ± 2.2250738585072014e-308 maks ± 1.7976931348623158e+308	8 byte

Tablica 2.5. Tipovi podataka u C

Uglate zagrade kod ključnih riječi označavaju da se ta ključna riječ može i izostaviti. Ključna riječ `unsigned` označava da će se koristiti samo pozitivni brojevi i nula (ne koristi se onaj jedan bit za označavanje predznaka broja).

Pretpostavimo da se žele deklarirati varijable `stranicaA`, `stranicaB` i `povrsina`. Neka ove varijable služe za vrijednosti koje će biti cijeli pozitivni brojevi (npr. tip podatka `unsigned int`). Neka varijabla `stranicaA` ima vrijednost 5, a varijabla `stranicaB` vrijednost 2. Neka se vrijednost varijable `povrsina` dobiva tako da se pomnože vrijednosti varijabli `stranicaA` i `stranicaB`. Sve navedeno bi u računalnome programu pisanom u programskom jeziku C izgledalo kako slijedi:

```
1.  main(){
2.    unsigned int stranicaA, stranicaB, povrsina;
3.    stranicaA = 5;
4.    stranicaB = 2;
5.    povrsina = stranicaA * stranicaB;}
```

Program 2.1.

Kako bi računalo znalo gdje se nalazi točka od koje treba početi izvoditi računalni program, u njemu se mora nalaziti funkcija `main` (1. linija koda). Unutar vitičastih zagrada (što predstavlja blok naredbi koje će računalo izvesti) izvedena je deklaracija varijabli (2. linija koda) te su im potom dodijeljene vrijednosti (3., 4. i 5. linija koda). Vidljivo je da su, prije same primjene varijabli (dodjele vrijednosti), one deklarirane. U protivnom bi računalo javilo pogrešku. Još se treba uočiti da naredba u C-u završava s točkom-zarez (;). Općenita deklaracija varijabli u C glasi:

```
Tip_podatka1 ime_Varijable11, imeVarijable21, ... imeVarijablen1;
Tip_Podatka2 ime_Varijable12, imeVarijable22, ... imeVarijablen2;
```

Dakle, navodi se tip podatka (njegova ključna riječ) te se nižu nazivi varijabli koji će biti toga tipa podatka. Sve završava s točka-zarez. Potom slijedi novi tip podatka i novi niz naziva varijabli. Već je rečeno, ali valja ponoviti – u programskome jeziku C nazivi varijabli su osjetljivi na mala i velika slova. To znači, ako je deklarirana varijabla naziva `stranicaA`, onda se u računalnome programu mora koristiti upravo takav naziv za tu varijablu, a ne npr. `STRANICAa`. U prethodnome poglavlju je navedeno koji znakovi se mogu koristiti u nazivu varijable i koji je njihov mogući raspored. Također je rečeno da identifikator ne može biti neka ključna riječ u programskome jeziku (riječ posebnoga značenja – npr. naredba). Ključnih riječi u C-u ima 32 (definirane su standardom ANSI C) i pišu se malim slovima. To su:

`auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile i while`.

Pri deklaraciji varijable treba paziti na to koje sve vrijednosti varijabla može poprimati tijekom izvođenja programa, a to ovisi o semantici (značenju) varijable. Npr. ako varijabla treba čuvati vrijednost koja predstavlja broj radnika na nekom projektu, tada je sigurno da će ta varijabla (npr. naziva `br_radnika`) biti cijeli pozitivan broj (`unsigned int`), dok će tečaj neke valute biti decimalni broj pa će varijabla (npr. `tecaj_eura`) biti tipa podatka `float` (ili možda `double`). Pogrešan izbor tipa podatka varijable vodi do pogreške tijekom izvođenja računalnoga programa.

U prethodnome je poglavlju spomenuta potreba za postojanjem posebnih vrsti varijabli – konstanti. U programskom jeziku C, konstante se mogu deklarirati primjenom ključne riječi `const` u deklaraciji varijable.

Neka se deklariraju sljedeće varijable:

`maloprodajnaCijenaGoriva`, `veleprodajnaCijenaGoriva`, `maloprodajnaCijenaMaterijala`, `veleprodajnaCijenaMaterijala`, `maloprodajnaCijenaUsluga`, `veleprodajnaCijenaUsluga`.

Sve varijable trebaju moći čuvati decimalne vrijednosti (cijene). Izračun maloprodajne cijene se dobiva tako da se veleprodajna poveća za porez na dodanu vrijednost. Vrijednost veleprodajnih cijena je definirana. Neka je porez na dodanu vrijednost definiran u konstanti. Računalni program u programskome jeziku C je:

```
1.  main(){
2.     const float PDV = 1.25;
3.     float maloprodajnaCijenaGoriva,veleprodajnaCijenaGoriva = 9.5;
4.     float maloprodajnaCijenaMaterijala,veleprodajnaCijenaMaterijala = 10.1;
5.     float maloprodajnaCijenaUsluga,veleprodajnaCijenaUsluga= 100.34;
6.     maloprodajnaCijenaGoriva = PDV * veleprodajnaCijenaGoriva;
7.     maloprodajnaCijenaMaterijala=PDV * veleprodajnaCijenaMaterijala;
8.     maloprodajnaCijenaUsluga = PDV * veleprodajnaCijenaUsluga;}
```

Program 2.2.

U 2. se liniji koda deklarira konstanta tipa podatka `float` (primjena ključne riječi `const`) čije ime je `PDV` i odmah joj se dodjeljuje vrijednost. U 3., 4. i 5. se liniji koda

deklariraju potrebne varijable, a nekima se odmah dodjeljuju i vrijednosti (moguća je ovakva dodjela u deklaraciji). U 6., 7. i 8. se liniji koda izračunavaju se maloprodajne cijene uz primjenu konstante `PDV` i spremaju se u odgovarajuće varijable. Kad bi se izvelo naknadno dodjeljivanje vrijednosti konstanti `PDV` (npr. u 6. liniji se stavi `PDV = 2`), računalo bi prilikom pokretanja računalnoga programa javilo pogrešku i izvođenje bi se zaustavilo.

2.4.2. Pokazivač, polje, niz znakova i zapis

U nastavku će se detaljnije pojasniti pojmovi pokazivač, polje, niz znakova i zapis. Pokazivači omogućavaju manipulaciju s varijablama, odnosno njihovim vrijednostima putem adrese na kojoj se vrijednost nalazi. Polje, niz znakova i zapis omogućavaju istovremeno čuvanje više različitih vrijednosti istoga (polje i niz znakova) ili različitoga (zapis) tipa podatka u istoj varijabli.

2.4.2.1. Pokazivač

Jedno od svojstava varijable je i njezina adresa, odnosno adresa rezervirane memorijske lokacije koja je dodijeljena varijabli. Svaka deklarirana varijabla ima pridruženu memorijsku adresu te samu vrijednost koju u sebi čuva.

Neka su deklarirane varijable `stranicaA, stranicaB i povrsina`. Neka ove varijable služe za čuvanje vrijednosti koje će biti cijeli pozitivni brojevi (npr. tip podatka `unsigned int`). Neka varijabla `stranicaA` ima vrijednost 5, a varijabla `stranicaB` vrijednost 2. Neka se vrijednost varijable `povrsina` dobiva tako da se pomnože vrijednosti varijabli `stranicaA` i `stranicaB`. Sve bi navedeno u računalnome programu pisanom u programskome jeziku C izgledalo kako slijedi:

```
1.  main(){
2.    unsigned int stranicaA, stranicaB, povrsina;
3.    stranicaA = 5;
4.    stranicaB = 2;
5.    povrsina = stranicaA * stranicaB;}
```

Program 2.3.

Nakon izvršavanja 5. linije koda, sadržaj memorije računala je:

100	200	300
5	2	10
stranicaA	stranicaB	povrsina

Dijagram 2.13. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.3.)

Pitanje koje se postavlja je kako doći do memorijske adrese (u prikazanome primjeru 100, 200 i 300) dodijeljene nekoj varijabli. Da bi se uzela vrijednost koju varijabla čuva, jednostavno se koristi identifikator varijable (npr. želimo li doći do vrijednosti varijable `stranicaA`, jednostavno koristimo njezin identifikator - vidjeti 5. liniju

koda). No, da bi se došlo do vrijednosti memorijske adrese na kojoj se čuva vrijednost varijable `stranicaA`, potrebno je koristiti adresni operator `&` (dakle, `&stranicaA`). Drugim riječima, kada bi se u 5. liniji koda navelo `povrsina = &stranicaA * stranicaB`; vrijednost koja bi se upisala u varijablu `povrsina` bi bila jednaka umnošku memorijske adrese varijable `stranicaA` i vrijednosti podatka koji čuva varijabla `stranicaB` (`povrsina = 100 * 2 = 200`).

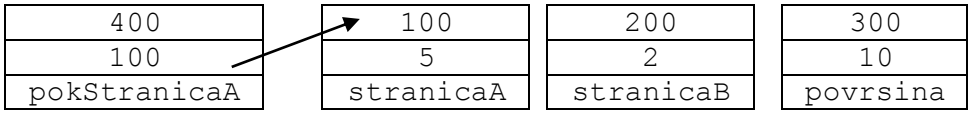
Postoji posebna vrsta varijabli, **pokazivači**, koji kao vrijednost čuvaju memorijsku adresu neke druge varijable (i pokazivači imaju svoju memorijsku adresu). Oni dakle pokazuju gdje se nalazi stvarna vrijednost podatka tako da u sebi čuvaju memorijsku adresu varijable koja čuva tu stvarnu vrijednost. Varijabla koja je tipa pokazivača se deklarira isto kao i klasična varijabla, s razlikom što se ispred imena varijable stavlja znak *****.

Neka se u prethodni program doda deklaracija pokazivača `pokStranicaA` koji će u sebi čuvati memorijsku adresu varijable `stranicaA`.

```
1.  main(){
2.    unsigned int stranicaA, stranicaB, povrsina;
3.    unsigned int *pokStranicaA;
4.    pokStranicaA = &stranicaA;
5.    stranicaA = 5;
6.    stranicaB = 2;
7.    povrsina = stranicaA * stranicaB;}
```

Program 2.4.

Nakon izvršavanja 7. linije koda, sadržaj memorije računala je:



Dijagram 2.14. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.4.)

U 3. je liniji koda izvršena deklaracija pokazivača. Ključno je da se pokazivač treba deklarirati istim tipom podatka koji ima varijabla čiju će adresu čuvati. Valja uočiti da i pokazivač ima svoju adresu (u primjeru je to 400) te da se do nje dolazi kako je već opisano (adresnim operatorom &). Opći oblik deklaracije pokazivača se stoga može zapisati kako slijedi:

```
Tip_podatka *ime_Varijable;
```

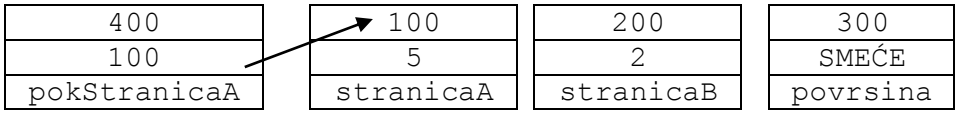
Osim putem identifikatora varijable, s vrijednosti koja se čuva u memoriji računala može se manipulirati i putem adrese memorijske lokacije na kojoj se ta vrijednost nalazi. U tu se svrhu koristi posebni operator ***** - operator dereferenciranja ili indirekcije. Ovaj operator se koristi nad pokazivačima, odnosno mrežnim adresama koje se čuvaju u pokazivačima.

Pristup vrijednosti preko memorijske adrese prikazuje sljedeći program:

```
1.  main(){
2.    unsigned int stranicaA, stranicaB, povrsina;
3.    unsigned int *pokStranicaA;
4.    pokStranicaA = &stranicaA;
5.    stranicaA = 5;
6.    stranicaB = 2;
7.    povrsina = *pokStranicaA * stranicaB;}
```

Program 2.5.

Nakon izvođenja 6. linije koda, sadržaj memorije računala je sljedeći:



Dijagram 2.15. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.5.)

Dakle, sve varijable, osim varijable `povrsina`, imaju neku vrijednost. U 7. se liniji koda vrijednost varijable `povrsina` izračunava tako da se uzme vrijednost koja se čuva na memorijskoj adresi koja se nalazi u varijabli `pokStranicaA`. Kako se u varijabli `pokStranicaA` nalazi vrijednost 100, to se uzima vrijednost s memorijske adrese 100, a to je broj 5. Taj broj 5 se množi s vrijednošću koja se nalazi u varijabli `stranicaB` (a to je 2), te se tako dobiva konačni rezultat koji se sprema u varijablu `povrsina` (a to je 10). Upravo `*pokStranicaA` (dakle operator *) kaže da se treba uzeti ne vrijednost koja se čuva u `pokStranicaA`, već ona vrijednost koja se nalazi na memorijskoj lokaciji na koju pokazuje `pokStranicaA`. Kad bi se u 7. liniji koda nalazilo `povrsina = pokStranica * stranicaB;` (dakle bez operatora dereferenciranja), za izračun bi se koristilo upravo ono što se čuva u `pokStranicaA` (dobilo bi se `povrsina = 100 * 2 = 200`).

Valja zapamtiti da prije uporabe pokazivača, osim njegove deklaracije (što vrijedi za sve varijable), treba izvesti dodjeljivanje vrijednosti, tj. odgovarajuće memorijske adrese (nakon deklaracije i u pokazivaču se odmah nalazi *SMEĆE*). Vrijednost *SMEĆE* koje po deklaraciji pokazivač poprima, se može interpretirati kao valjana adresa neke memorijske lokacije, ali ta memorijska lokacija nužno ne mora biti rezervirana za program koji se izvodi. Zbog toga se pri primjeni pokazivača sa *SMEĆEM* može dogoditi pristup nedopuštenoj memorijskoj lokaciji što u konačnici može izazvati prekid izvođenja programa.

2.4.2.2. Polje

Do sada je pokazano da jedna varijabla u jednome trenutku može u memoriji računala čuvati samo jednu vrijednost. Primjera radi:

```
1.  main(){
2.    unsigned int stranicaA, stranicaB, povrsina;
3.    stranicaA = 5;
4.    stranicaB = 2;
5.    povrsina = stranicaA * stranicaB;}
```

Program 2.6.

Nakon izvođenja 5. linije koda, sadržaj memorije računala je:

100	200	300
5	2	10
stranicaA	stranicaB	povrsina

Dijagram 2.16. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.6).

Vidljivo je da varijable čuvaju samo jednu vrijednost (5, 2 i 10). Pitanje koje se postavlja je – je li moguće da jedna varijabla u istome trenutku čuva više vrijednosti? Npr. varijabla `stranicaA`, osim što trenutno čuva broj 5, može li istovremeno čuvati i npr. broj 3.

Navedeno je moguće, deklaracijom varijable kao polja unutar kojega je istovremeno moguće čuvanje više različitih vrijednosti koje su sve istoga tipa podatka. Potom se do pojedine vrijednosti dolazi preko identifikatora varijable i indeksa na komu se vrijednost nalazi.

Npr. ako u varijabli `stranicaA`, `stranicaB` i `povrsina` želimo imati mogućnost istovremenoga čuvanja dviju vrijednosti, tada bi smo ove varijable deklarirali kao polja.

```
1.  main(){
2.    unsigned int stranicaA[2], stranicaB[2], povrsina[2];
3.    stranicaA[0] = 5;
4.    stranicaB[0] = 2;
5.    stranicaA[1] = 3;
6.    stranicaB[1] = 4
7.    povrsina[0] = stranicaA[0] * stranicaB[0];
8.    povrsina[1] = stranicaA[1] * stranicaB[1];}
```

Program 2.7.

Nakon 8. linije koda, sadržaj memorije računala je sljedeći:

100	104
5	3
stranicaA[0]	stranicaA[1]

200	204
2	4
stranicaB[0]	stranicaB[1]

300	304
10	12
povrsina[0]	povrsina[1]

Dijagram 2.17. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.7).

Ono je što se odmah zapaža da se u memoriji računala za pojedinu varijablu rezerviralo onoliko memorijskih lokacija iste veličine koliko će varijabla imati vrijednosti, odnosno koliko će polje imati elemenata. Svaka vrijednost varijable zapisana je u svoju memorijsku lokaciju koja slijede jedna za drugom (npr. za varijablu `stranicaA` memorijske lokacije imaju adrese 100 i 104 - ovo je stoga što tip podatka `unsigned int` zauzima 4 bajta, tj. 4 memorijske lokacije - dakle, ako je prva na adresi 100 onda druga koja joj slijedi mora biti na adresi 100 + 4 = 104). Do pojedine vrijednosti u varijabli (odnosno polju) dolazi se preko njezina indeksa. Indeksi idu od nule do deklariranoga broja elemenata minus 1.

U 7. se liniji koda izračunava vrijednost tako da se pomnože vrijednosti koje se čuvaju na indeksu 0 varijabli `stranicaA` i `stranicaB`, te se tako dobiveni rezultat smješta na indeks 0 varijable `povrsina`. Slično se događa i u 8. liniji.

Općenito se deklaracija varijable vrste polja može prikazati na sljedeći način:

```
Tip_podatka ime_Varijable[Broj_elemenata];
```

Polje koje je deklarirano nekim tipom podatka može čuvati samo vrijednosti toga tipa podatka. Dakle, nije moguće da se u polje tipa podatka npr. `unsigned int` (cijeli pozitivni broj) smjesti npr. decimalni broj (`float`). Sukladno prethodnome primjeru, nije moguće `stranicaA[0] = 0.4325`.

Do sada prikazano polje naziva se jednodimenzionalno polje budući da ima samo jednu dimenziju, odnosno jedan indeks. Jednodimenzionalno polje se može shvatiti kao niz podataka istoga tipa.

Postoje i dvodimenzionalna, trodimenzionalna i višedimenzionalna polja. To su polja koja imaju dvije, tri ili više dimenzija (odnosno dva, tri ili više indeksa). Dvodimenzionalno polje se može shvatiti kao tablica podataka istoga tipa, a trodimenzionalno polje kao listovi s tablicama podataka istoga tipa.

Do sada se vidjelo da svako deklarirano polje ima identifikator koji je zajednički svim elementima polja, te indeks preko koga se pristupa vrijednosti pojedinog elementa polja (npr. sa `stranicaA[0]` se pristupa vrijednosti koja se nalazi na indeksu 0 polja `stranicaA`). No, sam identifikator polja (u prikazanom primjeru `stranicaA`) u sebi čuva adresu prvoga elementa polja. Do adrese prvoga elementa polja može se doći i primjenom adresnoga operatora `&` (npr. `&stranicaA[0]`). Neka je dano sljedeće polje deklarirano tipom podataka `int`.

100	104
5	3
stranicaA[0]	stranicaA[1]

Dijagram 2.18. Prikaz sadržaja memorije računala za polje `stranicA` s dva elementa.

Tada identifikator `stranicaA` u sebi čuva adresu 100. Do ove se adrese može doći i s adresnim operatorom `&stranicaA[0]`. Do adrese 104 (dakle drugog elementa

polja) se može doći ili preko adresnoga operatora `&stranicaA[1]` ili operacijom `stranicaA+1`. Važno je reći da se identifikatoru polja ne može pridružiti nova adresa, tj. nije moguće napraviti `stranicaA = 108`. Identifikator polja je konstanta.

Sljedeći primjer prikazuje manipulaciju s vrijednostima polja preko pokazivača.

```
 1.  main(){
 2.    unsigned int stranicaA[2], stranicaB[2], povrsina[2];
 3.    unsigned int *pokStranica;
 4.    pokStranica = stranicaA;
 5.    *pokStranica = 5;
 6.    *(pokStranica + 1) = 3;
 7.    *stranicaB = 2;
 8.    *(stranicaB + 1) = 4;
 9.    povrsina[0] = stranicaA[0] * stranicaB[0];
10.    povrsina[1] = stranicaA[1] * stranicaB[1];}
```

Program 2.8.

U liniji 3. je deklariran pokazivač kojemu se u liniji 4. dodjeljuje adresa prvoga elementa polja `stranicaA` (nema adresnoga operatora & budući da se u samom identifikatoru nalazi adresa). U sljedećoj se liniji koda na memorijsku lokaciju adrese koja se čuva u pokazivaču (operator dereferenciranja *) smješta vrijednost 5. U biti se vrijednost 5 smješta u polje `stranicaA` na indeks 0 budući da pokazivač upravo pokazuje na tu adresu. U liniji 6. vrijednost 3 se smješta na adresu koja je dobivena tako da se adresa koju trenutno čuva pokazivač poveća za 1. U biti se na taj način dolazi do adrese sljedećega elementa polja `stranicaA`. Slično se može objasniti linija 7. i 8. samo što ovdje nije korištena posebna varijabla za pokazivač, već su iskorištene vrijednosti adresa koje se nalaze u identifikatoru polja.

2.4.2.3. Niz znakova

Niz znakova je posebna vrsta jednodimenzionalnoga polja koje u sebi čuva vrijednosti tipa podatka `char` (znak). Ponekad se niz znakova još naziva i `string`, a u nekim programskim jezicima postoji i posebni tip podatka `string`. Svrha niza znakova je omogućavanje čuvanja teksta kao niza znakova u memoriji računala.

Pretpostavimo da se u memoriji računala želi sačuvati riječ "PROGRAM". Računalni program pisan u programskome jeziku C izgleda kako slijedi:

```
 1.  main(){
 2.    char rijec[] = "PROGRAM";}
```

Program 2.9.

Sadržaj memorije računala nakon 2. linije koda je:

100	101	102	103	104	105	106	107
P	R	O	G	R	A	M	\0
rijec[0]	rijec[1]	rijec[2]	rijec[3]	rijec[4]	rijec[5]	rijec[6]	rijec[7]

Dijagram 2.19. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.9.)

U 2. liniji koda deklariran je niz znakova `rijec` i odmah mu je dodijeljena vrijednost "PROGRAM". Računalo je zbog toga u memoriji rezerviralo slijedno memorijske lokacije za svaki pojedini znak u riječi te za poseban *null* znak (`\0`) koji označava kraj niza. Ukupno je stoga rezervirano 8 memorijskih lokacija, odnosno polje tipa podatka `char` ima 8 elemenata. Vidljivo je da su adrese susjednih memorijskih lokacija udaljene točno za 1 (100, 101, 102, ..., 107). To je zbog toga što tip podatka `char` treba memorijsku lokaciju veličine jednog bajta pa je sljedeća adresa upravo jednaka prethodnoj + 1.

Želi li se dodijeliti neka vrijednost varijabli deklariranoj kao niz znakova, nakon njezine deklaracije, tada se to ne može učiniti jednostavnom primjenom operatora pridruživanja (=) varijabli budući da se radi o polju (identifikator polja u sebi čuva adresu prvoga elementa i uz to je konstanta – ne može mu se mijenjati vrijednost), već se svakom elementu polja treba dodjeljivati po jedan znak. Dakle, nije moguće učiniti kako slijedi:

```
1.  main(){
2.    char rijec[7+1];
3.    rijec = "PROGRAM";}
```

Program 2.10.

Potrebno je napraviti sljedeće:

```
 1.  main(){
 2.    char rijec[7+1];
 3.    rijec[0] = 'P';
 4.    rijec[1] = 'R';
 5.    rijec[2] = 'O';
 6.    rijec[3] = 'G';
 7.    rijec[4] = 'R';
 8.    rijec[5] = 'A';
 9.    rijec[6] = 'M';
10.    rijec[7] = '\0';}
```

Program 2.11.

Ovdje treba obratiti pažnju na liniju 10. gdje je kao zadnji element dodan null znak '`\0`'.

Jednostavnije se pridruživanje vrijednosti nizu znakova može postići primjenom posebne funkcije `strcpy`. U tome slučaju pridruživanje izgleda kako slijedi:

```
1.  main(){
2.    char rijec[7+1];
3.    strcpy(rijec ,"PROGRAM");}
```

Program 2.12.

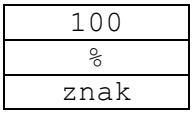
Ovdje treba obratiti pažnju da se vrijednost koja se dodjeljuje nizu znakova u funkciji `strcpy` stavlja u dvostruke navodnike (u primjeru "PROGRAM").

U narednim će se poglavljima prikazati neke funkcije u C-u koje se koriste u radu s nizom znakova (npr. za određivanje duljine niza znakova i sl.).

Odnos pokazivača i niza znakova je sličan kao i kod polja budući da niz znakova nije ništa drugo do polje tipa podatka `char`.

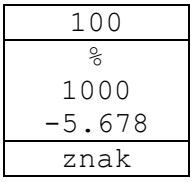
3.4.2.4. Zapis

Osnovno svojstvo svih do sada prikazanih varijabli je da one čuvaju vrijednost onoga tipa podatka kojom su deklarirane. Primjera radi ako je deklarirana varijabla `znak` tipom podatka `char`, tada se kao njezina vrijednost može pojaviti samo neki znak (npr. %). U memoriji računala postojala bi sljedeća situacija:



Dijagram 2.20. Prikaz sadržaja memorije računala za varijablu `zapis` koja čuva jednu vrijednost

Pitanje koje se postavlja je - je li moguće imati takvu varijablu koja bi u isto vrijeme mogla čuvati više vrijednosti, ali različitoga tipa podatka, tj. je li moguće imati sljedeću situaciju u memoriji računala:



Dijagram 2.21. Prikaz sadržaja memorije računala za varijablu `zapis` koja čuva više vrijednosti

Varijabla koja omogućava istovremeno čuvanje više vrijednosti različitih tipova podatka je zapis (slog, struktura). Dakle, zapis predstavlja skupinu podataka koji mogu biti različitoga tipa podatka. Zapis tvori novi složeni tip podatka koji se može koristiti u deklaraciji varijable.

Izrada zapisa se sastoji iz dva koraka:

1. Definiranje sastava zapisa - zapis se sastoji iz komponenti koje imaju svoj tip podatka i jedinstveno ime. U biti se radi se o pojedinačnoj deklaraciji varijabli. Definiranom se sastavu zapisa daje neki identifikator (pravila su ista kao i pri definiranju identifikatora varijabli). Ovaj identifikator je u

biti identifikator novoga složenog tipa podatka. Nepisano pravilo je da identifikator zapisa počinje velikim početnim slovom.

2. Deklaracija varijable složenim tipom podatka. Varijabla se deklarira tako što se navede identifikator zapisa (to je njezin tip podatka) te ime varijable.

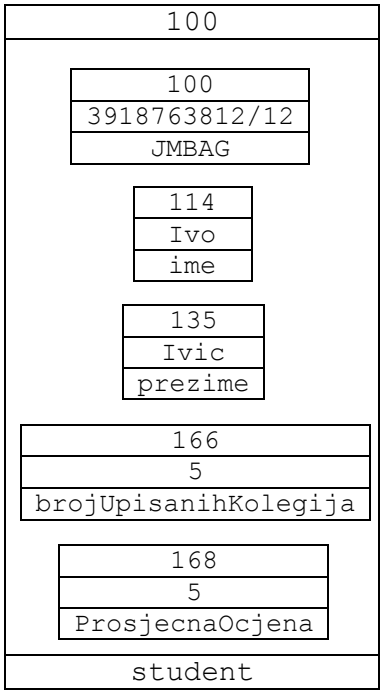
Neka se u memoriji računala želi sačuvati informacija o studentu koja se sastoji iz sljedećih podataka: JMBAG, ime, prezime, broj upisanih kolegija, prosječna ocjena. Računalni program koji ovo podržava je:

```
1.  typedef struct {
2.    char JMBAG[14];
3.    char ime[21];
4.    char prezime[31];
5.    unsigned short brojUpisanihKolegija;
6.    float prosjecnaOcjena;} InformacijaOStudentu;
7.
8.  main(){
9.    InformacijaOStudentu student={"3918763812/12","Ivo",
                                    "Ivic", 5, 4.65};}
```

Program 2.13.

U linijama koda pod rednim brojem 1, 2, 3, 4, 5 i 6 definira se sastav zapisa, odnosno novoga složenog tipa podatka kome je dana identifikacija (ime) `InformacijaOStudentu`. Vidljivo je da se zapis sastoji iz ukupno pet komponenti od kojih je svakoj definiran njezin tip podatka i njezina identifikacija (`JMBAG`, `ime`, itd.). Da bi se ovako definirani zapis primijenio, potrebno je deklarirati varijablu koja će biti toga novog složenoga tipa podatka (9. linija koda). Kao naziv tipa podatka koristi se identifikacija zapisa (`InformacijaOStudentu`). U istoj liniji koda je izvedena dodjela vrijednosti pojedinoj komponenti zapisa (iako je linija koda podijeljena u dva reda, i dalje se radi o istoj jednoj liniji koda). Sadržaj memorije računala nakon 9. linije prikazan je na dijagramu 2.22.

Može se vidjeti da je računalo rezerviralo memorijsku lokaciju za varijablu `student`, ali i za sve komponente iz kojih se sastoji zapis (`JMBAG`, `ime` itd.). Odnos slijednih memorijskih adresa odgovara potrebnoj veličini za odgovarajući tip podatka (npr. `JMBAG` se čuva na adresi 100, a `Ime` na adresi 114 zato što `JMBAG-u` treba 14 bajtova - tip podatka je `char`). Pristupanje pojedinoj komponenti zapisa izvodi se preko imena varijable i imena komponente koji su međusobno odvojeni točkom (npr. `student.JMBAG, student.ime` itd.).



Dijagram 2.22. Prikaz sadržaja memorije računala nakon izvođenja programskoga koda (program 2.13.)

Općenito se definicija zapisa može prikazati kako slijedi:

```
typedef struct {
   Tip_Podatka1 ime_Komponente1;
   Tip_Podatka2 ime_Komponente2;
   ...
   Tip_PodatkaN ime_KomponenteN;
} Identifikator_Zapisa;
```

Tada deklaracija varijable glasi:

```
Identifikator_Zapisa ime_Varijable;
```

A dolazak do neke komponente zapisa:

```
ime_Varijable.ime_Komponente;
```

Ovakvom notacijom se prethodni primjer može prikazati kako slijedi:

```
 1. typedef struct {
 2.   char JMBAG[14];
 3.   char ime[21];
 4.   char prezime[31];
```

```
 5.   unsigned short brojUpisanihKolegija;
 6.   float prosjecnaOcjena;} InformacijaOStudentu;
 7. main(){
 8.   InformacijaOStudentu student;
 9.   strcpy(student.JMBAG, "3918763812/12");
10.   strcpy(student.ime, "Ivo");
11.   strcpy(student.prezime, "Ivic");
12.   student.brojUpisanihKolegija = 5;
13.   student.prosjecnaOcjena = 4.65;}
```

Program 2.14.

Ako se u radu sa zapisima koristi njegova adresa, odnosno adresa memorijske lokacije rezervirane za zapis, tada se do pojedine njegove komponente može doći na dva načina.

Primjenom adrese zapisa, do komponente se dolazi notacijom ->. To pokazuje primjer koji slijedi:

```
 1. typedef struct {
 2.   char JMBAG[14];
 3.   char ime[21];
 4.   char prezime[31];
 5.   unsigned short brojUpisanihKolegija;
 6.   float prosjecnaOcjena;} InformacijaOStudentu;
 7. main(){
 8.   InformacijaOStudentu student, *pokazivac;
 9.   pokazivac = &student;
10.   strcpy(pokazivac->JMBAG, "3918763812/12");
11.   strcpy(pokazivac->ime, "Ivo");
12.   strcpy(pokazivac->prezime, "Ivic");
13.   pokazivac->brojUpisanihKolegija = 5;
14.   pokazivac->prosjecnaOcjena = 4.65;}
```

Program 2.15.

U liniji je 9. pokazivaču pridružena adresa zapisa. U linijama od 10, do 14. komponentama zapisa se pristupa preko pokazivača, odnosno njegove adrese, te se zbog toga koristi ->.

Pristup komponenti zapisa preko pokazivača, a uz primjenu notacije točke (.), moguć je ako se koristi operator dereferenciranja (*). To pokazuje sljedeći primjer.

```
 1. typedef struct {
 2.   char JMBAG[14];
 3.   char ime[21];
 4.   char prezime[31];
 5.   unsigned short brojUpisanihKolegija;
 6.   float prosjecnaOcjena;} InformacijaOStudentu;
 7.
 8. main(){
 9.   InformacijaOStudentu student, *pokazivac;
10.   pokazivac = &student;
```

```
11.   strcpy((*pokazivac).JMBAG, "3918763812/12");
12.   strcpy((*pokazivac).ime, "Ivo");
13.   strcpy((*pokazivac).prezime, "Ivic");
14.   (*pokazivac).brojUpisanihKolegija = 5;
15.   (*pokazivac).prosjecnaOcjena = 4.65;}
```

Program 2.16.

Ovdje treba obratiti pažnju da je u zagradama stavljeno dereferenciranje – `(*pokazivac)`. Time se računalu naredilo da prvo izvede dereferenciranje pa nakon toga da ide na komponentu zapisa. U protivnom bi se izvelo dereferenciranje komponente zapisa (koja uopće ne sadrži adresu već konkretnu vrijednost).

U prethodnim se primjerima može uočiti da komponenta zapisa može biti polje (u primjeru je to polje tipa podatka `char`, odnosno niz znakova). Komponenta zapisa može biti i drugi zapis. S druge strane i polje može biti deklarirano kao zapis. Npr. žele li se u polju sačuvati podaci o dva studenta, tada bi program izgledao kako slijedi:

```
 1. typedef struct {
 2.   char JMBAG[14];
 3.   char ime[21];
 4.   char prezime[31];
 5.   unsigned short brojUpisanihKolegija;
 6.   float prosjecnaOcjena;} InformacijaOStudentu;
 7.
 8. main(){
 9.   InformacijaOStudentu student[2];
10.   strcpy(student[0].JMBAG, "3918763812/12");
11.   strcpy(student[0].ime, "Ivo");
12.   strcpy(student[0].prezime, "Ivic");
13.   student[0].brojUpisanihKolegija = 5;
14.   student[0].prosjecnaOcjena = 4.65;
15.   strcpy(student[1].JMBAG, "3688753842/11");
16.   strcpy(student[1].ime, "Marko");
17.   strcpy(student[1].prezime, "Markovic");
18.   student[1].brojUpisanihKolegija = 3;
20.   student[1].prosjecnaOcjena = 3.75;}
```

Program 2.17.

Dakle, s poljima koja su deklarirana kao zapis, vrijedi sve ono što je do sada rečeno o poljima i zapisima.

2.5. Osnovne operacije s varijablama

U ovome će se poglavlju prikazati osnovne operacije s varijablama: operacija pridruživanja te algebarske i logičke operacije.

Operacija predstavlja izraz koji formira podatak. Iz izraza se može pročitati njegova semantička vrijednost koja čini (stvara) novu informaciju. Ovaj je izraz sažeta uputa računalu, tj. logički napisan skup **operanada** i **operatora** u cjelinu koji bi trebao rezultirati nekim podatkom. Operand je podatak koji sudjeluje u operaciji, a najčešće

je to neka konkretna vrijednost (npr. 3, -5.87, A itd.) ili vrijednost neke varijable. Operator je znak ili riječ koja određuje postupak koji se treba provesti nad operandom.

2.5.1. Operacija pridruživanja

Operacijom pridruživanja se nekoj varijabli pridružuje vrijednost. U tu se svrhu koristi operator pridruživanja (=). On varijabli, koja se nalazi na lijevoj strani operatora, pridružuje neku vrijednost, koja se nalazi na desnoj strani operatora.

Općenito se operacija pridruživanja može zapisati na sljedeći način:

```
ime_Varijable1 = ime_Varijable2=... ime_Varijablen = vrijednost;
```

Pri operaciji pridruživanja treba voditi brigu o tome da vrijednost koja se pridružuje odgovara tipu podatka kojim je varijabla deklarirana (npr. varijabli tipa podatka `char` se pridružuje vrijednost -35.6 – što nije ispravno). Dakle, operatorom pridruživanja se naređuje računalu da neku vrijednost spremi na memorijsku lokaciju koja je rezervirana za varijablu kojoj se vrijednost dodjeljuje.

Vrijednost može biti neka konkretna vrijednost (npr. 5, A, -0.574 itd.), vrijednost neke druge varijable, vrijednost koja je izračunata algebarskom ili logičkom operacijom te vrijednost koju vraća neka funkcija (npr. u C-u `sqrt` koja vraća drugi korijen broja).

Evo primjera ispravnih i neispravnih pridruživanja u programskome jeziku C:

Neispravna pridruživanja	Ispravna pridruživanja

`8 = 4 * 2`
`3 * 6 = 18`
`3.14159 = pi`
`a = b + 1 = c;`
	`a = b = c = 0;`
	`a = b = c + d;`
	`b = e = f - 3 * 4 + e - b;`

Tablica 2.5. Ispravna i neispravna pridruživanja

Primjer rezultata slijeda pridruživanja:

Slijed pridruživanja	Rezultat nakon svih obavljenih pridruživanja

`a = b = c = 1;`
`d = 2;`
`a = b = c + d;`
	a je 3
	b je 3
	c je 1
	d je 2

`b = e = f = 2;` `b = e = f - 3 * 4 + e - b;`	b je -10 e je -10 f je 2

Tablica 2.6. Slijed pridruživanja i rezultati

Vrijednost je varijabli moguće pridružiti odmah u istoj liniji koda u kojoj se varijabla deklarira:

Primjer pridruživanja	Ekvivalentno sa
`int a = 2;`	`int a;` `a = 2;`

`int a = 2;`
`a = 5;`
	`int a;`
	`a = 2;`
	`a = 5;`

`int a = 2;`
`int b = a;`
`int c = b + a;`
	`int a, b, c;`
	`a = 2;`
	`b = a;`
	`c = b + a;`

Tablica 2.7. Ekvivalentna pridruživanja

U tipu podatka `char` pridruživanje neke konkretne vrijednosti se izvodi tako da se ta vrijednost stavi u navodnike. U nekim jezicima ti navodnici su dvostruki ("), a u nekim jednostruki ('). U C-u se koriste jednostruki navodnici. Npr. želi li se varijabli `slovo` koja je tipa podatka `char` pridružiti vrijednost A, tada se to pridruživanje izvodi na sljedeći način: `slovo = 'A'`.

Pri varijablama koje su deklarirane kao polja, pridruživanje konkretnih vrijednosti se izvodi s pojedinim elementom. Nije moguće izvesti izravno dodjeljivanje cijeloga polja drugoj varijabli. Dakle, ako je deklarirano `int v1[3]` i `int v2[3]` nije moguće izvesti pridruživanje `v1=v2`, već se pridruživanje izvodi element po element polja – `v1[0] = v2[0], v1[1] = v2[1]` itd. Također je moguće izvesti dodjeljivanje `v1[2] = v2[2] = 100`.

Pri nizovima znakova za pridruživanje vrijednosti se može koristiti funkcija `strcpy` ili se pojedini znak pridružuje pojedinom elementu niza preko njegova indeksa (slično kako je to objašnjeno kod polja).

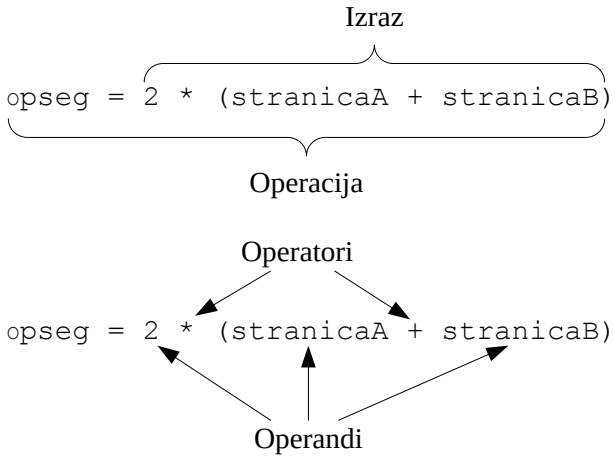
Pri zapisima moguće je vrijednost jednoga zapisa pridružiti vrijednosti drugoga zapisa izravnom primjenom operatora pridruživanja (vidjeti liniju 15. sljedećega programa).

```
 1. typedef struct {
 2.   char JMBAG[14];
 3.   char ime[21];
 4.   char prezime[31];
 5.   unsigned short brojUpisanihKolegija;
 6.   float prosjecnaOcjena;} InformacijaOStudentu;
 7.
 8. main(){
 9.   InformacijaOStudentu student, studentPomocni;
10.   strcpy(student.JMBAG, "3918763812/12");
11.   strcpy(student.ime, "Ivo");
12.   strcpy(student.prezime, "Ivic");
13.   student.brojUpisanihKolegija = 5;
14.   student.prosjecnaOcjena = 4.65;
15.   studentPomocni = student;}
```

Program 2.18.

2.5.2. Algebarske operacije

Algebarska operacija se sastoji iz varijable kojoj se pridružuje vrijednost izračunata nekim algebarskim izrazom. Algebarske izraze grade operandi i algebarski operatori. Primjer algebarske operacije je izračun opsega pravokutnika kome su dane stanice. Neka se rezultat izračuna želi spremiti u varijablu `opseg`, a vrijednosti stranica neka se čuvaju u varijablama `stranicaA` i `stranicaB`. Tada algebarska operacija za izračun opsega (stvaranje novoga podatka) i njegovo spremanje u varijablu `opseg` glasi:



Algebarski operatori se dijele na unarne i binarne. Unarni algebarski operatori su oni koji se primjenjuju u kombinaciji s jednim operandom, dok su binarni oni koji se primjenjuju u kombinaciji s dva operanda.

Unarni operatori su predznak (+, -), inkremencija (++) i dekremencija (--).

Predznak može biti unarni plus ili unarni minus. Primjer primjene ovoga operatora je dan u nastavku.

Algebarske operacije s unarnim operatorom predznaka	Vrijednost varijabli nakon izvođenja

`a = b = 1;`
`a = +a;`
`b = -b;`
	a je 1
	b je -1

`a = b = -1;`
`a = +a;`
`b = -b;`
	a je -1
	b je 1

Tablica 2.8. Primjeri algebarskih izraza s unarnim operatorom predznaka

Inkremencija može biti u obliku postfiks operatora (`X++`) ili prefiks operatora (`++X`). Postfiks operator naređuje računalu da za jedan poveća vrijednost varijable, ali tek nakon njezina korištenja. Prefiks operator naređuje računalu da za jedan poveća vrijednost varijable prije njezina korištenja. Slijede primjeri za operator inkremencije.

Algebarske operacije s unarnim operatorom inkremencije	Ekvivalentno sa	Rezultat ispisa
a = 1 ISPIŠI ++a	a = 1 a = a + 1 ISPIŠI a	2
a = -3 ISPIŠI a ISPIŠI ++a ISPIŠI a	a = -3 ISPIŠI a a = a + 1 ISPIŠI a ISPIŠI a	-3 -2 -2
x = 3 y = ++x ISPIŠI x ISPIŠI y	x = 3 x = x + 1 y = x ISPIŠI x ISPIŠI y	4 4
a = 1 ISPIŠI a++	a = 1 ISPIŠI a a = a + 1	1
a = -3 ISPIŠI a ISPIŠI a++ ISPIŠI a	a = -3 ISPIŠI a ISPIŠI a a = a + 1 ISPIŠI a	-3 -3 -2
x = 3 y = x++ ISPIŠI x ISPIŠI y	x = 3 y = x x = x + 1 ISPIŠI x ISPIŠI y	4 3

Tablica 2.9. Primjeri algebarskih izraza s unarnim operatorom inkremencije

Slično kao i inkremencija, i dekremencija može biti u obliku postfiks (`X--`) ili prefiks operatora (`--X`). Ovdje postfiks operator naređuje računalu da za jedan smanji vrijednost varijable, ali tek nakon njezina korištenja, dok prefiks operator naređuje računalu da za jedan smanji vrijednost varijable prije njezina korištenja. Slijede primjeri za operator dekremencije.

Algebarske operacije s unarnim operatorom dekremencije	Ekvivalentno sa	Rezultat ispisa
a = 1 ISPIŠI --a	a = 1 a = a - 1 ISPIŠI a	0
a = -3 ISPIŠI a ISPIŠI --a ISPIŠI a	a = -3 ISPIŠI a a = a - 1 ISPIŠI a ISPIŠI a	-3 -4 -4
x = 3 y = --x ISPIŠI x	x = 3 x = x - 1 y = x	2 2

`ISPIŠI y`	`ISPIŠI x` `ISPIŠI y`	

`a = 1`
`ISPIŠI a--`
	`a = 1`
	`ISPIŠI a`
	`a = a - 1`
		1

`a = -3`
`ISPIŠI a`
`ISPIŠI a--`
`ISPIŠI a`
	`a = -3`
	`ISPIŠI a`
	`ISPIŠI a`
	`a = a - 1`
	`ISPIŠI a`
		-3
		-3
		-4

`x = 3`
`y = x--`
`ISPIŠI x`
`ISPIŠI y`
	`x = 3`
	`y = x`
	`x = x - 1`
	`ISPIŠI x`
	`ISPIŠI y`
		2
		3

Tablica 2.10. Primjeri algebarskih izraza s unarnim operatorom dekrementacije

Binarni algebarski operatori su zbrajanje (+), oduzimanje (-), množenje (*), dijeljenje (/) te modulo (%).

Modulo predstavlja ostatak pri cjelobrojnome dijeljenju (npr 5 % 2 je 1 jer je 5 podijeljeno s 2 jednako 2 i ostatak je 1). Modulo radi isključivo na operandima s cjelobrojnim tipom podatka (nije moguće izvesti npr. operaciju 2.56 % 2).

Prilikom korištenja operatora dijeljenja treba se voditi briga o tome da djelitelj ne bude nula jer nije moguće izvesti npr. operaciju 45 / 0. Također, u programskome jeziku C kad se dijele dva cijela broja dobiva se cijeli broj, bez obzira što bi dijeljenje trebalo vratiti decimalni broj. To je stoga što se dijeljenjem cijelih brojeva izaziva cjelobrojno dijeljenje, a ne realno. Zbog toga je potrebno izazvati realno dijeljenje tako da se barem jedan operand ponaša kao realni broj. Opisana je situacija prikazana u sljedećoj tablici.

Algebarska operacija	Vrijednost varijabli nakon izvođenja

`int brojnik=5, nazivnik=3;`
`float rezultat;`
`rezultat = brojnik / nazivnik;`
	Varijabla `rezultat` ima vrijednost 1
	Treba biti 1.66666

`int brojnik=2, nazivnik=5;`
`float rezultat;`
`rezultat = brojnik / nazivnik;`
	Varijabla `rezultat` ima vrijednost 0
	Treba biti 0.4

Tablica 2.11. Dijeljenje dva cijela broja vraća cjelobrojni rezultat

Ista situacija, ali s izazvanim realnim dijeljenjem (primjena ključne riječi `float`).

Algebarska operacija	Vrijednost varijabli nakon izvođenja

`int brojnik=5, nazivnik=3;`
`float rezultat;`
`rezultat = (float) brojnik / nazivnik;`
	Varijabla `rezultat` ima vrijednost 1.66666

`int brojnik=2, nazivnik=5;`
`float rezultat;`
`rezultat = brojnik / (float) nazivnik;`
	Varijabla `rezultat` ima vrijednost 0.4

Tablica 2.12. Izazvano realno dijeljenje (ključnom riječi `float`) u dijeljenju cijelih brojeva

Ista situacija, ali s izazvanim realnim dijeljenjem (primjena decimalnoga tipa podatka u algebarskom izrazu – u primjeru je to broj 1.0).

Algebarska operacija	Vrijednost varijabli nakon izvođenja

`int brojnik=5, nazivnik=3;`
`float rezultat;`
`rezultat = 1.0 * brojnik / nazivnik;`
	Varijabla `rezultat` ima vrijednost 1.66666

`int brojnik=2, nazivnik=5;`
`float rezultat;`
`rezultat = 1.0 * brojnik / nazivnik;`
	Varijabla `rezultat` ima vrijednost 0.4

Tablica 2.13. Izazvano realno dijeljenje (primjena decimalnoga tipa podatka) u dijeljenju cijelih brojeva

Prilikom izračuna nekoga algebarskog izraza, računalo ima sljedeće prioritete (isto kao u matematici):

1. zagrade
2. operator množenja, dijeljenja i modulo
3. operator zbrajanja i oduzimanja

Primjera radi, algebarski će izraz 3 + 5 * 3 % 2 rezultirati brojem 4:

3 + 5 * 3 % 2
3 + 15 % 2
3 + 1
4

Prilikom definiranja neke algebarske operacije, u kojoj se vrijednost varijable računa preko njezine stare vrijednosti, može se koristiti skraćeni operator. Neki primjeri su dani u nastavku:

Duže pisanje	Kraće pisanje sa skraćenim operatorom
`a=a+10;`	`a+=10;`
`a=a*15;`	`a*=15;`
`a=a/2;`	`a/=2;`
`a=a-5;`	`a-=5;`

Tablica 2.14. Primjeri skraćenih operatora

2.5.3. Logičke operacije

Logičkom operacijom se računa vrijednost logičkoga izraza koja može biti istina (`True, 1`) ili neistina (`False, 0`). Logički izraz se gradi iz relacijskih i logičkih operatora.

Relacijski operatori (operatori uspoređivanja) su binarni operatori kojima se grade jednostavni logički izrazi. Njima se uspoređuju operandi (vrijednosti podataka). Sljedeća tablica prikazuje sve relacijske operatore i njihovo značenje.

Relacijski operator	Značenje
`a < b`	je li a manje od b
`a <= b`	je li a manje ili jednako b
`a > b`	je li a veće od b
`a >= b`	je li a veće ili jednako b
`a == b`	je li a jednako b
`a != b`	je li a različito od b

Tablica 2.15. Relacijski operatori

Posebnu pažnju treba usmjeriti na simbol za operator jednakosti (==) i simbol za operator različitosti (!=). Ovi simboli se koriste u programskome jeziku C i npr. Java. Posebno treba razlikovati operator pridruživanja (=) od operatora jednakosti (==).

Jednostavni logički izraz izgrađen je od operanada i nekoga relacijskog operatora. Primjer logičkih operacija s jednostavnim logičkim izrazima kao i rezultati njihova izračuna prikazani su u sljedećoj tablici.

Logičke operacije s jednostavnim logičkim izrazom	Vrijednost varijable nakon izvođenja
`unsigned short logickaVrijednost;` `logickaVrijednost = (1>0);`	`logickaVrijednost` je 1. Rezultat jednostavnoga logičkog izraza `1>0` je istina.
`unsigned short logickaVrijednost;` `logickaVrijednost = (10<-5.6);`	`logickaVrijednost` je 0. Rezultat jednostavnoga logičkog izraza `10<-5.6` je neistina.

`unsigned short logickaVrijednost;`
`int stranicaA = 2, stranicaB = 3;`
`logickaVrijednost = (stranicaA == stranicaB);`
	`logickaVrijednost` je 0. Rezultat jednostavnoga logičkog izraza `stranicaA == stranicaB`, odnosno 2==3 je neistina.

`unsigned short logickaVrijednost;`
`int stranicaA = 2, stranicaB = 3;`
`logickaVrijednost = (stranicaA != stranicaB);`
	`logickaVrijednost` je 1. Rezultat jednostavnoga logičkog izraza `stranicaA != stranicaB`, odnosno 2!=3 je istina.

`unsigned short logickaVrijednost;`
`int stranicaA = 2, stranicaB = 3;`
`logickaVrijednost=(stranicaA*stranicaB >= 6);`
	`logickaVrijednost` je 1. Rezultat jednostavnoga logičkog izraza `stranicaA*stranicaB >= 6`, odnosno 6>=6 je istina.

Tablica 2.16. Primjeri logičkih operacija s jednostavnim logičkim izrazom

Povezivanjem više jednostavnih logičkih izraza pomoću logičkih operatora dobivaju se složeni logički izrazi. Logički operatori su negacija (unarni operator), logičko I (binarni operator) te logičko ILI (binarni operator). Logički operatori djeluju na operande koji imaju logičku vrijednost istina ili neistina te stvaraju novu logičku vrijednost. Način transformacije jednih logičkih vrijednosti u druge preko logičkih operatora prikazuje tablica istinitosti logičkih operatora.

Logički operator negacije (!) je unarni operator koji negira logičku vrijednost. Negacija istine je neistina. Vrijedi i obrnuto – negacija neistine je istina. Slijedi tablica istinitosti logičkoga operatora negacije.

Logička vrijednost (L)	Negacija logičke vrijednosti (!L)
0	1
1	0

Tablica 2.17. Tablica istinitosti za negaciju

Logički operator I (&&) je binarni operator koji samo za kombinaciju ISTINA, ISTINA rezultira ISTINOM. Tablica istinitosti logičkoga operatora I glasi:

Logička vrijednost (L1)	Logička vrijednost (L2)	Logički operator I (L1 && L2)
0	0	0
0	1	0
1	0	0
1	1	1

Tablica 2.18. Tablica istinitosti za logički I

Logički operator ILI (||) je binarni operator koji samo u kombinaciji NEISTINA, NESITINA rezultira NEISTINOM. Tablica istinitosti logičkoga operatora ILI glasi:

Logička vrijednost (L1)	Logička vrijednost (L2)	Logički operator I (L1 || L2)
0	0	0
0	1	1
1	0	1
1	1	1

Tablica 2.19. Tablica istinitosti za logički ILI

Neka je dan sljedeći složeni logički izraz iz tri jednostavna logička izraza i dva logička operatora:

```
(a < b) && (c = = d) || (e > f)
```

Neka su zadane sljedeće vrijednosti varijabli:

```
a = b = 1
c = d = 2
e = 4
f = 3
```

Nakon uvođenja vrijednosti u složeni logički izraz dobiva se:

```
(1 < 1) && (2 = = 2) || (4 > 3)
```

Prednost u izračunu logičkih vrijednosti (1 ili 0) imaju jednostavni logički izrazi. Nakon izračuna jednostavnih logičkih izraza, rezultat je:

```
0 && 1 || 1
```

Redoslijed prednosti u izračunavanju logičke vrijednosti kog logičkih operatora je: negacija, logičko I, logičko ILI. Dakle u primjeru se prvo treba izračunati `0 && 1` pa tek onda ostatak. Zbog toga, a prema tablici istinitosti, dobiva se:

```
0 || 1
```

Odnosno, na kraju se dobiva 1, tj. početni složeni logički izraz će vratiti logičku istinitost uz zadane vrijednosti varijabli.

Neka je sada zadan sličan složeni logički izraz (postavljene su zagrade oko logičkog ILI):

```
(a < b) && ((c = = d) || (e > f))
```

Rezultat navedenoga izraza uz iste zadane vrijednosti varijabli je:

```
(1 < 1) && ((2 = = 2) || (4 > 3))
          0 && (1 || 1)
              0 && 1
                 0
```

Dakle, sada izraz daje logičku neistinu. Neka je sada zadan izraz kako slijedi (negacija je uključena u izraz):

```
!(a < b) && ((c = = d) || (e > f))
```

Tada se kao rezultat, a uz iste zadane vrijednosti varijabli, dobiva:

```
!(1 < 1) && ((2 = = 2) || (4 > 3))
          !0 && (1 || 1)
              1 && 1
                 1
```

Rezultat je sada logička istina.

Prilikom izračuna nekoga složenog logičkog izraza, računalo ima sljedeće prioritete:

1. zagrade
2. jednostavni logički izrazi
3. logička negacija
4. logičko I (konjunkcija)
5. logičko ILI (disjunkcija).

3. LINIJSKA ALGORITAMSKA STRUKTURA U C

Poglavlje **Linijska algoritamska struktura u C** odnosni se na implementaciju linijske algoritamske strukture u programskome jeziku C. Prikazan je osnovni kostur računalnoga programa pisanoga u C-u te njegovi glavni dijelovi (deklaracija varijabli, unos podataka, izračun, ispis podataka). Prikazane su naredbe za prikaz i unos podataka te neke matematičke funkcije i funkcije koje se koriste u radu s nizom znakova. Na kraju poglavlja nalazi se niz praktičnih zadataka.

Po završetku ovoga poglavlja čitatelj će moći:

- izraditi računalni program s linijskom algoritamskom strukturom prema zadanome problemu
- primijeniti naredbu `printf` za ispis na monitor
- primijeniti naredbu `scanf` za unos podataka
- objasniti svrhu pojedine linije koda
- prepoznati u kojoj se liniji koda odvija određena aktivnost (npr. deklaracija varijabli, unos podataka, izračuni, ispis podataka i sl.)
- objasniti važnosti pravovremenog uvođenja i dodjeljivanja vrijednosti varijablama
- izraditi pseudo kod iz danoga računalnog programa
- izraditi računalni program iz danoga pseudo koda
- formulirati primjer zadatka za linijsku algoritamsku strukturu
- identificirati greške u slijedu naredbi u danome računalnom kodu
- primijeniti neke matematičke funkcije C-a (npr. `sqrt`, `pow`, `sin`, `cos`, `tan`, `log`, `log10`, `exp`)
- primijeniti neke *string* funkcije C-a (npr. `puts`, `gets`, `strlen`, `strcpy`, `strcat`)
- izračunati rezultat računalnoga programa uz dane početne vrijednosti
- objasniti posljedice deklaracije varijabli na različitim mjestima u računalnome programu
- objasniti posljedice različitoga slijeda naredbi u računalnome programu.

3.1. Realizacija linijske algoritamske strukture u C

Osnovna je karakteristika linijske algoritamske strukture (slijed, sekvencija) da se u njoj algoritamski koraci izvode jedan za drugim – slijedno. Izvršni računalni program pisan u C-u (računalni program koji računalo može izravno izvesti njegovim pozivom) mora posjedovati glavnu funkciju – `main`. Ova funkcija govori računalu gdje se nalazi početak računalnoga programa, odnosno koja je prva naredba koja treba biti izvedena prilikom pokretanja računalnoga programa.

Neka se želi izraditi izvršni računalni program koji kada se pokrene, na zaslon monitora ispisuje poruku `"Hello World!"` – ispis ove poruke se često uzima za primjer pri izradi prvoga računalnog programa u nekome programskom jeziku. Takav računalni program mora imati glavnu funkciju `main` (jer je izvršni) i mora imati naredbu za ispis poruke `"Hello World"` koja će se izvesti po pokretanju računalnoga programa. Slijedi izvorni kod računalnoga programa.

```
1.  #include <stdio.h>
2.  main(){
3.     printf ("Hello World!");}
```

Program 3.1.

U liniji 2. je vidljiva je glavna funkcija `main`. Vitičaste zagrade predstavljaju granice bloka naredbi koje čine tijelo funkcije `main`. Računalo će izvesti upravo naredbe koje se nalaze u tijelu funkcije `main`. U liniji 3. je vidljiva naredba (funkcija) `printf` koja izvodi ispis teksta na zaslon monitora. Tekst je dan unutar navodnika. Također je vidljivo da naredba završava sa znakom ; što je obavezno kod C-a. Još jedan važan dio prikazanoga programa je linija koda 1. u kojoj se naređuje računalu da u izvođenju ovoga računalnog programa uključi vanjsku datoteku `stdio.h`. Ove datoteke se nazivaju zaglavne datoteke (.h označava header). Unutar njih se nalaze informacije potrebne za izvođenje nekih naredbi koje se koriste u računalnome programu. U prikazanome primjeru zaglavna datoteka `stdio.h` je potreba radi naredbe `printf`.

Ako se ovaj program pokrene unutar Dev C++ alata, uočit će se da će računalo otvoriti prozor konzole (prozor preko koga je omogućena interakcija s korisnikom programa putem teksta) i da će se nakon toga konzola odmah zatvoriti. U biti, korisnik neće vidjeti rezultat izvođenja ovoga programa. Razlog tome je u činjenici što računalo konzolu otvara odmah po pokretanju programa, dakle odmah po otkrivanju mjesta prve naredbe u programu (linija 3.). Potom računalo izvodi naredbu (dakle ispis poruke) i nakon toga odmah kreće u izvođenje sljedeće naredbe (slijed). Kako sljedeća naredba ne postoji (iza naredbe `printf` nema više naredbi), računalo završava s izvođenjem programa što uzrokuje zatvaranje konzole. Zbog brzine izvođenja naredbi, korisnik ne može vidjeti rezultat programa. Rješenje ovoga problema je da natjeramo računalo da nakon što izvede naredbu `printf` (ispis) pričeka s daljnjim izvođenjem programa. Ovo čekanje se može napraviti tako da računalo čeka nekakav odgovor od korisnika, odnosno pritisak bilo koje tipke s tipkovnice. To znači da je potrebna naredba kojom računalo čeka odgovor korisnika. U tu svrhu se može iskoristiti

naredba `getch()`. No, da bi se ova naredba koristila potrebna je zaglavna datoteka u kojoj se nalaze odgovarajuće informacije potrebne za izvođenje naredbe. Ta zaglavna datoteka je također `stdio.h`. Konačni program bi onda glasio:

```
1.  #include <stdio.h>
2.  main(){
3.    printf ("Hello World!");
4.    getch();}
```

Program 3.2.

Izvođenjem ovoga programa otvorit će se konzola u kojoj će biti ispisana poruka. Konzola se neće zatvoriti dok god korisnik ne pritisne neku tipku tipkovnice. Tek se nakon toga konzola zatvara jer iza linije 4. nema više naredbi i izvođenje programa završava. Blok naredbi koji čine tijelo funkcije `main` se sada sastoji iz dvije naredbe.

Zbog linijske algoritamske strukture prvo je bila izvedena naredba `printf` (ispis na zaslon monitora), pa tek nakon toga `getch` (čekanje na pritisak tipke). Ovo je ispravan slijed naredbi budući da se htjelo izraditi program koji prvo ispisuje poruku te nakon toga čeka odgovor korisnika. Sljedeći programski kod prikazuje zamjenu ovih dviju linija koda:

```
1.  #include <stdio.h>
2.  main(){
3.    getch();
4.    printf ("Hello World!");}
```

Program 3.3.

Sada će prilikom izvođenja programa prvo biti izvedena naredba `getch` (čekanje na korisnički odgovor, ali bez ispisa poruke) pa tek nakon toga `printf` (ispis poruke). Ali sada, nakon zadnje naredbe nema čekanja i program odmah završava. Dakle, pokretanjem programa pojavit će se prazna konzola i čekanje na korisnički odgovor. Nakon pritiska na neku tipku, računalo izvršava ispis pozdrava ali odmah zatvara konzolu. Ovo pokazuje koliko je bitan redoslijed napisanih naredbi, odnosno koliko je bitno razumijevanje linijskoga (slijednoga, sekvencijalnoga) izvođenja algoritamskih koraka.

Do sada je prikazan primjer računalnoga programa kojim se ispisuje jednostavna tekstualna poruka. Neka se sada želi izraditi program koji će ispisati sljedeći tekst `"13 * 17 = 221"`. Ovaj tekst bi se mogao ispisati jednostavno kao i poruka `"Hello World!"` ako se unaprijed izračuna rezultat računske operacije (221). No ono što će program napraviti je sljedeće, on će izračunati računsku operaciju `13 * 17`, te će dobiveni rezultat ispisati nakon ispisa `"13 * 17 = "`. I za ovaj će se ispis koristiti naredba `printf`. Slijedi kod računalnoga programa:

```
1.  #include <stdio.h>
2.  main(){
3.    printf ("13 * 17 = %d", 13 * 17);
4.    getch();}
```

Program 3.4.

Linija 3. je ključna za ispis. Prije je rečeno da će ono što se nalazi unutar navodnika naredbe `printf` (u prikazanome slučaju `"13 + 17 = %d"`) biti ispisano na zaslonu monitora. Ovdje treba obratiti pažnju na poseban dio teksta koji se ispisuje – `%d` – koji specificira format ispisa, odnosno konverzije (u sljedećemu će poglavlju biti više riječi o formatiranom ispisu) – znak konverzije `%d` se koristi pri ispisu cjelobrojnih tipova podataka (`int`). Naime, računalo neće na zaslon ispisati `%d`, već će na njegovo mjesto ispisati vrijednost koja slijedi iza teksta koji treba biti ispisan, a koji je odvojen sa zarezom (,). U prikazanom slučaju to je vrijednost do koje će računalo doći tako da izračuna rezultat računske operacije `13 * 17`. Prema tome, ono što će računalo napraviti u liniji 3. je da će prvo izračunati vrijednost računske operacije `13 * 17` te će zatim ispisati tekst koji se nalazi unutar navodnika, ali će umjesto znaka konverzije `%d` staviti rezultat izračuna, odnosno 221. Na kraju se dobiva ispis `"13 * 17 = 221"`.

Želi li se ispisati `"13 * 17 = 221, 67 + 32 = 99"`, ali uz izračun računskih operacija, tada će se naredba `printf` koristiti kako slijedi:

```
1.  #include <stdio.h>
2.  main(){
3.    printf ("13 * 17 = %d, 67 + 32 = %d", 13 * 17, 67 + 32);
4.  getch();}
```

Program 3.5.

U liniji 3. je vidljivo da u tekstu za ispis postoje dva znaka konverzije `%d`. Na mjesto prvoga znaka konverzije računalo stavlja prvu vrijednost koja se nalazi iza teksta za ispis (u našemu slučaju vrijednost `13 * 17`), a na mjesto drugoga znaka konverzije stavlja se druga vrijednost (u našem slučaju `67 + 32`). Na ovaj se način redaju vrijednosti koje će biti ispisane na odgovarajuća mjesta znakova konverzije.

Ono što se ispisuje na mjesto znaka konverzije mogu biti jednostavni, ali i složeni računski izračuni. Pri izboru znaka konverzije svakako treba voditi računa o tipu podatka koji je potrebno ispisati (`%d` je za cjelobrojne tipove podataka, `%f` za realne, %c za char, %s za niz znakova itd).

Neka se želi ispisati vrijednost drugoga korijena iz broja 2 tako da je na zaslonu prikaz `"Drugi korijen od 2 je 1.414"`. Sada je potrebno koristiti matematičke funkcije `sqrt` ili `pow`. Za ove funkcije je potrebno uključiti zaglavnu datoteku `math.h`. Program koji daje navedeni ispis glasi:

```
1.  #include <stdio.h>
2.  #include <math.h>
3.  main(){
```

```
4.    printf ("Drugi korijen od 2 je %f", sqrt(2));
5.  getch();}
```

Program 3.6.

U liniji 2. je uvedena potrebna zaglavna datoteka `math.h` budući da se koristi matematička funkcija `sqrt`. Ova funkcija se poziva tako da joj se kao argument šalje broj iz koga se vadi drugi korijen (broj unutar zagrada koje slijede nakon poziva funkcije `sqrt`) – linija 4. Funkcija `sqrt` vraća rezultat koji je realni tip podatka – zbog toga se koristi znak konverzije `%f`. Funkcija `pow` je općenitija od funkcije `sqrt` jer ona izvodi potenciranje zadanoga broja na zadani eksponent. Ona vraća realni tip podatka, a kao argument prima bazu i eksponent. Drugi korijen iz 2 je isto što i $2^{0.5}$. Program koji ispisuje drugi korijen broja 2 pomoću funkcije `pow` glasi:

```
1.  #include <stdio.h>
2.  #include <math.h>
3.  main(){
4.    printf ("Drugi korijen od 2 je %f", pow(2, 0.5));
5.  getch();}
```

Program 3.7.

Neke od matematičkih funkcija su prikazane u sljedećoj tablici.

Opis funkcije	Sintaksa
Drugi korijen nekoga broja	`sqrt(broj)`
Potenciranje broja	`pow(baza, eksponent)`
Sinus broja	`sin(kut u radijanima)`
Kosinus broja	`cos(kut u radijanima)`
Tanges broja	`tan(kut u radijanima)`
Logaritam broja (baza 10)	`log10(broj)`

Tablica 3.1. Neke matematičke funkcije u programskome jeziku C

Svi dosada prikazani programi nisu spremali vrijednost u memoriju računala (nije niti postojala potreba za time). Želi li se neka vrijednost sačuvati u memoriji računala, tada je potrebno koristiti varijable. Prije prve uporabe varijable, ona se mora deklarirati – definirati njezin tip podatka i identifikator. Neka se želi izraditi program kojim se računa i ispisuje opseg kvadrata čija je duljina stranice 5. Ovaj program će koristiti dvije varijable – u jednu će se spremiti duljina stranice kvadrata, a u drugu izračunati opseg. Slijedi programski kod:

```
1.  #include<stdio.h>
2.  main(){ //glavna funkcija
3.    int stranicaA,opseg; /*deklaracija dvije varijable
4.                           stranicaA i opseg tipom podataka int */
5.    stranicaA = 5;
6.    opseg = 4 * stranicaA;
7.    printf("Rjesenje zadatka je u sljedecem redu.\n");
8.    printf("Opseg kvarata stranice %d je %d.",stranicaA,opseg);
9.    getch();}
```

Program 3.8.

U liniji 2. se nalazi tekst koji je napisan nakon znakova `//`. Cijeli takav tekst računalo ignorira tijekom izvođenja programa, a njegova svrha je komentiranje dijelova programa u samom izvornome kodu. Uporabom komentara olakšava se razumijevanje programskoga koda. Znakom `//` moguće je napisati komentar u samo jednome retku. Ako postoji potreba za komentarom preko više redaka, tada se znakom `/*` otvara početak komentara, a znakom `*/` se komentar zatvara (linija 3. i 4.).

Prva naredba koja se izvodi u tijelu glavne funkcije `main` (linija 3.) je deklaracija dviju varijabli – `stranicaA` i `opseg`. Vidljivo je da će one u sebi čuvati vrijednosti tipa podatka `int` (računalo će rezervirati odgovarajući broj memorijskih lokacija), a preko identifikatora je razumljiva i njihova uloga u programu. U liniji 5. se varijabli `stranicaA` dodjeljuje vrijednost 5 (operatorom pridruživanja =), dok se u liniji 6. varijabli `opseg` pridružuje vrijednost koja je dobivena izračunom računske `operacije 4 * stranicaA` (što je formula za izračun opsega kvadrata). U liniji 7. se ispisuje poruka u kojoj se nalazi i specijalni znak `\n` koji označava prijelaz u novi redak (postoje još neki specijalni znakovi, kao npr. `\t` koji je oznaka za ispis taba itd.). U liniji 8. izvodi se ispis vrijednosti varijable `stranicaA` i `opseg` tako da ih se stavi na odgovarajući znak konverzije `%d`.

Do sada su prikazani programi koji rade određenu manipulaciju s brojčanim tipovima podataka. Slijede programi koji manipuliraju sa znakovnim tipom podatka (`char`). U prethodnim poglavljima je rečeno da se vrijednost znakovnoga tipa podatka (`char`) u memoriji računala čuva kao njegov ASCII kod.

Dodjeljivanje konkretne vrijednosti varijabli tipa podatka `char` izvodi se preko operatora pridruživanja i jednostrukih navodnika (`'`). Npr. program koji ispisuje ASCII kod znaka & može izgledati kako slijedi:

```
1.  #include<stdio.h>
2.  main(){
3.    char znak;
4.    znak = '&';
5.    printf("ASCII kod znaka %c je %d.", znak, znak);
6.    getch();}
```

Program 3.9.

U liniji 4. vidljivo je dodjeljivanje vrijednosti & varijabli `znak` koja je tipa podatka `char`. U liniji 5. izvodi se prvo ispis vrijednosti varijable `znak` interpretirane kao tip podatka `char` (to čini konverzijski znak `%c`) te nakon toga vrijednost iste varijable `znak` ali sada interpretirane kao tip podatka `int` (to čini konverzijski znak `%d`). Zbog toga se kao rezultat dobiva ASCII kod unesenoga znaka (za znak & to je broj 38).

Moguće je izraditi program koji radi obrnuti postupak, iz zadanoga ASCII koda ispisuje pripadajući znak. Budući da je za ASCII kod potrebno 8 bitova potrebno je deklarirati varijablu `ASCIIkod` koja će u memoriji računala upravo zauzeti potrebni broj bitova (moguće je deklarirati i varijablu s tipom podatka `int`, ali tada je zauzeta

prevelika količina memorije koja se sva nikada neće iskoristiti). Tip podatka koji zauzima 8 bitova je `char` pa će se varijabla `ASCIIkod` deklarirati upravo tim tipom podatka. Slijedi programski kod:

```
1.  #include<stdio.h>
2.  main(){
3.     char ASCIIkod;
4.     ASCIIkod = 38;
5.     printf("Za ASCII kod %d znak je %c.", ASCIIkod, ASCIIkod);
6.     getch();}
```

Program 3.10.

U liniji 4. varijabli `ASCIIkod` pridružena je vrijednost koja se pri ispisu u liniji 5. prvo interpretira kao broj (konverzijski znak `%d`), a nakon toga kao znak (konverzijski znak `%c`).

3.2. Unos, obrada i prikaz podataka

Jedan je od prethodno prikazanih programa računao opseg kvadrata čija je duljina stranice 5. Želi li se napraviti program koji računa opseg bilo koga kvadrata, računalo od korisnika treba doznati kolika je njegova duljina stranice, te tek nakon izvede izračun i ispis rezultata. Ovdje još nedostaje i provjera unesenih podataka (npr. duljina stranice može biti samo pozitivan cijeli broj), ali o načinu provjere podataka bit će riječi u narednim poglavljima. Slijedi izmijenjeni programski kod:

```
 1.  #include<stdio.h>
 2.  main(){ //glavna funkcija
 3.    int stranicaA,opseg; /* deklaracija dvije varijable
 4.    stranicaA i opseg tipom podataka int */
 5.    scanf("%d", &stranicaA);
 6.    opseg = 4 * stranicaA;
 7.    printf("Rjesenje zadatka je u sljedecem redu.\n");
 8.    printf("Opseg kvarata stranice %d je %d.",stranicaA,opseg);
 9.    getch();}
```

Program 3.11.

Promjena se dogodila u liniji 5. programskoga koda gdje sada stoji naredba `scanf` koja se koristi kako bi se omogućilo korisniku unos podatka o duljini stranice. Ova funkcija kao prvi argument unutar navodnika prima format unosa podatka (u prikazanom slučaju `"%d"` označava da se očekuje unos jednoga broja tipa podatka `int`) te adresu varijable u koju će računalo sačuvati podatak koji je korisnik unesao preko tipkovnice (u prikazanom slučaju se radi o adresi varijable `stranicaA` do koje dolazimo adresnim operatorom `&`).

Nakon pokretanja programa otvara se prazna konzola u koju korisnik treba unijeti duljinu stranice. Budući da je konzola prazna, ako korisnik nije unaprijed upoznat s time što se od njega očekuje, teško će moći dati ispravan podatak programu. Stoga je korisniku potrebno postaviti pitanje i tek onda od njega očekivati odgovor na to pitanje. U promatranome slučaju to pitanje može glasiti "`Unesite duljinu`

stranice kvadrata: ". Dakle, prije naredbe za unos podataka (naredba scanf), potrebno je ispisati navedeno pitanje (naredba printf). Konačno izmijenjeni program glasi:

```
 1. #include<stdio.h>
 2. main(){ //glavna funkcija
 3.   int stranicaA,opseg; /* deklaracija dvije varijable
 4.   stranicaA i opseg tipom podataka int */
 5.   printf("Unesite duljinu stranice kvadrata: ");
 6.   scanf("%d", &stranicaA);
 7.   opseg = 4 * stranicaA;
 8.   printf("Rjesenje zadatka je u sljedecem redu.\n");
 9.   printf("Opseg kvarata stranice %d je %d.",stranicaA,opseg);
10.   getch();}
```

Program 3.12.

Traženi ispis pitanja se nalazi u liniji koda 5. Prikazani program od korisnika zahtijeva unos duljine stranice jednoga kvadrata. Želi li se napraviti program kojim se od korisnika traži unos stranica od npr. 3 kvadrata, tada se u svrhu pamćenja stranica može iskoristiti polje dimenzije 3 u kome će se sačuvati duljine stranica te polje u koje će se sačuvati izračunata vrijednost opsega. Pored navedenoga, sada će se korisniku omogućiti unos realnih vrijednosti za duljine. Takav program je prikazan u nastavku.

```
 1. #include<stdio.h>
 2. main(){
 3.   float stranicaA[3],opseg [3];
 4.   printf("Unesite duljinu stranice 1. kvadrata: ");
 5.   scanf("%f", &stranicaA[0]);
 6.   printf("Unesite duljinu stranice 2. kvadrata: ");
 7.   scanf("%f", &stranicaA[1]);
 8.   printf("Unesite duljinu stranice 3. kvadrata: ");
 9.   scanf("%f", &stranicaA[2]);
10.   opseg [0] = 4 * stranicaA[0];
11.   opseg [1] = 4 * stranicaA[1];
12.   opseg [2] = 4 * stranicaA[2];
13.   printf("\nOpseg 1. kvadrata stranice %f je %f.",
              stranicaA[0], opseg[0]);
14.   printf("\nOpseg 2. kvadrata stranice %.3f je %.3f.",
              stranicaA[1], opseg[1]);
15.   printf("\nOpseg 3. kvadrata stranice %.2f je %.2f.",
              stranicaA[2], opseg[2]);
16.   getch();}
```

Program 3.13.

U liniji 3. deklarirana su dva polja (stranicaA i opseg) dimenzije 3 i tipom podatka float. U linijama od 4. do 9. nalaze se naredbe za ispis pitanja korisniku i prihvaćanje njegova odgovora (obratiti pažnju na %f u scanf funkciji) koji se sprema u odgovarajući element polja. U linijama od 10. do 12. izvodi se izračun opsega koji se sprema u odgovarajući element polja te se u zadnjim linijama izvodi ispis rezultata. Pri ispisu se može uočiti konverzijski znak %f radi ispisa realnog tipa

podatka (`float`). Ako se želi ispisati samo određeni broj znamenaka iza decimalne točke, tada se to može navesti brojem ispred znaka `f` (u primjeru `%.3f` označava ispis tri znamenke iza decimalne točke).

Neka se sada želi napraviti program koji će korisnika pitati unos nekoga znaka te će mu ispisati vrijednost ASCII koda za taj znak. Slijedi programski kod:

```
1.  #include<stdio.h>
2.  main(){
3.    char znak;
4.    printf("Unesite neki znak: ");
5.    scanf("%c", &znak);
6.    printf("ASCII kod znaka %c je %d.", znak, znak);
7.    getch();}
```

Program 3.14.

U liniji 3. deklarirana je varijabla imena `znak` koja će u sebi čuvati `char` tip podatka. Potom se korisniku ispisuje pitanje (linija 4.) te se od njega očekuje unos znakovnoga tipa podatka (linija 5. argument `"%c"` naredbe `scanf`) koji se smješta u varijablu `znak`. U liniji 6. se izvodi prvo ispis vrijednosti varijable `znak` interpretirane kao tip podatka `char` (to čini konverzijski znak `%c`), te nakon toga vrijednost iste varijable `znak`, ali sada interpretirane kao tip podatka `int` (to čini konverzijski znak `%d`). Zbog toga se kao rezultat dobiva ASCII kod unesenoga znaka (npr. za znak A to je broj 65).

Obrnuta verzija prethodnoga programa – iz unesenoga ASCII koda ispisuje se pripadajući znak prikazuje sljedeći programski kod. Unos korisnika se sada neće interpretirati kao `char`, već kao `int`.

```
1.  #include<stdio.h>
2.  main(){
3.    char ASCIIkod;
4.    printf("Unesite neki ASCII kod: ");
5.    scanf("%d", &ASCIIkod);
6.    printf("ASCII kod %d predstavlja znak %c.",ASCIIkod,ASCIIkod);
7.    getch();}
```

Program 3.15.

U liniji 4. izmijenjeno je postavljeno pitanje. Naredba `scanf` u liniji 5. sada je primila argument `"%d"` kojom unos interpretira kao `int` tip podatka koji sprema u varijablu `ASCIIkod`. Pri ispisu (linija 6.) prvo se ispisuje ASCII kod (vrijednost varijable `ASCIIkod` se konverzijskim znakom `%d` interpretira kao `int`), a nakon toga pripadajući znak (vrijednost varijable `ASCIIkod` se konverzijskim znakom `%c` interpretira kao `char`).

Do sada su prikazani programi koji manipuliraju samo jednim znakom. Želi li se izraditi program u kome će se od korisnika zatražiti unos imena, varijabla u koju će se spremiti taj podatak treba biti deklarirana kao niz znakova određene duljine. Također, računalo treba znati da korisnički unos treba interpretirati kao niz znakova. Slijedi

programski kod programa koji nakon unosa korisničkoga imena izvodi njegov ispis i ispis pozdravne poruke:

```
1.  #include<stdio.h>
2.  main(){
3.    char ime[20];
4.    printf("Unesite vase ime: ");
5.    scanf("%s", ime);
6.    printf("Vase ime je: %s. Zelim vam ugodan dan.", ime);
7.    getch();}
```

Program 3.16.

U liniji 3. deklarirana je varijabla `ime` kao niz od 20 znakova – bit će rezervirano memorijskoga prostora za 20 znakova ali će se efektivno moći koristiti 19 budući da je jedan znak (zadnji) uvijek rezerviran za poseban znak (`null` - `\0`) koji predstavlja kraj niza znakova. Nakon postavljenoga pitanja korisniku (linija 4.) slijedi čekanje na korisnički unos koji se interpretira kao niz znakova (argument `"%s"` u funkciji `scanf`) te se niz sprema u varijablu `ime`. Pažnju treba obratiti na činjenicu da ispred varijable `ime` nema adresnog operatora (`&`) budući da je varijabla `ime` u biti polje, a adresu prvoga elementa polja čuva sam njegov identifikator (dakle `ime`). U liniji 6. izvodi se ispis niza znakova koristeći konverzijski znak `%s`.

U prethodnome programu se od korisnika zahtijevao unos imena. Neka se sada zahtijeva unos imena i prezimena. Dakle i ovdje se radi o nizu znakova, ali u tome nizu znakova postoji i znak razmak (između imena i prezimena). Unos nizova znakova gdje se nalazi razmak nije moguć putem naredbe `scanf` budući da ona ovaj znak tretira kao kraj niza znakova. Zbog toga će se koristiti naredba `gets`. Prema tome, izmijenjeni program glasi:

```
1.  #include<stdio.h>
2.  main(){
3.    char ime[40];
4.    printf("Unesite vase ime i prezime: ");
5.    gets(ime);
6.    printf("Vase ime je: %s. Zelim vam ugodan dan.", ime);
7.    getch();}
```

Program 3.17.

U liniji koda 3. povećan je znakovni niz sa 20 na 40 znakova jer se u njemu osim imena sada treba čuvati i prezime. Nakon postavljenoga pitanja (linija 4.) slijedi naredba `gets` koja u varijablu `ime` sprema niz znakova (linija 5.). Nakon toga slijedi ispis.

Dodjeljivanje konkretnih vrijednosti nizu znakova nije moguće jednostavnom primjenom operatora pridruživanja. Dakle, nije moguće izvesti pridruživanje u obliku `ime = 'Marko Markovic'`. Za ovakva pridruživanja je potrebno koristiti funkciju `strcpy` koja izvodi kopiranje jednoga niza znakova u drugi niz znakova. Programski kod s ovom funkcijom bi mogao glasiti kako slijedi:

```
1.  #include<stdio.h>
2.  main(){
3.    char ime[40];
4.    strcpy (ime, "Marko Markovic");
5.    printf("Vase ime je: %s. Zelim vam ugodan dan.", ime);
6.    getch();}
```

Program 3.18.

U liniji 4. nalazi se poziv funkcije `strcpy` koja ima dva argumenta. Prvi argument je naziv varijable (niza znakova) u koji će se izvesti kopiranje stringa koji je dan drugim argumentom. Zbog toga se u varijablu `ime` kopira niz znakova "`Marko Markovic`" – obratiti pažnju da se ovdje koriste dvostruki navodnici (").

Postoji niz funkcija koje se mogu koristiti u radu s nizom znakova. Ovdje će se još prikazati `strlen` i `strcat`.

Funkcija `strlen` vraća duljinu nekoga niza znakova, dok funkcija `strcat` dodaje postojećemu nizu znakova novi niz znakova. Neka se želi napraviti program u kome se od korisnika traži prvo unos imena pa nakon toga i prezimena (vrijednosti se čuvaju u dvije varijable). Potom se ispisuje koliko korisnikovo ime, odnosno prezime ima znakova. Nakon toga se `ime` i `prezime` spajaju u novi niz znakova koji će sadržavati korisnikovo ime i prezime, ali i razmak među njima. Na kraju će se ispisati zadnje dobiveni niz znakova. Slijedi programski kod:

```
 1.  #include<stdio.h>
 2.  main(){
 3.    char ime[20];
 4.    char prezime[20];
 5.    int  ime_duljina, prezime_duljina;
 6.    char ime_i_prezime[40];
 7.    printf("Unesite vase ime: ");
 8.    scanf("%s", ime);
 9.    printf("Unesite vase prezime: ");
10.    scanf("%s", prezime);
11.    ime_duljina = strlen(ime);
12.    prezime_duljina = strlen(prezime);
13.    printf("Vase ime ima %d znakova. Vase prezime ima %d
                znakova", ime_duljina, prezime_duljina);
14.    ime_i_prezime[0] = '\0';
15.    strcat (ime_i_prezime, ime);
16.    strcat (ime_i_prezime, " ");
17.    strcat (ime_i_prezime, prezime);
18.    printf ("\nVase ime i prezime je: %s", ime_i_prezime);
19.    getch();}
```

Program 3.19.

Deklarirano je ukupno pet varijabli (linije 3., 4., 5., i 6.). Varijable `ime` i `prezime` će čuvati unesene korisničke podatke. Varijable `ime_duljina` i `prezime_duljina` će čuvati duljinu odgovarajućih nizova znakova, dok će varijabla `ime_i_prezime`

čuvati spojene nizove znakova koji se nalaze u varijablama `ime` i `prezime`. Nakon unosa imena i prezimena (linija 8. i 10.) slijedi poziv funkcije `strlen` (linija 11. i 12.) kojoj kao argument dostavljamo niz znakova, a ona nam vraća njegovu duljinu koja se zatim sprema u odgovarajuće varijable koje se ispisuju u liniji 13. Nakon ispisa slijedi spajanje nizova znakova `ime` i `prezime` u novi niz znakova `ime_i_prezime`. U tu se svrhu koristi funkcija `strcat`. No prije bilo kakvog spajanja, potrebno je varijablu `ime_i_prezime` inicijalizirati tako da ona u sebi čuva prazan niz znakova. To se radi tako da se kao element na indeksu 0 niza postavi `null` vrijednost `(\0)` što je i učinjeno u liniji 14. U protivnome bi se spajanje nizova znakova radilo sa *smećem* koje postoji u varijabli `ime_i_prezime` nakon njezine deklaracije. U liniji 15. spajamo niz znakova `ime_i_prezime` sa nizom znakova `ime`. Nakon ove naredbe u `ime_i_prezime` sada imamo niz znakova `ime`. U liniji 16. postojećemu nizu znakova u `ime_i_prezime` dodajemo razmak i nakon toga u liniji 17. i niz znakova `prezime`. Na kraju se ispisuje dobiveni niz znakova koji se sada čuva u varijabli `ime_i_prezime` (linija 18.).

Neka se sada želi izraditi program kojim će se unijeti `ime` i `prezime` studenta, broj ostvarenih bodova i broj mogućih bodova na nekom ispitu. Nakon unosa potrebno je ispisati postotak riješenosti ispita. Za ovaj će se program iskoristiti zapis koji će u sebi čuvati sve navedene podatke. Slijedi program:

```
 1.  typedef struct {
 2.    char ime[20+1];
 3.    char prezime[30+1];
 4.    float brojOstvarenihBodova;
 5.    float brojMogucihBodova;} InformacijaOStudentu;
 6.  main(){
 7.    InformacijaOStudentu student;
 8.    float postotakRijesenosti;
 9.    printf("Unesite ime studenta: ");
10.    scanf("%s", student.ime);
11.    printf("Unesite prezime studenta: ");
12.    scanf("%s", student.prezime);
13.    printf("Unesite ostvareni broj bodova: ");
14.    scanf("%f", &student.brojOstvarenihBodova);
15.    printf("Unesite moguci broj bodova: ");
16.    scanf("%f", &student.brojMogucihBodova);
17.    postotakRijesenosti=student.brojOstvarenihBodova/
                           student.brojMogucihBodova*100;
18.    printf ("Student %s %s je rijesio %.2f%% ispita.",
                 student.ime, student.prezime, postotakRijesenosti);
19.    getch();}
```

Program 3.20.

U linijama 7. i 8. deklarirane su potrebne varijable – zapis za čuvanje podataka o imenu, prezimenu, ostvarenim bodovima i mogućim bodovima te varijabla u koju će se izračunati postotak. Nakon toga slijedi postavljanje pitanja korisniku i prihvaćanje njegova odgovora koji se smješta u odgovarajuću komponentu zapisa. Nakon unosa podataka slijedi izračun postotka riješenosti ispita (linija 17.) te ispis rezultata (linija

18.). Pri ispisu rezultata valja pažnju usmjeriti na znakove `%%` koji slijede iza konverzijskoga znaka `%.2f`. Naime, želja je u ispisu imati i znak `%` budući da podatak predstavlja postotak. Kako je znak `%` specijalan znak (koristi se kod znaka konverzije), tada je za njegov ispis potrebno staviti `%%` - dva znaka. Zbog toga računalo zna da treba ispisati znak `%`, a ne da se radi o nekom konverzijskome znaku.

Sljedeći primjer programa pokazuje primjenu polja zapisa. Neka se želi izraditi program u kome se od korisnika traži unos duljine stranice tri pravokutnika te se izračunava njihova površina. Potom slijedi ispis unesenih dimenzija pravokutnika te pripadajuće površine. Za čuvanje informacija o pravokutnicima koristit će se zapis od dvije komponente (za svaku stranicu pravokutnika), a za čuvanje informacija o tri pravokutnika će se koristiti polje od tri elementa. Slijedi program:

```
 1. #include<stdio.h>
 2. typedef struct {
 3.   int stranicaA;
 4.   int stranicaB;} InformacijaOPravokutniku;
 5. main(){
 6.   InformacijaOPravokutniku pravokutnik[3];
 7.   int povrsina[3];
 8.   printf("Unesite duljinu stranice a 1. pravokutnika: ");
 9.   scanf("%d", &pravokutnik[0].stranicaA);
10.   printf("Unesite duljinu stranice b 1. pravokutnika: ");
11.   scanf("%d", &pravokutnik[0].stranicaB);
12.   printf("Unesite duljinu stranice a 2. pravokutnika: ");
13.   scanf("%d", &pravokutnik[1].stranicaA);
14.   printf("Unesite duljinu stranice b 2. pravokutnika: ");
15.   scanf("%d", &pravokutnik[1].stranicaB);
16.   printf("Unesite duljinu stranice a 3. pravokutnika: ");
17.   scanf("%d", &pravokutnik[2].stranicaA);
18.   printf("Unesite duljinu stranice b 3. pravokutnika: ");
19.   scanf("%d", &pravokutnik[2].stranicaB);
20.   povrsina[0]= pravokutnik[0].stranicaA*
                   pravokutnik[0].stranicaB;
21.   povrsina[1]= pravokutnik[1].stranicaA*
                   pravokutnik[1].stranicaB;
22.   povrsina[2]= pravokutnik[2].stranicaA*
                   pravokutnik[2].stranicaB;
23.   printf("\nPovrsina 1.pravokutnika dimenzije %dx%d iznosi%d.",
               pravokutnik[0].stranicaA,pravokutnik[0].stranicaB,
               povrsina[0]);
24.   printf("\nPovrsina 2. pravokutnika dimenzije %dx%d iznosi
              %d.", pravokutnik[1].stranicaA,
               pravokutnik[1].stranicaB, povrsina[1]);
25.   printf("\nPovrsina 3.pravokutnika dimenzije %dx%d iznosi
              %d.",pravokutnik[2].stranicaA,
               pravokutnik[2].stranicaB, povrsina[2]);
26.   getch();}
```

Program 3.21.

U linijama od 2. do 4. je definiran zapis naziva `InformacijaOPravokutniku`. U linijama 6. i 7. deklarirane su potrebne varijable – polje zapisa `pravokutnik`

dimenzije tri te polje `povrsina` koje će se koristiti u čuvanju izračunatih površina. Nakon toga, od 8. do 19. linije slijedi postavljanje pitanja korisniku i prihvaćanje odgovora te njegova spremanja u odgovarajuću varijablu. Potom se izvodi izračun površina čije vrijednosti se smještaju u odgovarajući indeks polja (linije od 20 do 22). Na kraju slijedi ispis podataka.

Već je rečeno da se pri radu s pokazivačima koriste dva operatora – adresni operator `(&)` i operator dereferenciranja ili indirekcije (*). Ova dva operatora se mogu iskoristiti i za ispis adrese memorijske lokacije koja je rezervirana za varijablu, odnosno za ispis vrijednosti koja se čuva na memorijskoj lokaciji ako je poznata adresa te memorijske lokacije. To prikazuje sljedeći primjer programa za izračun površine kruga:

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int radijus;
 4.    float *povrsina = &radijus;
 5.    printf("Unesite radijus kruga: \n");
 6.    scanf("%d", &radijus);
 7.    *povrsina = radijus * radijus * 3.14;
 8.    printf("Povrsina kruga radijusa %d iznosi %.2f.", radijus,
             *povrsina);
 9.    printf("\nAdresa memorijske lokacije varijable radijus je
             %d",&radijus);
10.    printf("\nAdresa memorijske lokacije varijable povrsina je
             %d",&povrsina);
11.    printf("\nAdresa memorijske lokacije na koju pokazuje
             pokazivac povrsina je %d",povrsina);
12.    getch();}
```

Program 3.22.

U liniji 3. se izvodi deklaracija varijable `radijus` koja će čuvati korisničku vrijednost čiji unos zahtijeva linija 6. U liniji je 4. deklarirana varijabla `povrsina` (pokazivač) i njoj je dodijeljena adresa varijable `radijus` (pokazivač mora imati dodijeljenu neku vrijednost jer se nakon deklaracije u njemu nalazi *SMEĆE*). U liniji 7. se izračunata površina kruga smješta na onu memorijsku lokaciju na koju pokazuje pokazivač `povrsina` (a to je upravo memorijska lokacija gdje se nalazi varijabla `radijus` jer se njezina adresa čuva u pokazivaču). Linije koda 8.,9.,10. i 11. ispisuju površinu kruga, adrese varijabli `radijus` i `povrsina`, te adresu koja se čuva u varijabli `povrsina` (a to je adresa varijable `radijus` budući se ona dodijelila pokazivaču u liniji koda 4.).

3.3. Praktični zadaci

Slijede primjeri riješenih i objašnjenih zadataka počevši od jednostavnijih prema složenijima. Korištene su sve vrste varijabli i operatora objašnjenih u prethodnim dijelovima poglavlja. Neki će se primjeri ponavljati iz prethodnih dijelova poglavlja, ali će biti ili riješeni na drugi način ili će se njima pojasniti i naglasiti neki drugi dio koda.

Prvi je primjer program koji pretvara unesenu temperaturu u stupnjevima Celzija u Fahrenheite. Formula za dobivanje Fahrenheita iz stupnjeva dana je sa: `F=9/5·C+32.` U nastavku je dan cijeli programski kod za unos stupnjeva Celzija i njihov ispis u Fahrenheitima na način da su obje potrebne varijable `C` i `F` deklarirane kao `float` (zbog formule je jasno da Fahrenheit može poprimiti realnu vrijednost jer je dijelom u obliku razlomka). Pri `scanf` i `printf` naredbi se stavlja oznaka `%f` zbog unosa, odnosno ispisa realnih vrijednosti. Oznaka `%.2f` znači da će realni broj biti ispisan tako da je zaokružen na dvije decimale.

Što bi se desilo da je varijabla `C` deklarirana kao `int`? Kada bi se za varijablu `C` unijela npr. vrijednost 24, varijabla `F` bi nakon izračuna imala vrijednost `75.00` umjesto `75.20`. To je zato što bi računalo izračunavalo sljedeći algebarski izraz 9/5·24+32. Kako je svaka vrijednost u ovom izrazu cijeli broj, odnosno tipa podatka `int`, to će i konačna izračunata vrijednost biti cijeli broj (u prikazanom slučaju će varijabla `F` imati vrijednost 75). Kako bi se dobila ispravna vrijednost, potrebno je "natjerati" računalo da prikazani algebarski izraz izračuna kao relani broj. To se može postići jednostavnim proglašenjem nekoga od brojeva iz formule realnim. Formula bi tada mogla glasiti: `F=9.0/5·C+32`. Slično se moglo postići eksplicitnim navođenjem pretvorbe tipa podatka `int` u `float` kako slijedi: `F=9/5·(float)C+32;`

```
1.  #include<stdio.h>
2.  #include<conio.h>
3.  main(){
4.     float C,F;
5.     printf("Unesi temperaturu u Celzijima\n");
6.     scanf("%f",&C);
7.     F=C*9/5+32;
8.     printf ("%.2f stupnjeva C  je %.2f F ",C,F);
9.     getch();}
```

Program 3.23.

Slijedi primjer računalnoga programa u kojem se unose `a,b,c`, tj. duljina, širina i visina kvadra te program ispisuje vrijednosti volumena (`V`) i oplošja (`OP`) po zadanim formulama:

`V=a·b·c; OP=2·(a·b+b·c+a·c);`

Računalo naredbe izvodi slijedno, jednu za drugom. Zbog toga je važno kojim se redoslijedom nižu naredbe u programskome kodu. Važnost nizanja naredbi je očigledna pri uvođenju varijabli i koncipiranja pojedinih izraza u kome sudjeluju uvedene varijable. Na primjer linija 11. ili 12. nije se mogla napisati nakon linije 13. jer se u liniji 13. u varijablama `V` i `OP` očekuju vrijednosti koje naredba `printf` treba ispisati. Isto tako se linije 11. i 12., u kojima se varijablama `V` i `OP` pridružuju vrijednosti izraza u kojima se koriste varijable `a`, `b` i `c`, ne mogu prebaciti ispred linija u kojima se unose vrijednosti za te varijable.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int OP,V,a,b,c;
 5.   printf("Unesite duljinu baze kvadra: ");
 6.   scanf("%d",&a);
 7.   printf("Unesite sirinu baze kvadra: ");
 8.   scanf("%d",&b);
 9.   printf("Unesite visinu kvadra: ");
10.   scanf("%d",&c);
11.   V=a*b*c;
12.   OP=2*(a*b+a*c+b*c);
13.   printf("Volumen kvadra iznosi %d a oplosje %d",V,OP);
14.   getch();}
```

Primjer 3.24.

Sljedeći primjer (program 3.25.) je program koji za uneseni iznos u eurima izračunava vrijednost tog iznosa u kunama. U zadatku su deklarirane dvije varijable, jedna za čuvanje unesenoga iznosa u eurima (`euro`), a druga za čuvanje unesenoga tečaja (`tecaj`). Izračun je ugrađen u ispis na mjesto gdje treba biti iznos u kunama. Ispis iznosa u eurima i kunama se prikazuje na dvije decimale(`%.2f`). Pri unosu vrijednosti treba paziti da se realni broj unosi s decimalnom točkom, a ne zarezom.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.    float euro,tecaj;
 5.    printf("Unesite zeljeni iznos u eurima: ");
 6.    scanf("%f",& euro);
 7.    printf("Unesite danasnji tecaj eura: ");
 8.    scanf("%f",& tecaj);
 9.    printf("%.2f eura iznosi %.2f kuna.", euro, euro * tecaj);
10.    getch();}
```

Program 3.25.

Slijedi primjer računalnoga programa gdje se prate promjene vrijednosti varijable `broj` te se u danom trenutku ispisuje `prethodnik` i `sljedbenik` prvotno unesene vrijednosti varijable `broj`. Odmah u liniji 7. se varijabli `pamti` pridruži prvotno unesena vrijednost za `broj`. Tada se nova vrijednost za varijablu `broj` dobije tako da se stara pomnoži sa 2 (linija 8., skraćeni operator `*=`) te se ispisuje ta nova vrijednost. Nakon toga se vrijednost koja se nalazi u varijabli `broj` umanji za 100 (linija 10. skraćeni operator `-=`) te se ispiše.

Na kraju, u liniji 15. se ispisuje prethodnik i sljedbenik prvotno unesene vrijednosti varijable `broj`, koja je spremljena u varijabli `pamti`.

Varijable `prethodnik` i `sljedbenik` pohranjuju vrijednosti prethodnika i sljedbenika (vrijednost im je pridružena u linijama 13. i 14.).

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int broj,sljedbenik,prethodnik,pamti;
 5.   printf("Unesite cijeli broj veći od 1000:");
 6.   scanf("%d",&broj);
 7.   pamti=broj;
 8.   broj*=2;
 9.   printf("Nova vrijednost varijable broj=%d\n",broj);
10.   broj-=100;
11.   printf("Nova vrijednost varijable b=%d\n",broj);
12.   printf("Sada zelimo ispisati prethodnik i sljedbenik
             originalnog cijelog broja broj.");
13.   prethodnik=pamti-1;
14.   sljedbenik=pamti+1;
15.   printf("prethodnik od %d je %d\nsljedbenik od %d je
             %d",pamti,prethodnik,pamti,sljedbenik);
16.   getch();}
```

Program 3.26.

Slijedi program 3.27. kojim se izračunava prosječna potrošnja goriva nekog automobila ako se zna količina potrošenoga goriva i prijeđena udaljenost u km (potrošnja će se izraziti na 100 km). Varijable prosječna potrošnja (`ppg`) i potrošeno gorivo (`pg`) su realnoga tipa, dok je varijabla za prijeđene kilometre (`km`) cjelobrojnoga tipa. U ovome su programu koraci koje korisnik treba izvesti numerirani i ispisani (naredbom `printf`) i oni zorno predočuju slijedno izvođenje algoritamskih koraka kojima se rješava praktični problem izračuna prosječne potrošnje goriva.

Formula za `ppg` glasi $ppg=(pg/km)\cdot 100$, gdje `pg/km` označava koliko je potrošeno goriva na 1 km. Pomnoženo sa 100 dobiva se traženi iznos za `ppg`.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int km;
 5.   float ppg,pg;
 6.   printf("1.Napunjen je spremnik goriva od automobila do
             vrha.\n");
 7.   printf("2.unesite koliko ste kilometara presli bez
             decimala:");
 8.   scanf("%d",&km);
 9.   printf("3.sada se opet ide napunit spremnik goriva do
             vrha.");
10.   printf("\n4.procitati na agregatu koliko litara goriva se
             natocilo.\n5.Unesite taj tocan broj s agregata:");
11.   scanf("%f",&pg);//pg=potroseno gorivo u litrima i decilitrima
12.   ppg=(pg/km)*100; //ppg prosječna potrošnja goriva na 100 km
13.   printf("\nZa %d km potrosio je %.2f l pa auto trosi %.2f na
            100km",km,pg,ppg);
14.   getch();}
```

Program 3.27.

Jednostavni primjer koji slijedi (program 3.28.) demonstrira značenje pokazivača. Kad se deklarira pokazivač on mora biti istog tipa podatka kao i vrijednost na čiju memorijsku lokaciju pokazuje. Isto tako pokazivaču se prije njegove uporabe mora dodijeliti adresa neke memorijske lokacije (npr. u liniji 4. mu je dodijeljena adresa varijable `x`). Oznaka `%p` označava ispis adrese memorijske lokacije (može se koristiti i `%d`), a iz linija 7. i 8. se vidi kako je adresa u pokazivaču `px` upravo adresa varijable `x`.

```
1.  #include<stdio.h>
2.  main(){
3.    int x,*px; // varijabla x i cjelobrojni pokazivač *px na x
4.    px=&x; /* pokazivač px sada ima adresu varijable x, on je
                 inicijaliziran*/
5.    printf("Unesite neki x : ");
6.    scanf("%d",&x);
7.    printf("x iznosi %d a adresa na kojoj je x je %p",x,px);
8.    printf("\n ili dobijemo isto\nx iznosi %d a adresa na kojoj
               je x je %p",x,&x);
9.    getch();}
```

Program 3.28.

Program 3.29. u nastavku opisuje primjenu pokazivača te operatora dereferenciranja (*). Deklarira se varijabla `c` tipa `char` (dakle 1 bajt) te pokazivač na tu varijablu `pc` (pri deklaraciji pokazivača uvijek se stavlja oznaka * kako bi ga se razlikovalo od ostalih varijabli tog tipa). U liniji 5 se varijabli `c` pridružuje vrijednost `'a'` (znak se pridružuje tako da se navede u jednostrukim navodnicima). Sadržaj varijable `c` promijenit će se preko pokazivača i operatora dereferenciranja s linijom 6. Linija 6. se interpretira kao: locirati adresu koju pohranjuje `pc`, pristupiti sadržaju te adrese preko operatora * te promijeniti njezin sadržaj. Kako je to adresa varijable `c`, mijenja se njezin sadržaj u znak `'b'`. Dakle `*pc` je sadržaj memorijske lokacije čija adresa je zapisana u pokazivaču `pc`.

Pa ipak, ovaj kod sadrži veliku grešku. Napravljena je pogreška jer se nigdje nije inicijalizirao pokazivač `pc` a išao se mijenjati sadržaj memorijske lokacije na koju ovaj pokazuje (linija 6.). Pokazivač se prije uporabe mora inicijalizirati tako da sadrži adresu neke memorijske lokacije koja je rezervirana za program koji se izvodi. Naime, prilikom deklaracije pokazivača u njemu se nalazi neka vrijednost koja se od prije nalazila u memoriji (*SMEĆE*). To *SMEĆE* se sigurno može interpretirati kao adresa neke memorijske lokacije. No problem je što tu memorijsku lokaciju računalo nužno nije rezerviralo za program koji se izvodi (već možda za neki drugi program ili lokacija ima oznaku da nije rezervirana). Ono što se događa ako se izvede operator dereferenciranja nad pokazivačem koji u sebi ima *SMEĆE* je pristup nedopuštenoj memorijskoj lokaciji što u konačnici izaziva pad izvođenja programa.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   char c,*pc;
 5.   c='a';
 6.   *pc='b';
 7.   printf("Na adresi %p smjesten je znak %c\n",&c,c);
 8.   printf("Na adresi %p smjesten je znak %c\n",pc,*pc);
 9.   getch();}
```

Program 3.29.

Ispravna varijanta ovoga programa prikazana je u programu 3.30 – ispravak je u liniji 6.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   char c,*pc;
 5.   c='a';
 6.   pc=&c;
 7.   *pc='b';
 8.   printf("Na adresi %p smjesten je znak %c\n",&c,c);
 9.   printf("Na adresi %p smjesten je znak %c\n",pc,*pc);
10.   getch();}
```

Program 3.30.

Slijedi još jedan primjer s pokazivačima i operatorom dereferenciranja (program 3.31). Deklarirane su dvije realne varijable s dvostrukom preciznošću te pokazivač `px`. Nakon inicijalizacije pokazivača s linijom 6., varijabli `y` se pridružuje vrijednost varijable `x`, ali preko dereferenciranja pokazivača `px` (linija 7.).

Provjera vrijednosti se izvodi ispisom u 8. i 9. liniji.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   double x=10,y; // varijable tipa double
 5.   double *px; // pokazivac na tip double
 6.   px=&x; /* pokazivač px sada ima adresu varijable x, on je
               inicijaliziran */
 7.   y=*px; /*varijabla y prima vrijednost varijable preko
               operatora dereferenciranja */
 8.   printf("Sada je jasno da je x=%5.1lf=y=%5.1lf\n",x,y);
 9.   printf("Moglo se pisati i ovako: \n");
10.   printf("Analogno se dobije koristeci dereferenciranje:
             x=%5.1lf=y=%5.1lf\n",x,*px);
11. getch();}
```

Program 3.31.

U narednome računalnom programu deklarira se polje od 5 elemenata te se koriste osnovne algebarske operacije na elementima polja. Indeksi koji određuju o kojem se elementu polja radi u ovome slučaju idu od 0 do 4. Prvi i drugi element polja unose se

pridruživanjem konkretnih cijelih brojeva (linije 5. i 6.), a treći i četvrti element unosi se preko naredbe `scanf` (korisnički unos). Moguće je preko naredbe `scanf` odjednom unijeti više vrijednosti koje se spremaju u različite varijable, ali je potrebno paziti koji je znak definiran kao znak koji ih odvaja (linija 8.).

Peti element polja dobiva se kao zbroj prva dva. U liniji 10. je navedeno pridruživanje vrijednosti elementu čiji indeks ne postoji u deklariranome polju (deklarirano je polje od 5 elemenata, dakle indeks zadnjeg elementa ovoga polja je 4). Iako računalo neće javiti grešku kad izvede liniju 10., ono što se dogodilo je da je računalo spremilo vrijednost u memorijsku lokaciju koja nije rezervirana za program koji se izvodi, što znači da tu memorijsku lokaciju može u nekome trenutku računalo dodijeliti drugome programu. U takvoj situaciji bi se dogodilo da program koji se izvodi želi zahvatiti memorijsku lokaciju koja je rezervirana za neki drugi program, odnosno dogodit će se nedopušteno pristupanje memorijskoj lokaciji koja u konačnici (slično kao i kod pokazivača) može izazvati rušenje programa. Programi izrađeni u nekim drugim programskim jezicima (npr. Java) bi kod dolaska na ovakvu liniju koda odmah prekinuli izvođenje i dojavili grešku.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int broj[5];
 5.    broj[0]=15;
 6.    broj[1]=10;
 7.    printf("Unesite 3. i 4. element polja odvojene zarezom: ");
 8.    scanf("%d,%d",&broj[2],&broj[3]);
 9.    broj[4]=broj[0]+broj[1];
10.    broj[5]=7;
11.    printf("Ispisi prvi element polja:%d\n",broj[0]);
12.    printf("Ispisi drugi element polja:%d\n",broj[1]);
13.    printf("Ispisi treci i cetvrti element polja: %d i
             %d\n",broj[2],broj[3]);
14.    printf("ispisi peti element polja:%d\n",broj[4]);
15.    printf("ispisi sesti element polja:%d\n",broj[5]);
16.    getch();}
```

Program 3.32.

Slijedi nekoliko računalnih programa vezanih za znakovne nizove ili stringove. U programu 3.33. deklarira se niz od 80 znakova (`str[80]`). Funkcija `puts` omogućava ispis *stringa* pod navodnicima a `gets` unos *stringa* kao niza znakova. Funkcijama `puts` i `gets` se kao argument daje niz znakova (kao vrijednost varijable ili kao konkretna vrijednost).

```
 1.  #include <stdio.h>
 2.  #include<conio.h>
 3.  main(){
 4.    char str[80];
 5.    puts ("Unesite string:");
 6.    gets (str);
 7.    puts ("Unijeli ste sljedeci string:");
 8.    puts (str);
 9.    getch();}
```

Program 3.33.

U računalnome programu koji slijedi demonstrirana je uporaba funkcija `strcpy()`, `strcat()` i `strlen()`. Uključena je zaglavna datoteka `string.h` zbog korištenja navedenih funkcija.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  #include<string.h>
 4.  main(){
 5.    char mojString[20];
 6.    char kopija[100];
 7.    strcpy( mojString, "Ivica i Marica" );
 8.    strcpy( kopija, mojString );
 9.    strcat( kopija, " se igraju ispred kuće." );
10.    puts(mojString);
11.    printf("String %s dugačak je %d znakova.\n", mojString,
              strlen(mojString));
12.    printf("%s\n", kopija);
13.    getch();}
```

Program 3.34.

U linijama koda 5. i 6. deklarirana su dva niza znakova (dva stringa): `mojString` od 20 znakova te `kopija` od 100 znakova. U liniji 7. izvedeno je kopiranje niza znakova `"Ivica i Marica"` u string `mojString`. `mojString` je mogao primiti dani niz znakova koji je manji od njegovoga kapaciteta od 20 znakova. U liniji 8. je sadržaj stringa `mojString` kopiran u string `kopija`. Uočava se da su `mojString` i `kopija` sada potpuno identičnoga sadržaja, tj. oba stringa sadrže niz `"Ivica i Marica"`. U liniji 9. stringu `kopija` se dodaje string `"Ivica i Marica"` te je aktualni sadržaj stringa `kopija` sada `"Ivica i Marica se igraju ispred šume"` i kapacitet od 100 znakova nije premašen. U liniji 10. ispisuje se niz `mojString`. U liniji 11. se sa `%s` konverzijom ispisuje string `mojString` te se izračunava broj znakova tog stringa korištenjem funkcije `strlen`. I na kraju, u liniji 12. ispisuje se cijeli sadržaj stringa `kopija`. Jasno je da se ovdje mogla koristiti i naredba `puts(kopija)`; koja bi generirala isti rezultat.

Sljedeći primjer koristi iste funkcije kao i prethodni.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include <string.h>
 4. main(){
 5.   char mz_moj[30], mz_moj1[70];
 6.   strcpy(mz_moj,"Programiranje u C-u");
 7.   strcpy(mz_moj1,mz_moj);
 8.   strcat(mz_moj1," studenti trebaju voljeti.");
 9.   puts(mz_moj);
10.   puts(mz_moj1);
11.   printf("%s",mz_moj1);
12.   getch();}
```

Program 3.35.

Na primjeru ispod prikazana je uporaba jednostavnoga zapisa kroz unos i ispis nekoga datuma koji se sastoji od dana, mjeseca i godine. Treba uočiti način na koji se dolazi do pojedine komponente zapisa (primjena točke).

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. typedef struct {
 4.   int dan;
 5.   int mjesec;
 6.   int godina;
 7. } Datum; //deklaracija zapisa imena Datum
 8. main(){
 9.   Datum danas;//danas je varijabla tipa podatka Datum
10.   printf("Unesi dan molim ");
11.   scanf("%d",&danas.dan);
12.   printf("Unesi mjesec molim ");
13.   scanf("%d",&danas.mjesec);
14.   printf("Unesi godinu molim");
15.   scanf("%d",&danas.godina);
16.   printf("danas je dan %d.%d.%d",danas.dan,
            danas.mjesec,danas.godina);
17.   getch();}
```

Program 3.36.

Program 3.37. je vrlo sličan prethodnom samo su komponente zapisa `dan` i `mjesec` sada nizovi znakova (*stringovi*). Komponenta `dan` se zadaje kao *string* od najviše 2 znaka (brojevi od 1-31) i `mjesec` kao *string* od najviše 9 znakova (mjeseci s najdužim nazivom imaju 9 znakova u genitivu). I jednom i drugom nizu znakova treba dodati još jedan znak za kraj niza (`\0`).

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. typedef struct {
 4.   int dan[2+1];
 5.   int mjesec[8+1];
 6.   int godina;
```

```
 7.  } Datum;
 8.  main(){
 9.    Datum danas;
10.    printf("Unesite dan kao niz dviju znamenki ");
11.    scanf("%s",danas.dan);
12.    printf("Unesite godinu ");
13.    scanf("%d",&danas.godina);
14.    printf("Unesite mjesec rijecima, npr. lipnja ");
15.    scanf("%s",danas.mjesec);
16.    printf("Danas je %s %s %d.",danas.dan,
              danas.mjesec,danas.godina);
17.    getch();}
```

Program 3.37.

U sljedećem računalnome programu je dan primjer u kome je stvoren novi tip podatka – zapis `Popis` čiji su članovi `naslov`, `autor`, `god_izdavanja` i `cijena`. Članovi `naslov` i `autor` su nizovi znakova (*stringovi*) od 100 odnosno 50 znakova. Član `god_izdavanja` je tipa podatka `int` (moglo se staviti i `unsigned int` jer godina ne može biti negativna) te je `cijena` tipa podatka `float` jer ona može biti u kunama i lipama.

U 12. liniji je deklarirana varijabla tipa podatka `Popis`, imena `vodic` te je inicijalizirana preko zadavanja vrijednosti članovima zapisa po točno definiranom redoslijedu. U 12. je liniji prikazano korištenje operatora točka (.) te je ispisan podatak o naslovu koji se nalazi unutar varijable `vodic` kao string. Kod ispisa je stavljen `%s` za string.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  typedef struct {
 4.     char naslov[100];
 5.     char autor[50];
 6.     int god_izdavanja;
 7.    float cijena;
 8.  } Popis;
 9.
10.  main(){
11.    Popis vodic={"Vodic za C","Marko Markovic ",2003,145.2};
12.    printf("Naslov knjige je : %s",vodic.naslov);
13.    getch();}
```

Program 3.38.

U sljedećemu primjeru se isti računalni program proširuje s korisničkim unosom vrijednosti koja se sprema u određenu komponentu zapisa.

U liniji 12. korisnički unos se kao *string* sprema u komponentu `naslov` varijable `knjiga`. U liniji 14. se korisnički unos kao realni tip podatka sprema u komponentu `cijena`. U liniji 15. se komponenti `cijena` varijable `knjiga` dodjeljuje konkretna vrijednost 67.32. U liniji 17. deklarira se varijabla `godina` (koja ne pripada nikakvoj

strukturi) te se njoj pridružuje vrijednost varijable `god_izdavanja` iz varijable `knjiga1` (`knjiga1` je deklarirana i inicijalizirana u liniji 15.).

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  typedef struct {
 4.     char naslov[100];
 5.     char autor[50];
 6.     int god_izdavanja;
 7.    float cijena;
 8.  } Popis;
 9.  main(){
10.     Popis knjiga;
11.     printf("Unesite naslov knjige, paziti, bez razmaka: " );
12.     scanf("%s",&knjiga.naslov);
13.     printf("Unesite cijenu knjige: " );
14.     scanf("%f",&knjiga.cijena);
15.     knjiga.cijena= 67.32;
16.     Popis knjiga1={"Programiranje C", "McGraw-Hill", 2003,345.2};
17.     int godina= knjiga1.god_izdavanja;
18.     printf("godina izdavanja od knjige1 jest %d\n",godina);
19.     printf ("Naslov je %s\n", knjiga1.naslov);
20.     printf ("%.2f kuna\n", knjiga1.cijena);
21.     getch();}
```

Program 3.39.

3.3.1. Zadaci za vježbu

1. Napisati računalni program kojim se unose proizvoljne vrijednosti za radijus baze te za visinu valjka. Ispisati vrijednosti iznosa oplošja te volumena valjka. $O = 2 \cdot r \cdot \pi \cdot v \quad V = r^2 \cdot \pi \cdot v$, gdje je r iznos polumjera baze a v iznos visine valjka.

2. Napisati računalni program za kalkulator konverzije dimenzija. Za unesenu decimalnu vrijednost duljine u metrima ispisati vrijednost duljine u milimetrima, centimetrima i kilometrima na tri decimale.

3. Napisati računalni program za izračun prijeđenoga puta u jednome mjesecu. Korisnik unosi udaljenost od kuće do posla (`float`) i nakon toga unosi broj dana (`int`) koliko u mjesecu ide na posao. Program mora ispisati prijeđenu kilometražu u jednome mjesecu (na dvije decimale) uzevši u obzir da se u jednom danu prijeđe put od kuće na posao i s posla kući.

4. Napisati računalni program za kalkulator konverzije valuta. Za unesenu decimalnu vrijednost novca u kunama potrebno je ispisati vrijednost u četiri strane valute (EUR, USD, CHF, GBP) na dvije decimale. Konverziju napraviti prema sljedećem tečaju:
   1 EUR = 7,55 KN
   1 USD = 5,85 KN
   1 CHF = 6,26 KN
   1 GBP = 9,32 KN

5. Prethodni zadatak treba riješiti tako da postoji varijabla za svaku vrijednost tečaja, tj. omogućeno je da za svaku unesenu vrijednost u kunama korisnik programa može unijeti vrijednost tečaja za sve četiri valute. Po unosu se ispisuju četiri iznosa u pripadajućim valutama, svaki u svojoj liniji.

6. Napraviti računalni program u komu se definira zapis Student koji se sastoji iz triju komponenti: ime, prezime i godina studija. Dalje treba deklarirati varijable za dva različita studenta, pridružiti svakome sve komponente te ispisati ih u tri retka od kojih je prvi naslovni a ostala dva vrijednosti komponenti za studenta. U ispisu odvajati tabulatorima vrijednosti pojedinih komponenti. Naslov ispisa je
   ```
   Ime         Prezime         God. studija
   ```

7. Prethodni zadatak treba izraditi tako da se deklarira polje od tri studenta. Korisnik treba unijeti sve tražene podatke za sva tri studenta. Potom je potrebno ispisati podatke o studentima. Naslov ispisa je:
   ```
   Ime         Prezime         God. studija
   ```

8. Napraviti računalni program kojim se deklariraju tri polja od po dva elemenata (obratiti pažnju na tip podatka – vidjeti primjer ispisa). Korisnik treba unijeti elemente prvih dvaju polja. Elementi trećega polja dobit će se zbrajanjem tako da je 1. element 3. polja zbroj 1. elementa 1. polja te 1. elementa 2. polja. Jednako se dobije i 2. element 3. polja. Ispisati dva retka sa tri vrijednosti – vrijednost prvoga, drugoga te trećega polja. Npr:
   ```
   2.1     3.2     5.3
   1.6     2.1     3.7
   ```

9. Napraviti računalni program kojim korisnik unosi dva *stringa*. Potrebno je deklarirati novi *string* u kojem će pisati tekst: `Jezik C`. Ispisati sve *stringove* i pored svakoga njegovu duljinu. Dodati 3. *stringu* tekst: `je svestran`. Ispisati novu vrijednost 3. *stringa*.

10. Napraviti računalni program u komu su deklarirane dvije cjelobrojne varijable te pokazivači na te varijable. Treba zbrojiti varijable (njihove vrijednosti unosi korisnik) tako da se zbroj pohrani u neku novu varijablu. Za novu varijablu također deklarirati pokazivač. Potrebno je ispisati vrijednosti svake pojedine varijable ali primjenom njihovih pokazivača.

4. RAZGRANATA ALGORITAMSKA STRUKTURA U C

Poglavlje **Razgranata algoritamska struktura u C** prikazuje implementaciju razgranate algoritamske strukture u programskome jeziku C. Prikazane su naredbe `if-else if-else` te `switch-case`. Na kraju se poglavlja nalazi niz praktičnih zadataka.

Po završetku ovoga poglavlja čitatelj će moći:

- prepoznati situaciju koja zahtijeva uvođenje razgranate strukture
- izraditi računalni program s razgranatom algoritamskom strukturom
- formulirati zadatak s razgranatom algoritamskom strukturom
- usporediti primjenu `if-else if-else` te `switch-case` naredbe
- izračunati rezultat računalnoga programa uz dane početne vrijednosti
- izraditi pseudo kod iz danoga računalnog programa
- izraditi računalni program iz danoga pseudo koda
- objasniti svrhu pojedine linije koda
- prepoznati ugnježđenu razgranatu algoritamsku strukturu
- prepoznati linijsku algoritamsku strukturu unutar razgranate
- demonstrirati izradu bloka naredbi
- objasniti odnos deklaracije varijable i bloka naredbi.

4.1. Realizacija razgranate algoritamske strukture u C

Razgranata algoritamska struktura (grananje, selekcija) omogućava uvjetno izvođenje niza algoritamskih koraka. Dakle, skup algoritamskih koraka (blok naredbi u programskome jeziku C) izvodi se tek onda kada je zadovoljen uvjet. Uvjet predstavlja neki logički izraz koji može poprimiti vrijednost istina ili neistina. Logički se izrazi grade iz relacijskih operatora (<, >, <=, >=, ==, !=) i logičkih operatora (`&&`, `||`, `!`) – (vidjeti poglavlje 2.5.3 Logičke operacije). Razgranata algoritamska struktura se u programskome jeziku C realizira preko naredbi `if-else if-else`, te `switch-case`.

4.1.1. Oblici naredbe `if-else if-else`

Ova će se naredba opisati kroz opis primjene naredbe `if`, potom naredbe `if-else`, i zatim naredbe `if-else if-else`.

4.1.1.1. Naredba `if`

Najjednostavniji oblik primjene naredbe `if-else if-else` je uporaba samo naredbe `if`, čime se omogućava realizacija preskoka nekoga bloka naredbi. Opći oblik naredbe `if` glasi:

```
1.  ...
2.  naredbaX
3.  if(logički_izraz)
4.  {
5.     blok_naredbi
6.  }
7.  naredbaY
8.  ...
```

Nakon izvođenja naredbe `naredbaX`, izvodi se naredba `if`. Prvo se izračunava `logički_izraz`. Ako on ima vrijednost istina (`True`) tada će se izvesti naredbe koje se nalaze u `blok_naredbi`. Dakle, izvest će se one naredbe koje se nalaze unutar vitičastih zagrada (između linija 4. i 6.). U slučaju da `logički_izraz` ima vrijednost neistina (`False`), `blok_naredbi` neće biti izveden. Radi čitljivosti, dobra je praksa blok naredbi podvući (tabulatorom) pod pripadajuću `if` naredbu – kako je to i učinjeno u prikazu općeg oblika naredbe. U bloku naredbi može se nalaziti nova `if` naredba sa svojim blokom naredbi itd. (ugnježđene `if` naredbe). Naredba `naredbaY` će se izvesti bez obzira je li `logički_izraz` istinit ili ne – dakle neovisno o tome jesu li izvedene naredbe koje se nalaze u `blok_naredbi`.

Sljedeći primjer prikazuje računalni program koji ispisuje svojstva unesenoga broja (pozitivan, negativan ili nula):

```
1.  #include<stdio.h>
2.  main(){
3.    int broj;
```

```
 4.   printf("Unesite neki cijeli broj: \n");
 5.   scanf("%d", &broj);
 6.   if (broj > 0) {
 7.     printf ("Broj %d je pozitivan.", broj);
 8.   }
 9.   if (broj < 0) {
10.     printf ("Broj %d je negativan.", broj);
11.   }
12.   if (broj == 0) {
13.     printf ("Broj %d je nula.", broj);
14.   }
15.   getch();}
```

Program 4.1.

U liniji 5. uneseni broj se sprema kao vrijednost varijable `broj`. U liniji 6. se izračunava logički izraz `broj > 0`. Ako je izračunata vrijednost istina, bit će izvedena linija koda 7. ,odnosno blok naredbi. Ako je pak izračunata vrijednost neistina, linija koda 7. neće biti izvedena. Nakon izvođenja 6. linije koda i eventualno 8. linije izvodi se 9. linija koda u kojoj se ponovno izvodi izračun ali sada logičkoga izraza `broj < 0`. U ovisnosti o njegovoj vrijednosti izvodi se ili ne linija koda 10. Nakon linije koda 9. (i 10. ako se izvodi) izvodi se linija koda 12. i izračun njezina logičkog izraza (obratiti pažnju da se u ovom logičkom izrazu koristi relacijski operator == kojim se provjerava jednakost, a ne = koji predstavlja operator pridruživanja vrijednosti varijabli).

Sljedeći računalni program ispisuje najmanji broj od tri unesena broja. U slučaju da su sva tri broja jednaka, ispisuje poruku "`Sva tri broja su jednaka`". U svrhu čuvanja vrijednosti triju brojeva koristit će se polje od tri elementa.

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int brojevi[3];
 4.    printf("Unesite 1. broj: ");
 5.    scanf("%d",&brojevi[0]);
 6.    printf("Unesite 2. broj: ");
 7.    scanf("%d",&brojevi[1]);
 8.    printf("Unesite 3. broj: ");
 9.    scanf("%d",&brojevi[2]);
10.    if((brojevi[0]<brojevi[1])&&(brojevi[0]<brojevi[2]))
11.      printf("Broj %d je najmanji.", brojevi[0]);
12.    if((brojevi[1]<brojevi[0])&&(brojevi[1]<brojevi[2]))
13.      printf("Broj %d je najmanji.", brojevi[1]);
14.    if((brojevi[2]<brojevi[0])&&(brojevi[2]<brojevi[1]))
15.      printf("Broj %d je najmanji.", brojevi[2]);
16.    if((brojevi[0]==brojevi[1])&&(brojevi[0]==brojevi[2]))
17.      printf("Brojevi su jednaki.");
18.    getch();}
```

Program 4.2.

Logički izraz koji se javlja u naredbi `if` sastoji se iz relacijskih i logičkih operatora. U osnovi se provjerava je li broj strogo manji od preostalih dvaju brojeva. Ako je, ispisuje se da je on najmanji broj (obratiti pažnju da naredba `prinf` nije u bloku naredbi – unutar vitičastih zagrada – ovo je moguće napraviti samo onda ako se blok naredbi, koji se treba izvesti u slučaju da je logički izraz istina, sastoji iz samo jedne naredbe – što je ovdje slučaj. U svim ostalim slučajevima naredbe treba staviti u blok naredbi). Poseban je uvjet u liniji 16. gdje se provjerava jesu li svi brojevi međusobno jednaki. Ovaj program ispravno ispisuje poruku ako se unesu tri različita (npr. 3, 5, 2), tri ista broja (npr. 3, 3, 3) ili dva ista broja koja nisu najmanja (npr. 4, 6, 6). No, ako se unesu dva ista broja koja su najmanja (npr. 6, 2, 2), program neće ispisati nikakvu poruku. To je zato što logički izraz niti u jednoj `if` naredbi neće imati vrijednost istina, a svaki ispis poruka je uvjetovan. Rješenje ovoga nedostatka je u nadopunjavanju odgovarajućih logičkih izraza dodatnim uvjetima te u izmjeni relacijskih operatora usporedbe (koristiti <=), a kako to pokazuje sljedeći primjer:

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int brojevi[3];
 4.    printf("Unesite 1. broj: ");
 5.    scanf("%d",&brojevi[0]);
 6.    printf("Unesite 2. broj: ");
 7.    scanf("%d",&brojevi[1]);
 8.    printf("Unesite 3. broj: ");
 9.    scanf("%d",&brojevi[2]);
10.    if(((brojevi[0]<=brojevi[1])&&(brojevi[0]<brojevi[2])) ||
         ((brojevi[0]<brojevi[1])&&(brojevi[0]<=brojevi[2])))
11.      printf("Broj %d je najmanji.", brojevi[0]);
12.    if(((brojevi[1]<=brojevi[0])&&(brojevi[1]<brojevi[2])) ||
         ((brojevi[1]<brojevi[0])&&(brojevi[1]<=brojevi[2])))
13.      printf("Broj %d je najmanji.", brojevi[1]);
14.    if(((brojevi[2]<=brojevi[1])&&(brojevi[2]<brojevi[0])) ||
         ((brojevi[2]<brojevi[1])&&(brojevi[2]<=brojevi[0])))
15.      printf("Broj %d je najmanji.", brojevi[2]);
16.    if((brojevi[0]==brojevi[1])&&(brojevi[0]==brojevi[2]))
17.      printf("Brojevi su jednaki.");
18.    getch();}
```

Program 4.3.

Navedeni program sada daje ispravnu poruku i kada su unesena dva ista broja koja su najmanja, ali se ta poruka ispisuje dva puta. To je zato što su u slučaju unosa dva ista najmanja broja zadovoljeni logički izrazi dvije `if` naredbe. Npr. ako se unesu brojevi 1, 1, 2, bit će zadovoljen logički izraz `if` naredbe u liniji 10., ali i u liniji 12. Zbog toga će biti izvedene linije koda 11. i 13., odnosno poruka će biti ispisana dva puta. Rješenje ovoga problema dano je u nastavku:

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int brojevi[3], najmanji;
```

```
 4.    printf("Unesite 1. broj: ");
 5.    scanf("%d",&brojevi[0]);
 6.    printf("Unesite 2. broj: ");
 7.    scanf("%d",&brojevi[1]);
 8.    printf("Unesite 3. broj: ");
 9.    scanf("%d",&brojevi[2]);
10.    if(((brojevi[0]<=brojevi[1])&&(brojevi[0]<brojevi[2])) ||
         ((brojevi[0]<brojevi[1])&&(brojevi[0]<=brojevi[2])))
11.      najmanji = brojevi[0];
12.    if(((brojevi[1]<=brojevi[0])&&(brojevi[1]<brojevi[2])) ||
         ((brojevi[1]<brojevi[0])&&(brojevi[1]<=brojevi[2])))
13.      najmanji = brojevi[1];
14.    if(((brojevi[2]<=brojevi[1])&&(brojevi[2]<brojevi[0])) ||
         ((brojevi[2]<brojevi[1])&&(brojevi[2]<=brojevi[0])))
15.      najmanji = brojevi[2];
16.    if((brojevi[0]==brojevi[1])&&(brojevi[0]==brojevi[2]))
17.      printf("Brojevi su jednaki.");
18.    if(!((brojevi[0]==brojevi[1])&&(brojevi[0]==brojevi[2])))
19.      printf("Broj %d je najmanji.", najmanji);
20.    getch();}
```

Program 4.4.

U navedenome primjeru izvele su se dvije promjene. U `if` naredbama (linije 10., 12. i 14.) više se ne ispisuje poruka, već se u varijablu `najmanji` pamti najmanji broj. U liniji 18. postoji `if` naredba koja se koristi za ispis poruke o najmanjemu broju. Poruka se ispisuje samo onda ako sva tri unesena broja nisu jednaka (koristi se logički operator `!` - negacija).

Sljedeći primjer prikazuje jednostavniji program koji ispisuje najmanji broj od tri unesena.

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int brojevi[3], najmanji;
 4.    printf("Unesite 1. broj: ");
 5.    scanf("%d",&brojevi[0]);
 6.    printf("Unesite 2. broj: ");
 7.    scanf("%d",&brojevi[1]);
 8.    printf("Unesite 3. broj: ");
 9.    scanf("%d",&brojevi[2]);
10.    najmanji = brojevi[0];
11.    if(brojevi[1]<najmanji)
12.      najmanji = brojevi[1];
13.    if(brojevi[2]<najmanji)
14.      najmanji = brojevi[2];
15.    if((brojevi[0]==brojevi[1])&&(brojevi[0]==brojevi[2]))
16.      printf("Brojevi su jednaki.");
17.    if(!((brojevi[0]==brojevi[1])&&(brojevi[0]==brojevi[2])))
18.      printf("Broj %d je najmanji.", najmanji);
19.    getch();}
```

Program 4.5.

Ideja je ovoga algoritma da se u posebnu varijablu `najmanji` sačuva vrijednost prvog unesenog broja (linija 10.). Potom se provjerava je li možda koji od preostalih

brojeva manji od vrijednosti koja se čuva u varijabli `najmanji`. Ako je, onda se `najmanji` mijenja (linije 11., 12., 13. i 14.). Na kraju se izvodi ili ispis da su svi brojevi jednaki ili ispis najmanjega broja.

U dosadašnjim primjerima programa za ispis najmanjeg od tri unesena broja koristilo se polje od tri elementa. U istu su se svrhu mogle koristiti i tri varijable ili zapis s tri komponente.

Slijedi primjer u kome je potrebno usporediti `char` tip podatka. RGB model (Red, Green, Blue) je najčešće korišteni sklopovski orijentirani model boja. Kombinacijom parova navedenih boja dobivaju se sljedeće boje: red + blue = magenta, red + green = yellow, green + blue = cyan. Izradit će se program koji od korisnika traži unos početnoga slova dvije od tri boje te ispisuje boju koja nastaje njihovom kombinacijom, odnosno ako su unesene iste dvije boje, ispisuje tu boju. U svrhu čuvanja dvije boje koristit će se zapis s dvjema komponentama.

```
 1.  #include<stdio.h>
 2.  typedef struct {
 3.    char boja1;
 4.    char boja2;} DvijeBoje;
 5.
 6.  main(){
 7.    DvijeBoje kombinacijaBoja;
 8.    printf ("Unesite 1. boju (R/r, G/g, B/b): ");
 9.    scanf("%c", &kombinacijaBoja.boja1);
10.    printf ("Unesite 2. boju (R/r, G/g, B/b): ");
11.    scanf(" %c", &kombinacijaBoja.boja2);
12.    if(kombinacijaBoja.boja1 == 'r' ||
           kombinacijaBoja.boja1 == 'R'){
13.       if(kombinacijaBoja.boja2=='b' ||
             kombinacijaBoja.boja2 == 'B'){
14.           printf("r + b = red + blue = magenta.");
15.       }
16.       if(kombinacijaBoja.boja2 =='g'||
              kombinacijaBoja.boja2=='G'){
17.           printf("r + g = red + green = yellow.");
18.       }
19.       if(kombinacijaBoja.boja2=='r'||
              kombinacijaBoja.boja2=='R'){
20.           printf("r + r = red.");
21.       }
22.    }
23.    if(kombinacijaBoja.boja1 == 'b' ||
           kombinacijaBoja.boja1 == 'B'){
24.       if(kombinacijaBoja.boja2 =='r'||
             kombinacijaBoja.boja2== 'R'){
25.           printf("b + r = blue + red = magenta.");
26.       }
27.       if(kombinacijaBoja.boja2=='g'||
              kombinacijaBoja.boja2 == 'G'){
28.           printf("b + g = blue + green = cyan.");
29.       }
30.       if(kombinacijaBoja.boja2=='b'||
```

```
          kombinacijaBoja.boja2=='B'){
31.         printf("b + b = blue.");
32.     }
33.   }
34.   if(kombinacijaBoja.boja1=='g'||
         kombinacijaBoja.boja1== 'G'){
35.     if(kombinacijaBoja.boja2=='r'||
          kombinacijaBoja.boja2== 'R'){
36.         printf("g + r = green + red = yellow.");
37.     }
38.     if(kombinacijaBoja.boja2=='b'||
          kombinacijaBoja.boja2=='B'){
39.         printf("g + b = green + blue = cyan.");
40.     }
41.     if(kombinacijaBoja.boja2=='g'||
          kombinacijaBoja.boja2=='G'){
42.         printf("g + g = green.");
43.     }
44.   }
45.   getch();}
```

Program 4.6.

Ovaj primjer prikazuje da se provjera je li varijabla tipa podatka `char` jednaka nekoj vrijednosti radi tako da se ta vrijednost (`znak`) stavi u jednostruke navodnike (`'`) – obratiti pažnju na provjeru unosa malog, odnosno velikoga slova. Primjer također pokazuje ugnježđene `if` naredbe. Posebnu pažnju u sklopu ovoga primjera treba dati liniji 11. u koju se unosi drugi znak. Naime, u liniji 9. je unesen prvi znak, ali je pritisnuta i tipka *Enter* kao potvrda unosa. No, tipka *Enter* je također znak pa zapravo u liniji 9. nije unesen jedan, već dva znaka.

To stvara problem pri unosu drugoga znaka jer je on već unesen pri prvome unosu. Rezultat je da nije omogućen unos drugoga znaka, već se izvođenje programa nastavlja na liniju 12. itd. Postoji više načina da se ovaj problem riješi (npr. primjena naredbe `getch()` umjesto `scanf()`). Ovdje je primijenjeno rješenje u kome se u liniji 11. ispred konverzijskoga znaka `%c` stavlja jedan razmak (*space*). Ovime se naredbi `scanf` nalaže da zanemari sve prazne znakove ispred znaka koji se unosi. Kako *Enter* spada u grupu praznih znakova, to on neće biti uzet u obzir kao drugi znak te će u konačnici i liniji 11. biti moguće unijeti stvarni drugi znak.

4.1.1.2. Naredba `if-else`

Razgranata algoritamska struktura realizirana preko naredbe `if-else` primjenjuje se onda kada postoje dva bloka naredbi od kojih se jedan treba izvesti kada je logički izraz istinit, a drugi kada je neistinit. Dakle, jedan od dva bloka naredbi će sigurno biti izveden. Opći oblik naredbe `if-else` glasi:

```
1.  ...
2.  naredbaX
3.  if(logički_izraz)
4.  {
```

```
 5.    blok_naredbi_1
 6.  }
 7.  else
 8.  {
 9.    blok_naredbi_2
10.  }
11.  naredbaY
12.  ...
```

Poslije izvođenja naredbe `naredbaX` izvodi se naredba `if`, odnosno izračunava se `logički_izraz`. Ako on ima vrijednost istina (`True`), tada će se izvesti naredbe koje se nalaze u `blok_naredbi_1`, a naredbe koje se nalaze u `blok_naredbi_2` neće biti izvedene. U obrnutome slučaju, kada `logički_izraz` ima vrijednost neistina (`False`), tada neće biti izvedene naredbe u `blok_naredbi_1`, već samo one koje se nalaze u `blok_naredbi_2`. I ovdje se u bloku naredbi može nalaziti nova `if` ili `if-else` naredba sa svojim blokom naredbi itd. (ugnježđene `if` naredbe). Naredba `naredbaY` će se izvesti bez obzira je li `logički_izraz` istinit ili ne – dakle, neovisno o tome jesu li izvedene naredbe koje se nalaze u `blok_naredbi_1` ili `blok_naredbi_2`. Treba obratiti pažnju da nakon ključne riječi `else` ne dolazi nikakav logički izraz koji bi se trebao računati budući da je on već dan iza ključne riječi `if`.

Sljedeći primjer prikazuje ispis svojstva unesenoga broja (pozitivan, negativan ili nula):

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int broj;
 4.    printf("Unesite neki cijeli broj: \n");
 5.    scanf("%d", &broj);
 6.    if (broj == 0) {
 7.      printf ("Broj %d je nula.", broj);
 8.    }
 9.    else {
10.      if (broj < 0) {
11.        printf ("Broj %d je negativan.", broj);
12.      }
13.      else{
14.        printf ("Broj %d je pozitivan.", broj);
15.      }
16.    }
17.    getch();}
```

Program 4.7.

U slučaju da je za broj unesena nula (linija koda 5.) u liniji koda 6. je zadovoljen uvjet `(broj == 0)` te će stoga biti izvršen ispis "`Broj 0 je nula.`" (linija koda 7.). Ako je pak za broj uneseno nešto što je različito od nula, tada uvjet u liniji 6. neće biti zadovoljen (bit će `False`) pa će biti izveden blok naredbi koji pripada `else` dijelu

(linija koda 9.). Ovdje će pak biti provjereno je li broj negativan (linija 10.) i ako je ispisuje se odgovarajuća poruka (linija 11.). Ako pak broj nije negativan, izvodi se blok odgovarajuće `else` naredbe (linija 13.). Zadnja linija koda koja se izvodi nakon izvedenih svih `if-else` naredbi je linija 17.

U slučajevima ugnježdenih `if` i `if-else` naredbi treba voditi brigu o tome kome `if`-u pripada koji `else`. U tu svrhu je preporučljivo naredbe `if` i `else` pisati jednu ispod druge i držati ih poravnatima. To zorno prikazuju sljedeća tri primjera s različitim formatiranjem istoga programskog koda.

```
if (A == 0)
if (B == 0)
printf("C");
else
printf("D");
```

```
if (A == 0)
   if (B == 0)
      printf("C");
   else
      printf("D");
```

```
if (A == 0){
   if (B == 0) {
      printf("C");
   }
   else {
      printf("D");
   }
}
```

Tablica 2.1.

Vidljivo je da naredba `else` ne pripada prvoj naredbi `if` (što bi se moglo pogrešno zaključiti iz prvoga neformatiranoga primjera programskoga koda). U drugom i trećemu primjeru jasno se vidi da naredba `else` pripada drugoj `if` naredbi koja se nalazi u bloku naredbi za prvi `if`.

U prethodnom je poglavlju prikazan jednostavniji primjer pronalaska najmanjega broja od njih tri. Slijedi isti primjer, ali sada s `if-else` naredbom. Za čuvanje vrijednosti triju brojeva koriste se tri varijable.

```
 1. #include<stdio.h>
 2. main(){
 3.   int broj1, broj2, broj3, najmanji;
 4.   printf("Unesite tri broja odvojena zarezom: ");
 5.   scanf("%d, %d, %d",&broj1, &broj2, &broj3);
 6.   najmanji = broj1;
 7.   if(broj2<najmanji){
 8.     najmanji = broj2;
 9.   }
10.   else {
11.     if (broj3<najmanji){
12.       najmanji = broj3;
13.     }
14.   }
15.   if((broj1==broj2)&&(broj1==broj3))
16.     printf("Brojevi su jednaki.");
17.   else
18.     printf("Broj %d je najmanji.", najmanji);
19.   getch();}
```

Program 4.8.

Brojevi se unose tako da se upišu u jednoj liniji i da se odvoje zarezom. To omogućava 5. linija koda. Osnovna je razlika ovoga primjera i primjera u prethodnome poglavlju što će se ovdje uvjet `broj3<najmanji` (linija koda 11.) provjeravati samo onda kada uvjet `broj2<najmanji` nije zadovoljen (linija koda 7.). U primjeru u prethodnome poglavlju uvjet `broj3<najmanji` uvijek se provjerava. Slično je i s 18. linijom koda. Ono što se ovdje postiglo naredbom `else` je smanjena potreba za obavljanjem izračuna logičke operacije, čime je u konačnici ubrzano izvođenje računalnoga programa.

4.1.1.3. Naredba `if-else if-else`

Razgranata algoritamska struktura realizirana preko naredbe `if-else if-else` primjenjuje se onda kada postoji više blokova naredbi od kojih se jedan treba izvesti kada je njegov pripadajući logički izraz istinit, a ostali se ne trebaju izvesti. Opći oblik naredbe `if-else if-else` glasi:

```
 1.  ...
 2.  naredbaX
 3.  if(logički_izraz_1){
 4.    blok_naredbi_1
 5.  }
 6.  else if (logički_izraz_2){
 7.    blok_naredbi_2
 8.  }
 9.  ...
10.  else if (logički_izraz_n){
11.    blok_naredbi_n
12.  }
13.  else{
14.    blok_naredbi_x
15.  }
16.  naredbaY
17.  ...
```

Poslije izvođenja naredbe `naredbaX` izvodi se naredba `if`, odnosno izračunava se `logički_izraz_1`. Ako je on zadovoljen, tada će se izvesti naredbe koje se nalaze u `blok_naredbi_1`, a naredbe u ostalim blokovima neće biti izvedene – u biti nakon izvođenja svih naredbi u `bloku_naredbi_1`, bit će izvedena `naredbaY`. U slučaju da nije zadovoljen `logički_izraz_1`, provjerava se je li zadovoljen `logički_izraz_2`. Ako je, izvodi se `blok_naredbi_2` i potom `naredbaY`. Ako nije, radi se provjera sljedećega logičkog izraza. Ako niti jedan logički izraz nije zadovoljen, onda će biti izveden `blok_naredbi_x`, budući da se on nalazi u bloku za `else`. U strukturi `if-else if-else` moguće je izostaviti dio `else`, dakle strukturu završiti s `else if`. U tome slučaju, da bi se izveo barem jedan blok naredbi, mora biti zadovoljen njegov (dakle jedan) logički izraz. U primjeni `else` dijela, niti jedan logički izraz ne mora biti zadovoljen. To znači da će sigurno neki blok naredbi biti izveden.

Sljedeći primjer prikazuje ispis svojstva unesenoga broja (pozitivan, negativan ili nula, te paran ili neparan – nula nije niti paran niti neparan):

```
 1. #include<stdio.h>
 2. main(){
 3.   int broj;
 4.   printf("Unesite neki cijeli broj: \n");
 5.   scanf("%d", &broj);
 6.   if (broj == 0) {
 7.      printf ("Broj %d je nula i nije niti paran niti neparan.",
 8.              broj);
 9.   }
10.   else if (broj < 0) {
11.      printf ("Broj %d je negativan ", broj);
12.      if (broj%2 == 0){
13.         printf ("i paran.");
14.      }
15.      else{
16.         printf ("i neparan.");
17.      }
18.   }
19.   else{
20.      printf ("Broj %d je pozitivan ", broj);
21.      if (broj%2 == 0){
22.         printf ("i paran.");
23.      }
24.      else{
25.         printf ("i neparan.");
26.      }
27.   }
28.   getch();}
```

Program 4.9.

Može se uočiti blok `if-else if-else` (linije koda 6., 9. i 18.) kojim se provjerava je li broj nula, negativan ili pozitivan. Postavljeni su logički uvjeti za provjeru je li broj nula (linija 6.) ili negativan (linija 9.). Ako nije niti jedno niti drugo onda je pozitivan i taj slučaj je uhvaćen naredbom `else`. U odgovarajućemu bloku za negativan i pozitivan broj izvodi se provjera je li broj paran, odnosno neparan (linije 11. i 14., odnosno 20. i 23.). Ispis se u navedenim blokovima radi tako da se prvo ispiše je li broj pozitivan ili negativan (linija 10., odnosno 19.), pa se na taj ispis dalje nalijepi novi ispis kojim se govori je li broj paran ili neparan (linije 12. i 15., odnosno 21. i 24.)

U sljedećem je primjeru prikazan računalni program kojim se unose dva broja i osnovna računska operacija koja se nad njima treba izvesti.

```
 1. #include<stdio.h>
 2. main(){
 3.   int broj1,broj2;
 4.   char operacija;
 5.   printf("Unesite prvi broj:");
 6.   scanf("%d",&broj1);
```

```
 7.    printf("Unesite drugi broj: ");
 8.    scanf("%d",&broj2);
 9.    printf("Racunske operacije su\n(+)zbrajanje\n(-)
             oduzimanje\n(*)mnozenje\n(/)djeljenje\n");
10.    printf("Unesite racunsku operaciju: ");
11.    scanf(" %c", &operacija);
12.    if(operacija == '+')
13.      printf("%d + %d = %d", broj1, broj2, broj1+broj2);
14.    else if(operacija == '*')
15.      printf("%d * %d = %d", broj1, broj2, broj1*broj2);
16.    else if(operacija == '-')
17.      printf("%d - %d = %d", broj1, broj2, broj1-broj2);
18.    else if (operacija == '/') {
19.      printf("%d / %d = ", broj1, broj2);
20.      if (broj2 == 0)
21.        printf("Djeljenje s nulom nije dozvoljeno.");
22.      else
23.        printf("%.2f", (float) broj1/broj2);
24.  }
25.    else
26.      printf("Nepostojeca racunska operacija.");
27.    getch();}
```

Program 4.10.

U računalnome programu treba uočiti da se u `else` dijelu `if-else if-else` strukture zahvaća slučaj kada je unesena nepostojeća računska operacija. Također treba uočiti da se u bloku naredbi koji se izvodi za računsku operaciju dijeljenja nalazi i provjera je li dijeljenje uopće moguće (drugi uneseni broj ne smije biti nula).

4.1.2. Naredba `switch-case`

Razgranata algoritamska struktura realizirana preko naredbe `switch-case` je slična realizaciji strukture preko naredbe `if-else if-else`. Osnovna razlika je što se u sklopu logičkog izraza u `switch-case` naredbi može naći samo cjelobrojna vrijednost (`char` ili `int`). Opći oblik naredbe `switch-case` glasi:

```
 1.  ...
 2.  naredbaX
 3.  switch (izraz){
 4.  case konstanta_1:
 5.       blok_naredbi_1
 6.       break
 7.  case konstanta_2:
 8.       blok_naredbi_2
 9.  ...
10.  case konstanta_n:
11.       blok_naredbi_n
12.  default:
13.       blok_naredbi_x
```

```
14.  }
15.  naredbaY
16.  ...
```

Nakon naredbe `naredbaX,` izračunava se `izraz` u `switch` naredbi. Izračunata vrijednost ovoga izraza može biti samo cjelobrojna vrijednost (`char` ili `int`). Nakon toga provjerava se je li izračunata vrijednost izraza jednaka konstanti `konstanta_1` (dakle prvom `case`). Ako je, izvodi se `blok_naredbi_1`. U sklopu ovoga bloka naredbi, kao zadnja naredba nalazi se `break`. Ona naređuje računalu da izađe iz `switch-case` strukture i izvede sljedeću naredbu – u primjeru je to naredba `naredbaY`. Kada `break` ne bi postojao, računalo bi nastavilo slijedno izvoditi sljedeći blok naredbi – u primjeru bi se izveo `blok_naredbi_2` itd. dok se ne bi došlo ili do `break` naredbe ili do zadnjega bloka `blok_naredbi_x`. Ova osobina `switch-case` strukture se naziva propadanje. Naredba `default` u strukturi `switch-case` označava blok naredbi koji treba biti izveden ako niti jedan `case` nije zadovoljen (slično kao `else` u strukturi `if-else if-else`). `Default` se može i izostaviti. Također se mogu izostaviti oznake za blok naredbi (vitičaste zagrade). No, treba uočiti da su obavezne vitičaste zagrade koje otvaraju nizanje elemenata `case` i `default`.

Slijedi primjer računalnoga programa iz prethodnoga poglavlja. Za potrebe spremanja dvaju brojeva i operacije izradit će se struktura s dvije komponente – brojevi (polje cjelobrojnih vrijednosti) i operacija (znak).

```
 1.  #include<stdio.h>
 2.  typedef struct {
 3.    int brojevi[2];
 4.    char operacija;} Izraz;
 5.  main(){
 6.    Izraz korisnickiPodaci;
 7.    printf("Unesite prvi broj: ");
 8.    scanf("%d",&korisnickiPodaci.brojevi[0]);
 9.    printf("Unesite drugi broj: ");
10.    scanf("%d",&korisnickiPodaci.brojevi[1]);
11.    printf("Racunske operacije su\n(+) zbrajanje\n(-)
             oduzimanje\n(*) mnozenje\n(/) djeljenje\n");
12.    printf("Unesite racunsku operaciju: ");
13.    scanf(" %c", &korisnickiPodaci.operacija);
14.    switch(korisnickiPodaci.operacija){
15.       case '+':
16.          printf("%d + %d = %d", korisnickiPodaci.brojevi[0],
                   korisnickiPodaci.brojevi[1],
                   korisnickiPodaci.brojevi[0]+korisnickiPodaci.brojevi[1]);
17.          break;
18.       case '*':
19.          printf("%d*%d=%d",korisnickiPodaci.brojevi[0],
                   korisnickiPodaci.brojevi[1],
                   korisnickiPodaci.brojevi[0]*korisnickiPodaci.brojevi[1]);
20.          break;
21.       case '-':
22.          printf("%d-%d=%d",korisnickiPodaci.brojevi[0],
                   korisnickiPodaci.brojevi[1],
                   korisnickiPodaci.brojevi[0]-korisnickiPodaci.brojevi[1]);
23.          break;
```

```
24.       case '/':
25.          printf("%d/%d=",korisnickiPodaci.brojevi[0],
                    korisnickiPodaci.brojevi[1]);
26.          if(korisnickiPodaci.brojevi[1] == 0)
27.             printf("Djeljenje s nulom nije dozvoljeno.");
28.          else
29.             printf("%.2f", (float)korisnickiPodaci.brojevi[0]/
                       korisnickiPodaci.brojevi[1]);
30.          break;
31.       default:
32.          printf("Nepostojeca racunska operacija.");
33.    }
34.    getch();}
```

Program 4.11.

U ovome primjeru treba uočiti da se u svakome bloku naredbi vezanom za pojedini `case` nalazi naredba `break`, dakle nema propadanja. Također treba primijetiti da u bloku naredbi za `default` nije potreban `break` jer se nakon izvođenja toga bloka naredbi i tako izlazi iz `switch-case` strukture.

4.2. Praktični zadaci

Slijedi računalni program u koji se unose tri cijela broja te se ispisuje najmanji i najveći uneseni broj. Primjenjuje se naredba `if-else`.

```
 1. #include<stdio.h>a
 2. main()
 3. {
 4.   int a,b,c;
 5.   printf("Unesite brojeve odvojene zarezom: ");
 6.   scanf("%d,%d,%d",&a,&b,&c);
 7.   if(a<b)
 8.   {
 9.      if(a<c)
10.         printf("%d je najmanji, ",a);
11.      else//onda je c<=a
12.         printf("%d je najmanji, ",c);
13.   }
14.   else//tu je b<=a
15.   {
16.      if(b<c)
17.         printf("%d je najmanji, ",b);
18.      else//c<=b
19.         printf("%d je najmanji, ",c);
20.   }
21.   if(a>b)
22.   {
23.      if(a>c)
24.         printf("%d je najveci ",a);
25.      else//onda je c>=a
26.         printf("%d je najveci",c);
27.   }
```

```
28.    else//tu je b>=a
29.    {
30.       if(b>c)
31.          printf("%d je najveci ",b);
32.       else//onda je c>=b
33.          printf("%d je najveci",c);
34.    }
35.    getch();}
```

Program 4.12.

U liniji se 6. vidi skraćeni način unosa više brojeva (vrijednosti varijabli), ali je bitno da se uneseni brojevi razdvoje sa znakom zarez, tj. upravo onim znakom koji je naveden u `scanf` naredbi. Da bi broj `a` bio najmanji, mora vrijediti `a<b` i `a<c`. Ako pak vrijedi da je `a<b`, ali ne i da je `a<c` (tj. vrijedi `c<=a`), onda je broj `c` najmanji. Opisano je u programu implementirano kroz linije koda 7., 9. i 11.

`Else` u liniji 14. izvodi se onda kada logički izraz iz linije 7. nije zadovoljen (pogledati početak i završetak bloka naredbi – odnosno vitičaste zagrade – koji se odnose na logički izraz linije 7.). To znači da u liniji 14. vrijedi logički izraz `b<=a`. U linijama 16. i 18. slijedi provjera je li najmanji broj `b` (za taj slučaj treba vrijediti `b<c`) ili broj `c` (za taj slučaj treba vrijediti `c<=b` – implementirano preko `else` u liniji 18.)

Od linije 21. opisani algoritam za pronalazak najmanjega od tri unesena broja koristi se i za pronalazak najvećeg, ali su sada logički izrazi izmijenjeni. Izlaz programa može se dodatno formatirati tako da se nakon svakoga `printf`-a koji ispisuje najmanji broj stavi posebni znak `\n` pa će ispis najvećeg broja započeti u novome redu.

Slijedi program koji za uneseni kut u stupnjevima ispisuje njegov kvadrant.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  main(){
 4.    int st;
 5.    printf("Unesi kut u stupnjevima: \n");
 6.    scanf("%d",&st);
 7.    if(st>0 && st<90)
 8.      printf("kut od %d stupnjeva je u 1. kvadrantu",st);
 9.    if(st>90 && st<180)
10.      printf("kut od %d stupnjeva je u 2. kvadrantu",st);
11.    if(st>180 && st<270)
12.      printf("kut od %d stupnjeva je u 3. kvadrantu",st);
13.    if(st>270 && st<360)
14.      printf("kut od %d stupnjeva je u 4. kvadrantu",st);
15.    if(st==0 || st ==90 || st ==180 || st ==270)
16.      printf("kut se proteze po osima");
17.    getch();}
```

Program 4.13.

U liniji 6. uneseni kut u stupnjevima sprema se u varijablu `st`, a potom se od linije 7. do linije 16. preko naredbi `if` i logičkih izraza koji se sastoje iz relacijskih (<, >, ==) i logičkih operatora (`&&`, `||`) opisuje što se treba ispisati u ovisnosti o unesenim vrijednostima.

U liniji 15. opisuje se slučaj kada je kut jedan od graničnih jer u linijama 7., 9., 11. i 13. nisu uključene granične vrijednosti.

Ipak, u ovome programu nije određeno što će biti ako se unese stupanj veći od 360. Program će nakon unosa vrijednosti stupnja od npr. 420 jednostavno skočiti na liniju koda 17., tj. naredbu `getch()` (jer niti jedan logički izraz u naredbama if nije zadovoljen).

Da bi se obuhvatile i te mogućnosti unosa i odgovarajućeg ispisa, dovoljno je u svakom logičkom izrazu na mjesto varijable `st`, staviti `st%360` (ostatak pri cjelobrojnome dijeljenju) jer se svaki kut koji je veći od 360, nakon punog okreta nalazi na istoj poziciji kao neki kut koji se dobije kao ostatak dijeljenja s 360. Npr. neka je `st=1034`. Pitanje koje se postavlja je koliko puta se uspije napraviti puni okret tj. 360 stupnjeva unutar te vrijednosti kuta. Rješenje je 1034/360=2 puna kruga i ostatak je 314, pa je stupanj 1034 u istom kvadrantu kao i stupanj 314, tj. broj koji se dobio kao ostatak pri dijeljenju s 360 ili 1034%360=314.

Još postoji jedna moguća vrijednost stupnjeva koja u prikazanome rješenju nije uzeta u obzir – unošenje negativnoga stupnja (npr. -546). Negativni stupnjevi su oni koji se broje u smjeru kazaljke na satu (zovu se negativni jer je u matematici pozitivno brojanje stupnjeva ono obrnuto od kazaljke na satu). Na primjeru za `st=-546`, kut u smjeru kazaljke na satu napravi jedan cijeli okret i dodatnih -186 stupnjeva – jer je -546%360=-186. Kut od -186 stupnjeva nalazi se u 2. kvadrantu, tj. tamo gdje se nalazi i kut od 360-186=174 stupnjeva, ili točnije napisano 360+(-186)=174.

U nastavku je dan dorađeni računalni program koji u obzir uzima negativne kutove i one veće od 360 stupnjeva.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.    int st, pamti;
 5.    printf("unesi kut u stupnjevima!\n");
 6.    scanf("%d",&st);
 7.    pamti =st;
 8.    if(st<0){
 9.       st=360+(st%360);
10.    }
11.    if ((st %360)>0 && (st%360)<90)
12.       printf("kut od %d stupnjeva je u 1. kvadrantu", pamti);
13.    if (st%360>90 && st%360<180)
14.       printf("kut od %d stupnjeva je u 2. kvadrantu", pamti);
15.    if (st%360>180 && st%360<270)
16.       printf("kut od %d stupnjeva je u 3. kvadrantu", pamti);
17.    if (st%360>270 && st%360<360)
```

```
18.       printf("kut od %d stupnjeva je u 4. kvadrantu", pamti);
19.    if( st%360==0 || st%360 ==90 || st%360 ==180 ||
20.       st%360 ==270)
21.       printf("kut se proteze po osima");
22.    getch();}
```

Program 4.14.

U sljedećemu računalnom programu (program 4.15.) izvodi se provjera mogu li tri unesene duljine biti duljine stranica nekoga trokuta te ako mogu, izvodi se ispis vrste trokuta.

Nakon unosa duljina, provjerava se je li zadovoljen uvjet o trokutu (zbroj duljina svakoga para stranica treba biti strogo veći od duljine treće stranice – linija 12.). Ako je taj uvjet zadovoljen, kreće se na kategorizaciju trokuta (jednakokračan, jednakostraničan ili raznostraničan). U liniji 13. provjerava se je li trokut jednakostraničan, potom u liniji 15. je li jednakokračan i nakon toga u liniji 17. je li raznostraničan.

Naredba `else` u liniji 20. odnosi se na naredbu `if` iz linije 12.. Dakle, ona se izvodi onda kada nije zadovoljen uvjet o trokutu, odnosno kada unesene duljine ne mogu biti duljine stranica nekoga trokuta.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int a,b,c;
 5.   printf("\nunesi 1.stranicu");
 6.   scanf("%d",&a);
 7.   printf("unesi 2.stranicu");
 8.   scanf("%d",&b);
 9.   printf("unesi 3.stranicu");
10.   scanf("%d",&c);
11.   printf("\n");
12.   if(a<b+c && b<a+c && c<a+b){
13.      if (a==b && a==c)
14.        printf("trokut je jednakostranican");
15.      else if(a==b || a==c || b==c)
16.        printf("trokut je jednakokracan");
17.      else
18.        printf("trokut je raznostranican");
19.   }
20.   else
21.   printf("ovo nisu stranice trokuta");
22.   getch();}
```

Program 4.15.

Računalni program u nastavku ispisuje jednadžbu pravca kroz dvije točke koja glasi: $y - y1 = \frac{y2-y1}{x2-x1}\ (x - x1)$, te informaciju raste li pravac ili pada. Ako se gleda i eksplicitni oblik pravca $y = a \cdot x + b$, uzimajući u obzir da su *x1*, *x2*, *y1*, *y2* poznate

vrijednosti, dobiva se da je $a = \frac{y2-y1}{x2-x1}$ i $b = -x1 \cdot a + y1$. Tako dobivene vrijednosti koristit će se pri ispisu eksplicitnog oblika jednadžbe pravca.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.    int x1,y1,x2,y2;
 5.    float a,b;
 6.    printf("unesite koordinate prve tocke odvojene zarezom: ");
 7.    scanf("%d,%d",&x1,&y1);
 8.    printf("unesite koordinate druge tocke odvojene zarezom: ");
 9.    scanf("%d,%d",&x2,&y2);
10.    a=(float)(y2-y1)/(x2-x1);
11.    b=-x1*a+y1;
12.    printf("a =%.2f   a b=%.2f\n",a,b);
13.    printf("jednadzba pravca glasi: ");
14.    printf("y=ax+b\n");
15.    if(b<0)
16.       printf("y=%.2f*x%.2f",a,b);
17.    else if(b>0)
18.       printf("y=%.2f*x+%.2f",a,b);
19.    else
20.       printf("y=%.2f*x",a);
21.    if(a<0)
22.       printf("\npravac pada");
23.    else
24.       printf("\npravac raste");
25.    getch();}
```

Program 4.16.

Deklarirane su četiri cjelobrojne varijable za koordinate dviju točaka (linija 4.). Nakon njihova unosa slijedi izračun i pridruživanje vrijednosti varijablama `a` i `b`, odnosno koeficijentima jednadžbe pravca (linije 10. i 11.). Treba primijetiti primjenu naredbe `float` u liniji 10. kako bi se izazvalo realno dijeljenje.

Kako se želi dobiti ispis jednadžbe u eksplicitnom obliku (kako je to navedeno i u ispisu u liniji 14.), formatiranje toga ispisa ovisi o vrijednosti koeficijenta `b`, odnosno o tome vrijedi li `b<0`, `b>0` ili `b==0`. Naime, žele se izbjeći ispisi oblika $y = a \cdot x + -b$ i $y = a \cdot x + 0$. Provjera, raste li pravac ili pada, izvodi se u linijama 21. i 23.

U sljedećemu računalnome programu se izračunava i ispisuje površina trokuta preko poluopsega `s`, a po formuli `p`=$\sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$. Varijabla `q` se deklarira i koristi zbog pojednostavljenja izraza pod korijenom. Program u liniji 10. ispituje vrijednosti varijable `q`. Naime, varijabla `q` ne smije poprimiti negativnu vrijednost jer se iz negativne vrijednosti ne može izračunati drugi korijen u skupu realnih brojeva.

```
 1. #include <stdio.h>
 2. #include <math.h>
 3. #include <conio.h>
 4. main(){
 5.    float a,b,c,s,p,q;
```

```
 6.    printf("Unesite a,b i c sa zarezom: \t");
 7.    scanf("%f,%f,%f",&a,&b,&c);
 8.    s = (( a + b + c ) / 2);
 9.    q = s*(s-a)*(s-b)*(s-c);
10.    if(q<0)
11.    {
12.      printf("\nIzraz pod korijenom mora biti >=0");
13.    }
14.    else
15.    {
16.      p= sqrt(q);
17.      printf("\n%.2f",p);
18.    }
19.    getch();}
```

Program 4.17.

U narednome programu ispisuju se rješenja kvadratne jednadžbe kao realni brojevi. Rješenja kvadratne jednadžbe se računaju po formuli: $x_{1,2=}\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. Imajući na umu da su *a, b, c* koeficijenti, tj. realni brojevi, može se odvojiti eksplicitno rješenje za $x_1=-b/(2\cdot a)+\sqrt{D}/(2\cdot a)$ te $x_2=-b/(2\cdot a)-\sqrt{D}/(2\cdot a)$, gdje je $D=b^2-4\cdot a\cdot c$, diskriminanta.

Rješenja kvadratne jednadžbe ovise o vrijednosti diskriminante te ako je $D<0$ rješenja glase kao i što je napisano. Za $D=0$ rješenja su jednaka, tj. $x1=x2=b/(2\cdot a)$, dok je malo složeniji slučaj kad je $D<0$. Naime u tome slučaju, pod korijenom se nalazi negativan broj. Korijen negativnoga broja ne postoji u skupu realnih brojeva, ali postoji u skupu kompleksnih brojeva, stoga su i rješenja kvadratne jednadžbe kompleksni brojevi. Uzimajući u obzir da je $i=\sqrt{-1}$, te da se svaki kompleksni broj *x* može prikazati kao $x=Re+Im\cdot i$, rješenje kvadratne jednadžbe će se dobiti tako da bude $Re=-b/(2\cdot a)$ a $Im=\sqrt{D}/(2\cdot a)$. Budući da je $D<0$, vrijednost *Im* će se u računalnome programu računati formulom $\sqrt{-D}/(2\cdot a)$.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  #include <math.h>
 4.  main (){
 5.    float a,b,c,x1,x2,D;
 6.    printf("unesi prvi broj:");
 7.    scanf ("%f",&a);
 8.    printf("unesi drugi broj:");
 9.    scanf ("%f",&b);
10.    printf("unesi treci broj:");
11.    scanf ("%f",&c);
12.    D=b*b-4*a*c;
13.    printf("diskriminanta za unesene parametre a b i c
             iznosi %.2f \n",D);
14.    if (D<0){
15.      float r,im;
16.      r=-b/(2*a);
```

```
17.      printf("Realni dio iznosi:%.2f",r);
18.      im=sqrt(-D)/2*a;
19.      printf("\nImaginarni dio iznosi :%.2f\n",im);
20.      printf("x1=%.2f+%.2fi\nx2=%.2f-%.2fi",r,im,r,im);
21.    }
22.    else if (D ==0 ){
23.      x1=(-b)/(2*a);
24.      printf("rjesenja x1=x2=%.2f",x1);
25.    }
26.    else {
27.      x1= (-b)/(2*a) + sqrt(D)/(2*a);
28.      x2= (-b)/(2*a) - sqrt(D)/(2*a);
29.      printf("x1=%.2f\nx2=%.2f",x1,x2);
30.    }
31.    getch();}
```

Program 4.18.

Nakon što su se unijele vrijednosti za `a,b` i `c`, izračunava se `D` te se ispisuje njezina vrijednost linija13. U liniji 14. provjerava se je li `D<0` (to je slučaj kada su rješenja kompleksni brojevi) te se za taj slučaj uvode i deklariraju dvije nove varijable, `re` i `im` koje će predstavljati ranije spomenuti realni i imaginarni dio rješenja. U liniji 20. ispisuju se rješenja koja su kompleksni brojevi. Prilikom ispisa vodi se briga o tome da se broj *i* postavi odmah nakon iznosa varijable `im`.

U liniji 23. izračunava se rješenje kvadratne jednadžbe za slučaj kada je `D==0`, odnosno u liniji 27. za slučaj kada je `D>0`.

U programu koji slijedi traži se upis iznosa računa, odabir način plaćanja (kartica, gotovina ili čekovi). Plaća li se karticom, dobiva se 5% popusta, za plaćanje gotovinom 10% popusta, dok se za plaćanje čekovima treba izračunati iznos jedne rate. Korisnik ima mogućnost odabrati između dvije do šest rata.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.     float iznos_racuna;
 5.     char np;
 6.     printf("UNESITE IZNOS RACUNA:\t");
 7.     scanf("%f",&iznos_racuna);
 8.     if(iznos_racuna > 0){
 9.        printf("\n\nODABERITE NACIN PLACANJA:\n");
10.        printf("'k' (KARTICA) / 'g' (GOTOVINA) / 'c' (CEK)\n\t");
11.        np=getch();
12.        printf("\n\nUNIJELI STE %c \n",np);
13.        if((np == 'c') || (np == 'C') ){
14.           int broj_rata;
15.           float iznos_rate;
16.           printf("\nOdabrali ste: CEK");
17.           printf("\nKupac bira broj rata (2-6)\n\t:");
18.           scanf("%d",&broj_rata);
19.           if ( (broj_rata >= 2 ) && (broj_rata <= 6 ) ){
20.              printf("\nODABRANO JE %d RATE",broj_rata);
21.              iznos_rate = iznos_racuna / broj_rata;
22.              printf("\nIZNOS RACUNA:\t %.2f",iznos_racuna);
```

```
23.             printf("\nIZNOS RACUNA PO 1 RATI:\t %.2f",iznos_rate);
24.          }
25.          else{
26.             printf("\nZAO NAM JE, ALI NE PRIHVACAMO TOLIKI BROJ RATA");
27.             printf("\nPROGRAM SE GASI");
28.          }
29.       }
30.       else if ( (np == 'k') || (np == 'K') ){
31.          float kartica;
32.          kartica = iznos_racuna * 0.95;
33.          printf("\n\nOdabrali ste: KARTICA");
34.          printf("\nKupac ima pravo na 5 posto popusta na iznos racuna");
35.          printf("\nIZNOS RACUNA:\t %.2f",iznos_racuna);
36.          printf("\nIZNOS RACUNA S POPUSTOM:\t %.2f",kartica);
37.       }
38.       else if ( (np == 'g') || (np == 'G') ){
39.          float gotovina;
40.          gotovina = iznos_racuna * 0.90;
41.          printf("\nOdabrali ste: GOTOVINA");
42.          printf("\nKupac ima pravo na 10 posto popusta na iznos
43.                   racuna");
44.          printf("\nIZNOS RACUNA:\t %.2f",iznos_racuna);
45.          printf("\nIZNOS RACUNA S POPUSTOM:\t %.2f",gotovina);
46.       }
47.       else{
48.          printf("\nKRIVI UNOS\n");
49.          printf("PROGRAM SE GASI");
50.       }
51.    }
52.    else{
53.       printf("\nUNOS NOVCA JE NEPRAVILAN");
54.       printf("\nPROGRAM SE GASI");
55.    }
56.    getch();}
```

Program 4.19.

Nakon unosa iznosa računa, provjerava se je li on strogo veći od nula (linija 8.). Ako je, od korisnika se traži unos načina plaćanja – prvo slovo riječi koja opisuje plaćanje (G ili g za gotovinu, k ili K za karticu te c ili C za ček). Varijabla *np* (način plaćanja) je tipa *char*, što znači da može pohraniti točno jedan znak. Slovo se za način plaćanja unosi preko tipkovnice te unesena vrijednost postaje sadržaj varijable `np` preko funkcije `getch()`. Budući da primjena funkcije `getch()` ne izaziva i ispis unesenoga slova, to se u liniji 12. naredbom `printf` izvodi ispis.

U liniji 21. provjerava se je li unesena vrijednost c ili C (to su dvije različite vrijednosti) jer i jedna i druga znače da se želi izvesti plaćanje preko čekova. Ako je unesena vrijednost c ili C, izvodi se odgovarajući blok naredbi u kome se deklariraju dvije nove varijable – jedna za broj rada, a druga za iznos rada. Nakon korisničkoga unosa, provjerava se je li broj rata u dopuštenom intervalu (dvije do šest rata). Ako je, izračunava se iznos rate i on se ispisuje korisniku (linije 21., 22., 23.).

Ukoliko se za način plaćanja unese slovo k ili K, program će ispisati iznos računa s popustom od 5% - u svrhu izračuna iznosa računa s popustom koristi se varijabla `kartica` čija vrijednost je 95% iznosa računa. Slično vrijedi i ako je uneseno plaćanje gotovinom.

`Naredba else` u liniji 47. odnosi se na slučaj kada se za vrijednost varijable `np` unese neki znak koji nije niti jedan od traženih, dok se naredba `else` u liniji 52. odnosi na unos iznosa računa koji je manji ili jedan nuli.

U nastavku je prikazan računalni program sa strukturom koji provjerava odnos pojedinih članova strukture preko naredbe `if-else`.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include <string.h>
 4. typedef struct {
 5.    char mz_ime[20];
 6.    char mz_prezime[30];
 7.    int mz_gr;
 8. } Student;
 9.
10. main(){
11.   Student mz_prvi;
12.   printf("Unesi ime od 1.studenta:\t");
13.   scanf("%s",&mz_prvi.mz_ime);
14.   printf("\nUnesi prezime od studenta:\t");
15.   scanf("%s",&mz_prvi.mz_prezime);
16.   printf("\nUnesi godiste od studenta:\t");
17.   scanf("%d",&mz_prvi.mz_gr);
18.   Student mz_drugi = {"Marko","Markovic",1991};
19.   if(mz_prvi.mz_gr<mz_drugi.mz_gr){
20.      printf("%s je stariji od %s", mz_prvi.mz_ime,mz_drugi.mz_ime);
21.    }
22.   else{
23.      printf("%s je stariji od %s", mz_drugi.mz_ime,mz_prvi.mz_ime);
24.   }
25.   getch();}
```

Program 4.20.

Računalni program (program 4.21) koristi pokazivače za ispis odgovarajuće relacije između dva unesena broja. Uneseni brojevi se spremaju u varijable `a` i `b` te se njihove adrese dodjeljuju pokazivačima `*pa` i `*pb` (linije 9. i 10.). U slučaju da je prvi uneseni broj veći od drugog, tada će pokazivač prvog unesenog broja pokazivati na adresu drugog unesenog broja i obrnuto (zamjena adresa).

U ovome računalnom programu ne dolazi do promjene sadržaja memorijskih lokacija prvotno unesenih brojeva, dakle sadržaj će varijable `a` i varijable `b` biti stalno u jednakim memorijskim ćelijama. Ono što se mijenja je sadržaj pokazivača (ovisi o ispunjavanju uvjeta u liniji 11.). Ako je uvjet ispunjen, pokazivač (varijabla) `pa` će sadržavati adresu varijable `b` te će sadržaj memorijske lokacije na koju pokazuje `pa`, tj `*pa` (operator dereferenciranja) biti ustvari vrijednost varijable `b`. I obrnuto, `*pb` će biti vrijednost varijable `a`. Tako će se i u ispisu u liniji 15. iskoristiti aktualna vrijednost sadržaja memorijske lokacije na koju pokazuje pokazivač `pa`, odnosno `pb`.

```
1. #include<stdio.h>
2. #include<conio.h>
3. main(){
4.   int a,b,*pa,*pb;
5.   printf("Unesi prvi broj:");
```

```
 6.   scanf("%d",&a);
 7.   printf("Unesi drugi broj:");
 8.   scanf("%d",&b);
 9.   pa=&a;
10.   pb=&b;
11.   if(a>b){
12.     pa=&b;
13.     pb=&a;
14.   }
15.   printf("%d <= %d",*pa,*pb);
16.   getch();}
```

Program 4.21.

Sljedeći računalni program ispisuje naziv mjeseca za unesen njegov redni broj (od 1 do 12). Koristi se naredba `switch-case`.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. int mj;
 4. main() {
 5.   printf("Unesite redni broj mjeseca: ");
 6.   scanf("%d",&mj);
 7.   switch(mj) {
 8.      case 1:
 9.        printf("\nsijecanj");
10.        break;
11.      case 2:
12.        printf("\nveljaca");
13.        break;
14.      case 3:
15.        printf("\nozujak");
16.        break;
17.      case 4:
18.        printf("\ntravanj");
19.        break;
20.      case 5:
21.        printf("\nsvibanj");
22.        break;
23.      case 6:
24.        printf("\nlipanj");
25.        break;
26.      case 7:
27.        printf("\nsrpanj");
28.        break;
29.      case 8:
30.        printf("\nkolovoz");
31.        break;
32.      case 9:
33.        printf("\nrujan");
34.        break;
```

```
35.       case 10:
36.          printf("\nlistopad");
37.          break;
38.       case 11:
39.          printf("\nstudeni");
40.          break;
41.       case 12:
42.          printf("\nprosinac");
43.          break;
44.       default:
45.          printf("\nkrivi mjesec");
46.    }
47.    getch();}
```

Program 4.22.

Dakle, kao što je vidljivo u programu, unosi se numerička vrijednost mjeseca te se odmah počinju nizati slučajevi s `case`. U liniji 7. stoji `switch(mj)`, što označava da se u ovisnosti o vrijednosti varijable `mj`, ispisuje naziv mjeseca. Ukupno može biti 12+1 slučaj – 12 slučajeva za mjesece i jedan slučaj (zadnji) kada nije ispunjen niti jedan od prethodnih 12 slučajeva (`caseova`).

U sintaksi naredbe `switch-case`, cijelo razmatranje slučajeva stoji u vitičastim zagradama što tvori tijelo naredbe `switch`, a posebnost sintakse su dvotočke nakon `case neka_vrijednost`, te naredba `break` na kraju svakog `casea`. U slučaju izostavljanja naredbe `break` bile bi izvršene naredbe pod svakim `caseom` niže uključujući `case` u kome je ispunjen uvjet (ova pojava se naziva propadanje). Npr. ako se u `case` za 10 ne bi stavila naredba `break`; a unese se 10, program bi ispisao:

```
listopad
studeni
prosinac
krivi mjesec
```

Sljedeći računalni program omogućuje korisniku unos rednoga broja države od koje treba ispisati glavni grad. Tu je nužno da se odmah po pokretanju programa ispišu kratke upute korisniku (linija 5.).

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  int drzava;
 4.  main() {
 5.    printf("Odaberite drzavu tako da unesete redni broj pored
             njenog imena:\n1. Hrvatska\n2. Njemacka\n3.
             Francuska\n4. Spanjolska\n5. Rusija\n\tunos:");
 6.    scanf("%d",&drzava);
 7.    switch(drzava) {
 8.       case 1:
 9.          printf("\n\nGlavni grad je: Zagreb");
10.          break;
11.       case 2:
12.          printf("\n\nGlavni grad je: Berlin");
```

```
13.          break;
14.       case 3:
15.          printf("\n\nGlavni grad je: Pariz");
16.          break;
17.       case 4:
18.          printf("\n\nGlavni grad je: Madrid");
19.          break;
20.       case 5:
21.          printf("\n\nGlavni grad je: Moskva");
22.          break;
23.       default:
24.          printf("\n\nKrivi odabir!");
25.    }
26.    getch();}
```

Program 4.23.

Program koji je dan u nastavku, u ovisnosti od unesene brojčane ocjene ispisuje njezinu opisnu vrijednost (nedovoljan, dovoljan, ...), a u slučaju da se unese ocjena koja nije u intervalu od jedan do pet, program oglašava zvučni alarm i ispisuje grešku.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.     int ocjena;
 5.     printf("unesite ocjenu: ");
 6.     scanf("%d",&ocjena);
 7.     switch(ocjena){
 8.        case 1:
 9.           printf("Ocjena: nedovoljan(%d)",ocjena);
10.           break;
11.        case 2:
12.           printf("Ocjena : dovoljan(%d )\n", ocjena);
13.           break;
14.        case 3:
15.           printf("Ocjena: dobar(%d) \n", ocjena);
16.           break;
17.        case 4:
18.           printf("Ocjena: vrlo dobar(%d)\n", ocjena);
19.           break;
20.        case 5:
21.           printf("Ocjena:odlican(%d)\n", ocjena);
22.           break;
23.        default: {
24.           printf("\a\a\a\aGreška. Ocjena mora biti izmedju 1 i
                     5\n");
25.           printf("pokrenite ponovo program i unesite valjanu
                      vrijednost ocjene");
26.        }
27.     }
28.     getch();}
```

Program 4.24.

U program se uvelo korištenje `\a` (zvuk u obliku jednog "bip-a" – linija 24.). Za dulji zvučni efekt koristilo se 4 uzastopna `\a` (četiri "bip-a").

Sljedeći računalni program koristi kombinaciju naredbi `switch-case` i `if`. Potrebno je izračunati jakost struje `I`, otpor `R` ili napon `U`, prema izboru. Formule Ohmovog zakona su: `I=U/R`, `U=I·R`, `R =U/I`. Vidljivo je da se u dvije formule pojavljuje razlomak. Zbog toga će biti nužno postaviti uvjet kojim će se provjeriti nazivnik jer nije dopušteno dijeljenje s nulom. Program će omogućiti korisniku izbor između slova I, R ili U te, u ovisnosti o izboru, tražit će unos vrijednosti potrebnih za izračun.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.    char izracun;
 5.    float I,R,U;
 6.    printf("Unesite slovo i ako zelite racunati jakost
 7.    struje:\n");
 8.    printf("Unesite slovo r ako zelite racunati otpor:\n");
 9.    printf("Unesite slovo u ako zelite racunati napon:\n");
10.    izracun=getch();
11.    printf("unijeli ste %c\n",izracun);
12.    switch(izracun){
13.       case 'i' :
14.          printf("Dakle zelite racunati jakost struje,unesite otpor
                    i napon:\n");
15.          printf("Unesite napon:");
16.          scanf("%f",&U);
17.          printf("Unesite otpor:");
18.          scanf("%f",&R);
19.          if(R==0){
20.             printf("ne moze otpor biti 0.");
21.          }
22.          printf("jakost struje I=%.2f",U/R);
23.          break;
24.       case 'r':
25.          printf("Dakle zelite racunati otpor,unesite jakost i
                   napon:\n");
26.          printf("Unesite napon:");
27.          scanf("%f",&U);
28.          printf("Unesite jakost:");
29.          scanf("%f",&I);
30.          if(I==0){
31.             printf("ne moze jakost biti 0.");
32.          }
33.          printf("Otpor R=%.2f",U/I);
34.          break;
35.       case 'u':
36.          printf("Dakle zelite racunati napon,unesite otpor i
                    jakost:\n");
37.          printf("Unesite otpor:");
38.          scanf("%f",&R);
39.          printf("Unesite jakost:");
40.          scanf("%f",&I);
41.          printf("Napon U=%.2f",I*R);
42.          break;
```

```
43.      default:
44.         printf("\a\a\a\aGreska!!!Unesite valjano slovo");
45.         printf("pokrenite ponovo program i unesite valjanu
                    vrijednost za izracun");
46.    }
47.   getch();}
```

Program 4.25.

Nakon što se na početku objasni korisniku način unosa slova za izračun pojedine vrijednosti, kreće naredba `switch-case`. Ako se odabere `'i'`, tada se vodi korisnika prema unosu preostale dvije vrijednosti, otpor i napon (od linija 14. do 18.). Budući da se u formuli vrijednost otpora nalazi u nazivniku, to se nakon unosa provjerava njegova vrijednost te se opisuje što činiti ako je ta vrijednost jednaka nula (ispisuje se poruka korisniku i program završava s izvođenjem).

4.2.1. Zadaci za vježbu

1. Napisati program za konverziju mjernih jedinica za duljinu. Za unesenu decimalnu vrijednost duljine u metrima ispisati vrijednost duljine u milimetrima, centimetrima ili kilometrima na tri decimale i to na način koji će ponuditi mogućnost odabira u koju jedinicu se želi izvesti konverzija. Nakon unosa vrijednosti u metrima, odabire se na sljedeći način:
   - Tipka m → pretvorba u mm
   - Tipka c → pretvorba u cm
   - Tipka k → pretvorba u km

   Po odabiru jedinice izračunati i ispisati vrijednost u novoj jedinici.

2. Napisati program koji će ponuditi mogućnost odabira valute u koju se želi izvesti konverzija. Nakon unosa vrijednosti u kunama, odabire se na sljedeći način:
   - Tipka e → pretvorba u EUR
   - Tipka d → pretvorba u USD
   - Tipka f → pretvorba u CHF
   - Tipka p → pretvorba u GBP

   Pritom se unese vrijednost tečaja za odabranu valutu te ispiše iznos konverzije.

3. Napisati program koji određuje slovnu ocjenu studenta. Učitavaju se tri ocjene studenta (ocjene su u intervalu 0 do 100) i prema srednjoj vrijednosti određuje se konačna slovna ocjena. Pravila dodjele slovne ocjene su:
   - Srednja vrijednost >= 90 → "Ocjena je A"
   - Srednja vrijednost >= 70 i < 90 → "Ocjena je B"
   - Srednja vrijednost >= 50 i < 70 → "Ocjena je C"
   - Srednja vrijednost < 50 → "Ocjena je F"

4. Potrebno je napraviti program koji za učitani broj u intervalu od 1 do 7 ispisuje dan u tjednu koji odgovara tom broju te prethodni i sljedeći dan. Broj 1 predstavlja

nedjelju. Npr. unese se broj tri – treba ispisati `utorak`, prije njega je ponedjeljak, a nakon njega je srijeda.

5. Napisati program koji posjeduje strukturu `Tocka`, s komponentama koordinatama *x* i *y*. Unijeti tri točke i odrediti pripadaju li one pravcu y=2x+1.

6. Napišite program za izračun jedne od veličina u formuli za pravocrtno gibanje: brzine - v, puta - s ili vremena - t. Potrebno je izraditi izbornik (ispisati na monitor) (v) brzina, (s) put, (t) vrijeme. U ovisnosti o unesenome broju s izbornika, unose se preostale dvije veličine. Ako korisnik greškom unese neki drugi znak, neka se ispiše prigodna poruka. Formula je $v = \frac{s}{t}$.

7. Potrebno je napraviti program koji računa trošak najma vozila. U program se unose sljedeći podaci:
   - standardna cijena jednoga dana najma,
   - ukupan broj dana najma (broj dana mora biti veći od 0, ako nije korisniku dati prigodnu poruku).

   U ovisnosti o broju dana najma kupcima se daju popusti po sljedećemu modelu prikazanom u tablici. Potrebno je ispisati cijenu najma bez popusta te s popustom. Do sedam dana nema popusta.

Broj dana najma	**Popust**
8-14	5%
15-30	10%
>30	20%

8. Napišite program koji računa udaljenost točke T(x_0, y_0) od pravca koji je zadan u obliku Ax + By + C = 0. Korisnik unosi koordinate točke T i koeficijente A, B i C. Udaljenost točke T od pravca računa se po formuli $d = -\frac{Ax_0 + By_0 + C}{\pm\sqrt{A^2 + B^2}}$ (ako je izraz u brojniku negativan, u nazivniku ide -, inače u nazivniku ostaje pozitivan izraz). Koeficijente A, B i C čuvati u strukturi naziva `Pravac`.

9. Potrebno je izraditi program kojim se unosi neki broj koji ne smije biti u intervalu od -100 do 100 i mora biti paran. Ako uneseni broj zadovoljava uvjete, izvesti ispis "Broj XX zadovoljava postavljene uvjete" (npr. za uneseni broj 122 ispis je "Broj 122 zadovoljava postavljene uvjete"). Ako uneseni broj ne zadovoljava uvjet, tada izvesti ispis "Broj XX ne zadovoljava postavljene uvjete" (npr. za uneseni broj -65 ispis je "Broj 65 ne zadovoljava postavljene uvjete".

10. Potrebno je izraditi program kojim se definira struktura Student koja se sastoji iz tri podatka: ime, prezime, godina studija. Korisnik treba unijeti podatke za dva studenta. Nakon toga je potrebno provjeriti koji student je na višoj godini i njegove podatke ispisati.

11. Treba napisati program za izračun konačne cijene u ovisnosti o načinu plaćanja kupca. Prvo korisnik unosi cijenu, a nakon toga i znak za način plaćanja: g za gotovinu, c za čekove, k za karticu. Pravila izračuna konačne cijene su:
    - Ako kupac plaća gotovinom, neka se ispiše: "za gotovinu 15% popusta" te se ispiše kolika je konačna cijena.
    - Ako plaća čekovima, a to se dozvoljava samo ako je cijena iznad 3000kn, unosi se broj rata (od 1 do 6) te izračunava iznos jedne rate.
    - Ako plaća karticom, dobije 5% popusta te korisnik bira broj rata: može plaćati od 1 do 12 rata bez kamata. Ispisuje se cijena jedne rate.

5. CIKLIČKA ALGORITAMSKA STRUKTURA U C

Poglavlje prikazuje **cikličku algoritamsku strukturu realiziranu u programskome jeziku C**. Prikazane su naredbe `for`, `while` i `do-while`. Prikazan je opći oblik ovih naredbi te specifični primjeri njihove primjene. Na kraju poglavlja nalazi se niz praktičnih zadataka.

Po završetku ovoga poglavlja čitatelj će moći:

- prepoznati potrebu korištenja cikličke algoritamske strukture
- objasniti sintaksu naredbi `for`, `while` i `do-while`
- pratiti vrijednosti varijabli
- usporediti `while` i `do-while` petlje
- skicirati rješenje danoga problema preko tri petlje
- napisati računalni kod s ugnježđenim petljama
- formulirati primjer zadatka za cikličku algoritamsku strukturu
- koristiti strukturu, polja i nizove znakova unutar petlji
- prepoznati različite algoritamske strukture unutar računalnoga programa
- objasniti svrhu pojedine linije koda
- izračunati rezultat računalnoga programa uz dane početne vrijednosti
- izraditi pseudo kod iz danoga računalnog programa
- izraditi računalni program iz danoga pseudo koda.

5.1. Realizacija cikličke algoritamske strukture u C

Ciklička algoritamska struktura (ponavljanje, iteracija, petlja) je takva struktura koja omogućava ponavljajuće izvođenje istoga bloka naredbi – tijela petlje. Ponavljanje se izvodi dok god je zadovoljen definirani logički uvjet. U svrhu realizacije cikličke algoritamske strukture u programskome jeziku C se koriste naredbe `for`, `while` i `do–while`.

5.1.1. Naredba `for`

Naredbom `for` se realizira ciklička algoritamska struktura s eksplicitnim brojačem. Opći oblik ove naredbe glasi:

```
1.  ...
2.  naredbaX
3.  for(inicijalizacija; logički_izraz; korak)
4.  {
5.     blok_naredbi;
6.  }
7.  naredbaY
8.  ...
```

Nakon naredbe `naredbaX` izvodi se naredba `for`. U njoj se ističu tri elementa međusobno razdvojena s točka-zarez (;). Ti elementi su `inicijalizacija`, `logički_izraz` i `korak`. U inicijalizaciji se izvodi postavljanje početne vrijednosti varijable koja će se dalje koristiti kao brojač. Inicijalizacija se u `for` naredbi izvodi samo jednom odmah na početku. Nakon inicijalizacije se izračunava `logički_izraz`. U logičkom se izrazu provjerava vrijednost brojača. O logičkom izrazu ovisi hoće li se i koliko puta izvesti naredbe u `blok_naredbi`. Ako je logički izraz istinit, izvodi se `blok_naredbi` (odnosno ponavlja se njegovo izvođenje). Ako je logički izraz neistinit ne izvodi se `blok_naredbi`, već naredba `naredbaY`.

U elementu `korak` u `for` naredbi definira se način promjene vrijednosti brojača (vrijednost može rasti ili padati – mogu se koristiti svi algebarski operatori, odnosno bilo kakav algebarski izraz). Do promjene vrijednosti brojača dolazi nakon izvođenja `blok_naredbi`. Nakon izvođenja `blok_naredbi` uvijek slijedi provjera `logičkog_izraza` radi utvrđivanja treba li se ponoviti izvođenje `blok_naredbi`. U toj provjeri se koristi upravo novo postavljena vrijednost brojača. Osim u elementu `korak`, vrijednost brojača se može promijeniti i unutar bloka naredbi.

Sljedeći primjer ispisuje pozdrav "Dobar dan" onoliko puta koliko je to naveo korisnik.

```
 1.  #include <stdio.h>
 2.  main(){
 3.     int brojac, brojPonavljanja;
 4.     printf("Unesite broj ponavljanja pozdrava \"Dobar dan\":");
 5.     scanf("%d",&brojPonavljanja);
```

```
 6.    for(brojac=1; brojac<=brojPonavljanja; brojac++){
 7.       printf("\nDobar dan");
 8.    }
 9.    printf("\nPetlja je zavrsila. Vrijednost varijable brojac je
                sada %d",brojac);
10.    getch();}
```

Program 5.1.

U liniji 5. uneseni broj se sprema u varijablu `brojPonavljanja`. U liniji 7. se `Dobar dan` ispisuje u navodnicima. U tu svrhu je ispred navodnika stavljen poseban znak `\`. To je zato što je navodnik specijalni znak koji se ne može izravno ispisati. U liniji se 6., prije deklarirana varijabla `brojac`, postavlja na vrijednost 1 (element `inicijalizacija`). Potom se provjerava je li vrijednost varijable `brojac` manja ili jednaka od unesenoga broja ponavljanja (element `logički_izraz`). Ako je provjera prošla, izvodi se blok naredbi (linija koda 7.). Povećanje brojača za jedan (element `korak`) dogodit će se nakon izvršenja bloka naredbi jer se povećanje radi postfiksnim operatorom ++ koji ima svojstvo da se prvo izvedu naredbe u bloku naredbi pa se tek onda izvodi povećanje. Dakle, vrijednost varijable `brojac` u prvom izvođenju boka naredbi je 1, zatim 2 itd. Treba uočiti da je vrijednost varijable `brojac` koja će biti ispisana u liniji 9. za jedan veća od unesenoga broja ponavljanja. Npr. ako se unese ponavljanje od 5, tada će po završetku ponavljanja varijabla `brojač` imati vrijednost 6 jer s tom vrijednosti nije zadovoljen logički izraz `brojac<=brojPonavljanja`, pa nema nastavka ponavljanja bloka naredbi.

Sljedeći primjer ispisuje sve parne brojeve između dva unesena broja. Uneseni brojevi se čuvaju u polju s dva elementa.

```
 1.  #include <stdio.h>
 2.  main(){
 3.    int graniceIntervala[2], pomocnaVarijabla, brojac;
 4.    printf("Unesite granicu intervala cijelih brojeva - dva broja
              razdvojena zarezom:");
 5.    scanf("%d, %d",&graniceIntervala[0], &graniceIntervala[1]);
 6.    if (graniceIntervala[0] > graniceIntervala[1]){
 7.       pomocnaVarijabla = graniceIntervala[0];
 8.       graniceIntervala[0] = graniceIntervala[1];
 9.       graniceIntervala[1] = pomocnaVarijabla;
10.    }
11.    printf("Parni brojevi izmedju %d i %d su:",
              graniceIntervala[0], graniceIntervala[1]);
12.    if(graniceIntervala[0]%2 != 0)
13.       graniceIntervala[0] = graniceIntervala[0] + 1;
14.    for(brojac=graniceIntervala[0];brojac<=graniceIntervala[1];
          brojac = brojac + 2) {
15.       if (brojac !=0)
16.          printf("\n%d.", brojac);
17.    }
18.    getch();}
```

Program 5.2.

Nakon unosa granice intervala, razdvojenih zarezom (linija 5.) provjerava se je li uneseni broj za donju granicu manji od drugoga broja koji predstavlja gornju granicu (linija 6.). Ako nije, izvodi se zamjena ovih dvaju brojeva. Cilj je na indeksu 0 polja `graniceIntervala` imati donju granicu intervala, a na indeksu 1 gornju. Ovu izmjenu rade linije koda 7., 8. i 9. Nakon pravilno postavljene donje i gornje granice, izvodi se ispis u liniji 11. Potom se u liniji 12. izvodi korekcija donje granice na način da ako je naveden neparan broj, onda se on povećava za jedan kako bi se dobio sljedeći paran broj (linija 13.). Potom se `for` naredbom ispisuju svi brojevi između korigirane donje granice intervala i gornje granice intervala ali sada s korakom 2 (`brojac = brojac + 2`). Dakle ispisuje se svaki drugi broj, a kako se počelo od parnoga broja (`brojac` ima početnu vrijednost postavljenu na korigiranu donju granicu intervala), to znači da će biti ispisani svi parni brojevi.

Sljedeći primjer prikazuje ispis parnih brojeva, ali od većega prema manjemu.

```
 1.  #include <stdio.h>
 2.  main(){
 3.     int graniceIntervala[2], pomocnaVarijabla, brojac;
 4.     printf("Unesite granicu intervala cijelih brojeva – dva broja
              razdvojena zarezom:");
 5.     scanf("%d, %d",&graniceIntervala[0], &graniceIntervala[1]);
 6.     if (graniceIntervala[0] > graniceIntervala[1]){
 7.        pomocnaVarijabla = graniceIntervala[0];
 8.        graniceIntervala[0] = graniceIntervala[1];
 9.        graniceIntervala[1] = pomocnaVarijabla;
10.     }
11.     printf("Parni brojevi izmedju %d i %d su:",
              graniceIntervala[0], graniceIntervala[1]);
12.     if (graniceIntervala[1]%2 != 0)
13.        graniceIntervala[1] = graniceIntervala[1] - 1;
14.     for(brojac=graniceIntervala[1]; brojac>=graniceIntervala[0];
           brojac = brojac - 2){
15.        if (brojac !=0)
16.           printf("\n%d.", brojac);
17.     }
18.     getch();}
```

Program 5.3.

Treba uočiti da je do promjena došlo u linijama koda 12. i 13. (sada se radi korekcija gornje granice), te u liniji koda 14. (`brojac` se sada inicijalizira na gornju granicu, logičkim izrazom se navodi da `brojac` treba biti veći ili jednak donjoj granici te se u koraku `brojac` sada smanjuje za 2).

Sljedećim se programom od korisnika zahtijeva broj temperaturnih očitanja koja se žele unijeti, a kako bi se na kraju izračunala njihova prosječna vrijednost.

```
 1.  #include <stdio.h>
 2.  main(){
 3.     int brojTemperaturnihOcitanja, brojac;
 4.     float suma, prosjecnaTemperatura, temperaturnoOcitanje;
 5.     for (brojac = 1; brojac == 1; brojac++) {
 6.        printf("\nUnesite broj temperaturnih ocitanja: ");
```

```
 7.       scanf("%d",&brojTemperaturnihOcitanja);
 8.       if (brojTemperaturnihOcitanja <= 0){
 9.          printf("Broj temperaturnih ocitanja treba biti veci od
                    nula.");
10.          brojac = brojac - 1;
11.       }
12.    }
13.    suma = 0;
14.    for (brojac = 1; brojac <= brojTemperaturnihOcitanja;
              brojac++){
15.       printf ("Unesite %d. temperaturno ocitanje: ", brojac);
16.       scanf("%f", &temperaturnoOcitanje);
17.       suma = suma + temperaturnoOcitanje;
18.    }
19.    prosjecnaTemperatura = suma / brojTemperaturnihOcitanja;
20.    printf ("Prosjecno temperaturno ocitanje iznosi: %.2f",
                prosjecnaTemperatura);
21.    getch();}
```

Program 5.4.

U ovome primjeru treba uočiti primjenu naredbe `for` kako bi se implementiralo ponavljanje pitanja na koje je korisnik pogrešno odgovorio (prva `for` naredba). Ovdje se vrijednost `brojaca` mijenja unutar bloka naredbi (linija koda 10.). Druga `for` naredba se koristi za unos temperaturnih očitanja i njihova sumiranja a kako bi se poslije mogla izračunati prosječna vrijednost očitanja (linija koda 19.). Treba uočiti inicijalizaciju varijable `suma` na nulu (linija koda 13.) koja je potrebna radi točne akumulacije suma temperaturnih očitanja.

Naredba `for` se često koristi pri radu s poljima. U sljedećemu se primjeru u polje unosi zapis koji se sastoji iz prezimena studenta i ostvarenoga postotka ocjene. Nakon unosa u dodatnu se komponentu zapisa za svakoga studenta unosi slovna ocjena. Potom se svi uneseni podaci ispisuju u formi tablice. Broj unosa određuje korisnik.

```
 1.  #include <stdio.h>
 2.  typedef struct {
 3.     char prezimeStudenta[30+1];
 4.     float postotakOcjene;
 5.     char slovnaOcjena[3];} StudentskiRezultat;
 6.  main(){
 7.    int brojac, brojStudentskihRezultata;
 8.    for (brojac = 1; brojac == 1; brojac++) {
 9.       printf("\nUnesite broj unosa studentskih rezultata: ");
10.       scanf("%d",& brojStudentskihRezultata);
11.       if (brojStudentskihRezultata <= 0){
12.          printf("Broj unosa studentskih rezultata treba biti veci
                     od nula.");
13.          brojac = brojac - 1;
14.       }
15.    }
16.    StudentskiRezultat rezultati[brojStudentskihRezultata];
17.    for(brojac=0;brojac<brojStudentskihRezultata;brojac++){
18.       printf ("%d. ZAPIS:", brojac+1);
19.       printf ("\nUnesite prezime studenta: ");
```

```
20.       scanf("%s", rezultati[brojac].prezimeStudenta);
21.       printf ("\nUnesite postotak ocjene: ");
22.       scanf("%f", &rezultati[brojac].postotakOcjene);
23.    }
24.    for(brojac=0;brojac<brojStudentskihRezultata;brojac++){
25.       if (rezultati[brojac].postotakOcjene >= 0){
26.          if (rezultati[brojac].postotakOcjene <= 29.9){
27.             rezultati[brojac].slovnaOcjena[0] = 'F';
28.             rezultati[brojac].slovnaOcjena[1] = '\0';
29.          }
30.          else if (rezultati[brojac].postotakOcjene <= 39.9) {
31.             rezultati[brojac].slovnaOcjena[0] = 'F';
32.             rezultati[brojac].slovnaOcjena[1] = 'X';
33.             rezultati[brojac].slovnaOcjena[2] = '\0';
34.          }
35.          else if (rezultati[brojac].postotakOcjene <= 49.9) {
36.             rezultati[brojac].slovnaOcjena[0] = 'E';
37.             rezultati[brojac].slovnaOcjena[1] = '\0';
38.          }
39.          else if (rezultati[brojac].postotakOcjene <= 59.9) {
40.             rezultati[brojac].slovnaOcjena[0] = 'D';
41.             rezultati[brojac].slovnaOcjena[1] = '\0';
42.          }
43.          else if (rezultati[brojac].postotakOcjene <= 69.9) {
44.             rezultati[brojac].slovnaOcjena[0] = 'C';
45.             rezultati[brojac].slovnaOcjena[1] = '\0';
46.          }
47.          else if (rezultati[brojac].postotakOcjene <= 79.9) {
48.             rezultati[brojac].slovnaOcjena[0] = 'B';
49.             rezultati[brojac].slovnaOcjena[1] = '\0';
50.          }
51.          else if (rezultati[brojac].postotakOcjene <= 100) {
52.             rezultati[brojac].slovnaOcjena[0] = 'A';
53.             rezultati[brojac].slovnaOcjena[1] = '\0';
54.          }
55.          else{
56.             rezultati[brojac].slovnaOcjena[0] = '-';
57.             rezultati[brojac].slovnaOcjena[1] = '\0';
58.          }
59.       }
60.       else {
61.          rezultati[brojac].slovnaOcjena[0] = '-';
62.          rezultati[brojac].slovnaOcjena[1] = '\0';
63.       }
64.    }
65.    printf ("\n\nISPIS ZAPISA:\n");
66.    for (brojac = 0; brojac < brojStudentskihRezultata;
           brojac++){
67.       printf("\n%s\t%.2f\t%s",
                  rezultati[brojac].prezimeStudenta,
                 rezultati[brojac].postotakOcjene,
                 rezultati[brojac].slovnaOcjena);
68.    }
69.    getch();}
```

Program 5.5.

U ovome primjeru postoje četiri cikličke strukture (linije koda 8., 17., 24., 66.). Prva `for` petlja se koristi kako bi se osiguralo da korisnik unese ispravni broj studentskih rezultata. Druga `for` petlja se koristi kako bi se unijeli podaci u polje. Ovdje treba obratiti pažnju na činjenicu da je `brojac` inicijaliziran na nulu. To je zato što se vrijednost brojača u pripadajućemu bloku naredbi koristi i kao indeks elemenata u polju, a indeksi elemenata polja počinju s nula (linija koda 20. ili 22.). Slično je i u preostalim `for` petljama. U trećoj petlji se ažurira komponenta `slovnaOcjena` na odgovarajuću ocjenu. Treba obratiti pažnju da je `slovnaOcjena` polje znakova od 3 elementa. Razlog je što postoji ocjena `"FX"` koja se sastoji iz dva znaka. Uvijek zadnji znak u nizu znakova treba biti rezerviran za null znak `"\0"` koji govori da je tu kraj niza znakova. Alternativa u prikazanom postavljanju niza znakova je uporaba funkcije `strcpy`. Tada bi pridruživanje izgledalo kako slijedi:

npr. umjesto:
```
rezultati[brojac].slovnaOcjena[0] = 'F';
rezultati[brojac].slovnaOcjena[1] = '\0';
```

bilo bi:
```
strcpy(rezultati[brojac].slovnaOcjena, "F");
```

Valja primijetiti kako je u liniji koda 16. deklarirano polje koje će sadržavati upravo onoliko elemenata koliko je to naveo korisnik svojim unosom, a koji se čuva u varijabli `brojStudentskihRezultata`.

Naredba `for` se može koristiti i u radu s nizom znakova. Sljedeći primjer prikazuje program koji ispisuje sve samoglasnike koji se pojavljuju u riječi koju korisnik unese te ispisuje unesenu riječ bez samoglasnika.

```
 1. #include <stdio.h>
 2. main(){
 3.    char rijec[100];
 4.    int brojac, brojSamoglasnika;
 5.    printf ("Unesite neku rijec: ");
 6.    scanf("%s", rijec);
 7.    brojSamoglasnika = 0;
 8.    for (brojac = 0; brojac < strlen(rijec); brojac++){
 9.       if (rijec[brojac] == 'a' || rijec[brojac] == 'A' ||
           rijec[brojac] == 'e' || rijec[brojac] == 'E' ||
           rijec[brojac] == 'i' || rijec[brojac] == 'I' ||
           rijec[brojac] == 'o' || rijec[brojac] == 'O' ||
           rijec[brojac] == 'u' || rijec[brojac] == 'U')
10.         brojSamoglasnika = brojSamoglasnika + 1;
11.    }
12.    printf("\nBroj samoglasnika u rijeci \"%s\" je %d", rijec,
             brojSamoglasnika);
13.    printf("\n\nIspis rijeci \"%s\" bez samoglasnika: ");
14.    for (brojac = 0; brojac < strlen(rijec); brojac++){
15.       if (!(rijec[brojac] == 'a' || rijec[brojac] == 'A' ||
             rijec[brojac] == 'e' || rijec[brojac] == 'E' ||
             rijec[brojac] == 'i' || rijec[brojac] == 'I' ||
             rijec[brojac] == 'o' || rijec[brojac] == 'O' ||
```

```
            rijec[brojac] == 'u' || rijec[brojac] == 'U'))
16.       printf("%c", rijec[brojac]);
17.    }
18.   getch();}
```

Program 5.6.

Ispis riječi bez samoglasnika izvodi se tako da se ispisuje znak po znak riječi koji nije samoglasnik. To je implementirano u linijama koda 15. i 16.. Funkcija `strlen` se koristi kako bi se otkrilo koliko znakova ima unesena riječ, odnosno do koje vrijednosti treba ići brojač u `for` petlji.

Neka se treba izraditi program kojim se provjerava ispravnost unos OIB-a. Neka je unos OIB-a ispravan ako je duljine 11 i ako u sebi ima samo znamenke. Slijedi program koji ispunjava navedeni zahtjev:

```
 1. #include <stdio.h>
 2. main(){
 3.    char OIB[11+1];
 4.    char ispravanOIB;
 5.    int brojac;
 6.    printf ("Unesite OIB za provjeru: ");
 7.    scanf("%s", OIB);
 8.    ispravanOIB = '1';
 9.    if (strlen(OIB) != 11){
10.       printf ("OIB treba biti duljine 11 znamenaka.");
11.       ispravanOIB = '0';
12.    }
13.    else{
14.       for (brojac = 0; brojac < strlen(OIB); brojac++){
15.          if (!(OIB[brojac] == '0' || OIB[brojac] == '1' ||
                 OIB[brojac] == '2' || OIB[brojac] == '3' ||
                 OIB[brojac] == '4' || OIB[brojac] == '5' ||
                 OIB[brojac] == '6' || OIB[brojac] == '7' ||
                 OIB[brojac] == '8' || OIB[brojac] == '9')){
16.             printf("OIB se sastoji samo iz znamenaka.");
17.             ispravanOIB = '0';
18.             break;
19.          }
20.       }
21.    }
22.    if (ispravanOIB == '1')
23.       printf("\nOIB \"%s\" je ispravan.", OIB);
24.    getch();}
```

Program 5.7.

U liniji koda 3. deklarira se niz znakova `OIB` koji se sastoji iz 11 + 1 znak (11 za OIB i 1 koji je potreban za poseban znak kraja niza). Deklarirana je i varijabla `ispravanOIB` koja će poslužiti kao indikator je li `OIB` prošao provjeru ili ne, a kako bi se na kraju programa mogla ispisati poruka (linija koda 22. i 23.). Nakon unosa `OIB-a` (linija 7.) provjerava se je li njegova duljina ispravna (linija koda 9.). Ako nije, ispisuje se odgovarajuća poruka te se u varijablu `ispravanOIB` sprema podatak 0 koji označava da je `OIB` neispravan. Ako je `OIB` prošao kontrolu duljine,

onda se kroz `for` petlju provjerava pojavljuju li se u njemu isključivo znamenke. Ako se pronađe da u `OIB-u` postoji znak koji nije znamenka (linija koda 15.) ispisuje se odgovarajuća poruka (linija koda 16.), u varijablu `ispravanOIB` se sprema informacija da je `OIB` neispravan i prekida se daljnja analiza znakova budući da ona dalje nije potreba. Ovo se postiže prekidom izvođenja cikličke algoritamske strukture. U tu svrhu se koristi naredba `break` (linija koda 18.). Ona spada u naredbe koje implementiraju algoritamsku strukturu bezuvjetnoga skoka i ona će biti objašnjena u posebnome poglavlju.

Naredbom `for` moguće je izraziti i cikličku algoritamsku strukturu koja će se izvoditi neograničen broj puta. Ovo se postiže na način da se izradi takav logički izraz koji je uvijek zadovoljen. U nastavku slijedi primjer programa u kome se javlja beskonačna petlja – program beskonačno mnogo puta ispisuje pozdrav "Dobar dan":

```
1.  #include <stdio.h>
2.  main(){
3.     int brojac;
4.     for(brojac=1; brojac==1; brojac = brojac + 0){
5.     printf("\nDobar dan");
6.     }
7.     getch();}
```

Program 5.8..

U `for` naredbi postignut je uvijek zadovoljen logički izraz (`brojac==1`) tako što je brojač kod svake iteracije ostao nepromijenjen (`brojac=brojac + 0`), odnosno brojač uvijek ima vrijednost 1. Ovakve beskonačne petlje nisu uvijek znak pogreške jer postoje programi u kojima je beskonačna petlja potrebna (npr. programi koji trebaju neprestano osluškivati događanje na svojoj periferiji – jedan od njih je i sami operacijski sustav računala).

5.1.2. Naredba `while`

Naredbom `while` se realizira ciklička algoritamska struktura s izlazom na vrhu. Opći oblik ove naredbe glasi:

```
1.  ...
2.  naredbaX
3.  while(logički_izraz)
4.  {
5.     blok_naredbi;
6.  }
7.  naredbaY
8.  ...
```

Nakon naredbe `naredbaX` izvodi se naredba `while`, odnosno izračunava se `logički_izraz`. Ako mu je vrijednost istina, izvode se naredbe u `blok_naredbi`. Po završetku izvođenja bloka naredbi, ponovno se izračunava `logički_izraz`. U slučaju da je `logički_izraz` neistinit izvodi se

naredbaY. Utjecaj na logički_izraz izvodi se tako da se u tijelu petlje događaju izmjene vrijednosti varijabli koje grade logički izraz.

Može se uočiti, slično kao i kod naredbe for, da logički izraz prvo mora biti zadovoljen kako bi se krenulo u izvođenje naredbi u bok_naredbi. Zato se ovakva ciklička algoritamska struktura i naziva struktura s izlazom na vrhu. U biti, naredba for se može izvesti pomoću naredbe while što je i prikazano u nastavku.

Opći oblik naredbe for:

```
1.  ...
2.  naredbaX
3.  for(inicijalizacija; logički_izraz; korak)
4.  {
5.     blok_naredbi;
6.  }
7.  naredbaY
8.  ...
```

Ekvivalentni oblik s naredbom while:

```
 1.  ...
 2.  naredbaX
 3.  inicijalizacija
 4.  while(logički_izraz)
 5.  {
 6.    blok_naredbi;
 7.    korak
 8.  }
 9.  naredbaY
10.  ...
```

Primjera radi, neka se treba izraditi program kojim se izračunava suma svih prirodnih brojeva do nekoga unesenog broja. Program bi primjenom naredbe for izgledao kako slijedi.

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int prirodniBroj, brojac, suma = 0;
 4.    printf("Unesite neki prirodni broj: ");
 5.    scanf("%d",&prirodniBroj);
 6.    if (prirodniBroj>0){
 7.       for (brojac=1; brojac<=prirodniBroj; brojac++){
 8.          suma = suma + brojac;
 9.       }
10.       printf("Suma prirodnih brojeva do broja %d iznosi %d.",
                  prirodniBroj, suma);
11.    }
12.    else{
13.       printf("Broj %d nije prirodni broj.");
14.    }
15.    getch();}
```

Program 5.9.

Slijedi isti program primjenom naredbe `while`.

```
 1. #include<stdio.h>
 2. main(){
 3.    int prirodniBroj, brojac, suma = 0;
 4.    printf("Unesite neki prirodni broj: ");
 5.    scanf("%d",&prirodniBroj);
 6.    if (prirodniBroj>0){
 7.       brojac=1;
 8.       while(brojac<=prirodniBroj){
 9.          suma = suma + brojac;
10.          brojac++;
11.       }
12.       printf("Suma prirodnih brojeva do broja %d iznosi %d.",
                  prirodniBroj, suma);
13.    }
14.    else{
15.       printf("Broj %d nije prirodni broj.");
16.    }
17.    getch();}
```

Program 5.10.

U oba se primjera provjerava je li uneseni broj prirodni broj (linija koda 6.) Ako je, izvodi se sumiranje prirodnih brojeva do unesenoga broja uz primjenu varijable `suma` (u prvome primjeru linija koda 8., odnosno u drugom linija koda 9.).

Naredni primjer ispisuje slovo koje je kodirano unesenim ASCII kodom. Slova su kodirana brojevima od 65 do 90 (velika slova) i od 97 do 122 (mala slova). Ako se unese broj koji ne predstavlja slovo, program ispisuje poruku i završava s izvođenjem.

```
 1. #include<stdio.h>
 2. main(){
 3.    char ASCIIkod;
 4.    printf("Unesite ASCII kod (0-255): ");
 5.    scanf("%d",&ASCIIkod);
 6.    while((ASCIIkod>=65 && ASCIIkod<=90) ||
             (ASCIIkod>=97 && ASCIIkod<=122))
 7.    {
 8.       printf("Slovo pod ASCII kodom %d je %c\n", ASCIIkod,
                ASCIIkod);
 9.       printf("Unesite ASCII kod (0-255): ");
10.       scanf("%d",&ASCIIkod);
11.    }
12.    printf("\nProgram zavrsava jer ASCII kod %d ne predstavlja
              slovo.", ASCIIkod);
13.    getch();}
```

Program 5.11.

Nakon prvog unosa ASCII koda (linija koda 5.) izvodi se provjera logičkoga izraza u `while` naredbi (linija koda 6.). Ako je logički izraz istina, izvršavaju se naredbe u pripadajućemu bloku naredbi. Vidljivo je da se vrijednost koja utječe na logički izraz mijenja u bloku naredbe (linija koda 10.). Kada logički izraz više nije zadovoljen – ima vrijednost neistina – izvodi se linija koda 12.

Sljedeći primjer povećava, odnosno smanjuje prikazani broj za jedan primjenom tipki + i - na tipkovnici. Za očitanje pritisnutoga znaka koristit će se naredba `getch()` koja vraća ASCII kod pritisnutoga znaka.

```
 1. #include<stdio.h>
 2. main(){
 3.    char pritisnutaTipka;
 4.    int brojac = 0;
 5.    pritisnutaTipka = 0;
 6.    printf("Sa + i - povecava se odnosno smanjuje prikazani
              broj.\nIz programa se izlazi ako se pritisne tipka
               ESC.\n\n");
 7.    while(pritisnutaTipka != 27) {//ASCII kod za ESC
 8.       pritisnutaTipka = getch();
 9.       system("cls");
10.       printf("Sa + i - povećava se odnosno smanjuje prikazani
                  broj.\nIz programa se izlazi ako se pritisne
                  tipka ESC.\n\n");
11.       if (pritisnutaTipka == 43) //ASCII kod za +
12.          brojac++;
13.       else if (pritisnutaTipka == 45) //ASCII kod za -
14.          brojac--;
15.        printf("%d", brojac);
16.    }
17.    printf("\nProgram zavrsava jer ste pritisnuli ESC.");
18.    getch();}
```

Program 5.12.

Nakon deklaracija varijabli i ispisa poruke provjerava se logički izraz (linija koda 7.) kojim se provjerava je li korisnik pritisnuo tipku ESC (ASCII kod 27). Ako nije, onda se očekuje pritisak neke tipke (linija koda 8.). Nakon pritiska tipke, briše se cijeli ekran konzole (linija koda 9.), ponovno se ispisuje poruka (jer je sada ekran obrisan), provjerava se koja je tipka pritisnuta (linije koda 11.. ili 12.) i u ovisnosti od ASCII koda povećava se ili smanjuje `brojac` za jedan. Potom se vrijednost `brojaca` ispisuje (linija koda 15.). U slučaju da je pritisnuta tipka ESC, dakle logički izraz je neistinit, bit će izvedena linija koda 17., odnosno ispis poruke.

5.1.3. Naredba `do-while`

Naredbom `do-while` realizira se ciklička algoritamska struktura s izlazom na dnu. Opći oblik ove naredbe glasi:

```
1. ...
2. naredbaX
3. do {
4.    blok_naredbi;
5. }
6. while(logički_izraz);
7. naredbaY
8. ...
```

Nakon izvođenja naredbe `naredbaX`, odmah se ulazi u `blok_naredbi` (dakle, bez obzira je li `logički_izraz` istinit ili ne). Ovo znači da će tijelo petlje `do-while`, odnosno pripadajući blok naredbi, biti sigurno izveden jednom. I ovdje se, slično kao i pri naredbi `while`, u bloku naredbi mijenjaju vrijednosti varijabli koje sudjeluju u logičkom izrazu. Nakon što se naredbe iz bloka naredbi izvedu, izračunava se logički izraz. Ako je istinit, ponavlja se izvođenje bloka naredbi. Ako je neistinit, izvodi se `naredbaY`.

Sljedeći primjer prikazuje primjenu `do-while` naredbe pri provjeri unosa korisničkih podataka. Program ispisuje naziv mjeseca čiji je redni broj unio korisnik. Ako uneseni broj nije valjan (nije u intervalu 1 do 12) pušta se zvučni signal, ispisuje se poruka i od korisnika se ponovno traži unos.

```
 1. #include<stdio.h>
 2. main(){
 3.    int redniBrojMjeseca;
 4.    do{
 5.       printf("\nUnesite redni broj mjeseca: ");
 6.       scanf("%d", &redniBrojMjeseca);
 7.       if (!(redniBrojMjeseca >=1 && redniBrojMjeseca <= 12))
 8.          printf ("\n\aRedni broj mjeseca treba biti u intervalu od
                    1 do 12.");
 9.       else
10.          break;
11.    }
12.    while(1==1);
13.       printf("\n\nMjesec pod rednim brojem %d se zove ",
                 redniBrojMjeseca);
14.    switch(redniBrojMjeseca){
15.       case 1 :
16.          printf("Sijecanj.");
17.          break;
18.       case 2 :
19.          printf("Veljaca.");
20.          break;
21.       case 3 :
22.          printf("Ozujak.");
23.          break;
24.       case 4 :
25.          printf("Travanj.");
26.          break;
27.       case 5 :
28.          printf("Svibanj.");
29.          break;
30.       case 6 :
31.          printf("Lipanj.");
32.          break;
33.       case 7 :
34.          printf("Srpanj.");
35.          break;
36.       case 8 :
37.          printf("Kolovoz.");
38.          break;
```

```
39.       case 9 :
40.          printf("Rujan.");
41.          break;
42.       case 10 :
43.          printf("Listopad.");
44.          break;
45.       case 11 :
46.          printf("Studeni.");
47.          break;
48.       case 12 :
49.          printf("Prosinac.");
50.          break;
51.    }
52.    getch();}
```

Program 5.13.

Ovdje treba uočiti da je za logički izraz izabran uvjet koji je uvijek zadovoljen – dakle, ovdje se radi o beskonačnoj petlji (linija koda 12.). Petlja se ponavlja sve dok korisnik unosi neispravni podatak (linija koda 7. i 8.) – uočiti znak `\a` unutar naredbe `printf` koji izaziva puštanje zvučnoga signala. Kada unese ispravni podatak, događa se naredba `break`, dakle prekid izvođenja petlje (linija koda 10.). Potom se izvodi naredba `switch-case` koja ispisuje naziv mjeseca.

Primjer koji slijedi koristi `do-while` petlju kako bi se provjerio korisnički unos OIB-a. Dopušta se samo unos znamenki i prihvaćanje jedanaestoznamenkastog OIB-a.

```
 1. #include<stdio.h>
 2. main(){
 3.    char OIB [11+1];
 4.    char znamenka;
 5.    int brojac = 0;
 6.    OIB[0] = '\0';
 7.    printf("\nUnesite neki OIB: ");
 8.    do{
 9.       znamenka = getch();
10.       if (znamenka >= 48 && znamenka <= 57){
11.          if (brojac <= 10){
12.             if (brojac < 0) brojac = 0;
13.             OIB[brojac] = znamenka;
14.             OIB[brojac+1] = '\0';
15.             brojac ++;
16.          }
17.          else{
18.             printf("\a");
19.             continue;
20.          }
21.       }
22.       else if (znamenka == 8) {
23.          if (brojac > 0){
24.             OIB[brojac-1] = '\0';
25.             brojac --;
```

```
26.         }
27.         else{
28.           printf("\a");
29.           continue;
30.         }
31.       }
32.       else if (znamenka == 13 && strlen(OIB) == 11){
33.         break;
34.       }
35.       else{
36.         printf("\a");
37.         continue;
38.       }
39.       system("cls");
40.       printf ("Unesite neki OIB: %s", OIB);
41.     }
42.     while(1==1);
43.     printf ("\n\nUneseni OIB je: %s", OIB);
44.     getch();}
```

Program 5.14.

Prikazani program koristi `do-while` petlju kako bi ograničio korisnički unos. Niz `if-else if-else` se koristi kako bi se svi mogući slučajevi korisničkog unosa uzeli u obzir i temeljem njih izvele određene akcije. Akcije su: dodaj novu znamenku u `OIB` (linija koda 13. i 14.), izbriši zadnju znamenku iz `OIB-a` (linija koda 24.), završi s unosom (linija koda 33.) te daj zvučni signal (linije koda 18., 28. i 36.). Ovdje treba obratiti pažnju na naredbu skoka `continue` koja preskače sve naredne linije koda do linije 42. kojom se provjerava uvjet za ponovno izvođenje petlje.

Dodatno se u ovaj program može uključiti i provjera zadnje kontrolne znamenke koja se računa prema sljedećem algoritmu (preuzeto sa www.regos.hr):

1. prva znamenka zbroji se s brojem 10
2. dobiveni se zbroj cjelobrojno (s ostatkom) podijeli brojem 10; ako je dobiveni ostatak 0 zamijeni se brojem 10 (ovaj broj je tzv. međuostatak)
3. dobiveni se međuostatak pomnoži brojem 2
4. dobiveni se umnožak cjelobrojno (s ostatkom) podijeli brojem 11; ovaj ostatak matematički nikako ne može biti 0 jer je rezultat prethodnoga koraka uvijek parni broj
5. sljedeća se znamenka zbroji s ostatkom u prethodnome koraku
6. ponavljaju se koraci 2, 3, 4 i 5 dok se ne potroše sve znamenke
7. razlika između broja 11 i ostatka u zadnjemu koraku je kontrolna znamenka; ako je ostatak 1 kontrolna znamenka je 0 (11-1=10, a 10 ima dvije znamenke)

Primjer za OIB (provjera ispravnosti OIB-a : 69435151530)

11-1=10; kontrolna znamenka je 0

Korak	Znamenka	Zbroj broja iz stupca b s ostatkom iz prethodnog koraka (brojem iz stupca f) ili s brojem 10 za korak 1	Ostatak od dijeljenja broja iz stupca c s 10 – ako je 0 stavi se 10	Umnožak broja iz stupca d s 2	Ostatak od dijeljenja broja iz stupca e s 11
a	b	c	d	e	f
1	6	16	6	12	1
2	9	10	10	20	9
3	4	13	3	6	6
4	3	9	9	18	7
5	5	12	2	4	4
6	1	5	5	10	10
7	5	15	5	10	10
8	1	11	1	2	2
9	5	7	7	14	3
10	3	6	6	12	1

Algoritam 5.1.

5.2. Praktični zadaci

Dan je računalni program koji ispisuje brojeve između dva unesena cijela broja. Pretpostavlja se da će korisnik unijeti prvi broj koji je manji od drugog. Ispis brojeva između dva unesena postigao se tako da je postavljen korak petlje na `x++`, a uneseni brojevi su uključeni u ispis.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.   int x, broj1, broj2;
 5.   printf("Upisi prvi broj: ");
 6.   scanf("%d", &broj1);
 7.   printf("Upisi drugi broj: ");
 8.   scanf("%d", &broj2);
 9.   for(x=broj1;x<=broj2;x++) {
10.     printf("\n%d",x);
11.   }
12.   getch();}
```

Program 5.15.

Prethodni računalni program se može prepraviti tako da u obzir uzme veličinu unesenih brojeva

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.   int x, broj1, broj2;
 5.   printf("Upisi prvi broj: ");
```

```
 6.    scanf("%d", &broj1);
 7.    printf("Upisi drugi broj: ");
 8.    scanf("%d", &broj2);
 9.    if(broj1<=broj2){
10.       for(x=broj1;x<=broj2;x++) {
11.          printf("\n%d",x);
12.       }
13.    }
14.    else{
15.       for(x=broj2;x<=broj1;x++){
16.          printf("\n%d",x);
17.       }
18.    }
19.    getch();}
```

Program 5.16.

Uvođenjem naredbe `if–else` riješena je situacija u kojoj korisnik za prvi broj unese vrijednost je veću od drugoga broja. Ključni dio je u `for` petlji `brojac` inicijalizirati na ispravnu vrijednost (vidjeti linije 10. i 15.).

Slijedi računalni program (program 5.17.) koji ispisuje sve brojeve između dva unesena broja, djeljive s 3 i 5. Pretpostavlja se da je prvi uneseni broj manji od drugog. U liniji se 10. provjerava je li vrijednost koja se nalazi u varijabli `x` istodobno djeljiva s 3 i 5. Ako je, onda će se vrijednost varijable `x` ispisati.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int x, Broj1, Broj2, Pom;
 5.    printf("Upisi prvi broj: ");
 6.    scanf("%d", &Broj1);
 7.    printf("Upisi drugi broj: ");
 8.    scanf("%d", &Broj2);
 9.    for(x=Broj1;x<=Broj2;x++) {
10.       if (x % 3 == 0  &&  x % 5 == 0){
11.          printf("\n%d",x);
12.       }
13.    }
14.    getch();}
```

Program 5.17.

Program u nastavku traži od korisnika unos deset vrijednosti (koristi se varijabla `br`). Pri svakome unosu se vrijednost pribraja varijabli `suma`. Nakon unosa deset vrijednosti, ispisuje se njihov zbroj i njihov prosjek (aritmetička sredina).

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  main(){
 4.    int br,x,suma=0;
 5.    float ars;
```

```
 6.    for(x=1;x<=10;x++){
 7.      printf("Upisi %d. broj ",x);
 8.      scanf("%d",&br);
 9.      suma=suma+br;
10.    }
11.    ars=(float)suma/10;
12.    printf("%d\t\t%.2f",suma,ars);
13.    getch();}
```

Program 5.18.

U programu se uvodi varijabla `suma` te se odmah pri deklaraciji postavlja na vrijednost 0. To se mora napraviti zbog linije 9., gdje se varijabli `suma` pridružuje vrijednost koja je izračunata tako da se aktualna vrijednost varijable `suma` poveća za unesenu vrijednost varijable `br`. A aktualna vrijednost varijable `suma` u prvome prolasku (ciklusu) petlje bila bi *SMEĆE* da se nije inicijalizirala na 0. Linija 9. se može skraćeno pisati `suma+=br;`. Dakle, svakim novim prolaskom kroz tijelo petlje dobiva se nova vrijednost varijable `suma` (staroj vrijednosti se dodaje nova vrijednost varijable `br`). Tako se formira vrijednost varijable `suma`, koja po izlasku petlje, dakle nakon deset ponavljanja, sadrži upravo zbroj svih unesenih vrijednosti varijable `br` (varijabla `br` je deset puta promijenila vrijednost).

Sljedeći računalni program ispisuje svaki drugi element polja koji je korisnik unio. Za razliku od prethodnoga programa u kome nisu sačuvane vrijednosti koje je korisnik unio, ovdje će se koristiti polje koje omogućava istovremeno čuvanje više vrijednosti, ali istoga tipa podatka. I ovdje korisnik treba unijeti deset brojeva.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int polje[10],suma=0,i;
 5.    printf("ovo su elementi polja\n");
 6.    for (i=0;i<10;i++){
 7.      scanf("%d",&polje[i]);
 8.    }
 9.    printf("\nsvaki drugi clan je");
10.    for(i=0;i<10;i=i+2){
11.      printf("\n%d",polje[i]);
12.      suma=suma+polje[i];
13.    }
14.    printf("suma svakog drugog člana polja je %d",suma);
15.    getch();}
```

Program 5.19.

Koriste se dvije `for` petlje – prva se `for` petlja (linija 6.) koristi za unos vrijednosti u polje, a druga (linija 10.) za ispis svakog drugog elementa polja te za izračun njihove sume.

Naredni računalni program traži od korisnika unos deset brojeva koji se spremaju u polje. Potom se ispisuje najveći i najmanji uneseni broj te njihov prosjek. Varijabla `x`

predstavlja indekse elemenata polja. U varijable `min` i `max` će se nakon uspoređivanja svih članova, smjestiti upravo najmanji, odnosno najveći uneseni broj. Varijabla `suma` će biti zbroj svih elemenata polja.

Algoritam za pronalazak najmanjeg i najvećega broja već je korišten u poglavlju Razgranata algoritamska struktura u C – zadatak 4.8. Dakle, `min` i `max` se postave na vrijednost prvog elementa u polju (linija 9.). `Suma` također (linija 10.). Tada slijedi petlja `for` (linija 11.) koja uzima preostalih devet elemenata polja (dakle indeksa `x` od 1 do 9). U liniji 12. se provjerava je li element polja na indeksu `x` manji od trenutne vrijednosti varijable `min`; ukoliko je, varijabli `min` se pridružuje vrijednost tog `x`-tog elementa polja; ukoliko nije, vrijednost `min` se ne mijenja.

Slično se radi s varijablom `max` u liniji 13. U liniji 15. vrijednost varijable `suma` povećava se za vrijednost `x`-tog elementa polja. Bitno je naglasiti da ovdje `suma` nije postavljena na 0, već na prvi element polja (indeks 0) jer u petlji `for` brojanje indeksa započinje s jedan. Moguće je programski kod preurediti tako da varijabla `suma` bude inicijalizirana na nula.

U liniji 18. izvršen je ispis prosjeka unesenih vrijednosti kao kvocijent `suma/10`.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int broj[10], x, min, max, suma;
 5.    for(x=1; x<=10; x++){
 6.      printf("Upisi %d. broj: ", x);
 7.      scanf("%d", &broj[x-1]);
 8.    }
 9.    min = max = broj[0];
10.    suma = broj[0];
11.    for(x=1;x<=9;x++){
12.      if(broj[x] < min)  min = broj[x];
13.      if(broj[x] > max)  max = broj[x];
14.      suma = suma + broj[x];   // suma += broj[x];
15.    }
16.    printf("\nMinimum = %d", min);
17.    printf("\nMaximum = %d", max);
18.    printf("\nProsjek = %f", (float)suma/10);
19.    getch();}
```

Program 5.20.

Računalni program u nastavku ispisuje u dva stupca sve unesene brojeve te njihovo odstupanje od prosjeka tog istog uzorka brojeva. Koriste se dvije petlje. Prva petlja se koristi za unos deset brojeva u polje te se u toj petlji izračunava njihova suma koja je kasnije potrebna za izračun prosjeka (linija 11.). Oznaka `\t` u liniji 7. je oznaka za tabulator, a koristi se pri formatiranju ispisa.

Vrijednost varijable `ars` se dobije kao kvocijent `sume` (čija vrijednost se lokalno pretvara u tip podatka `float`) i broja deset (broja unesenih vrijednosti).

Druga petlja (linija 12.) ispisuje deset puta liniju teksta koji se sastoji od `x`-tog elementa polja, dva tabulatora te broja koji je razlika prosjeka sa `x`-tim elementom polja (odstupanje od prosjeka).

Ispisivanjem linija teksta koji ima praznine (2 tabulatora) stječe se dojam da se radi o dvama stupcima. Ključno je u naredbi `printf` postaviti oznaku novoga retka `\n`, na početak ili kraj, kako bi se kod svakoga ponavljanja ispis radio u novome retku (inače bi ispisi bili slijepljeni).

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int br[10],x,suma=0;
 5.   float ars;
 6.   for(x=0;x<10;x++){
 7.     printf("unesi %d. broj\t",x+1);
 8.     scanf("%d",&br[x]);
 9.     suma=suma+br[x];
10.   }
11.   ars=(float)suma/10;
12.   for(x=0;x<10;x++)
13.     printf("%d\t\t%.2f\n",br[x],ars-br[x]);
14.   getch();}
```

Program 5.21.

U računalnome programu koji slijedi od korisnika se zahtijeva unos deset brojeva. Potom se ispisuje najveći broj, suma unesenih brojeva i korijen sume. Algoritam za pronalazak najvećega među unesenim brojevima je već prikazan, kao i način izračuna sume unesenih brojeva. Izračunavanje korijena sume odvojilo se u dva slučaja – vađenje korijena iz pozitivnoga i iz negativnoga broja. Slučaj vađenja korijena iz negativnoga broja (slučaj kada je `suma<0)` riješen je tako da se izračunava korijen od `suma*(-1)` te se kod ispisa, pored izračunatog korijena, ispisuje `i` (linija 15.).

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include<math.h>
 4. main(){
 5.   int i, max=0,a,suma=0;
 6.   printf("unesite 10 brojeva: \n");
 7.   for(i=1;i<=10;i++){
 8.     printf("%d.\t",i);
 9.     scanf("%d",&a);
10.     if(max<a)
11.       max=a;
12.     suma+=a;
13.   }
14.   if(suma<0)
15.     printf("max broj je %d\nsuma je %d\nkorjen je %.2fi",max,
```

```
                suma,sqrt(suma*(-1)));
16.   else
17.      printf("max broj je %d\nsuma je %d\nkorjen je %.2f",max,
                suma,sqrt(suma));
18.   getch();}
```

Program 5.22.

U sljedećemu računalnom programu unose se elementi polja te se odmah po unosu kategoriziraju u parne, neparne, pozitivne ili negativne i kao takvi se pribrajaju u odgovarajuće sume: suma parnih (`spar`), suma neparnih (`snepar`), suma pozitivnih (`spoz`) te suma negativnih (`sneg`). Nakon toga ispisuju se posebno parne i posebno neparne vrijednosti polja.

Pri deklaraciji varijabli za sve se sume odmah izvodi njihova inicijalizacija na vrijednost nula. Potom slijedi petlja za unos elemenata polja u sklopu koje se element polja odmah ispituje (paran/neparan ili pozitivan/negativan) te se njegova vrijednost dodaje u odgovarajuću sumu.

Nakon petlje slijedi ispis suma. Program završava s dvije `for` petlje – prva za ispis parnih, a druga za ispis neparnih elemenata polja.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.   int a[10],i,spar=0,snepar=0,spoz=0,sneg=0;
 5.   for(i=0;i<=9;i++){
 6.      printf("unesite %d element polja",i+1);
 7.      scanf("%d",&a[i]);
 8.      if(a[i]%2==0) spar+=a[i];
 9.      if(a[i]%2!=0) snepar+=a[i];
10.      if(a[i]>=0) spoz+=a[i];
11.      if(a[i]<0) spar+=a[i];
12.   }
13.   printf("suma parnih je %d\n",spar);
14.   printf("suma neparnih je %d\n",snepar);
15.   printf("suma pozitivnih je %d\n",spoz);
16.   printf("suma negativnih je %d\n",sneg);
17.   printf("parni su:\n");
18.   for(i=0;i<=9;i++){
19.      if(a[i]%2==0)
20.         printf("\n%d",a[i]);
21.   }
22.   printf("\nNeparni su:\n");
23.   for(i=0;i<=9;i++){
24.      if(a[i]%2!=0)
25.         printf("\n%d",a[i]);
26.   }
27.   getch();}
```

Program 5.23.

Sljedeći računalni program ispisuje oblik pravokutnika koristeći znak *. Unose se dimenzije pravokutnika (duljina i širina) te se ispisuje onoliko redaka znaka * koliki je iznos za duljinu, a svaki redak se sastoji iz onoliko znakova * koliki je iznos za širinu. Npr. za unos duljine od 5 a širine od 8 ispisat će se sljedeće:

```
********
********
********
********
********
```

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int a,b,i,j;
 5.   printf("unesite visinu pravokutnika od zvjezdica: ");
 6.   scanf("%d",&a);
 7.   printf("unesite sirinu pravokutnika od zvjezdica: ");
 8.   scanf("%d",&b);
 9.   for(i=1;i<=a;i++){
10.     for(j=1;j<=b;j++){
11.       printf("* ");
12.     }
13.     printf("\n");
14.   }
15.   getch();}
```

Program 5.23.

Petlja `for` u liniji 9. služi za brojanje redaka, tj. visine pravokutnika, pa ide od jedan do vrijednosti varijable `a`. Za svaki redak, tj. za svaki `i`, ispisuje se `b` zvjezdica, a to je zadatak druge petlje `for` iz linije 10. Petlja `for` iz linije 10. se nalazi u tijelu petlje `for` iz linije 9. Ovo je primjer ugniježđenih petlji, što znači da za svaki prolazak prve petlje, dakle za svaki `i`, izvodi se cijela druga petlja. Za svaki `i`, nakon što se `b` puta ispiše znak *,ispisuje se i prelazak u novi red kako bi sljedeći ispis znakova * započeo u novome retku (linija 13.).

Izuzetno je važno brinuti se o tome kojem bloku naredbi (definiranom s vitičastim zagradama) pripada koja naredba. Tako linija 13. pripada bloku naredbi prve petlje (ne druge).

Sljedeći računalni program ispisuje samo rub pravokutnika zadane dimenzije. Npr. za duljinu 5 i širinu 8 ispis će biti kako slijedi:

```
********
*      *
*      *
*      *
********
```

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int a,b,i,j;
 5.   printf("unesi visinu praznog pravokutnika : ");
 6.   scanf("%d",&a);
 7.   printf("unesi sirinu praznog pravokutnika : ");
 8.   scanf("%d",&b);
 9.   for(i=1;i<=b;i++){
10.     printf("* ");
11.   }
12.   printf("\n");
13.   for(i=2;i<=a-1;i++){
14.     printf("* ");
15.     for(j=2;j<=b-1;j++){
16.        printf("  ");
17.     }
18.     printf("*\n");
19.   }
20.   for(i=1;i<=b;i++){
21.    printf("* ");
22.   }
23.   getch();}
```

Program 5.24.

Algoritam koji je primijenjen prvo iscrta prvi redak pravokutnika (prva `for` petlja). Potom se crtaju "unutarnji" redci pravokutnika (druga ugniježđena `for` petlja) – unutarnji redci su oni koji nisu prvi i zadnji redak pravokutnika. Zadnja for petlja crta zadnji redak pravokutnika.

Isti računalni program se može izraditi primjenom drugačijeg algoritma koji koristi samo jednu ugnježđenu petlju. Slijedi prikaz takvoga računalnog programa.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int a,b,i,j;
 5.   printf("unesite visinu pravokutnika: ");
 6.   scanf("%d",&a);
 7.   printf("unesite sirinu pravokutnika: ");
 8.   scanf("%d",&b);
 9.   for(i=1;i<=a;i++){
10.     for(j=1;j<=b;j++){
11.       if (i==1 || i==a || j==1 || j==b)
12.         printf("* ");
13.       else
14.         printf("  ");
15.     }
16.     printf("\n");
17.   }
18.   getch();}
```

Program 5.25.

Računalni program u nastavku traži od korisnika unos pet brojeva. Potom izrađuje tablicu umnožaka unesenih brojeva. Format ispisa je sljedeći:

```
*|3  2  1  3  2
----------
3|9  6  3  9  6
2|6  4  2  6  4
1|3  2  1  3  2
3|9  6  3  9  6
2|6  4  2  6  4
```

Uneseni brojevi se spremaju u polje od pet elemenata. Kako bi se izradila njihova tablica množenja, koristi se ugniježđena petlja. Prva se `for` petlja (linija 6.) koristi za unos pet brojeva. Druga se `for` petlja (linija 12.) koristi za ispis prvoga retka tablice. Preostale se dvije ugniježđene petlje (linija 16.) koriste za formatirani ispis tablice i izračun umnožaka.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int a[5];
 5.    int i, j;
 6.    for(i = 1 ; i <= 5 ; i++){
 7.      printf("Unesite %d broj:\t",i);
 8.      scanf("%d",&a[i]);
 9.    }
10.    printf("\n\n");
11.    printf("* |");
12.    for(i = 1 ; i <= 5 ; i++){
13.      printf("\t%d",a[i]);
14.    }
15.    printf("\n--|----------------------------------------");
16.    for(j = 1 ; j <= 5 ; j++){
17.      printf("\n%d |", a[j]);
18.      for(i = 1 ; i <= 5 ; i++){
19.        printf("\t%d", a[j] * a[i]);
20.      }
21.    }
22.    getch();}
```

Program 5.26.

Ukoliko se želi izraditi tablica množenja koja u recima ima jedan niz od `m` brojeva, a u stupcima drugi niz od `n` brojeva (dakle, tablica nije kvadratna, odnosno nema isti broj redaka i stupaca kao u prethodnom primjeru), tada je potrebno deklarirati dva polja – jedan dimenzije `m`, a drugi dimenzije `n`. Slijedi računalni program u kome je `m=5`, a `n=7`, dakle radi se o tablici s 5 redaka i 7 stupaca.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.    int a[5];
 5.    int b[7];
 6.    int i, j;
 7.    printf("Unesite elemente prvog polja\n");
 8.    for(i = 1 ; i <= 5 ; i++) {
 9.       printf("unesi %d. od 5\n",i);
10.       scanf("%d",&a[i]);
11.    }
12.    printf("Unesite elemente drugog polja\n");
13.    for(i = 1 ; i <= 7 ; i++){
14.       printf("unesi %d. od 7\n",i);
15.       scanf("%d",&b[i]);
16.    }
17.    printf("* |");
18.    for(i = 1 ; i <= 7 ; i++){
19.       printf("\t%d",b[i]);
20.    }
21.    printf("\n--|-----------------------------------------");
22.    for(j = 1 ; j <= 5 ; j++){
23.       printf("\n%d |", a[j]);
24.       for(i = 1 ; i <= 7 ; i++){
25.          printf("\t%d", a[j] * b[i]);
26.       }
27.    }
28.    getch();}
```

Program 5.27.

U sljedećemu primjeru se prikazuje korištenje funkcije `rand()` koja generira slučajne brojeve od nule do `RAND_MAX`, cijelog broja koji je definiran u zaglavnoj datoteci `stdlib.h`. Općenito se s funkcijom `rand()` uz primjenu operatora modulo, mogu dobiti slučajno generirani brojevi u nekome željenom intervalu brojeva.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include <stdlib.h>
 4. #include <time.h>
 5. main(){
 6.    int slucajni, x;
 7.    srand( (unsigned)time( NULL ) );
 8.    printf("evo primjera nekog slucajno generiranog
                broja %d",rand());
 9.    printf("\nsada ce se generirati 10 slucajnih brojeva od 1-
              20\n");
10.    for(x=1;x<=10;x++){
11.       slucajni = (rand() % 20)  +  1;
12.       printf("%d\n", slucajni);
13.    }
14.    getch();}
```

Program 5.28.

Zaglavne su datoteke, uključene na početku, potrebne radi korištenja funkcije `rand()` (`stdlib.h`) i funkcije `time()`(`time.h`). Linija 7. je potrebna kako bi se pri svakome pokretanju računalnoga programa nanovo inicijalizirao generator slučajnih brojeva. Kada bi program bio bez ove linije koda, računalo bi pri svakome pokretanju programa generiralo uvijek isti slijed slučajnih brojeva.

U liniji 8. se poziva funkcija `rand()` te ispisuje slučajno generirani broj.

U liniji 10. pokreće se `for` petlja kako bi se ispisalo deset slučajno generiranih brojeva, ali sada u intervalu od 1 do 20. Točno se taj raspon brojeva dobiva preko operatora modulo (`%`). Dakle, slučajno će generirani broj % 20, za rezultat dati brojeve u intervalu od 0 do 19 (ostaci pri dijeljenju s 20), a dodavanjem broja 1 cijeli interval se pomiče na interval od 1 do 20.

Sljedeći računalni program korisniku omogućava pogađanje slučajnoga broja iz intervala od 1 do 10 u najviše pet pokušaja. Ograničenje od pet pokušaja je implementirano kroz `for` petlju. U tijelu se petlje nalaze provjere odnosa unesenoga broja (varijabla `pokusaj`) i slučajno generiranoga broja (varijabla `x`). U ovisnosti o njihovom odnosu, ispisuje se odgovarajuća poruka. Treba primijetiti naredbu `break` (linija 22.) koja prekida izvođenje petlje kada je slučajni broj je pogođen. Isto tako, treba primijetiti kako se u linijama 27. i 29. koristi vrijednost varijable `i` (brojača petlje). Ako je petlja prekinuta naredbom `break` (broj pogođen), u varijabli `i` će biti upravo vrijednost koja odgovara rednom broju pokušaja (neće biti izveden korak `i++` u petlji). Ako je pak petlja završila jer uvjet petlje `i<=5` nije ispunjen, tada vrijednost u varijabli `i` neće biti redni broj zadnjega pokušaja (u primjeru je to 5). To je zato što je izveden korak petlje `i++` (u primjeru će stoga vrijednost biti 6).

```
 1. #include <stdlib.h>
 2. #include <stdio.h>
 3. #include <conio.h>
 4. #include <time.h>
 5. main(){
 6.   int i,x,pokusaj;
 7.   printf("Pogodite broj od 1-10 u 5 pokusaja\n");
 8.   srand(time(0));
 9.   x = rand() % 10 + 1 ;
10.   for(i = 1 ; i <= 5 ; i++){
11.     printf("\nUnesite %d pokusaj broja:\t",i);
12.     scanf("%d",&pokusaj);
13.     if(pokusaj < x){
14.       printf("\nBroj X je veci od unesenog.\n");
15.     }
16.     else if(pokusaj > x){
17.       printf("\nBroj X je manji od unesenog.\n");
18.     }
19.     else if(pokusaj == x){
20.       printf("\nBravo! Pogodili ste broj.\n");
21.       pokusaj=x;
```

```
22.         break;
23.       }
24.    }
25.    printf("--------------------------------------");
26.    if(pokusaj==x)
27.       printf("\nPogodili ste u %d pokusaja,X=pogodak=%d",
                i,pokusaj);
28.    else
29.       printf("\nNiste uspjeli pogoditi u %d pokusaja",i-1);
30.    getch();}
```

Program 5.29.

U programskome se kodu često koriste varijable koje signaliziraju neko stanje. Takve se varijable nazivaju zastavice (*engl. flag*). One često imaju dvije vrijednosti (`true`, `false`; `1`, `0`).

U sljedećem računalnom programu je potrebno ispisati svaki drugi višekratnik broja 3 od danih 15 brojeva. Koristi se varijabla `svdrugi`, tipa `char`, jer je to tip koji zauzima najmanje memorijskih lokacija – jedan bajt, kao varijabla koja će signalizirati potrebu ispisa broja jer je on drugi koji ispunjava uvjet da je višekratnik broja 3. Varijabla će poprimiti vrijednost `0` ili `1`.

U računalnome se programu 15 cijelih brojeva čuva u polju odgovarajuće duljine (linija 8.). Petlja `for` se koristi kako bi se ispitalo je li vrijednost elemenata polja na indeksu `i` djeljiva s 3. Djeljivost se provjerava primjenom operatora modulo (`%`) – izračunava se ostatak pri cjelobrojnome dijeljenju koji treba biti jednak nuli (`==0`) ako drugi operand dijeli prvi (linija 8.). Za provjeru nedjeljivosti, operator modulo treba dati rezultat koji je različit od nula (`!=0`).

Budući se želi ispisati svaki drugi broj iz polja koji je djeljiv s 3, potrebno je pamtiti stanje ispisa. Naime, kada se ispiše prvi broj, varijabla `svdrugi` poprima vrijednost 0 (linija 11.) što označava da se ispis ne treba izvesti. Kada se ponovno pronađe broj koji ispunjava traženi uvjet, on neće biti ispisan (jer nije drugi, odnosno varijabla `svdrugi` ima vrijednost 0), ali će sada varijabla `svdrugi` poprimiti vrijednost 1 (linija 13.). To sada znači da će sljedeći broj koji ispunjava uvjet biti ispisan. Ovim "spuštanjem" i "podizanjem" zastavice, odnosno promjenom vrijednosti varijable `svdrugi` izvodi se ispis svakog drugog višekratnika broja 3, odnosno izvodi se ispis samo onih brojeva kod kojih je zastavica "podignuta".

```
 1. #include <conio.h>
 2. #include<stdio.h>
 3. main(){
 4.    int i;
 5.    char svdrugi=1;
 6.    int a[15] = {0, 4, 6, 9, 11, 12, 19, 21, 24, 27, 30, 33, 36,
                    39, 42};
 7.    for(i=0;i<15;i++){
 8.       if(a[i]%3==0){
 9.          if(svdrugi==1){
```

```
10.            printf("%d\n",a[i]);
11.            svdrugi=0;
12.          }
13.          else svdrugi=1;
14.       }
15.    }
16.    getch();}
```

Program 5.30.

Računalni program koji slijedi omogućava korisniku unos 10 cijelih brojeva koji se spremaju u polje odgovarajuće dimenzije. Ispisuje se svaki drugi parni broj. Ponovno se koristi varijabla `svdrugi` kao zastavica.

```
 1. #include <stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int a[10],i;
 5.   printf("Unesite sada po zelji 10 brojeva:\n\n");
 6.   for(i=0;i<10;i++){
 7.     printf("unesite %d od 10:\t",i+1);
 8.     scanf("%d",&a[i]);
 9.   }
10.   printf("e sad ispisuj svaki drugi parni\n");
11.   char svdrugi=1;
12.   for(i=0;i<10;i++){
13.     if(a[i]%2==0) {
14.       if(svdrugi==1){
15.         printf("\n%d",a[i]);
16.         svdrugi=0;
17.       }
18.       else svdrugi=1;
19.     }
20.    }
21. getch();}
```

Program 5.31.

U narednome računalnom programu korisnik unosi cijeli broj. Potom se izvodi provjera je li uneseni broj prost ili ne. Broj je prost onda i samo onda kada je djeljiv samo s brojem 1 i samim sobom. U programskome se kodu koristi varijabla `brojac` koja ima ulogu zastavice jer se njome signalizira dostizanje stanja u kojem broj nije prost.

Svaki je broj djeljiv s 1 i samim sobom pa se postavlja pitanje kako programskim kodom izraziti uvjet da je prost broj onaj koji je djeljiv samo s 1 i samim sobom? To se može postići tako da se traže djelitelji unesenoga broja koji su različiti od 1 i samoga tog broja. Pronalazak takvoga djelitelja se signalizira zastavicom, odnosno varijablom `brojac`. U slučaju da zastavica signalizira postojanje djelitelja, to označava da broj nije prost. Inače je broj prost.

Promatrajući cijele brojeve, može se zaključiti da su mogući djelitelji nekoga broja n od 1 do n/2. Primjera radi, broj 16 ima djelitelje koji se nalaze u intervalu od 2 do 8 (osim 1 i samoga 16), broj 19 ima djelitelje u intervalu od 2 do 9 (osim 1 i samoga 19) itd.

U prikazanome programskom kodu (program 5.32.) nakon korisničkoga unosa (varijabla `broj`) slijedi `for` petlja u kojoj se traži djelitelj unesenoga broja koji je u intervalu od broja 2 do unesenoga broja podijeljenoga s 2 (`broj/2`). U slučaju da se pronađe djelitelj (linija koda 8.), podiže se zastavica (linija koda 9.) te se prekida izvođenje petlje (linija koda 10.) jer je pronađen barem jedan djelitelj. Ako se nakon izvođenja svih ponavljanja tijela petlje ne pronađe djelitelj, zastavica će ostati spuštena (varijabla `brojac` će imati vrijednost 0) što će označiti da je uneseni broj prost.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  main(){
 4.    int broj,i,brojac=0;
 5.    printf("Unesi neki broj \n");
 6.    scanf("%d",&broj);
 7.    for (i=2;i<=broj/2;i++){
 8.      if((broj%i)==0){
 9.        brojac++;
10.        break;
11.      }
12.    }
13.    if(brojac==0)
14.      printf("\n%d je prost broj",broj);
15.    else
16.      printf("\n%d nije prost broj",broj);
17.    getch();}
```

Program 5.32.

Računalni program koji slijedi ispisuje sve proste brojeve do nekoga unesenog broja. Primjera radi, ako se unese broj 12, program će ispisati:

```
1
2
3
5
7
11
```

Provjera je li broj prost ili ne, koja je opisana u prethodnome zadatku, ovdje se mora raditi za svaki broj u intervalu od 1 do unesenoga broja. Stoga u programskome kodu postoji ugniježđena petlja. Prva se `for` petlja (linija 8.) koristi za izbor broja iz intervala. Ulogu zastavice ima varijabla `potvrda`. Ona se pri izboru broja odmah postavlja na vrijednost 0 (linija 9.). To se može interpretirati i kao pretpostavka da je izabrani broj prost. Potom slijedi druga `for` petlja (linija 10.) kojom se traži djelitelj

izabranoga broja koji nije 1 niti sam taj broj. Ako se takav djelitelj pronađe, zastavica se "podiže", odnosno varijabla `potvrda` poprima vrijednost 1. Nakon provjere djelitelja, slijedi provjera je li zastavica "spuštena" ili "podignuta" (linija 16.). Ako je spuštena (ispunjen uvjet `potvrda==0`) izabrani se broj ispisuje (izabrani broj se nalazi u varijabli `i`).

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.   int unos,i,dj,potvrda;
 5.   printf("Unesi pozitivan broj:\t");
 6.   scanf("%d",&unos);
 7.   printf("Popis prostih brojeva do %d je:\n",unos);
 8.   for (i = 1 ; i <= unos ; i++) {
 9.     potvrda = 0;
10.     for(dj=2 ; dj <= i/2 ; dj++ ) {
11.       if(i % dj == 0 ) {
12.         potvrda = 1;
13.       break;
14.       }
15.     }
16.     if ( potvrda == 0) {
17.       printf("%d\n",i);
18.     }
19.   }
20. getch();}
```

Program 5.33.

Prvi računalni program s `while` petljom ispisuje znak s unesenim ASCII kodom. Korisnik unosi ASCII kodove iz intervala od 0 do 255.

Kako je `while` petlja, petlja s izlaskom na vrhu, to uvjet za prvo izvođenje tijela petlje mora biti odmah ispunjen. Zbog toga se prije izvođenja petlje traži korisnički unos ASCII koda (linija 6.). Unos nekoga ASCII koda izvan intervala od 0 do 255 neće ispuniti uvjet `while` petlje te će biti izvedena linija 12. Ukoliko se pak unese ASCII kod između brojeva 0 i 255, bit će ispunjen uvjet `while` petlje te će se izvesti blok naredbi koji čini tijelo petlje.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. main(){
 4.   int broj,i;
 5.   printf("unesi ASCII kod: ");
 6.   scanf("%d",&broj);
 7.   while(broj<=255 && broj>=0){
 8.     printf("znak s ASCII kodom %d je %c\n",broj,broj);
 9.     printf("unesi broj: ");
10.     scanf("%d",&broj);
11.   }
```

```
12.    printf("\nRazmatranje se zavrsava jer ste unijeli ASCII
              kod izvan granica!");
13.    getch();}
```

Program 5.34.

U sljedećemu računalnom programu korisnik unosi cijele brojeve, sve dok ne unese 0. Program potom ispisuje sumu i prosjek svih unesenih brojeva, ne uzimajući u obzir zadnju unesenu vrijednost. I ovdje se prvi upit za korisničkim unosom nalazi izvan tijela petlje (linija 8.) kako bi varijabla `broj` imala vrijednost s kojom će biti ispitano ispunjenje uvjeta u liniji 9.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int suma=0, broj, brojac=1;
 5.    float prosjek;
 6.    printf("UPISUJTE BROJEVE...ZA KRAJ UNESITE 0, ISPISUJE
              SE  SUMA UNESENIH BROJEVA I NJIHOV PROSJEK\n");
 7.    printf("Upisi broj: ");
 8.    scanf("%d", &broj);
 9.    while(broj != 0){
10.      suma = suma + broj;
11.      printf("Upisi broj: ");
12.      scanf("%d", &broj);
13.      brojac++;
14.    }
15.    printf("Suma = %d\nprosjek=%.2f", suma,
              (float)suma/(brojac-1));
16.    getch();}
```

Program 5.35.

Računalni program (program 5.36.) traži od korisnika unos cijelih brojeva sve dok se ne unese broj koji je manji ili jednak nuli. Nakon svakoga unosa, računalo ispisuje trenutnu sumu unesenih brojeva (varijabla `suma`) i njihov trenutni prosjek (varijabla `ars`). U varijabli `i` se čuva redni broj unosa. Po završetku unosa brojeva ispisuje se njihova konačna suma i prosjek.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int broj,suma=0,i=1;
 5.    float ars;
 6.    printf("Unesi broj: \n");
 7.    scanf("%d",&broj);
 8.    while(broj>0){
 9.      suma+=broj;
10.      printf("Trenutna suma je %d",suma);
11.      ars=(float)suma/i;
12.      printf("\nTrenutni prosjek je %.2f",ars);
13.      printf("\nPonovno unesi broj: \n");
```

```
14.      scanf("%d",&broj);
15.      i++;
16.    }
17.    printf("\nKonacna suma je %d",suma);
18.    printf("\nKonacni prosjek je %.2f",ars);
19.    getch();}
```

Program 5.36.

Sljedeći računalni program (program 5.37.) traži od korisnika unos cijeloga broja. Potom računalo izračunava i ispisuje sumu znamenki unesenoga broja. Primjera radi, ako je unesen broj -4201, onda će računalo ispisati da je zbroj znamenki 7 (predznak broja se ne uzima u obzir).

Pitanje je kako izolirati znamenku po znamenku nekoga unesenog broja koji može imati bilo koji broj znamenki. Rješenje je u tome da se broju uvijek uzima znamenka na koja se nalazi na mjestu jedinica. Naime, bilo koji cijeli broj pri dijeljenju s brojem 10 daje ostatak koji je jednak znamenki na mjestu jedinice. Dakle, potrebno je koristiti operator modulo (`%`). Kada se izolira znamenka na mjestu jedinica, unesenome broju bi se sada ta znamenka trebala "izrezati" te nad novo dobivenim brojem ponoviti operaciju modulo s brojem 10. Izrezivanje znamenke se dobiva cjelobrojnim dijeljenjem unesenoga broja s 10.

Slijedi opisani algoritam primijenjen na primjeru broja -4201:

```
 1.  -4201%10=-1                 (znamenka na mjestu jedinica)
 2.  -4201/10=-420               (broj s izrezanom znamenkom)
 3.  -420%10=0                   (znamenka na mjestu jedinica)
 4.  -420/10=-42                 (broj s izrezanom znamenkom)
 5.  -42%10=-2                   (znamenka na mjestu jedinica)
 6.  -42/10=-4                   (broj s izrezanom znamenkom)
 7.  -4%10=-4                    (znamenka na mjestu jedinica)
 8.  -4/10=0                     (nema broja s izrezanom znamenkom)
 9.  (-1) + 0 + (-2) + (-4) = -7 (suma znamenki)
10.  -7 * (-1) = 7               (rješenje)
```

Algoritam 5.1.

Može se zaključiti da se postupak pronalaska znamenke koja se nalazi na mjestu jedinica, te njezino izrezivanje, ponavljaju sve dok se izrezivanjem ne dobije broj 0. Potom se izvodi sumiranje dobivenih znamenki te se, ako je suma negativna, množi s -1 kako bi se dobila pozitivna suma.

Varijabli `or_broj` pridružuje se vrijednost varijable `broj` (vrijednost unesenoga broja). Na taj će se način uneseni broj sačuvati budući da se vrijednost varijable broj mijenja u tijelu petlje `while`.

U tijelu petlje u varijablu `znamenka` se sprema znamenka na mjestu jedinica koja pripada trenutnoj vrijednosti varijable `broj` (varijabla `broj` čuva vrijednost iz koje je izrezana znamenka na mjestu jedinica). Dobivena znamenka se pribraja varijabli `suma`, a varijabli `broj` se ta znamenka ukloni (linija 12.).

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.     int broj, znamenka, suma ,or_broj;
 5.     printf("Unesite broj ciju sumu znamenaka zelite izracunati:");
 6.     scanf("%d", &broj);
 7.     or_broj=broj;
 8.     suma = 0;
 9.     while(broj !=0){
10.        znamenka = broj % 10;
11.        suma = suma + znamenka;
12.        broj = broj/10;
13.     }
14.     if(or_broj<0)
15.        printf ("Broj %d ima zbroj znamenki = %d", or_broj,suma*(-1));
16.     else
17.        printf ("Broj %d ima zbroj znamenki = %d", or_broj,suma);
18.     getch();}
```

Program 5.37.

Računalni program 5.38. također izvodi obradu znamenki unesenoga broja, ali se sada znamenke spremaju u polje odgovarajuće dimenzije. Nakon korisničkoga unosa nekog broja, ispisuje se: broj znamenki unesenoga broja, znamenke redom prema mjestu jedinica (od prve do zadnje znamenke) odvojene zarezom te se ispisuje suma znamenki. Primjera radi, za uneseni broj 345796 ispisuje se:

```
Broj znamenki, x=6
3,4,5,7,9,6
3+4+5+7+9+6=32
```

U liniji se 6. zahtijeva unos broja, pozitivnog ili negativnog, s maksimalno 10 znamenki jer je deklarirano polje `Zn[10]` upravo dimenzije 10. Nakon osiguranja da će se u daljnjoj obradi koristiti samo pozitivni broj (linija 7.), slijedi algoritam kojim se izolira znamenka na mjestu jedinica i sprema u odgovarajući element polja (linija 11.) te se pridodaje sumi (linija 12.)

Nakon završetka petlje `while`, ispisuje se ukupan broj znamenki (linija 16.) te se u petlji `for` (linija 18.) izvodi ispis znamenaka od prve do zadnje. Kako se u elementima polja nalaze znamenke, ali u redoslijedu od jedinica na više, dakle, na indeksu 0 polja se nalazi znamenka na mjestu jedinica, na indeksu 1 znamenka na mjestu stotica itd., to je ispis potrebno izvesti tako da se elementi polja ispisuju od najvišeg indeksa prema najnižem. Budući da se pri ispisu između znamenki treba nalaziti zarez (linija 19.), posljednja znamenka treba biti ispisana bez zareza (linija 21.).

Iza ispisa znamenki slijedi ispis sume znamenki preko `for` petlje (linija 22.). Između znamenki se treba nalaziti znak + (linija 23.), osim kod zadnje znamenke iza koje se treba nalaziti znak = (linija 25.). Potom slijedi vrijednost sume koja se nalazi u varijabli `suma`.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
```

```
 4.    int b, x, zn[10], suma = 0;
 5.    printf("Upisi broj: ");
 6.    scanf("%d", &b);
 7.    if(b<0)
 8.       b*=-1;
 9.    x=0;
10.    while(b>0){
11.       zn[x] = b % 10;
12.       suma += zn[x];
13.       b /= 10;
14.       x++;
15.    }
16.    printf("Broj znamenki, x=%d\n",x);
17.    int p=x-1;
18.    for(x--; x>0; x--){
19.       printf("%d,", zn[x]);
20.    }
21.    printf("%d\n" ,zn[x]);
22.    for(p; p>0; p--){
23.       printf("%d+", zn[p]);
24.    }
25.    printf("%d=%d" ,zn[p],suma);
26.    getch();}
```

Program 5.38.

Sljedeći računalni program traži od korisnika unos nekoga cijelog broja koji se sprema u varijablu `br`. Potom se izračunava faktorijela unesenoga broja. Faktorijela za uneseni broj n glasi n!=1·2·...·(n-1)·n. S tim da je definirano da je 0! = 1 i 1!=1.

Korisniku je omogućeno pet unosa (linija koda 5.). U slučaju da se unese negativan broj, korisniku se ispisuje poruka te se uneseni negativan broj pretvara u pozitivan (linija 9.). Varijabla `fakt` je varijabla u kojoj se izračunava vrijednost faktorijela unesenoga broja. Kako se varijabla `fakt` koristi za akumuliranje umnožaka (nova vrijednost se dobije tako da se stara vrijednost pomnoži s brojem – linija 19.), to se njezina početna vrijednost treba postaviti na 1 (linija 17.). Akumuliranje se umnožaka izvodi sve dok ima brojeva s kojima se treba množiti, dakle dok god je ispunjen uvjet `while` petlje (linija 18.). U liniji 23. se ispisuje uneseni broj i vrijednost njegove faktorijele.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  main(){
 4.     int fakt,br,konstbr,i;
 5.     for(i=1;i<6;i++){
 6.        printf("Unesi broj: ");
 7.        scanf("%d",&br);
 8.        konstbr=br;
 9.        if(br<0){
10.           printf("\nGreska. Ne postoji faktorijel negativnog broja, ");
11.           printf("ali ce se uneseni broj pomnoziti sa (-1)");
12.           br=br*-1;
13.        }
```

```
14.       if(br==1||br==0)
15.          fakt=1;
16.       else{
17.          fakt=1;
18.          while(br>1){
19.             fakt=fakt*br;
20.             br=br-1;
21.          }
22.       }
23.       printf("\nZa uneseni broj %d faktorijela je %d\n",konstbr,fakt);
24.    }
25.   getch();}
```

Program 5.39.

Računalni program 5.40. pretvara uneseni pozitivni cijeli broj iz dekadskoga u binarni zapis. Postupak pretvaranja pozitivnoga cijelog broja iz dekadskoga u binarni sustav sastoji se iz dijeljenja unesenoga broj s bazom binarnoga sustava, odnosno brojem 2. Dobiveni ostatak predstavlja jednu znamenku u binarnome zapisu. Potom se izračunava količnik tako da se broj iz prethodnoga koraka cjelobrojno podijeli s 2. Ponovno se od dobivenoga količnika izračunava ostatak pri dijeljenju s 2 koji predstavlja drugu binarnu znamenku itd. Postupak se ponavlja dok količnik ne postane 0.

Primjera radi, neka se broj 26 treba pretvoriti u binarni zapis. Slijedi postupak (algoritam):

```
 1.  26%2 = 0   (binarna znamenka)
 2.  26/2 = 13  (količnik)
 3.  13%2 = 1   (binarna znamenka)
 4.  13/2 = 6   (količnik)
 5.  6%2 = 0    (binarna znamenka)
 6.  6/2 = 3    (količnik)
 7.  3%2 = 1    (binarna znamenka)
 8.  3/2 = 1    (količnik)
 9.  1%2 = 1    (binarna znamenka)
10.  1/2 = 0    (količnik - kraj)
11.  11010      (binarni zapis broja 26 – čitanje obrnuto od
                redoslijeda izračuna znamenki)
```

Algoritam 5.2.

Varijabla `i` predstavlja indekse elemenata polja `bin[]` koji će sadržavati ostatke pri dijeljenju vrijednosti broja `b` s brojem 2. Nakon unosa broja, njegova se vrijednost čuva u dvije varijable `org` i `b`. Prva varijabla neće mijenjati svoju vrijednost u ostatku programa, dok druga hoće jer se koristi u algoritmu izračuna binarnoga broja. Izračun binarnoga broja (izračun binarne znamenke i izračun količnika) izvodi se u tijelu petlje while (linija 10.) sve dok je ispunjen uvjet da je količnik različit od nula (`b!=0`).

Nakon završetka `while` petlje, izvodi se petlja `for` (linija 16.) kojom se ispisuje izračunati binarni broj. Vodeći se bit binarnoga broja nalazi na zadnjem indeksu polja. Stoga ispis treba biti takav da se elementi polja ispisuju od najvećega prema najmanjem indeksu.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. main(){
 4.   int i=0,b,org;
 5.   int bin[30];
 6.   printf("Unesite cijeli pozitivan broj u dekadskom
 7.          sustavu:\t");
 8.   scanf("%d",&b);
 9.   org=b;
10.   while (b!=0){
11.     bin[i] = b % 2;
12.     b = b / 2;
13.     i++;
14.   }
15.   printf("\nBroj %d u dekadskom je ",org);
16.   for(i-- ; i >= 0 ; i--){
17.     printf("%d",bin[i]);
18.   }
19.   printf(" u binarnom.");
20.   getch();}
```

Program 5.40.

Računalni program 5.41. pretvara uneseni binarni zapis broja u njegov dekadski oblik. Postupak pretvaranja binarnoga zapisa broja u dekadski zapis izvodi se množenjem znamenki binarnoga broja s odgovarajućim težinskim faktorom. Zbroj dobivenih umnožaka predstavlja dekadski zapis binarnoga broja. Primjera radi, neka je dan binarni broj 11010. Slijedi postupak (algoritam):

```
1. 0·2^0 = 0·1 = 0   (umnožak binarne znamenke i težinskog faktora)
2. 1·2^1 = 1·2 = 2   (umnožak binarne znamenke i težinskog faktora)
3. 0·2^2 = 0·4 = 0   (umnožak binarne znamenke i težinskog faktora)
4. 1·2^3 = 1·8 = 8   (umnožak binarne znamenke i težinskog faktora)
5. 1·2^4 = 1·16 = 16 (umnožak binarne znamenke i težinskog faktora))
6. 0+2+0+8+16 = 26 (suma umnožaka predstavlja broj u dekadskom
                    zapisu)
```

Algoritam 5.3.

Računalni program zahtijeva unos binarnoga broja kao niza od najviše 20 znakova (linija 6. i 9.). Varijabla `k` se koristi za čuvanje eksponenta na koji se broj 2 treba potencirati kako bi se dobio odgovarajući težinski faktor.

Izračun umnožaka binarne znamenke i njezina težinskoga faktora te sumiranje dobivenih umnožaka izvode se u `for` petlji (linija 12.). Varijabla `ch` se koristi u izračunu umnožaka (ona predstavlja binarnu znamenku u tipu podatka `int`) jer se uneseni binarni broj koji se čuva u polju `a` (tip podatka `char`) sastoji iz znakova koji se kao takvi ne mogu koristiti u algebarskim izrazima (u prikazanom primjeru unutar umnožaka binarnih znamenki i težinskih faktora). Stoga se umjesto znaka iz polja `a` koristi broj iz varijable `ch` (linija 13., 14. i 15.).

Treba uočiti da se u tijelu petlje koristi funkcija `pow` (potrebna zaglavna datoteka `math.h`) koja vraća rezultat potenciranja broja na neki eksponent (broj je 2, a eksponent se mijenja u ovisnosti o težinskome faktoru koji pak ovisi o rednomu broju binarne znamenke u binarnome broju).

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. #include<string.h>
 4. #include<math.h>
 5. main(){
 6.    char a[20];
 7.    int b=0,i=0,k,ch;
 8.    printf("Unesi broj koji se treba prevest u dekadski: ");
 9.    scanf("%s",a);
10.    k=strlen(a)-1;
11.    printf("\n broj znamenki je %d\n",strlen(a));
12.    for(i=0;i<strlen(a);i++){
13.       if(a[i]=='0') ch=0;
14.       else ch=1;
15.       b=b+(ch*pow(2,k));
16.       printf("%d\n",b);
17.       k--;
18.    }
19.    printf("%s u binarnom zapisu je %d u dekadskom zapisu",a,b);
20.    getch();}
```

Program 5.41.

Naredni računalni program također rješava problem pretvorbe binarnoga zapisa broja u njegov dekadski zapis. No sada se za čuvanje korisničkoga broja koristi varijabla tipa podatka `int`. Dolazak do pojedine znamenke binarnoga broja izvodi se prema opisanom algoritmu (algoritam 5.1.), odnosno na način kao da se radi o dekadskome broju.

Petlja `for` se koristi za "dekadsko" izdvajanje znamenki (linija 12.) koje se potom množe odgovarajućim težinskim faktorom i pribrajaju varijabli `dek` (linija 13.). Na kraju se ispisuje početni binarni broj (sačuvan u varijabli `pamti`) i njegov dekadski zapis (sačuvan u varijabli `dek`) – linija 17.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. #include<string.h>
 4. #include<math.h>
 5. main(){
 6.    int bin;
 7.    int dek=0,i=0,zn,pamti;
 8.    printf("Unesi binarni broj koji se treba prevest u
              dekadski zapis: ");
 9.    scanf("%d",&bin);
10.    pamti=bin;
11.    while(bin>0){
12.       zn=bin%10;
```

```
13.      dek+=zn*pow(2,i);
14.      i++;
15.      bin=bin/10;
16.   }
17.   printf("%d u binarnom zapisu je %d u dekadskom zapisu",
            pamti,dek);
18.   getch();}
```

Program 5.42.

Sljedeći računalni program traži od korisnika da pogodi slučajno generirani broj. Korisniku je na raspolaganju pet pokušaja. Kontrola broja pokušaja implementirana je preko petlje `while`. U slučaju pogotka, zastavica `pogodak` se podiže (postavlja se na vrijednost 1), ispisuje se poruka i prekida se izvođenje petlje (linija 16., 17., 18. i 19.). U ostalim se slučajevima ispisuje odgovarajuća poruka (linija 15. i 22.). Nakon završetka petlje provjerava se je li zastavica spuštena (`pogodak==0`). Ako jest, to znači da broj nije pogođen te da treba ispisati drugu poruku.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include <stdlib.h>
 4. #include <time.h>
 5. main(){
 6.    int zamisljeni, broj, pokusaj=0, pogodak=0;
 7.    srand( (unsigned)time( NULL ) );
 8.    zamisljeni = (rand() % 20)  +  1;
 9.    while(pokusaj < 5){
10.       printf("\n%d.Pokusaj:  ",pokusaj+1);
11.       printf("Upisi broj od 1-20: ");
12.       scanf("%d", &broj);
13.       pokusaj = pokusaj + 1;
14.       if(broj>zamisljeni)
15.          printf("Uneseni broj je veci od zamisljenoga.\n");
16.       else if(broj==zamisljeni){
17.          pogodak=1;
18.          printf("\nBravo pogodili ste u samo %d pokusaja,
                zamisljeni broj je bio %d",pokusaj,zamisljeni);
19.          break;
20.       }
21.       else
22.          printf("Uneseni broj je manji od zamisljenoga.\n");
23.    }
24.    if(pogodak==0)
25.       printf("\n U %d pokusaja niste uspjeli pogoditi broj koji
                iznosi:%d",pokusaj,zamisljeni);
26.    getch();}
```

Program 5.43.

Računalni program koji slijedi (program 5.44.) omogućava unos cijelih brojeva sve do trenutka kada su unesena tri parna broja. Nakon toga se ispisuje koliko je ukupno brojeva uneseno te koliko je od njih parnih (treba ih biti tri), a koliko neparnih. Ponavljajući je unos brojeva implementiran preko `do-while` petlje koja ima

svojstvo da se prvo izvodi tijelo petlje pa se tek onda provjerava ispunjenost uvjeta za ponovno izvođenje (petlja s izlaskom na dnu).

Varijabla `count` koristi se za prebrojavanje onih unesenih brojeva koji su parni i o njezinoj vrijednosti ovisi kada će se prekinuti ponavljanje izvođenja tijela petlje (linija15.). Varijabla `ukupno` se koristi za prebrojavanje svih unesenih brojeva.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include <math.h>
 4. main(){
 5.   int unos;
 6.   int ukupno=0;
 7.   int count=0;
 8.   do{
 9.     printf("\nUNESITE BROJ:\t");
10.     scanf("%d",&unos);
11.     ukupno++;
12.     if (unos % 2 == 0){
13.        count++;
14.     }
15.   }while(count < 3);
16.   printf("\n Unijeli se %d brojeva, od kojih je %d parnih
            i %d neparnih.",ukupno,count,ukupno-count);
17.   getch();}
```

Program 5.44.

Sljedeći računalni program traži od korisnika unos brojeva dok se ne unese broj 0. Nakon svakoga unosa računa se i ispisuje trenutna suma i trenutni broj unesenih brojeva (linija 14., 15. i 16.). Nakon izvođenja petlje, ispisuje se ukupna suma i prosjek unesenih brojeva.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. #include <math.h>
 4. main(){
 5.   int unos;
 6.   int buff=0;
 7.   int count=0;
 8.   float prosjek;
 9.   do{
10.     printf("\nUNESITE BROJ:\t");
11.     scanf("%d",&unos);
12.     buff = buff + unos;
13.     count++;
14.     printf("\n(buff = %d) | (count = %d)\n",buff,count);
15.   }while(unos != 0);
16.   prosjek = (float) buff / count;
17.   printf("\nZbroj unesenih brojeva je %d",buff);
18.   printf("\nProsjek unesenih brojeva je %.2f",prosjek);
```

```
19.   getch();}
```

Program 5.45.

Računalni program koji slijedi omogućava pogađanje broja koji je unio neki korisnik. Kako bi uneseni broj bio nevidljiv igraču, koristi se naredba `system("cls")` koja briše ekran. Igrač treba pogoditi broj unutar šest pokušaja. Ograničenje od šest pokušaja implementirano je petljom `do-while`. Za izlaz iz petlje u slučaju pogotka broja ne koristi se naredba `break`, već je to jedan od uvjeta izvođenja petlje (`x!=y`) gdje se u varijabli `x` čuva broj koji treba pogoditi, a u `y` broj koji je igrač unio.

```
 1. #include <stdio.h>
 2. #include <stdlib.h>
 3. #include <conio.h>
 4. main() {
 5.   int x,y,brojac=0,uspjeh=0;
 6.   scanf("%d",&x);
 7.   system("cls");
 8.   do {
 9.     printf ("\nUnesite  %d. broj: ",brojac+1);
10.     scanf("%d",&y);
11.     brojac++;
12.     if (x > y)
13.       printf ("Broj je veci od zamisljenoga");
14.     else if (x < y)
15.       printf ("Broj je manji od zamisljenog");
16.     else
17.     {
18.       printf ("Pogodili ste broj");
19.       uspjeh=1;
20.    }
21.   }while (x != y && brojac<6);
22.   if(uspjeh==1)
23.     printf("\nUspjeli ste pogoditi broj u %d pokusaja od
              maksimalnih 6",brojac);
24.   else
25.     printf("\nNiste uspjeli pogoditi broj u 6 pokusaja.");
26.   getch();}
```

Program 5.46.

U poglavlju 2., Varijable, spomenulo se postojanje dvodimenzionalnih, trodimenzionalnih i višedimenzionalnih polja. U osnovi, dimenziju polja određuje broj indeksa kojima se identificira element polja. Dvodimenzionalno polje od m×n elemenata predstavlja tablicu koja ima m redaka i n stupaca. Zorno se dvodimenzionalno polje može prikazati razmještajem sjedećih mjesta u kino dvorani. Element polja je sjedeće mjesto, a njegova identifikacija je redak u kome se nalazi i pozicija u tome retku (ovo vrijedi ako svaki redak ima jednak broj sjedećih mjesta). Ponekad se dvodimenzionalno polje naziva i dvodimenzionalna matrica. Opći je oblik deklaracije dvodimenzionalnoga polja dan u nastavku.

```
Tip_podatka ime_Varijable[Broj_redaka][Broj_stupaca];
```

Do sada je prikazana primjena jedne `for` petlje u radu s jednodimenzionalnim poljem. Ona se koristi kako bi se mijenjala vrijednost indeksa elementa polja. Za rad s dvodimenzionalnim poljem potrebne su dvije `for` petlje (ugnježđene), jedna za indekse retka, a druga za indekse stupca.

Primjer računalnoga programa koji slijedi prikazuje dvodimenzionalno polje čiji su elementi slučajno generirani brojevi. Korisnik definira dimenziju dvodimenzionalnoga polja. Potom se kao elementi polja (tablice, matrice) postavljaju slučajno generirani brojevi u intervalu od 1 do 10. Nakon definiranja vrijednosti polja, izračunava se i ispisuje suma svih njegovih elemenata, pronalazi se i ispisuje najveći element te, ako je matrica kvadratna (broj redaka je jednak broju stupaca), ispisuju se elementi glavne dijagonale (elementi kod kojih su oba indeksa jednaka).

Nakon unosa dimenzije matrice (linija 8. i 10.) deklarira se dvodimenzionalno polje odgovarajuće dimenzije. Prva ugniježđena `for` petlja (linija 14.) koristi se kako bi se slučajno generirani broj smjestio na odgovarajuću poziciju u matrici (linija 16.) te kako bi se izračunala suma svih elemenata matrice (linija 17.). Narednom ugniježđenom `for` petljom (linija 21.) ispisuju se elementi dvodimenzionalnoga polja. Posljednja `for` petlja (linija 32.) koja se izvodi samo onda ako se radi o kvadratnoj matrici (linija 30.) ispisuje elemente na glavnoj dijagonali matrice.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. #include<stdlib.h>
 4. #include<time.h>
 5. main(){
 6.   int m,n;
 7.   printf("Unesite koliko redaka ima matrica:");
 8.   scanf("%d",&m);
 9.   printf("Unesite koliko stupaca ima matrica:");
10.   scanf("%d",&n);
11.   int matrica[m][n];
12.   srand(time(0));
13.   int i,j,suma=0,max;
14.   for(i=0;i<m;i++){
15.     for(j=0;j<n;j++){
16.       matrica[i][j]=rand()%10+1;
17.       suma+=matrica[i][j];
18.     }
19.   }
20.   max=matrica[1][1];
21.   for(i=0;i<m;i++){
22.     for(j=0;j<n;j++){
23.       if(matrica[i][j]>max)max=matrica[i][j];
24.       printf("%d\t",matrica[i][j]);
25.     }
```

```
26.       printf("\n");
27.     }
28.     printf("Suma svih elemenata matrice iznosi: %d",suma);
29.     printf("\nNajveci element matrice je %d",max);
30.     if(m==n){
31.       printf("\nElementi na dijagonali su: ");
32.       for(i=0;i<m-1;i++){
33.         printf("%d,",matrica[i][i]);
34.       }
35.       printf("%d",matrica[i][i]);}
36.     else
37.       printf("\nMatrica nije kvadratna pa nema glavnu
                  dijagonalu");
38.     getch();}
```

Program 5.47.

Sljedeći računalni program omogućava korisniku unos imena i prezimena koji su odvojeni posebnim znakom podcrte (_). Program ispisuje koliko slova ima u imenu, a koliko u prezimenu. Niz znakova (string) unosi se u 8. liniji. U `for` petlji se izračunava koliko slova ima do znaka podcrte (_) – linija 10. i 11. Kada se u nizu znakova otkrije podcrta, prekida se izvođenje petlje (linija 13.). Program završava ispisom rezultata.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. #include<string.h>
 4. main(){
 5.   char ime[40];
 6.   int br=0,i;
 7.   printf("Unesite ime i prezime odvojene znakom _: ");
 8.   scanf("%s",ime);
 9.   for(i=0;i<=strlen(ime)-1;i++){
10.     if(ime[i]!='_')
11.       br++;
12.     else
13.       break;
14.   }
15.   printf("Duljina imena je %d slova",br);
16.   printf("\nDuljina prezimena je %d",strlen(ime)-br-1);
17.   getch();}
```

Program 5.48.

Računalni program koji slijedi provjerava je li unesena riječ palindrom. Palindrom je riječ koja je simetrična, odnosno koja ima jednako značenje pri čitanju s desna i s lijeva (oko, ana, neven itd.). Nakon unosa riječi, primjenom `while` petlje, provjeravaju se odgovarajuća slova u riječi. Ako su slova u riječi, na odgovarajućim pozicijama, različita, zastavica se spušta (linija 13.) što označava da riječ nije palindrom. U tome se slučaju izvođenje petlje prekida (u uvjetu petlje stoji

`flag==1`). Nakon završetka izvođenja petlje, a u ovisnosti o tome je li zastavica podignuta ili ne (linija 16. i 18.) izvodi se ispis poruke.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  #include <string.h>
 4.  main(){
 5.    char s[30];
 6.    int i,flag;
 7.    i=0;
 8.    printf("Unesite rijec: ");
 9.    scanf("%s", s);
10.    flag = 1;
11.    while ((i<(strlen(s)/2)) && (flag==1)){
12.      if(s[i]!=s[strlen(s)-i-1]){
13.        flag = 0;}
14.      i++;
15.    }
16.    if(flag==1)
17.      printf("Rijec je palindrom.\n");
18.    else
19.      printf("Rijec nije palindrom");
20.    getch();}
```

Program 5.49.

Sljedeći računalni program traži od korisnika unos znamenaka (broj od 0 do 9) sve dok korisnik ne unese znak (slovo) a. Nakon završetka unosa znamenki ispisuje se broj unosa pojedine znamenke. Broj unosa pojedine znamenke čuva se u polju od 10 elemenata čije su vrijednosti `for` petljom (linija 7.) inicijalizirane na 0 (linija 8.). Ponavljanje unosa znamenki izvodi se `while` petljom (linija 11.). Unesena se znamenka sprema u varijablu `znamenka`. Potom se provjerava je li unesena vrijednost znamenka (linija 14.) te se, ako jest, onda povećava odgovarajuća suma u polju `brojaci` (linija 14.). Do odgovarajuće se sume dolazi preko indeksa koji je upravo jednak znamenki na koju se odnosi (npr. za znamenku 0 suma se akumulira na indeksu 0 itd.). Indeks se računa tako da se ASCII kod unesenoga znaka umanji za ASCII kod znaka '0' (`znamenka-'0'`) – linija 14. Nakon završetka unosa znamenki `for` petljom (linija 16.) ispisuje se znamenka i njezin broj unosa.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int i ;
 5.    char znamenka;
 6.    int brojaci[10];
 7.    for(i=0;i<10;i++){
 8.      brojaci[i] = 0;
 9.    }
10.    printf("Pocnite unositi znamenke.\n");
11.    while(znamenka !='a'){
12.      scanf(" %c",&znamenka);
```

```
13.       if (znamenka >= '0' && znamenka <= '9')
14.          brojaci[znamenka-'0']++;
15.    }
16.    for(i = 0; i < 10; i++){
17.       printf("Znamenku %d se unijelo -- %d puta\n",i,brojaci[i]);
18.    }
19.    getch();}
```

Program 5.50.

Računalni program koji slijedi prikazuje rad s pokazivačima na polje. Deklarirano je polje od pet elemenata koje je i inicijalizirano (linija 4.). Samo ime polja u sebi čuva adresu na kojoj se nalazi vrijednost prvoga elementa polja (dakle onoga s indeksom 0). Stoga se adresa koja je dodijeljena pokazivaču `pa` preko imena polja `a` (linija 5.) odnosi na adresu gdje se nalazi vrijednost prvoga elementa polja. Petlja `for` koja slijedi (linija 6.) ispisuje vrijednost elementa polja, adresu na kojoj se element nalazi (izračunata preko operacije zbrajanja s pokazivačima – `pa+i`, treba primijetiti da ovo zbrajanje nije poznato matematičko zbrajanje dva broja) te adresu dobivenu preko adresnog operatora (`&a[i]`). Pri ispisu vrijednosti pokazivača koristio se konverzijski znak `%p` koji adresu prikazuje u heksadecimalnome zapisu. Ako se koristi `%d`, dobit će se dekadski zapis adrese.

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  main(){
4.     int i,a[5]={1,5,2,12,-5};
5.     int *pa=a;
6.     for(i=0;i<=5;i++){
7.        printf("%d.element polja=%d i ima adresu
                %p=%p\n",i,a[i],pa+i,&a[i]);
8.     }
9.     getch();}
```

Program 5.51.

Slijedi računalni program koji prikazuje odnos veličine memorijske lokacije i određenoga tipa podatka. Deklarirana su i inicijalizirana tri polja dimenzije tri, različitih tipova podataka. Potom je izveden ispis elemenata polja i adresa memorijskih lokacija na kojima se nalaze. Kako se za polje slijedno rezerviraju memorijske lokacije, njihova se veličina može uočiti u razlici između prethodne i trenutne adrese. Elementima polja pristupa se preko pokazivača i operatora dereferenciranja (*).

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    int i,j,k;
 5.    char a[3]={'a','b','c'};
 6.    short b[]={253,124,-50};
 7.    int c[3]={2533411,1111,-5000};
 8.    char *pa=a;
```

```
 9.    short *pb=b;
10.    int *pc=c;
11.    printf("Program demonstrira odnos tipa podatka i velicine
              i memorijskih lokacija.\n");
12.    printf("U polju su unaprijed definirane tri vrijednosti
              tipa podatka char.\n");
13.    printf("Broj\t\tAdresa\n");
14.    for(i=0;i<3;i++){
15.      printf("%c\t\t%p\n",*(pa+i),pa+i);
16.    }
17.    printf("U polju su unaprijed definirane tri vrijednosti
              tipa podatka short.\n");
18.    printf("Broj\t\tAdresa\n");
19.    for(i=0;i<3;i++){
20.      printf("%d\t\t%p\n",*(pb+i),pb+i);
21.    }
22.    printf("U polju su unaprijed definirane tri vrijednosti
              tipa podatka int.\n \n");
23.    printf("Broj\t\tAdresa\n");
24.    for(i=0;i<3;i++){
25.      printf("%d\t\t%p\n",*(pc+i),pc+i);
26.    }
27.    getch();}
```

Program 5.52.

Računalni program koji slijedi koristi pokazivače na nizu znakova (polju tipa podatka `char`). Nakon deklariranja nizova znakova i njihove inicijalizacije (linija 4. i 5.) te deklaracije dvaju pokazivača (linija 6.), ispisuje se sadržaj niza znakova `a` (linija 8.). Vrijednost se pokazivača postavlja na adresu varijable `a`, odnosno adresu varijable `b` (linija 9. i 10.). Petlja `while` se koristi za prolazak kroz niz znakova `a` i njihovo kopiranje u odgovarajuće mjesto u nizu znakova `b` (linija 12., 13. i 14.). Na kraju programa ispisuje se sadržaj niza znakova `b` (linija 18.).

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  main(){
 4.    char a[30]="Programiranje u jeziku C";
 5.    char b[50]="pocetak programiranja i proba";
 6.    char *pa,*pb;
 7.    printf("Ovo je vrijednost stringa a\n");
 8.    puts(a);
 9.    pa=a;
10.    pb=b;
11.    while(*pa!='\0'){
12.      *pb=*pa;
13.      pa++;
14.      pb++;
15.    }
16.    *pb='\0';
```

```
17.    printf("Ovo je vrijednost stringa b, nakon kopiranja
               znak po znak iz stringa a\n");
18.    puts(b);
19.    getch();}
```

Program 5.53.

5.2.1. Zadaci za vježbu

1. Napisati program koji za upisani broj ispisuje sve brojeve do toga broja i pored njih ispisuje riječ PARAN ili NEPARAN u ovisnosti o tome ako je ispisani broj paran ili neparan. Nula nije paran niti neparan broj.
2. Izraditi program kojim se unose brojevi sve dok se ne unese broj 1. Nakon završetka unosa potrebno je izračunati i ispisati sumu svih parnih i umnožak svih neparnih brojeva te broj unesenih brojeva.
3. Napisati program koji od korisnika očekuje unos 10 brojeva i izračunava sumu, prosječnu vrijednost, maksimalnu i minimalnu vrijednost.
4. Napisati program koji će od korisnika tražiti unos `n` pozitivnih brojeva. Korisnik prvo unosi broj `n`. Program će se zaustaviti ukoliko je unesen negativan broj. Inače će izračunati i ispisati zbroj unesenih brojeva.
5. Napisati program koji traži od korisnika željeni broj unosa te nakon toga traži unos toliko brojeva. Po unosu ispisuje sumu brojeva, srednju vrijednost te najveći i najmanji do tada uneseni broj. Koristiti polja.
6. Izraditi program koji će tražiti unos dvaju brojeva te izračunati i ispisati zbroj svih brojeva između unesenih brojeva, uključujući i te brojeve.
7. Prethodni program proširiti tako da se traži ponavljanje unosa sve dok prvi uneseni broj ne bude manji od drugoga.
8. Napisati program koji za upisani broj ispisuje sve brojeve od 1 do toga broja koji su djeljivi s 3 te pored broja ispisuje poruku DJELJIV S 3 i brojeve koji su djeljivi sa 7 te pored toga broja ispisuje poruku DJELJIV SA 7. Ukoliko je broj djeljiv s oba broja, treba ispisati obje poruke. Npr.
   ```
   6 DJELJIV S 3
   14 DJELJIV SA 7
   21 DJELJIV S 3 DJELJIV SA 7
   ```
9. Napisati program za učenje množenja. Program generira dva slučajna cijela broja od 0 do 30 i ispisuje ih na ekran. Od korisnika se traži unos produkta ispisana dva broja sve dok se ne unese točan produkt. Na kraju je potrebno ispisati broj pokušaja.
10. Izraditi program koji ispisuje tri stupca: prvi stupac su brojevi od 10-100 ali svaki peti, drugi stupac su njihove polovice, a treći njihovi kubovi. Potrebno je ispisati i naslovni redak:
    ```
    Broj          Polovica             KUB
    ```

11. Napisati program koji će na ekranu ispisati trokut načinjen od zvjezdica. Korisnik treba unijeti broj redaka. Izgled trokuta od 5 redaka je:
```
*
**
***
****
*****
```

12. Napisati program kojim se unose brojevi dok se ne unesu točno 3 parna ili točno 3 neparna broja (0 ne uzeti u obzir). Nakon završetka unosa, potrebno je ispisati sumu svih parnih i sumu svih neparnih brojeva.

13. Napisati program kojim korisnik unosi PIN od četiri znamenke (npr. 0451). Unutar programa u varijablu `kod` spremiti ispravni PIN. Korisnik ima tri pokušaja kako bi unio ispravan PIN. U slučaju ispravnoga unosa ispisati poruku `Otkljucano`. Ako korisnik unutar tri pokušaja nije unio ispravan PIN, ispisati poruku `Zakljucano`.

14. Napisati program kojim se unose dva niza znakova (dva stringa). Nakon unosa, program treba ispisati onaj niz znakova koji ima više suglasnika (samoglasnici su a, e, i ,o ,u, A, E, I, O, U).

15. Napisati program koji generira i ispisuje matricu dimenzije 10×10 u kojoj su svi elementi prvoga reda 1, svi elementi drugoga reda 2, svi elementi trećega reda 3 itd. Generiranje matrice napraviti pomoću petlji.

16. Napisati program u koji se unosi matrica dimenzije m×n. Korisnik unosi dimenziju matrice. Potrebno je izračunati i ispisati sumu svih elemenata u matrici. Program također ispisuje sumu svakog pojedinog reda te pronalazi i ispisuje red u kojem je suma najmanja.

17. Napisati program koji uz pomoć petlji ispisuje sljedeće brojeve:
```
1   3   5   7   9   11  13  15  17  19
2   4   6   8   10  12  14  16  18  20
1   3   5   7   9   11  13  15  17  19
2   4   6   8   10  12  14  16  18  20
1   3   5   7   9   11  13  15  17  19
2   4   6   8   10  12  14  16  18  20
```

18. Napisati program koji ispisuje sljedeći uzorak. Ispis izvesti preko petlje.
```
000000
000001
000011
000111
001111
011111
111111
```

19. Napisati program koji generira jediničnu matricu n-tog reda (n×n – korisnik unosi dimenziju matrice). Jedinična matrica ima 1 na svim mjestima gdje je broj retka jednak broju stupca, a na ostalim mjestima ima 0. Primjera radi, jedinična matrica dimenzije 4×4 izgleda kako slijedi:
```
1000
0100
0010
0001
```

20. Napisati program za transponiranje matrice. Dimenziju matrice kao i njezine elemente unosi korisnik. Transponirana matrica se dobije tako da redci postanu stupci, odnosno prvi redak postaje prvi stupac, drugi redak drugi stupac itd. Primjer transponirane matrice:
```
12          135
34          246
56
```

21. Napisati program kojim se stvara matrica dimenzije n×n (korisnik unosi dimenziju matrice i njezine elemente). Nakon unosa, program izračunava traga matrice. Trag matrice je zbroj elemenata na glavnoj dijagonala. Elementi na glavnoj dijagonali su oni elementi koji imaju jednake indekse.

22. Napisati program koji će koristiti strukturu `Artikl` koja će imati dvije komponente – `sifra` i `cijena`. Korisnik treba unijeti pet artikala (koristiti petlju). Nakon unosa program treba pronaći i ispisati artikl (šifru i njegovu cijenu) s najmanjom cijenom.

23. Napravite program koji od korisnika traži unos imena (bez prezimena). Nakon unosa program treba ispisati iz koliko slova se sastoji ime, koliko ima samoglasnika i suglasnika te ime treba ispisati naopako s desna na lijevo.

24. Napisati program kojim se u ispisu znamenke pomiču za jedno mjesto. Ispis izvesti primjenom petlje. Izgled ispisa je kako slijedi:
```
123456789
912345678
891234567
789123456
678912345
567891234
456789123
345678912
234567891
```

25. Napisati program koji utvrđuje je li broj koji je korisnik unesao prost. Korisnik unosi cijeli broj. Ako uneseni broj nije prost, program treba ispisati sve brojeve s kojima je uneseni broj djeljiv. Broj je prost ako je djeljiv s 1 i samim sobom.

26. Napisati program u kome se traži unos dvoznamenkastoga broja. Korisnik treba unositi broj sve dok se ne unese broj čija je suma znamenki jednaka 5 (npr. 14,23,…). Program treba ispisati broj pokušaja.

27. Napisati program koji ispisuje prikazanu strukturu. Koristiti petlje.

```
++++*++++
+++*|*+++
++**|**++
+***|***+
****|****
```

28. Napisati program koji generira matricu dimenzije 10×10. U njoj se u neparnim redovima nalaze znakovi 'N', a u parnim 'P'. Na glavnoj dijagonali se nalaze znakovi 'X'. Prvi red u matrici, premda ima indeks 0 tretirati kao prvi red, odnosno neparni red. Generiranu matricu program treba ispisati.

29. Napisati program koji će ispisati trokut sastavljen iz zvjezdica. Korisnik unosi broj zvjezdica u zadnjem retku trokuta. Broj treba biti neparan. Primjer ispisa za trokut sa sedam zvjezdica u zadnjem retku:

```
   *
  ***
 *****
*******
```

30. Napisati program koji definira strukturu točke u koordinatnome sustavu. Struktura mora sadržavati `x` i `y` koordinatu točke. Napisati program kojim se unosi niz od n točaka (korisnik unosi broj točaka). Za svaku točku se unosi x i y koordinata. Nakon unosa svih točaka, korisnik unosi dva broja koji predstavljaju indeks neke dvije unesene točke. Program potom izračunava i ispisuje udaljenost dviju točaka koje se nalaze na unesenim indeksima niza. Udaljenost se točaka računa po formuli $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

6. ALGORITAMSKA STRUKTURA BEZUVJETNOGA SKOKA U C

Ovo poglavlje prikazuje **algoritamsku strukturu bezuvjetnoga skoka** realiziranu u programskome jeziku C. Prikazane su naredbe `exit`, `break`, `continue`, `return` i `goto`. Prikazan je opći oblik ovih naredbi te specifični primjeri njihove primjene.

Po završetku ove nastavne cjeline moći će se:

- objasniti naredbu `continue` i primijeniti ju u cikličkoj algoritamskoj strukturi
- objasniti naredbu `break` i primijeniti ju u cikličkoj i razgranatoj algoritamskoj strukturi
- objasniti i primijeniti naredbu `goto`
- objasniti i primijeniti naredbu `exit`
- objasniti i primijeniti naredbu `return`.

6.1. Realizacija algoritamske strukture bezuvjetnoga skoka u C

Algoritamska struktura bezuvjetnoga skoka je takva struktura koja omogućava bezuvjetno preskakanje niza naredbi. Naredna naredba koja će biti izvedena ovisi o korištenoj naredbi skoka. U svrhu realizacije algoritamske strukture bezuvjetnoga skoka, u programskome jeziku C koriste se `break`, `continue`, `return`, `exit` i `goto`.

6.1.1. Naredba `break`

Naredba `break` koristi se u sklopu `switch–case` naredbe te naredbe za realizaciju cikličke algoritamske strukture (`for`, `while` i `do–while`). U nastavku su prikazani opći oblici primjene naredbe `break`.

```
 1.  ...
 2.  naredbaX
 3.  switch (izraz){
 4.  case konstanta_1:
 5.    blok_naredbi_1
 6.    break;
 7.  ...
 8.  default:
 9.    blok_naredbi_x
10.  }
11.  naredbaY
12.  ...
```

Pri `switch–case` naredbi, `break` se nalazi u sklopu nekoga bloka naredbi (u `default` nije potrebna naredba `break` jer se i tako završetkom `bloka_naredbi_x` izlazi iz cijeloga bloka `switch–case`). Izvođenje naredbe `break` uzrokuje preskakanje svih narednih naredbi u sklopu strukture `switch–case` te izvođenje naredbe `naredbaY`.

```
 1.  ...
 2.  naredbaX
 3.  for(inicijalizacija; logički_izraz; korak){
 4.    naredbe_1;
 5.    if (if_logički_izraz)
 6.       break;
 7.    naredbe_2;
 8.  }
 9.  naredbaY
10.  ...
```

```
 1.  ...
 2.  naredbaX
 3.  while(logički_izraz){
 4.    naredbe_1;
 5.    if (if_logički_izraz)
```

```
 6.      break;
 7.    naredbe_2;
 8.  }
 9.  naredbaY
10.  ...
```

```
 1.  ...
 2.  naredbaX
 3.  do  {
 4.    naredbe_1;
 5.    if (if_logički_izraz)
 6.      break;
 7.    naredbe_2;
 8.  } while(logički_izraz);
 9.  naredbaY
10.  ...
```

U svim trima realizacijama cikličke algoritamske strukture, naredba `break` uzrokuje prekid izvođenja tijela petlje, izlazak iz petlje i izvođenje one naredbe koja slijedi nakon petlje. U prikazanim primjerima to je `naredbaY`. Naredba `break` bit će izvedena ako je zadovoljen uvjet `if_logički_izraz`. U tome slučaju bit će preskočeno izvođenje `naredbe_2`. Dakle, `naredbe_2` bit će izvedene samo onda kada `if_logički_izraz` nije istinit.

Sljedeći primjer prikazuje primjenu naredbi `switch–case` i `break` u programu kojim se ispisuje koliko uneseni mjesec (njegov redni broj) ima dana.

```
 1.  #include<stdio.h>
 2.  main(){
 3.    int redniBrojMjeseca;
 4.    do{
 5.      printf("\nUnesite redni broj mjeseca: ");
 6.      scanf("%d", &redniBrojMjeseca);
 7.      if(!(redniBrojMjeseca >=1 && redniBrojMjeseca <= 12))
 8.        printf ("\n\aRedni broj mjeseca treba biti u intervalu od
                1 do 12.");
 9.      else
10.        break;
11.    }
12.    while(1==1);
13.    switch(redniBrojMjeseca){
14.      case 1 :
15.      case 3 :
16.      case 5 :
17.      case 7 :
18.      case 8 :
19.      case 10 :
20.      case 12 :
21.        printf("\n\nMjesec pod rednim brojem %d. ima 31
                dan.", redniBrojMjeseca);
22.        break;
23.      case 2 :
```

```
24.          printf("\n\nMjesec pod rednim brojem 2. ima 28 ili 29
                     dana, ovisi je li godina prijestupna ili ne.");
25.          break;
26.       case 4 :
27.       case 6 :
28.       case 9 :
29.       case 11 :
30.          printf("\n\nMjesec pod rednim brojem %d. ima 30
                     dana.", redniBrojMjeseca);
31.    }
32. getch();}
```

Program 6.1.

Izvođenje bilo koje `break` naredbe u `switch–case` bloku uzrokuje skok na liniju koda 32., odnosno na prvu naredbu koja slijedi iza `switch–case` naredbe. U zadnjemu slučaju (`case` 11 – linija koda 29.) nije stavljena `break` naredba jer nakon izvođenja pripadajućega bloka naredbi i tako slijedi izlazak iz `switch–case` bloka i izvođenje linije 32. Također se može vidjeti i primjena `break` naredbe u `do–while` petlji (linija koda 10.) koja se izvodi onda kada je unesen ispravan redni broj mjeseca. U biti, njezinim izvođenjem se prekida izvođenje tijela petlje i same petlje te se izvodi prva naredba izvan `do–while` bloka, a to je linija koda 13.

U sljedećem računalnome programu koristi se naredba `break` u `for` petlji kako bi se pronašla i ispisala vrijednost prvoga temperaturnog očitanja koje je ispod nule te redni broj toga očitanja. Ukupno je memorirano pet temperaturnih očitanja.

```
 1. #include <stdio.h>
 2. main(){
 3.    int redniBrojOcitanja;
 4.    float temperaturnoOcitanjeIspodNula;
 5.    float temperaturnaOcitanja [] = {3.5, 6.7, -3.2, -5, 0};
 6.    for (redniBrojOcitanja = 0; redniBrojOcitanja < 5;
 7.         redniBrojOcitanja ++) {
 8.      if (temperaturnaOcitanja[redniBrojOcitanja] < 0)
 9.         break;
10.     }
11.    printf ("Prvo temperaturno ocitanje ispod nula je ono rednoga
                broja %d i iznosi %.2f.", redniBrojOcitanja + 1,
                temperaturnaOcitanja[redniBrojOcitanja]);
12.    getch();}
```

Program 6.2.

U tijelu petlje `for` provjerava se je li temperaturno očitanje manje od nula i ako je, izvodi se naredba `break` (linije koda 8. i 9.). Ako se izvede naredba `break`, prekida se izvođenje tijela petlje i sama petlja te se izvodi prva naredba iza `for` petlje, a to je linija koda 11. u kojoj se ispisuje vrijednost elementa polja na indeksu koji je vrijedio u tijelu petlje prije izvođenja `break` naredbe.

Sljedećim se programom unosi redni broj mjeseca te se ispisuje njegov broj dana. Unos se ponavlja dok god se ne unese nula koja označava kraj programa.

```
 1.  #include<stdio.h>
 2.  main(){
 3.     int redniBrojMjeseca;
 4.     while (1==1){
 5.        do{
 6.           printf("\nUnesite redni broj mjeseca: ");
 7.           scanf("%d", &redniBrojMjeseca);
 8.           if (!(redniBrojMjeseca >=0 && redniBrojMjeseca <= 12))
 9.              printf ("\n\aRedni broj mjeseca treba biti u intervalu
                       od 1 do 12; 0 je izlaz iz programa.");
10.           else
11.              break;
12.        }
13.        while(1==1);
14.        if (redniBrojMjeseca == 0) break;
15.        switch(redniBrojMjeseca){
16.           case 1 :
17.           case 3 :
18.           case 5 :
19.           case 7 :
20.           case 8 :
21.           case 10 :
22.           case 12 :
23.              printf("\n\nMjesec pod rednim brojem %d. ima 31
                       dan.", redniBrojMjeseca);
24.              break;
25.           case 2 :
26.              printf("\n\nMjesec pod rednim brojem 2. ima 28 ili 29
                       dana, ovisi je li godina prijestupna ili ne.");
27.              break;
28.           case 4 :
29.           case 6 :
30.           case 9 :
31.           case 11 :
32.              printf("\n\nMjesec pod rednim brojem %d. ima 30
                       dana.", redniBrojMjeseca);
33.        }
34.     }
35.     printf("Kraj programa.");
36.     getch();}
```

Program 6.3.

Ključni dio ovoga programa je linija 14. koja provjerava je li uneseni broj za redni broj mjeseca 0. Ako jest, izvodi se `break` naredba. Treba uočiti da se ova `break` naredba odnosi na prvu `while` petlju (linija koda 4.) te da se njezinom izvedbom događa skok na prvu naredbu izvan `while` petlje. Ta se naredba nalazi na liniji koda 35.

Sljedećim se programom od korisnika traži unos svih samoglasnika. Nakon što su uneseni svi samoglasnici, program ispisuje broj pokušaja.

```
 1.  #include<stdio.h>
 2.  main(){
 3.     char samoglasnici[] = {'A', 'E', 'I', 'O', 'U'};
```

```
 4.    char uneseniSamoglasnici[5], samoglasnik;
 5.    int brojSamoglasnika = 5, indeksUnesenogSamoglasnika = 0;
 6.    int brojPokusaja = 0, brojac;
 7.    char jeLiSamoglasnik, jeLiSamoglasnikVecUnesen;
 8.    while (1==1){
 9.       brojPokusaja++;
10.       printf("Unesite samoglasnik (%d. pokusaj): ", brojPokusaja);
11.       scanf(" %c", &samoglasnik);
12.       jeLiSamoglasnik = '0';
13.       jeLiSamoglasnikVecUnesen = '0';
14.       for (brojac = 0; brojac < brojSamoglasnika; brojac ++){
15.          if (samoglasnik == samoglasnici[brojac] ||
                samoglasnik - 32 == samoglasnici[brojac]){
16.             jeLiSamoglasnik = '1';
17.             break;
18.          }
19.       }
20.       if (jeLiSamoglasnik == '1'){
21.          for (brojac = 0; brojac < brojSamoglasnika; brojac ++){
22.             if (samoglasnik == uneseniSamoglasnici[brojac] ||
                    samoglasnik - 32 == uneseniSamoglasnici[brojac]){
23.                jeLiSamoglasnikVecUnesen = '1';
24.                break;
25.             }
26.          }
27.          if(jeLiSamoglasnikVecUnesen == '0'){
28.             uneseniSamoglasnici[indeksUnesenogSamoglasnika] =
                samoglasnik;
29.             if(indeksUnesenogSamoglasnika == 4)
30.                break;
31.             else
32.                indeksUnesenogSamoglasnika++;
33.             }
34.          }
35.    }
36.    printf("\n\nUnijeli ste sve samoglasnike. Ukupan broj pokusaja
              je %d.", brojPokusaja);
37.    printf("\n\nSamoglasnike ste unijeli ovim redoslijedom: ");
38.    for (brojac = 0; brojac < brojSamoglasnika; brojac ++){
39.       printf ("%c", uneseniSamoglasnici[brojac]);
40.       if (brojac < brojSamoglasnika - 1) printf (", ");
41.    }
42.    getch();}
```

Program 6.4.

U prethodnome primjeru treba uočiti niz `break` naredbi od kojih svaka pripada nekoj petlji (npr. `break` u liniji 17. pripada `for` petlji iz linije 14.). Ovdje treba obratiti pažnju kako se provjerava je li samoglasnik unesen kao veliko ili kao malo slovo (pomak u ASCII kodu između velikih i malih slova je 32) – linija 15. i 22.

6.1.2. Naredba `continue`

Naredba `continue` koristi se u sklopu naredbi za realizaciju cikličke algoritamske strukture (`for`, `while` i `do–while`). U nastavku su prikazani opći oblici primjene naredbe `continue`.

```
 1.  ...
 2.  naredbaX
 3.  for(inicijalizacija; logički_izraz; korak){
 4.    naredbe_1;
 5.    if (if_logički_izraz)
 6.       continue;
 7.    naredbe_2;
 8.  }
 9.  naredbaY
10.  ...
```

```
 1.  ...
 2.  naredbaX
 3.  while(logički_izraz){
 4.    naredbe_1;
 5.    if (if_logički_izraz)
 6.       continue;
 7.    naredbe_2;
 8.  }
 9.  naredbaY
10.  ...
```

```
 1.  ...
 2.  naredbaX
 3.  do  {
 4.    naredbe_1;
 5.    if (if_logički_izraz)
 6.       continue;
 7.    naredbe_2;
 8.    }
 9.  while(logički_izraz);
10.  naredbaY
11.  ...
```

U svim trima realizacijama cikličkih algoritamskih struktura, naredba `continue` uzrokuje prekid izvođenja tijela petlje, ali ne uzrokuje izlazak iz nje (kao što to radi naredba `break`), već se izvodi izračun uvjeta `logički_izraz` (u `for` petlji se prije izvođenja izračuna ažurira `korak`), čime se nastavlja ili ne nastavlja izvođenje tijela petlje (ovisno o tome je li uvjet petlje ispunjen ili ne). Naredba `continue` bit će izvedena ako je zadovoljen uvjet `if_logički_izraz`. U tome slučaju bit će preskočeno izvođenje `naredbe_2`. Dakle, `naredbe_2` bit će izvedene samo onda kada `if_logički_izraz` nije istinit (slično kao i kod naredbe `break`).

Sljedeći program ispisuje prosječnu negativnu i prosječnu pozitivnu vrijednost temperaturnog očitanja (nula se smatra pozitivnom vrijednošću):

```
 1.  #include <stdio.h>
 2.  main(){
 3.     int redniBrojOcitanja;
 4.     int brojNegativnihOcitanja = 0, brojPozitivnihOcitanja = 0;
 5.     float sumaNegativnihOcitanja = 0, sumaPozitivnihOcitanja = 0;
 6.     float temperaturnaOcitanja [] = {3.5, 6.7, -3.2, -5, 0};
 7.     for (redniBrojOcitanja = 0; redniBrojOcitanja < 5;
              redniBrojOcitanja ++) {
 8.        if (temperaturnaOcitanja[redniBrojOcitanja] < 0){
 9.           sumaNegativnihOcitanja = sumaNegativnihOcitanja +
              temperaturnaOcitanja[redniBrojOcitanja];
10.           brojNegativnihOcitanja ++;
11.           continue;
12.        }
13.        sumaPozitivnihOcitanja = sumaPozitivnihOcitanja +
           temperaturnaOcitanja[redniBrojOcitanja];
14.        brojPozitivnihOcitanja ++;
15.     }
16.     printf ("\nProsjecna negativan vrijednost temperaturnog
                 ocitanja iznosi %.2f.", sumaNegativnihOcitanja/
                 brojNegativnihOcitanja);
17.     printf ("\nProsjecna pozitivna vrijednost temperaturnog
                 ocitanja iznosi %.2f.", sumaPozitivnihOcitanja/
                 brojPozitivnihOcitanja);
18.     getch();}
```

Program 6.5.

Naredba se `continue` ovdje koristi kako bi se preskočile linije koda 13. i 14. (jer se radi o negativnom očitanju) i kako bi se izvela obrada sljedećega očitanja u polju `temperaturnaOcitanja`. Dakle, nakon izvođenja naredbe `continue`, izvodi se `redniBrojOcitanja++` (korak u `for` petlji), a nakon toga i provjera logičkog izraza `redniBrojOcitanja<5` (obje naredbe u liniji koda 7.)

Sljedeći program izvodi provjeru je li uneseno neko veliko slovo.

```
 1.  #include<stdio.h>
 2.  main(){
 3.     char velikoSlovo;
 4.     while (1==1){
 5.        printf ("\nUnesite neko veliko slovo: ");
 6.        scanf (" %c", &velikoSlovo);
 7.        if (!(velikoSlovo >= 65 && velikoSlovo <= 90)){
 8.           printf ("\n\nNiste unijeli veliko slovo. Pokusajte ponovno.");
 9.           continue;
10.        }
11.        break;
12.     }
13.     printf("\n\nUnijeli ste veliko slovo.");
14.     getch();}
```

Program 6.6.

U primjeru se naredba `continue` (linija koda 9.) koristi kako bi se preskočila naredba `break` (linija koda 11.) te kako bi se ponovno izvela petlja, koja se izvodi jer je uvjet `1==1` uvijek ispunjen. Naredba `break` izvodi se onda kada je uneseno veliko slovo, odnosno kada logički izraz u liniji koda 7. nije zadovoljen. Velika slova imaju ASCII kod u intervalu od 65 do 90.

Sljedeći primjer, slično kao i prethodni, provjerava je li unesena neka znamenka. Znamenke imaju ASCII kod u intervalu od 48 do 57.

```
 1. #include<stdio.h>
 2. main(){
 3.    char znamenka;
 4.    do {
 5.       printf ("\nUnesite neku znamenku: ");
 6.       scanf (" %c", &znamenka);
 7.       if (!(znamenka >= 48 && znamenka <= 57)){
 8.          printf ("\n\nNiste unijeli znamenku. Pokusajte ponovno.");
 9.          continue;
10.       }
11.       break;
12.    }
13.    while (1==1);
14.    printf("\n\nUnijeli ste znamenku.");
15.    getch();}
```

Program 6.7.

Opis programa analogan je opisu prethodnoga programa (program 6.6.).

6.1.3. Naredba `exit`

Naredba `exit` koristi se kada se želi prekinuti izvođenje cijeloga programa. U nastavku je prikazan opći oblik primjene ove naredbe.

```
1. ...
2. naredbaX
3. exit(status);
4. naredbaY
5. ...
```

Naredba `exit` može se koristiti u bilo kome dijelu programskoga koda. Ona trenutno prekida izvođenje programa. U gornjem, općem obliku primjene naredbe, `naredbaY` se nikada neće izvesti. `Status` predstavlja neku vrijednost koja će se vratiti procesu koji je pokrenuo program. U primjerima prikazanim u ovoj knjizi to je proces u operacijskome sustavu, ali to uvijek ne mora biti slučaj, budući da neki korisnički program može biti taj koji je pozvao neki drugi korisnički program koji će prvome vratiti status kada se u njemu izvede naredba `exit`. Uobičajeno je vratiti vrijednost 1 ili veću, ako se želi reći da je u programu koji se prekida sve ispravno izvedeno, odnosno 0 ako se želi dojaviti neki problem iz programa koji se prekida.

U sljedećemu se primjeru provjerava je li korisnik unesao znamenku. Ako nije, pita ga se želi li ponoviti pokušaj ili prekinuti program.

```
 1.  #include<stdio.h>
 2.  main(){
 3.    char znamenka;
 4.    char ponovi;
 5.    do {
 6.      printf ("\nUnesite neku znamenku: ");
 7.      scanf (" %c", &znamenka);
 8.      if (!(znamenka >= 48 && znamenka <= 57)){
 9.        printf ("\n\nNiste unijeli znamenku. Zelite ponovno
                     pokusati (d/n)?");
10.        do{
11.          ponovi = getch();
12.          if (ponovi == 'n' || ponovi == 'N')
13.            exit(1);
14.        }
15.        while (ponovi != 'd' && ponovi != 'D' &&
                 ponovi != 'n' && ponovi != 'N');
16.        continue;
17.      }
18.      break;
19.    }
20.    while (1==1);
21.    printf("\n\nUnijeli ste znamenku.");
22.    getch();}
```

Program 6.8.

Naredba `exit` koristi se u liniji koda 13. koja se izvodi onda kada je korisnik izabrao da ne želi ponoviti pokušaj unosa znamenke (linija koda 12.). Izvođenje naredbe `exit` odmah prekida izvođenje program.

6.1.4. Naredba `goto`

Naredba `goto` omogućava eksplicitno navođenje linije koda na koju se treba skočiti i od koje se dalje treba nastaviti izvođenje programa. U nastavku je prikazan opći oblik primjene ove naredbe.

```
1.  ...
2.  naziv_labele: naredbaX
3.  naredbaY
4.  ...
5.  goto naziv_labele;
6.  naredbaZ
7.  ...
```

Iza naredbe `goto` navodi se `naziv_labele` koji predstavlja simbolički naziv linije koda na koju se želi skočiti. U prikazanom općem obliku izvođenje naredbe `goto` izazvalo bi skok i izvođenje naredbe `naredbaX`, iza koje bi se dalje izvodila `naredbaY` itd. `NaredbaZ` se nikada ne bi izvela jer je ispred nje skok. U praksi se ne preporuča primjena naredbe `goto`, budući da je iščitavanje i shvaćanje

programskoga koda u kome se nalaze naredbe `goto` izrazito otežano. To se može vidjeti iz primjera koji slijedi, a u kome je program 6.8. sada izrađen isključivo primjenom `goto` naredbe.

```
 1.  #include<stdio.h>
 2.  main(){
 3.    char znamenka;
 4.    char ponovi;
 5.    unosZnamenke: printf ("\nUnesite neku znamenku: ");
 6.    scanf (" %c", &znamenka);
 7.    if (!(znamenka >= 48 && znamenka <= 57)){
 8.      printf ("\n\nNiste unijeli znamenku. Zelite ponovno
                 pokusati (d/n)?");
 9.      ponovniPokusaj: ponovi = getch();
10.      if (ponovi == 'n' || ponovi == 'N')
11.        goto krajPrograma;
12.      if (ponovi != 'd' && ponovi != 'D' && ponovi != 'n' &&
             ponovi != 'N')
13.        goto ponovniPokusaj;
14.      goto unosZnamenke;
15.    }
16.    goto tocanUnos;
17.    if (1==1)
18.      goto unosZnamenke;
19.    tocanUnos: printf("\n\nUnijeli ste znamenku.");
20.    getch();
21.    krajPrograma: exit(1);}
```

Program 6.9.

Iako je program dobro formatiran (blokovi naredbi su podvučeni pod odgovarajuće naredbe), a nazivi labela (simboličkih imena linija koda) iskazuju semantiku linije koda, teško je pratiti sve navedene skokove. Treba uočiti da se, nakon izvođenja naredbe `goto`, izvodi ona linija koda koja ima odgovarajuće simboličko ime (npr. izvođenjem linije 13. dolazi do skoka i izvođenja linije koda 9.) te da se dalje nastavlja slijedno izvođenje narednih linija koda (nakon linije 9. izvodi se linija 10. itd.).

6.1.5. Naredba `return`

Naredba `return` omogućava povratak na onu liniju koda koja je pozvala neku korisničku proceduru. Procedure se detaljnije pojašnjavaju u narednome poglavlju. Povratkom se može vratiti i neka vrijednost iz pozvane korisničke procedure. Opći oblik primjene ove naredbe je kako slijedi.

```
1.  korisnicka_procedura(){
2.    naredba1
3.    return povratna_vrijednost;
4.    naredba2}
5.
6.  main(){
```

```
 7.   naredbaX
 8.   vrijednost = korisnicka_procedura();
 9.   naredbaY
10.   ...}
```

Nakon izvođenja naredbe `naredbaX` u glavnome dijelu programa (`main()`), poziva se `korisnicka_procedura`. Ovim pozivom provodi se izvođenje naredbe `naredba1` unutar bloka naredbi koji pripada korisničkoj proceduri. Izvođenje naredbe `return` dolazi do skoka, odnosno izlaska iz bloka naredbi korisničke procedure (`naredba2` se neće izvesti). Ako naredba `return` vraća vrijednost, tada se ona (u primjeru je to `povratna_vrijednost`) smješta u varijablu `vrijednost` (u glavnome dijelu programa) i nakon toga se izvodi `naredbaY`. Praktični primjeri ove naredbe slijede u sljedećemu poglavlju.

7. PROCEDURE U RAČUNALNOME PROGRAMU

Poglavlje opisuje osnovne pojmove koji se odnose na **procedure u računalnome programu**. Prikazana je svrha izrade procedura, način njihova definiranja i komunikacije s mjestom poziva. Prikazan je opći oblik definicije procedure i njezina poziva. Poglavlje završava praktičnim primjerima računalnih programa i zadacima za vježbu.

Po završetku ovoga poglavlja čitatelj će moći:

- definirati proceduru u računalnome kodu
- navesti i objasniti svojstva procedure
- opisati sintaksu deklaracije i definicije procedure
- objasniti što su argumenti u definiciji procedure
- objasniti proceduru sa i bez vraćanja vrijednosti
- skicirati primjer procedure koja vraća i koja ne vraća vrijednost
- definirati proceduru koja vrijednost vraća kroz argument
- objasniti pozivanje procedure
- objasniti rekurzivni poziv procedure
- objasniti trajanje vrijednosti varijable deklarirane u proceduri
- preoblikovati zadani računalni program tako da se primjenjuju procedure
- preoblikovati zadanu proceduru tako da se vrijednost vraća kroz argument, odnosno kroz povratni tip
- objasniti i primijeniti slanje polja u proceduru i njegovo vraćanje
- objasniti i primijeniti slanje zapisa u proceduru i njegovo vraćanje
- objasniti i primijeniti slanje nizova znakova u proceduru i njegovo vraćanje
- objasniti i primijeniti slanje pokazivača u proceduru i njegovo vraćanje.

7.1. Pojam procedure u računalnome programu

Za pojam procedure u računalnome se programu često koristi i pojam funkcija, rutina, potprogram, metoda (zadnji se pojam koristi u objektno orijentiranom programiranju). Procedura predstavlja imenovani skup algoritamskih koraka (naredbi) kojim se rješava neki manji problem. Izvođenje toga skupa naredbi (**tijela procedure**) izvodi se pozivom njegova imena – i to se naziva **poziv procedure**. Proceduri je moguće poslati podatke. Podaci se proceduri šalju kroz njezine **argumente** (**parametre**). Ti se podaci zatim mogu koristiti unutar naredbi tijela procedure. Također je moguće da nakon izvođenja tijela procedure, ona vrati podatke mjestu koje ju je pozvalo. Vraćanje vrijednosti je moguće kroz argumente (parametre) procedure i/ili kroz njezin **povratni tip**.

Procedure predstavljaju temeljni koncept u proceduralnom programiranju kod koga se osnovni složeni problem razbija u manje jednostavnije probleme koji se rješavaju pisanjem procedura. Potom se te procedure pozivaju, čime se rješavaju manji problemi te se, nakon riješenih svih manjih problema, rješava i početni osnovni. Neku proceduru poziva druga procedura. No, moguće je i da procedura poziva samu sebe – to se naziva **rekurzivni poziv procedure**.

Iako se naziv funkcija koristi kao sinonim proceduri, ipak postoji razlika. Naime, funkcija je matematički pojam za koji je bitna činjenica da ona prima neke podatke ali obavezno i neke podatke vraća. Dakle, može se reći da je funkcija samo specijalni oblik procedure jer procedura ne mora vraćati podatke mjestu koje ju je pozvalo.

Još treba reći da nakon što se izvede tijelo procedure, sve varijable koje su deklarirane u tijelu procedure, te svi njezini deklarirani argumenti, nestaju iz memorije računala. Žele li se neke varijable sačuvati u memoriji računala, tada ih se treba deklarirati na poseban način (u programskome jeziku C to je ključna riječ `static`).

7.2. Realizacija procedura u C

Opći je oblik definicije procedure u C dan u nastavku.

```
 1.  povratni_tip naziv_procedure(const tip_argumenta1 naziv_argumenta1,
                                  tip_argumenta2 naziv_argumenta2,
                                  ...
                                  tip_argumentan naziv_argumentan){
 2.     naredba1
 3.     return povratna_vrijednost;
 4.     naredba2
 5.  }
 6.  neka_procedura(){
 7.     naredbaX
 8.     vrijednost = naziv_procedure (konkretna_vrijednost,
                                      varijabla1,
                                      ...
                                      varijablan);
 9.     naredbaY
10.     ...
11.  }
```

U prikazanoj općoj definiciji procedure prikazani su svi njezini dijelovi: **povratni tip, naziv procedure, popis argumenata** i **tijelo procedure**.

Povratni tip predstavlja tip podatka koji procedura vraća. Ako je naveden tip podatka, onda u tijelu procedure (u bloku naredbi koji je ograničen vitičastim zagradama) mora postojati naredba skoka `return` i vrijednost koja se vraća (u prikazu to je `povratna_vrijednost` koja može biti neka konkretna vrijednosti ili vrijednost koja se čuva u nekoj varijabli). Ako procedura ne vraća vrijednost kroz povratni tip (dakle primjenom naredbe `return`), tada se kao povratni tip stavlja `void`. U tome slučaju u tijelu procedure nije potrebna naredba `return`, no ona se može koristiti za prekid izvođenja procedure.

Naziv procedure je naziv preko koga će se procedura pozivati bilo iz glavnoga programa (`main`) bilo iz neke druge procedure. Važno je napomenuti da procedura koja se poziva mora obavezno biti definirana prije procedure u kojoj se poziv izvodi. To je vidljivo iz općeg oblika definicije procedure. Procedura `naziv_procedure` poziva se iz procedure `neka_procedura`. Stoga je procedura `naziv_procedure` definirana iznad (prije) procedure `neka_procedura`.

Popis argumenata predstavlja deklaraciju varijabli (argumenata, parametara) preko kojih će podaci ući u tijelo procedure (moguće i izaći iz procedure). Svaka deklaracija se sastoji iz tipa podatka (primjera radi `tip_argumenta1`) i naziva argumenta (primjera radi `naziv_argumenta1`). Podaci koji su poslani proceduri s mjesta njezina poziva koriste se u tijelu procedure tako da se argumenti procedure koriste kao obične varijable. Redoslijed nizanja argumenata je važan u pozivu procedure. Dakle, ako je neki argument procedure namijenjen određenome podatku (primjera radi postotku popusta), tada taj podatak u proceduru mora ući upravo kroz taj argument.

Moguće je kroz argumente procedure vratiti neki podatak mjestu koje je proceduru pozvalo. U osnovi, argumentima se u proceduru mogu poslati podaci (tada se to naziva **slanje podataka po vrijednosti** – tako se šalju jednostavni tipovi podataka) ili adrese gdje se podaci nalaze (tada se to naziva **slanje reference na podatke** – primjera radi, tako se šalju polja). Kroz argumente kojima se podaci šalju po vrijednosti, nije moguće podatke vratiti mjestu poziva, dok je to moguće kroz argumente kojima se šalje referenca (adresa) na podatke. U svrhu eksplicitnoga slanja reference na podatke, argument se treba definirati kao pokazivač na neki tip podatka. Želi li se onemogućiti promjena podatka u tijelu procedure kroz tako deklarirani argument, on se treba deklarirati ključnom riječi `const`. Procedura može biti i bez argumenata. Tada se nakon njezina imena navodi prazan popis, odnosno `()`.

Tijelo procedure predstavlja blok naredbi koji će se izvesti kada se pozove procedura svojim imenom. Ako procedura posjeduje argumente, tada se i oni trebaju pojaviti u tijelu procedure. Ako procedura ima naveden povratni tip koji nije `void`, tada se u tijelu procedure mora nalaziti naredba `return`. Tijelo procedure može se sastojati iz bilo koje kombinacije algoritamskih struktura.

U proceduri `neka_procedura` postoji poziv na proceduru `naziv_procedure` kojoj se kroz argumente šalju podaci (primjera radi `konkretna_vrijednost`, `varijabla1` itd.). Ono što procedura `naziv_procedure` vrati kroz svoj povratni tip, to će se sačuvati u varijabli naziva `vrijednost`.

Imenovanje procedura i njezinih argumenata provodi se po pravilima imenovanja varijabli što je opisano u poglavlju **2. Varijable**. Dobra praksa je da se za naziv procedure bira glagol, čime se dodatno naglašava da svojim pozivom procedura čini neku radnju.

Najjednostavnija procedura nema povratni tip i nema argumente. Slijedi primjer u kome korisnik, unosom rednoga broja procedure, bira koja će procedura biti izvedena.

```
 1.  #include<stdio.h>
 2.  void procedura1(){
 3.    printf("\n\nPozvana je prva procedura.");
 4.  }
 5.
 6.  void procedura2(){
 7.    printf("\n\nPozvana je druga procedura.");
 8.  }
 9.
10.  main(){
11.    char redni_broj_procedure;
12.    do {
13.      printf ("\nUnesite redni broj procedure koju želite
                 izvesti (1 ili 2): ");
14.      scanf (" %c", &redni_broj_procedure);
15.      if(!(redni_broj_procedure=='1'||
               redni_broj_procedure == '2')){
16.        printf ("\n\nRedni broj treba biti 1 ili 2");
17.        continue;
18.      }
19.      break;
20.    }
21.    while (1==1);
22.    if (redni_broj_procedure == '1'){
23.      procedura1();
24.    }
25.    else{
26.      procedura2();
27.    }
28.    getch();}
```

Program 7.1.

Nakon unosa ispravnoga rednog broja (1 ili 2), izvodi se linija koda 19. U ovisnosti o tome koji je redni broj unesen, poziva se `procedura1` ili `procedura2` (linija koda 23. ili 26.). Nakon poziva neke od procedura, izvode se naredbe koje se nalaze u njihovu tijelu (linije koda 3. ili 7.). Po završetku izvođenja tijela procedure, izvodi se

linija koda 28. Dakle, kontrola se izvođenja programa vraća mjestu koje je proceduru pozvalo. Treba obratiti pažnju da su obje procedure definirane ispred mjesta njihova poziva, dakle, ispred procedure `main`. Također treba uočiti da niti jedna od procedura nema argumente, a kao povratni tip ima stavljeno `void`, dakle, ne vraća nikakvu vrijednost mjestu koje ju je pozvalo.

Slijedi program u kome postoji procedura koja prima dva unesena broja i ispisuje njihov odnos.

```
 1.  #include<stdio.h>
 2.
 3.  void provjeraIIspisOdnosaDvaBroja(int broj1, int broj2){
 4.    if (broj1 < broj2){
 5.       printf ("Broj %d je manji od broja %d.", broj1, broj2);
 6.    }
 7.    else if (broj1 > broj2){
 8.       printf ("Broj %d je veci od broja %d.", broj1, broj2);
 9.    }
10.    else{
11.       printf ("Brojevi %d i %d su isti.", broj1, broj2);
12.    }
13.  }
14.
15.  main(){
16.    int korisnickiBroj1, korisnickiBroj2;
17.    printf ("\nUnesite dva broja odvojena zarezom: ");
18.    scanf ("%d, %d", &korisnickiBroj1, &korisnickiBroj2);
19.    provjeraIIspisOdnosaDvaBroja(korisnickiBroj1,korisnickiBroj2);
20.    getch();}
```

Program 7.2.

Nakon unosa dvaju brojeva koje se sprema u varijable `korisnickiBroj1` i `korisnickiBroj2`, poziva se procedura `provjeraIIspisOdnosaDvaBroja` (linija koda 19.). Ovoj se proceduri kroz njezine argumente dostavljaju vrijednosti varijabli `korisnickiBroj1` i `korisnickiBroj2`. Vrijednosti navedenih varijabli prepisuju se u argumente procedure (`broj1` i `broj2`) – slanje podataka je po vrijednosti jer je `int` jednostavni tip podatka. Na taj su se način vrijednosti koje je korisnik unio u proceduri `main` prenijele u tijelo procedure u kome se izvodi provjera odnosa brojeva i ispis rezultata (linije od 4. do 11.). Po završetku izvođenja tijela procedure, izvodi se linija 20.

Sljedeći primjer prikazuje važnost redoslijeda argumenata. Program posjeduje proceduru koja ispisuje rezultat računske operacije nad dva operanda. Proceduri se prosljeđuju dva operanda i računska operacija.

```
 1.  #include<stdio.h>
 2.
 3.  void racunskaOperacija(int operand1,int operand2,char operacija){
 4.    if(operacija== '+'){
 5.      printf("%d+ %d=%d",operand1,operand2,operand1 + operand2);
 6.    }
 7.    else if (operacija== '-'){
 8.      printf("%d-%d = %d",operand1,operand2,operand1 - operand2);
 9.    }
10.    else if (operacija== '*'){
11.      printf("%d*%d = %d",operand1,operand2,operand1 * operand2);
12.    }
13.    else if (operacija== '/'){
14.      if (operand2 == 0)
15.        printf ("Nije moguce dijeljenje s nulom.");
16.      else
17.        printf ("%d / %d = %.2f", operand1, operand2,
                     (float)operand1 / operand2);
18.    }
19.    else if (operacija== '%'){
20.      if (operand2 == 0)
21.        printf ("Operaciju modulo nije moguce izvesti s nulom.");
22.      else
23.        printf ("%d % %d = &d", operand1, operand2,
                      operand1 % operand2);
24.    }
25.    else{
26.      printf ("Nepostojeca racunska operacija");
27.    }
28.  }
29.  main(){
30.    int operand1, operand2;
31.    char operacija;
32.    printf ("\nUnesite dva broja odvojena zarezom: ");
33.    scanf ("%d, %d", &operand1, &operand2);
34.    printf ("\nUnesite racunsku operaciju: ");
35.    scanf (" %c", & operacija);
36.    racunskaOperacija(operand1, operand2, operacija);
37.    getch();}
```

Program 7.3.

U navedenome je primjeru izrađena procedura naziva `racunskaOperacija` koja ima tri argumenta. Kroz prva dva argumenta proceduri se dostavlja vrijednost operanada, dok se kroz treći dostavlja računska operacija. To pokazuje kojim se redoslijedom podaci trebaju dostaviti proceduri (linija koda 36.). Jasno je da prilikom poziva procedure `racunskaOperacija` nije moguće na prvome mjestu staviti podatak o računskoj operaciji jer bi on u proceduru ušao kao argument `operand1` i to bi izazvalo potpuno pogrešno izvođenje ostatka tijela procedure. U ovome programu još treba uočiti da su u proceduri `main` deklarirane varijable koje imaju isto ime kao i argumenti procedure. Ovime se htjelo pokazati da deklaracija varijabli u proceduri `main` (ili bilo kojoj drugoj koja poziva neku proceduru) i deklaracija

argumenata pozivane procedure su potpuno neovisni, odnosno mogu ali i ne moraju imati isti naziv (identifikator).

Sljedeći program posjeduje proceduru kojoj se dostavi cijena s PDV-om, a ona kroz povratni tip vrati iznos cijene bez PDV-a.

```
 1. #include<stdio.h>
 2.
 3. float cijenaBezPDV(float cijenaSPDV){
 4.   int PDV = 25;
 5.   float cijena;
 6.   cijena = cijenaSPDV/(1+(float)PDV/100);
 7.   return cijena;
 8. }
 9. main(){
10.   float korisnickaCijenaSPDV;
11.   float korisnickaCijenaBezPDV;
12.   printf ("\nUnesite neku cijenu s PDVom: ");
13.   scanf ("%f", &korisnickaCijenaSPDV);
14.   korisnickaCijenaBezPDV = cijenaBezPDV(korisnickaCijenaSPDV);
15.   printf("\n\n Cijena s PDVom: %.2f\nCijena bez PDVa: %.2f",
            korisnickaCijenaSPDV, korisnickaCijenaBezPDV);
16.   getch();}
```

Program 7.4.

Nakon korisničkog unosa cijene s `PDV-om`, ona se kroz argument dostavlja proceduri `cijenaBezPDV` (linija koda 14.). Vrijednost varijable `korisnickaCijenaSPDV` ulazi u tijelo procedure u kome se izračunava cijena bez PDV-a koja se naredbom `return` vraća mjestu poziva (linija koda 7.). Na mjestu poziva postoji operator pridruživanja (=) koji ono što pozvana procedura vraća povratnim tipom pridružuje varijabli `korisnickaCijenaBezPDV` (linija 14.). Nakon toga se izvodi ispis vrijednosti koje se nalaze u varijablama `korisnickaCijenaSPDV`, `korisnickaCijenaBezPDV`.

Program koji slijedi posjeduje proceduru koja samo kroz argument vraća vrijednost mjestu poziva. Proceduri se dostavlja cijena i iznos popusta, a ona izračuna novu cijenu s popustom.

```
1.  #include<stdio.h>
2.
3.  void cijenaSPopustom(float cijena, int popust, float
4.  *novaCijena){
5.    *novaCijena = cijena * ((100 - popust)/100.0);
6.  }
7.  main(){
8.    float cijena, novaCijena;
9.    int popust;
10.   printf ("\nUnesite neku cijenu i popust odvojeno zarezom: ");
11.   scanf ("%f, %d", &cijena, &popust);
12.   cijenaSPopustom (cijena, popust, &novaCijena);
13.   printf("\n\nCijena bez popusta (Kn): %.2f\nPopust(%):
            %d\nCijena s popustom (Kn): %.2f",  cijena, popust,
            novaCijena);
```

```
14.    getch();}
```

Program 7.5.

Nakon unosa `cijene` i `popusta` poziva se procedura `cijenaSPopustom` (linija koda 12.). Proceduri se dostavljaju uneseni podaci preko prvih dvaju argumenata, a kroz treći argument se dostavlja adresa varijable u kojoj se očekuje da procedura upiše izračunatu vrijednost (adresa varijable se dostavlja adresnim operatorom `&`). Procedura prima dva unesena podatka i adresu varijable. U tijelu se procedure izvodi izračun nove cijene koja se postavlja u memorijsku lokaciju s dostavljenom adresom (linija koda 5.) – koristi se operator dereferenciranja (`*`). Potom se kontrola izvođenja programa vraća u proceduru `main`, odnosno na liniju koda 13. Kako je procedura `cijenaSPopustom` izvela upis vrijednosti na memorijsku lokaciju s dostavljenom adresom, do te se vrijednosti u `main` proceduri dolazi primjenom varijable `novaCijena` jer se upravo njezina adresa poslala proceduri.

Do sada su se za tip podataka argumenata procedure i za njezin povratni tip, koristili jednostavni tipovi podataka. Kod njih se podatak prenosi po vrijednosti. Slijede primjeri sa složenim tipovima podataka (niz znakova, polje i zapis). Kod njih se podatak prenosi po referenci.

Sljedeći program prima niz znakova (riječ), a vraća niz znakova bez samoglasnika. Niz znakova unosi korisnik.

```
 1.  #include<stdio.h>
 2.
 3.  void izbaciSamoglasnike(char rijec[], char * rijecBezSamoglasnika){
 4.     int brojac = 0;
 5.     int brojacBezSamoglasnika = 0;
 6.     while(rijec[brojac] != '\0'){
 7.        if (!(rijec[brojac] == 'a' || rijec[brojac] == 'A' ||
                 rijec[brojac] == 'e' || rijec[brojac] == 'E' ||
                 rijec[brojac] == 'i' || rijec[brojac] == 'I' ||
                 rijec[brojac] == 'o' || rijec[brojac] == 'O' ||
                 rijec[brojac] == 'u' || rijec[brojac] == 'U')){
 8.           rijecBezSamoglasnika[brojacBezSamoglasnika] = rijec[brojac];
 9.           brojacBezSamoglasnika++;
10.        }
11.        brojac++;
12.     }
13.     rijecBezSamoglasnika[brojacBezSamoglasnika] = '\0';
14.  }
15.  main(){
16.     char rijec [50+1], rijecBezSamoglasnika[50+1];
17.     printf ("\nUnesite neku rijec: ");
18.     scanf ("%s", rijec);
19.     izbaciSamoglasnike (rijec, rijecBezSamoglasnika);
20.     printf("\n\nUnesena rijec: %s\nRijec bez samoglasnika: %s",
21.              rijec, rijecBezSamoglasnika);
22.     getch();}
```

Program 7.6.

Kako se složeni tipovi podataka procedurama dostavljaju referencama (adresama gdje se podaci nalaze), to deklaracija argumenata kroz koje se dostavljaju složeni tipovi podatka omogućava da se kroz iste te argumente izvodi i vraćanje vrijednosti. Želi li

se ovo izbjeći, argument se treba deklarirati s ključnom riječi `const`. U gornjemu primjeru vidljiva su dva načina definiranja argumenata. I kroz jedan i kroz drugi način deklaracije argumenta moguće je proceduri dostaviti niz znakova, odnosno iz procedure je moguće dobiti niz znakova.

Želi li se polje (niz znakova, polje brojeva, polje zapisa itd.) vratiti iz procedure kroz povratni tip, tada se to treba učiniti također vraćanjem adrese. Kako se ovim postupkom vraća adresa lokalne varijable definirane u tijelu procedure, a koja nakon izvođenja procedure nestaje iz memorije računala, takvu je varijablu potrebno deklarirati s ključnom riječi `static` koja omogućava zadržavanje lokalne varijable iz tijela procedure u memoriji računala i nakon izvođenja procedure. To prikazuje naredni program koji obavlja isti posao kao i prethodni program, ali sada procedura vraća niz znakova kroz povratni tip.

```
 1.  #include<stdio.h>
 2.
 3.  char * izbaciSamoglasnike(const char rijec[]){
 4.    int brojac = 0;
 5.    int brojacBezSamoglasnika = 0;
 6.    static char rijecBezSamoglasnika[50+1];
 7.    while(rijec[brojac] != '\0'){
 8.       if (!(rijec[brojac] == 'a' || rijec[brojac] == 'A' ||
                rijec[brojac] == 'e' || rijec[brojac] == 'E' ||
                rijec[brojac] == 'i' || rijec[brojac] == 'I' ||
                rijec[brojac] == 'o' || rijec[brojac] == 'O' ||
                rijec[brojac] == 'u' || rijec[brojac] == 'U')){
 9.          rijecBezSamoglasnika[brojacBezSamoglasnika] =
             rijec[brojac];
10.          brojacBezSamoglasnika++;
11.       }
12.       brojac++;
13.    }
14.    rijecBezSamoglasnika[brojacBezSamoglasnika] = '\0';
15.    return rijecBezSamoglasnika;
16.  }
17.  main(){
18.    char rijec [50+1], *rijecBezSamoglasnika;
19.    printf ("\nUnesite neku rijec: ");
20.    scanf ("%s", rijec);
21.    rijecBezSamoglasnika = izbaciSamoglasnike (rijec);
22.    printf("\n\nUnesena rijec: %s\nRijec bez samoglasnika: %s",
               rijec, rijecBezSamoglasnika);
29.    getch();}
```

Program 7.7.

Program prikazuje proceduru koja ima argument deklariran ključnom riječi `const`. Ovo znači da se vrijednost argumenta `rijec` neće moći mijenjati u tijelu procedure (što nije bio slučaj u programu 7.6.). Nadalje, procedura ima povratni tip koji je adresa na tip podatka `char`. U tijelu procedure postoji naredba `return` kojom se vraća adresa lokalno deklarirane varijable `rijecBezSamoglasnika` (linija 15.). Budući da po završetku izvođenja tijela procedure, sve lokalno deklarirane varijable nestaju iz

memorije računala, mjesto poziva procedure bi dobilo adresu varijable koja više ne postoji u memoriji računala. Da se to ne bi dogodilo, lokalna varijabla `rijecBezSamoglasnika` mora ostati u memoriji računala nakon izvođenja tijela procedure. To se čini tako da se varijabla deklarira ključnom riječi `static` (linija 6.). U pozivu procedure (linija 21.) dostavlja se adresa prvoga elementa niza znakova (u nazivu varijable `rijec` krije se adresa jer je varijabla deklarirana kao polje – linija 18.). Ono što procedura `izbaciSamoglasnike` vraća (a vraća adresu na tip podatka `char`) sprema se u pokazivač `rijecBezSamoglasnika` (deklariran u liniji 18.). Nakon toga izvodi se ispis sadržaja varijabli u liniji 22.

U sljedećemu primjeru proceduri se šalje polje brojeva. Procedura kroz argument vraća najmanji i najveći broj unutar polja.

```
 1. #include<stdio.h>
 2.
 3. void najmanjiINajveci(const int brojevi[], int zadnjiIndeks,
                          int *najmanji, int *najveci){
 4.   int brojac;
 5.   *najmanji = *najveci = brojevi[0];
 6.   for (brojac = 0; brojac <= zadnjiIndeks; brojac ++){
 7.      if (*najmanji > brojevi[brojac])
 8.         *najmanji = brojevi[brojac];
 9.      if(*najveci < brojevi[brojac])
10.         *najveci = brojevi[brojac];
11.   }
12. }
13. main(){
14.   int brojevi[100];
15.   int brojac, najmanji, najveci;
16.   char izbor;
17.   for (brojac = 0; brojac < 100; brojac ++){
18.      printf ("Unesite neki broj: ");
19.      scanf("%d", &brojevi[brojac]);
20.      do{
21.         printf("\nZelite unijeti jos jedan broj (d/n): ");
22.         scanf(" %c", &izbor);
23.      }
24.      while (!(izbor == 'd' || izbor == 'D' || izbor == 'n' ||
                  izbor == 'N'));
25.      if(izbor == 'n' || izbor == 'N')
26.         break;
27.   }
28.   if (brojac == 100) brojac --;
29.   najmanjiINajveci(brojevi, brojac, &najmanji, &najveci);
30.   printf ("\n\nNajmanji medju unesenim brojevima je %d, a
              najveci je %d.", najmanji, najveci);
31.   getch();}
```

Program 7.8.

U prikazanome programu procedura `najmanjiINajveci` prima četiri argumenta. Kroz prvi prima adresu na kojoj se nalazi prvi element polja, kroz drugi zadnji indeks

u polju te kroz preostala dva adrese na kojima se očekuje rezultat koji procedura treba izračunati. U liniji koda 29. poziva se procedura s odgovarajućim varijablama.

Neka se najmanji i najveći element želi spremiti u zapis naziva `NajmanjiNajveci` koji se sastoji iz dviju komponenti. Sljedeći program pokazuje kako se ovakav zapis može vratiti kroz povratni tip procedure.

```
 1.  #include<stdio.h>
 2.  typedef struct {
 3.    int najmanji;
 4.    int najveci;
 5.  } NajmanjiNajveci;
 6.    NajmanjiNajveci najmanjiINajveci(const int brojevi[],
                                        int zadnjiIndeks){
 7.    int brojac;
 8.    NajmanjiNajveci rezultat;
 9.    rezultat.najmanji = rezultat.najveci = brojevi[0];
10.    for (brojac = 0; brojac <= zadnjiIndeks; brojac ++){
11.       if (rezultat.najmanji > brojevi[brojac])
12.          rezultat.najmanji = brojevi[brojac];
13.       if (rezultat.najveci < brojevi[brojac])
14.          rezultat.najveci = brojevi[brojac];
15.    }
16.    return rezultat;
17.  }
18.  main(){
19.    int brojevi[100];
20.    int brojac;
21.    NajmanjiNajveci rezultat;
22.    char izbor;
23.    for (brojac = 0; brojac < 100; brojac ++){
24.       printf ("Unesite neki broj: ");
25.       scanf("%d", &brojevi[brojac]);
26.       do{
27.          printf("\nZeline unijeti jos jedan broj (d/n): ");
28.          scanf(" %c", &izbor);
29.       }
30.       while (!(izbor == 'd' || izbor == 'D' || izbor == 'n' ||
                    izbor == 'N'));
31.       if (izbor == 'n' || izbor == 'N')
32.          break;
33.    }
34.    if (brojac == 100) brojac --;
35.    rezultat = najmanjiINajveci(brojevi, brojac);
36.    printf ("\n\nNajmanji medju unesenim brojevima je %d, a
              najveci je %d.", rezultat.najmanji, rezultat.najveci);
37.    getch();}
```

Program 7.9.

Ovdje treba uočiti da je povratni tip procedure `najmanjiINajveci` deklariran kao zapis `NajmanjiNajveci`. U tijelu se procedure naredbom `return` vraća vrijednost koja odgovara povratnome tipu procedure (linija koda 16.). Nakon poziva procedure, vrijednost koju ona vrati se pridružuje varijabli koja je tipa podatka zapisa

`NajmanjiNajveci` (linija koda 35.). Potom se izvodi ispis vrijednosti njezinih komponenti (linija koda 36.).

U sljedećemu se programu proceduri `izracun` dostavlja polje zapisa. Unutar zapisa nalaze se duljine stranica pravokutnika. Procedura ide od zapisa do zapisa, računa površinu pravokutnika i tu površinu upisuje u odgovarajuću komponentu zapisa. Unosi se ukupno pet pravokutnika. Nakon završetka izračuna, unutar procedure `izracun` poziva se nova procedura ispis u kojoj se ispisuju duljine stranica pravokutnika te njihova površina.

```
 1.  #include<stdio.h>
 2.  typedef struct {
 3.    int stranicaA;
 4.    int stranicaB;
 5.    int povrsina;
 6.  } Pravokutnik;
 7.  void ispis (const Pravokutnik pravokutnici[]){
 8.     int brojac;
 9.     system("cls");
10.     printf ("a\tb\tPovrsina");
11.     for (brojac = 0; brojac < 5; brojac ++){
12.        printf("\n%d\t%d\t%d", pravokutnici[brojac].stranicaA,
                  pravokutnici[brojac].stranicaB,
                  pravokutnici[brojac].povrsina);
13.     }
14.  }
15.  void izracun(Pravokutnik pravokutnici[]){
16.     int brojac;
17.     for (brojac = 0; brojac < 5; brojac ++){
18.      pravokutnici[brojac].povrsina=pravokutnici[brojac].stranicaA *
                                       pravokutnici[brojac].stranicaB;
19.     }
20.     ispis(pravokutnici);
21.  }
22.  main(){
23.     Pravokutnik pravokutnici[5];
24.     int brojac;
25.     for (brojac = 0; brojac < 5; brojac ++){
26.        printf("%d. PRAVOKUTNIK:", brojac+1);
27.        printf ("\nUnesite stranicu a: ");
28.        scanf("%d", &pravokutnici[brojac].stranicaA);
29.        printf ("\nUnesite stranicu b: ");
30.        scanf("%d", &pravokutnici[brojac].stranicaB);
31.     }
32.     izracun (pravokutnici);
33.     getch();}
```

Program 7.10.

Prikazani program posjeduje dvije procedure. Prva se procedura `izracun` poziva iz glavnoga dijela programa (procedure `main`) – linija koda 32. Argument kroz koji se prosljeđuju podaci proceduri je upravo tipa podatka zapisa `Pravokutnik`. Argument nije deklariran kao `const` budući da se u ovoj proceduri treba izvesti izmjena nekih podataka koji su stigli kroz argument (treba se izračunati površina i rezultat zapisati u odgovarajuću komponentu zapisa). Unutar procedure `izracun` poziva se druga procedura `ispis` kojoj se prosljeđuju izmijenjeni zapisi. Argument je

i ove procedure deklariran tipom podatka `Pravokutnik`, ali kao `const` jer se u njoj ne trebaju izvesti izmjene pristiglih podataka, već samo njihov ispis.

Kao što je već rečeno, procedura može pozvati i samu sebe i onda se to naziva rekurzivni poziv procedure. Sljedeći program prikazuje izračun faktorijela broja primjenom rekurzivnoga poziva procedure. Faktorijel broja n se računa kako slijedi n!=1·2·3·...n. S tim da je 0! = 1! = 1.

```
 1. int faktorijel(int broj){
 2.   if (broj == 0)
 3.      return 1;
 4.   return broj * faktorijel(broj-1);
 5. }
 6. main(){
 7.   int broj;
 8.   do{
 9.      printf("Unesite neki broj od 0 do 10: ");
10.      scanf("%d", &broj);
11.      if (broj>=0 && broj <=10)
12.         break;
13.   }
14.   while (1==1);
15.   printf ("\n\n%d! = %d", broj, faktorijel(broj));
16.   getch();}
```

Program 7.11.

Nakon unosa ispravnoga broja izvodi se ispis rezultata. Na mjesto konverzijskoga znaka `%d` stavit će se vrijednost koju će vratiti procedura `faktorijel` (moguće je izravno pozvati proceduru koja vraća vrijednost kroz povratni tip u sklopu `printf` naredbe) – linija 15. U tijelu procedure `faktorijel` vidljivo je da ona ponovno poziva proceduru `faktorijel` (dakle samu sebe), ali je podatak koji se šalje drugačiji (manji je za jedan) – linija 4. Pri rekurzivnome pozivu treba paziti da postoji izlaz iz rekurzije (u prikazanome primjeru izlaz je u liniji 2., odnosno 3.). Ovaj izlaz jamči da neće doći do besprekidnoga rekurzivnog poziva koji će uzrokovati rušenje programa jer će se napuniti memorijski prostor računala u kome se spremaju adrese poziva procedura (stog).

7.3. Praktični zadaci

Računalni program koji slijedi demonstrira pozivanje jednostavne procedure `pozdrav` koja ispisuje `Dobar dan`. Prikazan je način izrade procedure kod koga je razdvojen dio deklaracije procedure (linija 3.) i dio definicije procedure (linija 13., 14. i 15.). Poziv procedure se višestruko izvodi u glavnome dijelu programa (`main()`). Treba uočiti da se svakim pozivom procedure izvodi tijelo procedure u kome se nalazi naredba za ispis pozdrava (linija 14.). Također treba uočiti da je povratni tip procedure `void`, što znači da procedura ne vraća vrijednost mjestu koje ju je pozvalo.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  void pozdrav();
 4.
 5.  main(){
 6.    printf("Sada ce se pozvati procedura pozdrav nekoliko
             puta\n");
 7.    pozdrav();
 8.    pozdrav(); pozdrav();
 9.    printf("\nNovi red\n");
10.    pozdrav(); pozdrav(); pozdrav();
11.    getch();}
12.
13.  void pozdrav(){
14.    printf("Dobar dan !");
15.  }
```

Program 7.12.

Naredni računalni program od korisnika traži unos dvaju brojeva koji se šalju proceduri `zbroj` (linija 3.) koja ispisuje zbroj proslijeđena dva broja (linija 4.). Procedura je pozvana iz glavnoga dijela programa tri puta. U prvome se pozivu proceduri šalje vrijednost koja je spremljena u varijablama, u drugome pozivu proceduri se šalju konkretne vrijednosti te se u trećemu pozivu proceduri šalju vrijednosti koje se prvo izračunavaju (linija 12). Treba uočiti da vrijednost koja se šalje proceduri mora biti odgovarajućega tipa podatka koji je specificiran u deklaraciji argumenta procedure (u prikazanome primjeru radi se o tipu podatka `int`) – linija 3.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  void zbroj(int a,int b){
 4.    printf("Zbroj %d i %d je %d\n",a,b,a+b);
 5.  }
 6.  main(){
 7.    int c,d;
 8.    printf("unesi 1. broj:");
 9.    scanf("%d",&c);
10.    printf("unesi 2. broj:");
11.    scanf("%d",&d);
12.    zbroj(c,d);zbroj(-23,78); zbroj (c+2, d*3);
13.    getch();}
```

Program 7.13.

Primjer računalnoga programa koji slijedi od korisnika traži unos koordinata dviju točaka. Potom program izračunava i ispisuje njihovu udaljenost. Izračun se izvodi u proceduri `udaljenost` (linija 4.). U tijelu te procedure koristi se funkcija `sqrt()` kako bi se izračunao drugi korijen (linija 9.). Zbog toga je uključena zaglavna datoteka `math.h` (linija 3.). Procedura kroz svoje argumente prima četiri koordinate, a kroz

povratni tip (`float`) vraća rezultat. Formula po kojoj se računa udaljenost točaka T1(x1, y1) te T2(x2, y2) glasi: d=$\sqrt{(x2-x1)^2+(y2-y1)^2}$.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  #include<math.h>
 4.  float udaljenost(int x1, int y1,int x2,int y2){
 5.    int d1,d2;
 6.    d1=x2-x1;
 7.    d2=y2-y1;
 8.    float D;
 9.    D=sqrt(d1*d1+d2*d2);
10.    return D;
11.  }
12.  main(){
13.    int a,b,c,d;
14.    printf("Unesi koordinate tocke T1\n");
15.    scanf("%d",&a);
16.    scanf("%d",&b);
17.    printf("Unesi koordinate tocke T2\n");
18.    scanf("%d",&c);
19.    scanf("%d",&d);
20.    printf("Udaljenost tocaka je %.2f",udaljenost(a,b,c,d));
21.    getch();}
```

Program 7.14.

Računalni program 7.15 traži od korisnika unos dvaju brojeva. Potom ispisuje koji je broj veći. Provjera i ispis izvode se u proceduri `veci`. Deklaracija i definicija procedure su razdvojeni. Treba primijetiti da je prilikom deklaracije procedure za deklaraciju njezinih argumenata dovoljno navesti samo tip podatka – nije nužno navođenje naziv argumenta (linija 3.). No, pri definiciji procedure nužno je navesti tip podatka i naziv argumenta (linija 13.). Procedura se u glavnome dijelu programa poziva dva puta – jednom s vrijednostima koje je unio korisnik, a drugi puta s konkretnim vrijednostima.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  void veci(int, int);
 4.
 5.  main(){
 6.    int x,y;
 7.    printf("Unesite dva cijela broja:");
 8.    scanf("%d %d", &x,&y);
 9.    veci(x,y);
10.    veci(56,100);
11.    getch();}
12.
13.  void veci(int x, int y){
14.    int z;
```

```
15.   if(x>=y) z=x; else z=y;
16.   printf("Najveci je %d",z);
17. }
```

Program 7.15.

Računalni se program može sastojati iz proizvoljnoga broja procedura. Jedino je ograničenje da se procedura definira prije mjesta poziva, odnosno ako se prije mjesta poziva izvodi deklaracija procedure, tada mjesto njezine definicije nije bitno. Program koji slijedi sastoji se iz dviju procedure. Procedura `ispisi` ispisuje pozdravnu poruku `Dobar dan` onoliko puta koliko je to poslano proceduri kroz njezin argument (linija 4.). Procedura `zbroji` zbraja dva broja koja je procedura dobila svojim argumentima te rezultat vraća kroz povratni tip (linija 11.).

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3.
 4. void ispisi(int broj){
 5.   int x;
 6.   printf("\n");
 7.   for (x=1; x<=broj; x++)
 8.   printf ("\nDobar dan.");
 9. }
10.
11. int zbroji (int prvi, int drugi){
12.   int rez;
13.   rez=prvi+drugi;
14.   return rez;
15.   }
16.
17. main(){
18.   int puta,zbroj,a,b;
19.   printf("\nNapisi koliko puta zelite ispis pozdrava: ");
20.   scanf ("%d",&puta);
21.   ispisi(puta);
22.   printf("\nUnesi brojeve za zbroj\n");
23.   scanf("%d\t%d",&a,&b);
24.   zbroj=zbroji(a,b);
25.   printf ("\nZbroj=%d",zbroj);
26.   printf("\nZbroji brojeve 5 i 3.");
27.   zbroj=zbroji(5,3);
28.   printf("\n5+3=%d",zbroj);
29.   getch();}
```

Program 7.16.

Računalni program koji slijedi traži od korisnika unos baze i pozitivnog eksponenta. Potom se izračunava i ispisuje rezultat potenciranja baze na eksponent. Izračun se izvodi u posebnoj proceduri `potencija` (linija 4.). Procedura prima tri argumenta – kroz prva dva prima bazu i eksponent, a kroz treći adresu na koju mora poslati rezultat potenciranja (ne vraća ništa kroz povratni tip pa je on stoga `void`). Algoritam

izračuna potencije glasi: pomnožiti bazu sa samom sobom onoliko puta koliki je eksponent. Navedeni algoritam je implementiran `for` petljom (linija 6. i 7.). U prikazanome se računalnome programu lako može primijetiti važnost redoslijeda argumenata. Naime, procedura `potencija` u prvom argumentu očekuje vrijednost baze, u drugome vrijednost eksponenta, a u trećemu memorijsku adresu. Zamijene li im se mjesta pri pozivu procedure, rezultat potenciranja će biti pogrešan.

U pozivu procedure `potencija`, pored baze i eksponenta, dostavlja joj se i adresa memorijske lokacije na kojoj se očekuje rezultat u glavnome dijelu programa (`main()`). Može se uočiti da se rezultat očekuje u varijabli `pot` pa je stoga njezina adresa poslana proceduri primjenom adresnog operatora `&`. U proceduri se primjenom operatora dereferenciranja (`*`) pristupa memorijskoj lokaciji čija je adresa dostavljena proceduri te se u njoj spremaju međurezultati izračuna kao i konačni rezultat (linija 8.)

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3.
 4. void potencija (int baza,int exp, int *rezultat){
 5.   int i;
 6.   *rezultat=1;
 7.   for(i=1;i<=exp;i++){
 8.     *rezultat = *rezultat * baza;
 9.   }
10. }
11.
12. main(){
13.   int b,exp,pot;
14.   printf("Unesite bazu: ");
15.   scanf("%d",&b);
16.   printf("Unesite eksponent: ");
17.   scanf("%d",&exp);
18.   potencija(b,exp, &pot);
19.   printf("%d na %d. potenciju iznosi %d",b,exp,pot);
20.   getch();}
```

Program 7.17.

Naredni računalni program traži od korisnika unos deset cijelih brojeva. Nakon unosa ispisuju se uneseni brojevi, njihovo odstupanje od prosjeka te njihov kvadrat. U svrhu izračuna kvadrata koristi se posebna procedura `pot` (linija 4.) koja kroz povratni tip vraća potenciju broja na neki eksponent. Treba uočiti kako se procedura `pot` poziva iz funkcije `printf`. To je moguće učiniti samo s procedurama koje rezultat vraćaju kroz svoj povratni tip, s tim da je povratni tip jednostavni tip podatka.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3.
 4. int pot(int a,int b){
 5.   int p=1,i;
```

```
 6.    for(i=1;i<=b;i++)
 7.      p*=a;
 8.    return p;
 9.  }
10.
11.  main(){
12.    int a[10],i,suma=0;
13.    float ars;
14.    printf("Unesite deset cijelih brojeva:\n");
15.    for (i=0;i<10;i++){
16.      scanf("%d",&a[i]);
17.      suma+=a[i];
18.    }
19.    ars=(suma*1.0)/i;
20.    printf("Prosjek je = %.2f\n",ars);
21.    printf("\nBROJ\t\tOdstupanje\tKvadrat");
22.    printf("\n_____________________________________________");
23.    for (i=0;i<10;i++)
24.    printf("\n%d\t\t%.2f\t\t%d",a[i],ars-a[i],pot(a[i],2));
25.    getch();}
```

Program 7.18.

Računalni program koji slijedi demonstrira poziv procedure iz glavnoga programa koja se nalazi u zaglavnoj datoteci `pomocne_procedure.h`. Sadržaj datoteke `pomocne_procedure.h` prikazan je u programu 7.20. To je računalni program koji ne posjeduje glavnu funkciju `main` iz čega slijedi da se ne radi o izvršnome programu (programu koji korisnik pokrene), nego se radi o programu koji koristi neki drugi program. Zaglavna datoteka `pomocne_procedure.h` se sastoji iz definicije jedne procedure imena `potencija`. Moguće je u istoj zaglavnoj datoteci definirati više različitih procedura.

U izvršnome programu u kome se nalazi glavna funkcija `main` (zadatak 7.19.) naredbom `#include` uključena je zaglavna datoteka `pomocne_procedure.h` (linija 3.) te je zbog toga moguće pozvati proceduru `potencija` (linija 9.). Treba obratiti pažnju na to da je naziv zaglavne datoteke naveden pod navodnicima (`"`). To znači da se ova zaglavna datoteka nalazi u istoj mapi u kojoj se nalazi i datoteka s glavnim programom. U slučaju da se datoteka nalazi u nekoj drugoj mapi, pod navodnicima bi se trebala navesti njezina cjelokupna putanja. Uključivanje je zaglavnih datoteka korištenjem znakova < i > moguće samo onda ako se te zaglavne datoteke nalaze u posebnoj mapi na koju pokazuju opcije kompajliranja samoga alata Dev-C++. Ovakav modularni način pisanja programskoga koda, odnosno njegova grupiranja u posebne datoteke, omogućava bolju preglednost budući se semantički povezani dijelovi programskoga koda nalaze u istoj datoteci. Također je olakšano ponovno korištenje (*engl. reuse*) već napravljenih procedura u različitim računalnim programima. Primjera radi prikazani računalni program (zadatak 7.19.) koristi proceduru `potencija` iz zaglavne datoteke `pomocne_procedure.h`. No, tu istu

proceduru (dakle ponovno korištenje) moguće je iskoristiti u bilo kome drugome računalnom programu jednostavnim uključivanjem zaglavne datoteke.

Računalni program (program 7.19.) ispisuje sve eksponente u intervalu od 1 do 10 te potencije broja 2 i -3 na ispisani eksponent. U izračunu se potencije koristi procedura `potencija` koja je definirana u zaglavnoj datoteci `pomocne_procedure.h` (linija 9.).

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. #include "pomocne_procedure.h"
 4.
 5. main(){
 6.   int i;
 7.   printf("Exponent\tPotencija od 2\tPotencija od -3\n");
 8.   for(i=1;i<=10;i++){
 9.     printf("%d\t\t%d\t\t%d\n",i,potencija(2,i),
              potencija(-3,i));
10.   }
11.   getch();}
```

Program 7.19.

Datoteka `pomocne_procedure.h`

```
1. int potencija(int baza,int n){
2.   int i,pot;
3.   pot=1;
4.   for(i=1;i<=n;i++)
5.     pot*=baza;
6.   return pot;
7. }
```

Program 7.20.

Naredni računalni program traži od korisnika unos dvaju cijelih brojeva. Zahtijeva se da je prvi broj manji od drugog. Nakon toga se izračunava i ispisuje prosjek brojeva između dva unesena broja, potom svaki broj i njegovo odstupanje od prosjeka i na kraju sumu svih odstupanja. Unos se zahtijeva u glavnome dijelu programa, dok se izračun i ispis izvodi u proceduri `odstupanje`.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3. void odstupanje(int a, int b){
 4.   int i,suma=0,br=0;
 5.   float ars,odstupanje,sumodst=0;
 6.   for (i=a;i<=b;i++){
 7.     suma+=i;
 8.     br++;
 9.   }
10.   ars=(suma*1.0)/br;
11.   printf("Prosjek brojeva od %d do %d je %.2f\n",a,b,ars);
```

```
12.   for(i=a;i<=b;i++){
13.     odstupanje=ars-i;
14.     sumodst+=odstupanje;
15.     printf("Broj je %d, a odstupanje je %.2f\n",i,
              odstupanje);
16.   }
17.   printf("\nSuma odstupanja je %.2f",sumodst);
18. }
19.
20. main(){
21.   int d,e;
22.   printf("Unesite neka dva broja odvojena tabom (prvi manji
          od drugog). Program će izračunati prosjek i
          odstupanja\n");
23.   scanf("%d\t%d",&d,&e);
24.   odstupanje(d,e);
25.   getch();}
```

Program 7.21.

Računalni program koji slijedi (program 7.22.) traži od korisnika unos nekoga malog slova engleske abecede. Program ispisuje odgovarajuće veliko slovo. Pretvorba maloga u veliko slovo izvodi procedura naziva `maloUVeliko`. Algoritam pretvorbe koristi činjenicu da je razlika ASCII koda maloga i odgovarajućega velikog slova ista za sva slova te da se ASCII kodovi velikih slova nalaze iza ASCII kodova malih slova. Stoga, ako je poznato malo slovo, poznat je i njegov ASCII kod. Da bi se došlo do ASCII koda odgovarajućega velikog slova, treba se ASCII kodu maloga slova dodati pomak. Pomak se može izračunati tako da se od ASCII koda nekoga velikog slova (primjera radi veliko slovo A) oduzme ASCII kod odgovarajućega malog slova (primjera radi malo slovo a). Ovaj algoritam je implementiran u proceduri `maloUVeliko` (linija 6.). Procedura kroz argument prima vrijednost maloga slova, a kroz povratni tip vraća vrijednost.

Treba uočiti uvjet koji je postavljen kako bi se kod izračuna velikoga slova, odnosno njegova ASCII koda, u obzir uzimala samo mala slova, a ne neki drugi znakovi (linija 5.). Naime, uvjetom se provjerava je li ASCII kod koji se nalazi u varijabli `malo` unutar intervala ASCII kodova koji se odnose na mala slova a i z.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3.
 4. char maloUVeliko (char malo){
 5.   if(malo>='a' && malo<='z'){
 6.     return malo+'A'-'a';
 7.   }
 8.   else
 9.     return malo;
10.   }
11.
12. main(){
```

```
13.   char maloSlovo;
14.   printf("Unesite neko malo slovo:\n");
15.   scanf("%c",&maloSlovo);
16.   printf("\nVeliko slovo je %c", maloUVeliko(maloSlovo));
17.   getch();}
```

Program 7.22.

Sljedeći računalni program računa i ispisuje faktorijel unesenoga broja. Faktorijel se računa u posebnoj proceduri `faktorijel` koja kroz argument prima korisnički broj te izračunava i ispisuje njegovu faktorijelu.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3. void faktorijel(int a){
 4.   int x;
 5.   int F=1;
 6.   for(x=2;x<=a;x++)
 7.     F*=x;
 8.   printf("Faktorijela od %d je %d",a,F);
 9. }
10.
11. main(){
12.   printf("Unesite broj ciji se faktorijel racuna: ");
13.   int m;
14.   scanf("%d",&m);
15.   faktorijel(m);
16.   getch();}
```

Program 7.23.

Sljedeći računalni program (program 7.24.) izračunava i ispisuje ukupan broj kombinacija koje se sastoje od k elemenata iz skupa od n elemenata. Prirodne brojeve k i n unosi korisnik. Ograničenje je da je broj k strogo manji od broja n. Formula za izračun glasi $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Zorni primjer izračuna ukupnoga broja kombinacija je košarkaška momčad nekoga kluba koja broji 5 članova. Neka je 10 ukupan broj igrača iz kluba koji mogu biti dio momčadi. Tada se od njih 10 može sastaviti ukupno $\binom{10}{5} = \frac{10!}{5!(10-5)!} = \frac{3628800}{120 \cdot 120} =$ 252 različite momčadi.

Za izračun faktorijele broja izrađena je posebna procedura `fakt` (linija 4.)

```
1. #include<stdio.h>
2. #include<conio.h>
3.
4. long fakt(int a){
5.   long i,rez=1;
6.   for(i=2;i<=a;i++)
7.     rez*=i;
8.   return rez;
```

```
 9.  }
10.
11.  main(){
12.    int n,k;
13.    printf("Unesite broj igraca u klubu:");
14.    scanf("%d",&n);
15.    printf("Unesite broj igraca u ekipi:");
16.    scanf("%d",&k);
17.    long povrh;
18.    povrh=fakt(n)/(fakt(k)*fakt(n-k));
19.    printf("Broj raznih momcadi od %d igraca je %ld",k,
             povrh);
20.    getch();}
```

Program 7.24.

Slijedi računalni program kojim se izračunava i ispisuje skalarni umnožak dvodimenzionalnih vektora. Skalarni umnožak vektora je jednak $[x_i] \cdot [y_i]=x_1 \cdot y_1 + x_2 \cdot y_2 + \cdots + x_n \cdot y_n$. U računalnome programu broj n je jednak 2.

Dvodimenzionalni vektori čuvaju se u jednodimenzionalnim poljima s dva elementa (linija 10). Nakon unosa komponenti jednoga i drugoga vektora (linija 12., 13., 15. i 16.), polja se prosljeđuju posebnoj proceduri `skalarni`. Argumenti ove procedure deklarirani su kao jednodimenzionalna polja (linija 3.).

Treba uočiti da se proceduri koja kroz svoje argumente prima polja, dostavlja adresa prvog elementa polja (linija 17.) jer se koriste nazivi polja (`vektor1`, `vektor2`), a nazivi polja predstavljaju konstante koje u sebi čuvaju upravo adresu prvog elementa polja. To znači da se proceduri podaci dostavljaju preko reference (adrese gdje se podaci nalaze), a ne preko vrijednosti (kopiranjem vrijednosti u memorijsku lokaciju rezerviranu za deklarirani argument). Kako se podaci dostavljaju preko reference, oni mogu u proceduri biti izmijenjeni i kao takvi vraćeni mjestu poziva (u primjeru vraćeni `main` funkciji). Kako bi se to spriječilo, valja argumente čija se vrijednost ne smije mijenjati deklarirati ključnom riječi `const`. Time argumenti postaju konstante koje se u tijelu procedure ne mogu mijenjati.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  int skalarni(int v1[],int v2[]){
 4.    int skalar;
 5.    skalar=v1[0]*v2[0]+v1[1]*v2[1];
 6.    return skalar;
 7.  }
 8.
 9.  main(){
10.    int vektor1[2],vektor2[2];
11.    printf("Unesite 1. vektor: ");
12.    scanf("%d",&vektor1[0]);
13.    scanf("%d",&vektor1[1]);
14.    printf("Unesite 2. vektor: ");
```

```
15.    scanf("%d",&vektor2[0]);
16.    scanf("%d",&vektor2[1]);
17.    printf("Skalarni umnožak vektora je %d",
               skalarni(vektor1,vektor2));
18.    getch();}
```

Program 7.25.

Naredni računalni program traži od korisnika unos deset cijelih brojeva. Nakon unosa ispisuje se koji je uneseni broj najveći. U svrhu čuvanja unesenih brojeva koristi se jednodimenzionalno polje veličine 10 elemenata (linija 6.). Nakon unosa, polje, odnosno adresa prvoga elementa polja se kroz argument dostavlja proceduri `najveci` (linija 13.). Unutar procedure traži se najveći uneseni broj primjenom petlje `for` (linija 16. i 17.) te se na kraju ispisuje rezultat (linija 19.)

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.  void najveci(int a[]);
 4.
 5.  main(){
 6.    int a[10],i;
 7.    printf("Unesite 10 elemenata polja: \n");
 8.    for(i=0;i<=9;i++){
 9.      scanf("%d", &a[i]);}
10.    najveci(a);
11.    getch();}
12.
13.  void najveci (int a[]){
14.    int z=0,veci,i;
15.    veci =a[0];
16.    for(i=1;i<10;i++){
17.      if(a[i]> veci) veci =a[i];
18.    }
19.    printf("Najveci broj je %d",veci);
20.  }
```

Program 7.26.

Računalni program koji slijedi izračunava i ispisuje sumu pet unesenih cijelih brojeva. Brojeve unosi korisnik. Jednodimenzionalno polje od pet elemenata koristi se za spremanje unesenih brojeva (linija 7.). Nakon unosa brojeva, polje koje ih sadrži dostavlja se proceduri `zbr` (linija 17.). Osim polja, proceduri se dostavlja i adresa na kojoj se očekuje rezultat (linija 12.). U proceduri se preko petlje `for` izvodi sumiranje unesenih elemenata polja te se međurezultati i konačni rezultat spremaju na adresu na koju pokazuje pokazivač `suma` (linija 19., 20. i 21.).

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.
 4.  void zbr(int a[], int *suma);
 5.
```

```
 6.  main(){
 7.    int a[5], i, b;
 8.    printf("Ubacite vrijednosti koje zelite sumirati:  \n");
 9.    for(i=0;i<5;i++){
10.      scanf("%d", & a[i]);
11.    }
12.    zbr(a, &b);
13.    printf("\nSuma svih unesenih brojeva je= %d\n", b);
14.    getch();
15.  }
16.
17.  void zbr(int a[], int *suma){
18.    int i;
19.    *suma=0;
20.    for(i=0; i<5; i++){
21.      *suma+=a[i];
22.    }
23.  }
```

Program 7.27.

Sljedeći računalni program predstavlja poopćenje prethodnoga računalnog programa (program 7.27.). Naime, prethodni računalni program sumirao je pet unesenih cijelih brojeva. Sada se omogućava korisniku da definira koliko brojeva želi sumirati te da toliko brojeva i unese.

Nakon korisničkoga definiranja broja unosa (linija 9.), deklarira se jednodimenzionalno polje s upravo definiranim brojem elemenata (linija 10.). Potom preko petlje `for` slijedi korisnički unos elemenata polja (linija 12.i 13.). Ispunjeno polje i ukupan broj elemenata polja prosljeđuje se proceduri `zbr` (linija 15.) koja kroz povratni tip vraća rezultat (sumu unesenih brojeva). Kako se proceduri dostavlja adresa prvoga elementa polja, argument kroz koji adresa ulazi u proceduru, može se deklarirati i kao pokazivač na odgovarajući tip podatka (linija 19.) – do sada je argument kroz koji se proceduri dostavljalo polje bio deklariran kao polje bez broja elemenata polja. Ako se polje prosljeđuje proceduri preko pokazivača, tada treba voditi brigu o tome kako se primjenom pokazivača pristupa pojedinom elementu polja.

Ako je `a` neko jednodimenzionalno polje s `n` elemenata, tada vrijedi:

`a + i == &a[i]`

`*(a + i) == a[i]`

za svaki `i = 0, 1, ..., n-1`.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.
 4.  int zbr(int *a, int brBrojeva);
 5.
 6.  main(){
 7.    int i, b, brBrojeva;
 8.    printf("Koliko brojeva zelite sumirati? ");
```

```
 9.    scanf("%d", &brBrojeva);
10.    int a[brBrojeva];
11.    printf("Unesite vrijednosti koje zelite sumirati:  \n");
12.    for(i=0;i<brBrojeva;i++){
13.       scanf("%d", &a[i]);
14.    }
15.    b=zbr(a, brBrojeva);
16.    printf("\nSuma svih unesenih brojeva je %d\n", b);
17.    getch();}
18.
19.  int zbr(int *a, int brBrojeva){
20.    int suma, i;
21.    suma=0;
22.    for(i=0; i<brBrojeva; i++){
23.       suma+=a[i];
24.    }
25.    return suma;
26.  }
```

Program 7.28.

Računalni program koji slijedi prikazuje potrebu deklariranja argumenta kao pokazivača kada je on jednostavnoga tipa podatka (`char`, `int`, `float` itd.). Naime, kroz argumente s jednostavnim tipovima podataka, podaci se proceduri dostavljaju preko vrijednosti. Stoga ove podatke nije moguće u proceduri izmijeniti i kao takve ih kroz argumente vratiti mjestu poziva procedure.

Računalni program od korisnika traži unos dvaju cijelih brojeva koja se potom prosljeđuju proceduri `zamijeni` koja im treba zamijeniti vrijednosti i potom ih vratiti u glavni program koji ih ispisuje. U proceduri `zamijeni` u računalnome programu 7.29. argumenti su deklarirani tako da se kroz njih podaci prenose preko vrijednosti (linija 4.). Nakon unosa dvaju brojeva i poziva procedure, njihove se vrijednosti kopiraju u argumente procedure (kopiraju se u memorijske lokacije rezervirane za argumente procedure). Procedura `zamijeni` u svome tijelu koristi i mijenja vrijednosti koje se nalaze u memorijskim lokacijama rezerviranima za argumente. Zbog toga, kada se završi izvođenje procedure, nisu zahvaćene varijable `a` i `b` iz glavnoga dijela programa jer se one nalaze na nekim drugim memorijskim lokacijama. Stoga vrijednosti varijabli `a` i `b` nisu izmijenjene.

U računalnome programu (program 7.30.) procedura `zamijeni` ima argumente koji su deklarirani kao pokazivači (linija 4.). Kod poziva, proceduri se dostavljaju adrese na kojima se nalaze vrijednosti s kojima procedura treba izvesti određenu radnju (linija 18.) – koristi se adresni operator `&`. Sada procedura u svome tijelu mijenja vrijednosti koje se nalaze na memorijskim lokacijama čije adrese su proceduri dostavljene kroz argumente (linija 6., 7. i 8.) – koristi se operator dereferenciranja `*`. Nakon završetka izvođenja tijela procedure, u glavnome dijelu programa izvodi se ispis sadržaja varijabli `a` i `b` (linija 19.). No, njihov sadržaj je sada izmijenjen jer je to

učinila procedura tako što je varijablama `a` i `b`, odnosno njihovim memorijskim lokacijama pristupila putem memorijskih adresa.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.
 4.  void zamijeni(int x, int y){
 5.    int z;
 6.    z=x;
 7.    x=y;
 8.    y=z;
 9.  }
10.
11.  main(){
12.    int a,b;
13.    printf("Unesite neki broj a:");
14.    scanf("%d",&a);
15.    printf("Unesite neki drugi broj b:");
16.    scanf("%d",&b);
17.    printf("Redoslijed brojeva je a=%d, b=%d\n", a, b);
18.    zamijeni(a,b);
19.    printf("Nakon poziva procedure redoslijed brojeva je
             a=%d,b=%d\n", a, b);
20.    getch();}
```

Program 7.29.

```
 1.  #include <stdio.h>
 2.  #include <conio.h>
 3.
 4.  void zamijeni(int *x, int *y){
 5.    int z;
 6.    z=*x;
 7.    *x=*y;
 8.    *y=z;
 9.  }
10.
11.  main(){
12.    int a,b;
13.    printf("Unesite neki broj a:");
14.    scanf("%d",&a);
15.    printf("Unesi neki drugi broj b:");
16.    scanf("%d",&b);
17.    printf("Redoslijed brojeva je a=%d, b=%d\n", a, b);
18.    zamijeni(&a,&b);
19.    printf("Nakon poziva procedure redoslijed brojeva je
             a=%d, b=%d\n", a, b);
20.    getch();}
```

Program 7.30.

Računalni program koji slijedi (program 7.31.) traži od korisnika unos sedam cijelih brojeva. Brojevi se spremaju u jednodimenzionalno polje od sedam elemenata (linija 20.). Potom se ispunjeno polje dostavlja proceduri `sumaProsjek` (kroz argument koji je deklariran ključnom riječi `const`) koja izračunava ukupan broj parnih brojeva, sumu i prosjek unesenih brojeva (linija 4.). Broj parnih brojeva vraća kroz povratni tip, dok ostale izračunate vrijednosti vraća kroz dva argumenta deklarirana kao pokazivači.

Treba uočiti kako je procedura `sumaProsjek` pozvana unutar naredbe `printf`. Nakon njezina poziva slijede varijable `zbroj` i `prosjek` (linija 25.). To je moguće jer procedura ima povratni tip kroz koji vraća vrijednost (broj parnih brojeva) koja se ispisuje u naredbi `printf` te je procedura pozvana prije korištenja varijabli `zbroj` i `prosjek`, tako da je stigla izvesti izračun i vrijednosti spremiti u navedene varijable prije njihova ispisa.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.
 4.  int sumaProsjek(const int a[],int n,int *zbroj,float *prosjek){
 5.    int i,br=0;
 6.    *zbroj=0;
 7.    for(i=0;i<n;i++){
 8.      if(a[i]%2==0){
 9.        br++;}
10.      *zbroj+=a[i];
11.    }
12.    *prosjek=(float)*zbroj/n;
13.    return br;
14.  }
15.
16.  main(){
17.    int n,zbroj;
18.    float prosjek;
19.    n=7;
20.    int i,a[n];
21.    for(i=0;i<n;i++){
22.      puts("Unesi broj: ");
23.      scanf("%d",&a[i]);
24.    }
25.    printf("Uneseno je %d brojeva, od kojih je %d parnih.
           Ukupna suma brojeva je %d a prosjek %.2f", n,
           sumaProsjek(a,n,&zbroj,&prosjek),zbroj,prosjek);
26.    getch();}
```

Program 7.31.

Slijedi računalni program koji generira n slučajnih brojeva iz nekoga korisnički zadanog intervala brojeva. Broj generiranih slučajnih brojeva (n) definira korisnik. Za generiranje slučajnih brojeva koriste se funkcije `srand` (ona inicijalizira generator slučajnih brojeva) i `rand` (ona generira slučajni broj). Obje se funkcije nalaze u tijelu procedure `genA` koja je izrađena u svrhu generiranja i ispisa željenoga broja slučajno

generiranih brojeva iz definiranoga intervala brojeva (linija 6.). Generirani se brojevi spremaju u polje koje se potom ispisuje.

Ograničenje generiranja brojeva na neki interval izvedeno je primjenom operacije modulo i zbrajanja. Naime, funkcija `rand` vraća slučajno generirani broj koji je u intervalu od 0 do konstante `RAND_MAX` (vrijednost joj je 32767 – definirana je u zaglavnoj datoteci `stdlib.h`). Kako bi se ograničio interval slučajno generiranih brojeva, koristi se sljedeća formula: `generirani_broj = rand()%(širina_intervala + 1) + donja_granica`. Primjera radi, ako se žele generirati brojevi u intervalu od 5 do 11, tada je formula `generirani_broj = rand()%7 + 5`. Naime, `rand()%7` može poprimiti brojeve 0, 1, 2, 3, 4, 5 i 6. Konačni mogući generirani broj dobiva se tako da se ovim mogućim brojevima dodaje 5. Prema tome, slučajni broj može poprimiti vrijednost 5, 6, 7, 8, 9, 10 i 11, a to je upravo željeni interval brojeva od 5 do 11. Razlika intervala se računa u liniji 23.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  #include<stdlib.h>
 4.  #include<time.h>
 5.
 6.  void genA(int n,int c,int b ) {
 7.     int i, a[n];
 8.     srand(time(NULL));
 9.     for(i=0;i<n;i++){
10.     a[i]=rand()%c+b;
11.     printf("%d\n",a[i]);
12.     }
13.  }
14.
15.  main(){
16.     int n,x,y,g;
17.     printf("Unesite koliko zelite brojeva izgenerirati:");
18.     scanf("%d",&n);
19.     printf("Unesite donju granicu intervala generiranja:");
20.     scanf("%d",&y);
21.     printf("Unesite gornju granicu intervala generiranja:");
22.     scanf("%d",&g);
23.     x=g-y+1;
24.     printf("Sad ce se izgenerirati %d brojeva u intervalu
              %d-%d\n",n,y,g);
25.     genA(n,x,y);
26.     getch();}
```

Program 7.32.

Računalni program koji slijedi traži od korisnika pet unosa dana i mjeseca što je implementirano petljom `for`. Nakon svakoga unosa, podaci o danu i mjesecu dostavljaju se proceduri `godisnja_doba` (linija 25.). U ovisnosti o unesenome

danu i mjesecu, ova procedura ispisuje u kome se godišnjemu dobu nalazi uneseni datum.

```
 1.  #include<stdio.h>
 2.  #include<conio.h>
 3.  #include<math.h>
 4.
 5.  void godisnja_doba(int d,int m){
 6.     if(((d>=1 && d<=31) && (m==7 || m==8)) || (m==6 && d>=21) ||
           (m==9 && d<=22)){
 7.        printf("Datum %d.%d. je u ljetnom razdoblju.",d,m);
 8.     }
 9.     else if(((d>=1 && d<=31) && (m==10 || m==11)) ||
                (m==9 && d>=23) || (m==12 && d<=20)){
10.        printf("Datum %d.%d. je u jesenjem razdoblju.",d,m);
11.     }
12.     else if(((d>=1 && d<=31) && (m==1 || m==2)) ||
                (m==12 && d>=21) || (m==3 && d<=20)){
13.        printf("Datum %d.%d. je u zimskom razdoblju.",d,m);}
14.     else
15.        printf("Datum %d.%d. je u proljetnom razdoblju.",d,m);
16.  }
17.
18.  main(){
19.     int m,d,i;
20.     for(i=1;i<5;i++){
21.        printf("\nUnesite dan: ");
22.        scanf("%d",&d);
23.        printf("Unesi mjesec: ");
24.        scanf("%d",&m);
25.        godisnja_doba(d,m);
26.     }
27.     getch();}
```

Program 7.33.

Naredni računalni program demonstrira način na koji se zapis dostavlja proceduri. Procedura `pocetno` sprema u komponente zapisa podatke o imenu, dobi i plaći koji joj se dostavljaju njezinim drugim, trećim i četvrtim argumentom (linija 19. i 23.). Prvim joj se argumentom dostavlja adresa na kojoj mjesto poziva procedure očekuje rezultat. Treba uočiti kako je treći argument (`char *name`) deklariran kao pokazivač. To je zato što se ovim argumentom u proceduru dostavlja podatak o imenu koji u biti predstavlja niz znakova. U tijelu procedure `pocetno` se mogu uočiti dva načina pristupa komponenti zapisa preko adrese zapisa (`->` i `.`) – linija 30., 31. i 32. Nakon poziva procedure `pocetno`, u glavnom se dijelu programa poziva procedura `ispis_imena` kojoj se dostavlja zapis čije je komponente napunila procedura `pocetno` (linija 20. i 24.). Argument procedure `ispis_imena` nije pokazivač na zapis. Stoga se vrijednost zapisa proceduri prosljeđuje po vrijednosti, a ne po referenci. Time je onemogućena izmjena zapisa u tijelu procedure. Isto vrijedi i za ostale dvije procedure – `godine` i `novac`.

Treba uočiti poziciju na kojoj je izvršena deklaracija varijabli `prvi` i `drugi` (linija 11.). Zbog pozicije deklaracije, ove varijable su dostupne iz bilo koje procedure i njihova trajnost je koliko i trajnost cijeloga programa. Stoga je cijeli prikazani računalni program moguće izraditi bez argumenata kojima se procedurama dostavljaju zapisi `prvi` i `drugi`.

```
 1. #include <stdio.h>
 2. #include <string.h>
 3. #include <conio.h>
 4.
 5. typedef struct {
 6.    char ime[40];
 7.    int dob;
 8.    float placa;
 9. } Osoblje;
10.
11. Osoblje prvi,drugi;
12.
13. void pocetno(Osoblje *osoba, int godine,
                 char *ime, float novac);
14. void godine(Osoblje osoba);
15. void novac(Osoblje osoba);
16. void ispis_imena(Osoblje osoba);
17.
18. int main(){
19.    pocetno(&prvi, 24, "Ivan", 3245.56);
20.    ispis_imena(prvi);
21.    godine(prvi);
22.    novac(prvi);
23.    pocetno(&drugi, 28, "Marko",5555.56);
24.    ispis_imena(drugi);
25.    godine(drugi);
26.    novac(drugi);
27.    getch();}
28.
29. void pocetno(Osoblje *osoba, int godine,
                 char *name, float novac){
30.    strcpy(osoba->ime, name);
31.    osoba->dob = godine;
32.    (*osoba).placa=novac;
33. }
34.
35. void godine(Osoblje osoba){
36.    printf("\n%s ima %d godina.\n",osoba.ime, osoba.dob);
37. }
38.
39. void ispis_imena(Osoblje osoba){
40.    printf("Ime osobe je %s\n", osoba.ime);
41. }
```

```
42.
43. void novac(Osoblje osoba){
44.   printf("\n%s ima %.2f kuna placu.\n",
45.          osoba.ime,osoba.placa);
46. }
```

Program 7.34.

Računalni program koji slijedi traži od korisnika unos koordinata točke te pomak. Koordinate točke se spremaju u komponente zapisa `Koordinata` (linija 20. i 22.). Nakon unosa koordinata i pomaka, podaci se dostavljaju proceduri `pomakni` koja računa nove koordinate tako da starima nadoda pomak te ih spremi na mjesto starih koordinata (linija 9. i 10.). Treba uočiti kako je proceduri dostavljena adresa zapisa preko pokazivača `pt1`. Njemu je adresa zapisa dodijeljena u liniji 15. Nakon poziva procedure slijedi ispis novih koordinata (linija 24.).

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3.
 4. typedef struct {
 5.   int x;
 6.   int y; } Koordinata;
 7.
 8. void pomakni(Koordinata *s,int pomak){
 9.   (*s).x+=pomak;
10.   (*s).y+=pomak;
11. }
12.
13. int main(){
14.   Koordinata *pt1, t1;
15.   pt1=&t1;
16.   int x,pomak, y;
17.   printf("Unesite pomak komponenti strukture: ");
18.   scanf("%d",&pomak);
19.   printf("Unesite x komponentu: ");
20.   scanf("%d",&t1.x);
21.   printf("Unesite y komponenti t1: ");
22.   scanf("%d",&t1.y);
23.   pomakni(pt1,pomak);
24.   printf("x dio je= %d, y dio je = %d\n", t1.x, t1.y);
25.   getch();}
```

Program 7.35.

Naredni računalni program demonstrira svrhu ključne riječi `static` u deklaraciji varijable unutar tijela procedure. Pri izvođenju neke procedure računalo rezervira potrebne memorijske lokacije koje se oslobađaju odmah nakon završetka izvođenja tijela procedure. Samo varijable iz tijela procedure koje su deklarirane ključnom riječi `static` ostaju u memoriji računala i nakon izvođenja procedure.

Računalni se program sastoji iz procedure `primjer_st` koja svakim svojim pozivom povećava vrijednost `static` varijable za jedan (linija 6.) te ju ispisuje (linija 7.). U glavnome dijelu programa procedura se poziva tri puta. Kako je prilikom deklaracije `static` varijable ona postavljena na vrijednost 5, to program ispisuje brojeve 7, 8 i 9. Naime, svakim pozivom procedure `primjer_st` dodaje se jedan na vrijednost koju je `static` varijabla poprimila po završetku prethodnoga poziva procedure `primjer_st`.

Uklanjanjem riječi `static` iz deklaracije varijable u tijelu procedure, uzrokovalo bi se uklanjanje te varijable iz memorije računala po završetku izvođenja procedure. Stoga bi novi poziv procedure jednostavno ponovno rezervirao memorijsku lokaciju za varijablu i s novom varijablom izveo izračun. Zbog toga bi računalni program ispisao bojeve 7, 7, 7.

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3.
 4. void primjer_st(){
 5.   static int x = 6;
 6.   x++;
 7.   printf("%d\n", x);
 8. }
 9.
10. int main(){
11.   primjer_st();primjer_st();
12.   primjer_st();
13.   getch();}
```

Program 7.36.

Računalni program koji slijedi koristi svojstvo `static` varijable kako bi izračunao sumu pet unesenih brojeva. Nakon pojedinog unosa broja, računalni program ispisuje dotadašnju sumu. Broj unosa je implementiran petljom `for` (linija 15), dok se sam unos i sumiranje izvode u posebnoj proceduri `suma` (linija 4.).

```
 1. #include<stdio.h>
 2. #include<conio.h>
 3.
 4. void suma(){
 5.   int broj;
 6.   static int s=0;
 7.   printf("\nUnesi neki broj : ");
 8.   scanf("%d",&broj);
 9.   s+=broj;
10.   printf("\nSuma je %d",s);
11. }
12.
13. main(){
14.   int i;
15.   for(i=0;i<=4;i++)
```

```
16.       suma();
17.    getch();}
```

Program 7.37.

Naredni računalni program demonstrira rekurzivni poziv procedure. Računa se suma prvih `n` prirodnih brojeva (broj `n` unosi korisnik). Nakon unosa, poziva se procedura `suma` (linija 11.) koja vraća rezultat kroz povratni tip. U tijelu procedure nalazi se njezin poziv (poziva samu sebe) – linija 19. Treba uočiti uvjet za izlaz iz rekurzije (linija 16.) bez koga rekurzija ne bi imala kraja.

```
 1. #include <stdio.h>
 2. #include <conio.h>
 3.
 4. unsigned int suma(unsigned int n);
 5.
 6. main(){
 7.    unsigned int b;
 8.    unsigned int sumb;
 9.    printf("Unesite neki prirodni broj:\n");
10.    scanf("%d",&b);
11.    sumb=suma(b);
12.    printf("Suma prvih %d prirodnih brojeva je %d", b, sumb);
13.    getch();}
14.
15. unsigned int suma(unsigned int n){
16.    if(n==1)
17.       return 1;
18.    else
19.       return n+suma(n-1);
20. }
```

Program 7.38.

7.3.1. Zadaci za vježbu

1. Napisati program u kome korisnik unosi neki broj. Program treba ispisati slučajno generirani broj iz intervala 0 do unesenoga broja. Generirani broj treba vratiti procedura `slucajniBroj`, kojoj se dostavlja uneseni broj.
2. Napisati program u kome korisnik unosi tri broja. Potrebno je ispisati umnožak unesenih brojeva. Umnožak treba vratiti procedura `umnozakTriBroja` kojoj se dostavljaju uneseni brojevi.
3. Napisati program koji sadrži proceduru `kvadriraj`. Proceduri se dostavlja adresa varijable `broj` koja je deklarirana u glavnome dijelu programa. Procedura izračunava kvadrat vrijednosti koja se nalazi na proslijeđenoj adresi i izračunatu vrijednost glavnome programu vraća kroz isti argument. Vrijednost varijable `broj` unosi korisnik. U glavnome dijelu programa potrebno je ispisati vrijednost varijable `broj` prije i nakon poziva procedure `kvadriraj`.
4. Napisati program koji sadrži proceduru `povecajZaX`. Proceduri se dostavlja adresa na varijablu `broj` i vrijednost varijable `pomak`. Procedura treba kroz

posebni argument glavnome programu vratiti vrijednost koja je jednaka vrijednosti varijable `broj` uvećane za `pomak`. U glavnome dijelu programa korisnik unosi vrijednost varijable `broj` i `pomak`. Nakon poziva procedure, potrebno je ispisati rezultat u formatu `broj+pomak=rezultat`.

5. Napisati program koji sadrži proceduru `izmijeni`. Proceduri se dostavljaju dva podatka `broj1` i `broj2` – prvi preko reference, a drugi preko vrijednosti. Oba navedena podatka unosi korisnik u glavnome dijelu programa. Procedura treba izvesti kvadriranje vrijednosti varijable `broj2` i tu vrijednost smjestiti u varijablu `broj1`. U glavnome dijelu programa treba ispisati vrijednosti varijabli `broj1` i `broj2` prije i nakon poziva procedure.

6. Napisati program koji sadrži funkciju `recip`. Proceduri se iz glavnoga programa prosljeđuje `broj` koji je unio korisnik. Ona izračunava vrijednost `1/broj` i tu vrijednost vraća kroz argument u glavni programa. U glavnome programu treba izvesti ispis u formatu `1/broj=rezultat`.

7. Napisati program u kome u glavnome programu korisnik unosi jediničnu cijenu i količinu artikla. Potrebno je izraditi proceduru `ukupna_cijena` koja vraća ukupnu cijenu artikla ako joj se proslijede njegova jedinična cijena i količina. U slučaju da je količina veća od 10, treba na ukupnu cijenu aplicirati 5% popusta (izračun ukupne cijene treba se obaviti u proceduri). U glavnome dijelu programa, nakon poziva izrađene procedure, treba ispisati:
`Ukupna cijena artikla s jediničnom cijenom ... kn i količinom ... iznosi: ... kn`

8. Napisati program koji simulira izvlačenje loto brojeva (primjera radi, 7/39 – `brojBrojeva/najveciBroj`). Program treba sadržavati proceduru `izvuciBroj` kojoj se dostavlja `najveciBroj` koji može biti generiran. Procedura kroz argument vraća slučajno generirane brojeve (sprema ih u polje) u intervalu od 1 do `najveciBroj`. Vrijednosti `najveciBroj` i `brojBrojeva` unosi korisnik u glavnome dijelu programa. U glavnome dijelu programa treba ispisati slučajne brojeve koje je vratila procedura `izvuciBroj`. U ispisu treba biti toliko brojeva kolika je vrijednost `brojBrojeva`. Generirani se brojevi ne smiju ponavljati.

9. Napisati program u kome se nalazi procedura `provjera` kojoj se dostavlja cijeli broj, a ona vraća P ako je broj paran, odnosno N ako je neparan. Ako je nula, vraća se 0. U glavnome dijelu programa korisnik treba unijeti željeni broj unosa. Potom se uneseni brojevi spremaju u polje. Nakon unosa, u glavnome se dijelu programa izvodi ispis u dva stupca. U prvome se stupcu nalazi element polja, a u drugome vrijednost koju će kroz povratni tip vratiti procedura `provjera` kojoj se dostavlja element polja.

10. Napisati program u koji se unosi polje cijelih brojeva od `n` elemenata (korisnik definira `n`). Potom je potrebno pozvati proceduru `okreni` kojoj se dostavlja

polje. Procedura stvara novo polje tako da okreće elemente dostavljenoga polja. U novome je polju na prvome mjestu element koji je u dostavljenome polju bio na zadnjem mjestu itd. Novo je polje potrebno vratiti kroz povratni tip u glavnome dijelu programa u kome se novo polje treba i ispisati.

11. Napisati program u koji se unosi polje cijelih brojeva od `n` elemenata (korisnik definira `n`). Potom je potrebno pozvati proceduru `zamjeni` kojoj se dostavlja polje. Procedura treba izraditi novo polje tako da na indeksu na kome je u dostavljenome polju bio pozitivan broj bude znak +, odnosno znak – ako je negativan broj te 0 ako je nula. Novo polje je potrebno vratiti kroz argument u glavni dio programa u kome se treba u dva stupca ispisati staro i novo polje.

12. Napisati program koji sadrži proceduru `brojSlova_a`. Procedura treba vratiti broj pojavljivanja slova a (u obzir uzeti i veliko slovo A) u nizu znakova koji je proslijeđen proceduri. Niz znakova unosi korisnik u glavnome dijelu programa. Nakon poziva procedure, u glavnome dijelu programa treba ispisati rezultat koji je procedura vratila kroz povratni tip.

13. Napisati program koji sadrži funkciju `gdjeJe_d`. Procedura treba vratiti redni broj na kome se u nizu znakova prvi puta pojavljuje slovo d (u obzir uzeti i veliko slovo D). Niz znakova unosi korisnik u glavnome dijelu programa te se on prosljeđuje proceduri. Procedura vraća rezultat kroz argument. U glavnome dijelu programa, nakon poziva procedure, treba ispisati dobiveni rezultat.

14. Napisati program koji sadrži funkciju `koliko_X`. Proceduri se dostavlja polje cijelih brojeva. Korisnik u glavnome dijelu programa definira broj unosa te toliko cijelih brojeva unosi u polje. Potom unosi broj koji se traži. Polje i broj koji se traži prosljeđuju se proceduri. Ona treba izračunati koliko se puta traženi broj pojavljuje u polju i to vratiti kroz povratni tip. Nakon poziva procedure, u glavnome dijelu programa je potrebno ispisati rezultat koji je procedura vratila.

15. Napisati program koji računa volumen kvadra ($V=a \cdot b \cdot c$). Unose se širina (a), visina (b) i duljina (c) pravokutnika. Napisati proceduru `volumen_kvadra` koja kroz argument vraća volumen ako joj se proslijede širina (stranica a), visina (stranica b) i duljina (stranica c). Procedura još treba vratiti 1 ako se radi o kocki (uvjet a=b=c) ili 0 ako se ne radi o kocki (uvjet a<>b ili a<>c ili b<>c). Nakon poziva procedure, a u ovisnosti radi li se o kocki ili ne, u glavnome dijelu programa treba ispisati:

```
Volumen kocke iznosi: ...
```

Ili

```
Volumen kvadra iznosi: ...
```

8. RAD S DATOTEKAMA

Ovo poglavlje opisuje **rad s datotekama** u programskome jeziku C. Opisane su tekstualne i binarne datoteke. Prikazane su naredbe `fopen`, `fclose`, `fprintf`, `fscanf`, `fread`, `fwrite`. Prikazan je opći oblik ovih naredbi te primjeri njihove primjene.

Po završetku ovoga poglavlja čitatelj će moći:

- definirati pojam datoteke
- objasniti razliku između binarne i tekstualne datoteke
- objasniti i primijeniti pristupanje datoteci u programskome jeziku C
- objasniti i primijeniti pisanje u datoteku u programskome jeziku C
- objasniti i primijeniti čitanje datoteke u programskome jeziku C.

8.1. Pojam datoteke

Datoteka predstavlja imenovano memorijsko područje u kome su trajno zapisani podaci. Fizički se ona najčešće nalazi na nekoj vanjskoj memoriji (tvrdi disk, optički disk, memorijske kartice itd.). U osnovi, podaci koji se tijekom izvođenja nekoga računalnog programa nalaze u RAM memoriji računala, a koji se trebaju trajno spremiti, spremaju se u datoteku na vanjskoj memoriji (npr. Word dokument, baza podataka, sama izvršna verzija računalnog programa itd.). Općenito se razlikuju dvije vrste datoteka – tekstualne i binarne datoteke.

Tekstualne datoteke čuvaju nizove znakova koji su razdijeljeni u više redaka. U svakome retku ne nalazi se niti jedan ili se nalazi više znakova. Svaki redak završava posebnim znakom za novi red (u C to je `\n`). Namjena je tekstualnih datoteka pohrana tekstualnih podataka (nizova znakova). Nije im namjera pohraniti brojčane tipove podataka, zapisa itd.

Binarne datoteke čuvaju niz okteta (bajtova) podataka. U C-u bajt (8 bita) je predstavljen tipom podatka `char`. U biti, binarna datoteka sadrži podatke točno onako kako su oni zapisani u internoj RAM memoriji računala. Stoga se u njih mogu upisati bilo koji jednostavni (npr. u C-u: `int`, `float` itd.) ili složeni tipovi podataka (slog, polje).

Rad s datotekama, neovisno jesu li tekstualne ili binarne, prati sljedeća tri koraka:

1. otvori datoteku
2. čitaj iz datoteke ili piši u datoteku
3. zatvori datoteku.

8.2. Rad s datotekama u C

U programskome jeziku C postoji niz naredbi koje se koriste u radu s datotekama. Ovdje će biti prikazane osnovne naredbe za otvaranje i zatvaranje datoteke te za njezino čitanje i pisanje. Primjena naredbi za rad s datotekama zahtijeva uključivanje zaglavne datoteke `stdio.h`.

8.2.1. Otvaranje i zatvaranje datoteke

Otvaranje datoteke se izvodi naredbom (funkcijom) `fopen`. Opći oblik primjene ove naredbe glasi:

```
1.  ...
2.  FILE *datoteka;
3.  datoteka = fopen(ime_datoteke, vrsta_otvaranja);
4.  if (datoteka == NULL){
5.    blok_naredbi_za_gresku;
6.  }
7.  else {
8.    blok_naredbi_za_rad_s_datotekom;
9.  }
```

```
10.  fclose(datoteka);
11.  ...
```

Funkciji `fopen` se prosljeđuju dva podatka kroz dva argumenta. Prvim se argumentom funkciji dostavlja ime datoteke koju se želi otvoriti (uključujući i putanju do datoteke ako se ona ne nalazi u mapi u kojoj se nalazi datoteka izvršnoga programa koji se izvodi). Kroz drugi se argument dostavlja vrsta otvaranja datoteke. Sljedeća tablica prikazuje moguće vrste otvaranja datoteke.

Vrsta otvaranja datoteke	Značenje
`r`	Postojeća se datoteka otvara samo za čitanje.
`w`	Stvara se nova datoteka samo za pisanje.
`a`	Postojeća se datoteka otvara samo za dodavanje teksta.
`r+`	Postojeća se datoteka otvara za čitanje i pisanje.
`w+`	Stvara se nova datoteka za pisanje i čitanje.
`a+`	Postojeća se datoteka otvara za čitanje i dodavanje teksta.

Posebno za vrstu otvaranja `w`, `w+`, `a` i `a+` vrijedi:

- ako se postojeća datoteka otvara s `w` ili `w+`, njezin će sadržaj biti prebrisan novim sadržajem.
- ako datoteka koja se otvara s `a` ili `a+`, ne postoji, bit će stvorena nova datoteka.
- ako se datoteka otvara s `a` ili `a+`, novi će tekst biti dodan na kraj datoteke, odnosno na kraj teksta koji se nalazi u datoteci.

Datoteka koja se otvara s prikazanim vrstama otvaranja bit će tekstualna datoteka. Ako se svakoj od vrsta doda i oznaka `b`, tada će datoteka biti otvorena kao binarna datoteka. Dakle, vrste otvaranja binarne datoteke su:

Vrsta otvaranja datoteke	Značenje
`rb`	Postojeća se binarna datoteka otvara samo za čitanje.
`wb`	Stvara se nova binarna datoteka samo za pisanje.
`ab`	Postojeća se binarna datoteka otvara samo za dodavanje.
`rb+`	Postojeća se binarna datoteka otvara za čitanje i pisanje.
`wb+`	Stvara se nova binarna datoteka za pisanje i čitanje.
`ab+`	Postojeća se binarna datoteka otvara za čitanje i dodavanje.

I za vrstu otvaranja `wb`, `wb+`, `ab` i `ab+` vrijedi ono što je napisano za `w`, `w+`, `a` i `a+`.

Funkcija `fopen` vraća pokazivač (adresu) na tip podatka `FILE`. Adresa se treba spremiti u odgovarajući pokazivač (u općemu je primjeru to varijabla `datoteka` – linija 2.). Prije daljnjega rada s varijablom `datoteka` potrebno je provjeriti je li u njoj spremljena neka adresa ili vrijednost `NULL` (linija 4.). Vrijednost `NULL` u varijabli `datoteka` upućuje na to da datoteka s navedenim imenom nije mogla biti otvorena i stoga je potrebno izvesti blok naredbi `blok_naredbi_za_gresku` (u kojoj se, primjera radi, može nalaziti ispis poruke o grešci). Ako pak varijabla datoteka ima vrijednost različitu od `NULL`, tada će biti izveden blok naredbi `blok_naredbi_za_rad_s_datotekom` (primjera radi, čitanje iz datoteke).

Cjelokupan rad s datotekom obavezno mora završiti njezinim zatvaranjem za što se koristi naredba `fclose` kojoj se dostavlja ime datoteke (linija 10.).

8.2.2. Čitanje i pisanje tekstualne datoteke

U programskome jeziku C postoji niz naredbi (funkcija) za čitanje i pisanje tekstualne datoteke. Ovdje će se prikazati naredbe `fprintf` (naredba za pisanje) i `fscanf` (naredba za čitanje) Obje se naredbe koriste pri formatiranom pisanju i čitanju datoteke.

Opći oblik naredbe `fprintf` glasi:

```
1.  ...
2.  fprintf(datoteka, format, podatak1, ...);
3.  ...
```

Prije primjene funkcije `fprintf` datoteka mora biti otvorena u nekoj vrsti koja omogućava pisanje u datoteku (na otvorenu datoteku pokazuje pokazivač `datoteka`, koja se funkciji dostavlja kroz prvi argument). Drugi argument funkcije prima niz znakova koji predstavlja format ispisa (slično kao i u naredbi `printf`). Primjera radi, format može biti "%s %d", ili "Broj je: %d". Iza drugog argumenta slijede argumenti kojima se funkciji dostavljaju podaci koji trebaju biti ispisani na mjestu konverzijskoga znaka u formatu. Primjera radi, funkcija `fprintf(datoteka, "Ime: %s Ocjena: %d\n", "Marko", "5")` će u datoteku upisati sljedeći niz znakova `"Ime: Marko Ocjena: 5"` i novi redak.

Opći oblik naredbe `fscanf` glasi

```
1.  ...
2.  int broj_Objekata;
3.  broj_Objekata = fscanf(datoteka, format, podatak1, ...);
4.  ...
```

Prije primjene naredbe `fscanf` potrebno je datoteku otvoriti u nekoj vrsti koja omogućava njezino čitanje (pokazivač `datoteka` pokazuje na otvorenu datoteku – prvi argument funkcije). Drugi argument funkcije je format čitanja. Primjera radi, format čitanja može biti "%s %d". Iza drugoga argumenta slijede varijable u koje će se prebaciti podaci iz datoteke (slično kao kod naredbe `scanf`). Primjera radi, funkcija `fscanf(datoteka, "%s %d", ime, &ocjena)` će iz datoteke učitati podatke tako da će prvi podatak u retku biti smatran nizom znakova koji će biti spremljen u varijablu `ime`, dok će drugi podatak u retku biti smatran cijelim brojem koji će biti spremljen u varijablu `ocjena`. Funkcija `fscanf` vraća `broj_objekata` koji su učitani ili `EOF` što označava grešku ili kraj datoteke.

Neka je sadržaj tekstualne datoteke sljedeći:

```
Ime: Marko Ocjena: 5
Ime: Ana Ocjena: 4
```

Format koji bi se trebao definirati u `fscanf` funkciji, a koji bi čitao podatke je sljedeći "`%s %s %s %d`". Funkcija bi trebala četiri varijable u koje bi upisala podatke iz jednoga retka (podaci u jednome retku razdijeljeni su jednim razmakom).

Neka se treba izraditi program koji korisniku ispisuje izbornik – (1) spremi podatke u datoteku i (2) učitaj i ispiši podatke iz datoteke. Ako se izabere (1) korisnika se traži da unese pet studenata (ime, prezime i postotak ocjene). Potom se uneseni podaci spremaju u tekstualnu datoteku (`Studenti.txt`), svaki u svoj redak. Podaci su razdijeljeni znakom tabulatorom. Ako se izabere (2), učitavaju se podaci iz datoteke i ispisuju se u obliku tablice. U računalnome programu koristi se polje od pet elemenata u kome se nalazi struktura `Student`.

Slijedi računalni program.

```
 1. #include<stdio.h>
 2. typedef struct{
 3.     char ime[30+1];
 4.     char prezime[30+1];
 5.     float ocjena;
 6. } Student;
 7.
 8. void spremanjePodataka(){
 9.     Student studenti[5];
10.     int brojac;
11.     FILE *datoteka;
12.     system("cls");
13.     printf("UNOS I SPREMANJE PODATAKA:\n\n");
14.     for (brojac = 0; brojac < 5; brojac++){
15.        printf("Unos %d. studenta.\n", brojac + 1);
16.        printf("Ime studenta: ");
17.        scanf("%s", studenti[brojac].ime);
18.        printf("Prezime studenta: ");
19.        scanf("%s", studenti[brojac].prezime);
20.        printf("Ocjena studenta: ");
21.        scanf("%f", &studenti[brojac].ocjena);
22.     }
23.     datoteka = fopen("Studenti.txt", "w");
24.     if (datoteka==NULL){
25.        printf ("\nGRESKA U STVARANJU DATOTEKE.");
26.     }
27.     else{
28.        for (brojac = 0; brojac < 5; brojac++){
29.           fprintf(datoteka, "%s\t%s\t%f\n", studenti[brojac].ime,
                 studenti[brojac].prezime, studenti[brojac].ocjena);
30.        }
31.     }
32.     fclose (datoteka);
33.     getch();
34. }
35.
36. void citanjePodataka(){
37.     Student studenti[5];
38.     int brojStudenata = 0;
39.     int brojac;
40.     FILE *datoteka;
41.     system("cls");
42.     printf("CITANJE I ISPIS PODATAKA:\n\n");
43.     datoteka = fopen("Studenti.txt", "r");
```

```
44.     if (datoteka==NULL){
45.        printf ("\nGRESKA U OTVARANJU DATOTEKE.");
46.     }
47.     else{
48.        while (fscanf(datoteka, "%s\t%s\t%f",
                        studenti[brojStudenata].ime,
                        studenti[brojStudenata].prezime,
                        &studenti[brojStudenata].ocjena) != EOF){
49.           brojStudenata++;
50.        }
51.        printf("\n\nIme\tPrezime\tOcjena\n")  ;
52.        for (brojac = 0; brojac < brojStudenata; brojac++){
53.           printf("%s\t%s\t%f\n",
                     studenti[brojac].ime,
                     studenti[brojac].prezime,
                     studenti[brojac].ocjena);
54.        }
55.     }
56.     fclose (datoteka);
57.     getch();
58. }
59. main(){
60.     int izbor;
61.     do{
62.     system("cls");
63.     printf("(1)Spremi podatke u datoteku\n");
64.     printf("(2)Ucitaj i ispisi podatke iz datoteke\n");
65.     printf("(3)Izadji iz programa ");
66.     scanf("%d", &izbor);
67.     if(!( izbor >=1 && izbor <= 3))
68.        printf ("\n\aNepostojeci izbor.");
69.     else if (izbor == 1)
70.        spremanjePodataka();
71.       else if (izbor == 2)
72.        citanjePodataka();
73.       else
74.        break;
75.     } while (1==1);
76.     getch();}
```

Program 8.1.

Nakon stvaranja datoteke `Student.txt`, ona se može otvoriti nekim računalnim programom za pregled tekstualnih datoteka (primjera radi *Notepad*). Sadržaj datoteke je potpuno čitljiv.

8.2.3. Čitanje i pisanje binarne datoteke

Za čitanje i pisanje binarne datoteke u programskome jeziku C koriste se naredbe `fread` (za čitanje) i `fwrite` (za pisanje).

Opći oblik naredbe `fread` glasi:

```
1. ...
2. fread(podatak, velicina_podatka, broj_podataka, datoteka);
3. ...
```

Prije primjene funkcije `fread` datoteka treba biti otvorena na dogovarajući način (pokazivač `datoteka` se funkciji dostavlja kroz četvrti argument). Funkcija iz

datoteke čita onoliko podataka koliko je navedeno u argumentu `broj_podataka`. Veličina tih podataka, odnosno broj bajtova iz kojih se sastoje, naveden je u drugom argumentu `velicina_podatka`. Pročitani se podaci spremaju na adresu na koju pokazuje argument (pokazivač) `podatak`.

Opći oblik naredbe `fwrite` glasi:

```
...
fwrite(podatak, velicina_podatka, broj_podataka, datoteka);
...
```

Slično kao i pri funkciji `fread`, prije primjene funkcije `fwrite` datoteka treba biti otvorena na dogovarajući način (pokazivač na datoteku se funkciji dostavlja kroz četvrti argument – `datoteka`). Funkcija u datoteku čita onoliko podataka koliko je navedeno u argumentu `broj_podataka`. Veličina podataka (broj bajtova) naveden je u drugom argumentu `velicina_podatka`. Podaci koji se trebaju spremiti u datoteku nalaze se na adresi koju pokazuje argument `podatak`.

Slijedi računalni program analogan onom iz programa 8.1., u kome se koristi binarna datoteka.

```
#include<stdio.h>
typedef struct{
   char ime[30+1];
   char prezime[30+1];
   float ocjena;
} Student;

void spremanjePodataka(){
   Student studenti[5];
   int brojac;
   FILE *datoteka;
   system("cls");
   printf("UNOS I SPREMANJE PODATAKA:\n\n");
   for (brojac = 0; brojac < 5; brojac++){
      printf("Unos %d. studenta.\n", brojac + 1);
      printf("Ime studenta: ");
      scanf("%s", studenti[brojac].ime);
      printf("Prezime studenta: ");
      scanf("%s", studenti[brojac].prezime);
      printf("Ocjena studenta: ");
      scanf("%f", &studenti[brojac].ocjena);
   }
   datoteka = fopen("Studenti.bin", "w");
   if (datoteka==NULL){
      printf ("\nGRESKA U STVARANJU DATOTEKE.");
   }
   else{
      fwrite (studenti, sizeof(studenti), 1, datoteka);
   }
   fclose (datoteka);
   getch();
}
```

```
33.
34.  void citanjePodataka(){
35.     Student studenti[5];
36.     int brojStudenata = 0;
37.     int brojac;
38.     FILE *datoteka;
39.     system("cls");
40.     printf("CITANJE I ISPIS PODATAKA:\n\n");
41.     datoteka = fopen("Studenti.bin", "r");
42.     if (datoteka==NULL){
43.        printf ("\nGRESKA U OTVARANJU DATOTEKE.");
44.     }
45.     else{
46.        fread(studenti, sizeof(studenti), 1, datoteka);
47.        printf("\n\nIme\tPrezime\tOcjena\n")     ;
48.        for (brojac = 0; brojac < 5; brojac++){
49.           printf("%s\t%s\t%f\n",
                      studenti[brojac].ime,
                      studenti[brojac].prezime,
                      studenti[brojac].ocjena);
50.        }
51.     }
52.     fclose (datoteka);
53.     getch();
54.  }
55.  main(){
56.     int izbor;
57.     do{
58.        system("cls");
59.        printf("(1)Spremi podatke u datoteku\n");
60.        printf("(2)Ucitaj i ispisi podatke iz datoteke\n");
61.        printf("(3)Izadji iz programa ");
62.        scanf("%d", &izbor);
63.        if(!( izbor >=1 && izbor <= 3))
64.           printf ("\n\aNepostojeci izbor.");
65.        else if (izbor == 1)
66.           spremanjePodataka();
67.        else if (izbor == 2)
68.           citanjePodataka();
69.        else
70.           break;
71.     } while (1==1);
72.     getch();}
```

Program 8.2.

Nakon stvaranja datoteke `Student.bin`, ona se također može otvoriti nekim računalnim programom za pregled tekstualnih datoteka. No, sada njezin sadržaj nije čitljiv. U računalnome programu treba uočiti primjenu funkcije `sizeof` koja vraća veličinu tipa podatka neke varijable (linija 28. i 46.). Ona je korištena kako bi se funkcijama `fwrite`, odnosno `fread` dostavila veličina podatka (broj bajtova) koji se trebaju upisati u datoteku, odnosno iz nje pročitati.

9. PRONALAŽENJE I ISPRAVLJANJE POGREŠAKA U RAČUNALNOME PROGRAMU

Poglavlje opisuje **pronalaženje i ispravljanje pogrešaka u računalnome programu** pisanom u programskome jeziku C uz primjenu alata Dev-C++. Prikazani su sljedeći pojmovi: alat za pronalaženje pogrešaka (*engl. debugger*), točka prekida izvođenja programa (*engl. break point*), izvedi naredni korak (*engl. next step*), izvedi unutrašnje korake (*engl. step into*), nastavi izvršavanje (*engl. continue*) i dodaj varijablu na promatranje (*engl. add watch*).

Po završetku ovoga poglavlja čitatelj će moći:

- objasniti vrste pogrešaka u računalnome programu
- objasniti pojam alata za pronalaženje pogrešaka (*engl. debugger*)
- objasniti sljedeće pojamove: točka prekida izvođenja programa (*engl. break point*), izvedi naredni korak (*engl. next step*), izvedi unutrašnje korake (*engl. step into*), nastavi izvršavanje (*engl. continue*) i dodaj varijablu na promatranje (*engl. add watch*).
- primijeniti alat za pronalaženje pogrešaka.

9.1. Pogreške u računalnome programu

U računalnome se programu mogu pojaviti dvije osnovne vrste pogrešaka – sintaktičke i semantičke. Sintaktičke pogreške su one koje su nastale tijekom pisanja računalnoga programa kojim su se naredbe koristile na nedopušteni način. Primjera radi, u programskome jeziku C linija koda nije završila s točka zarez (;), blok naredbi nije zatvoren vitičastom zagradom, primjenjuje se varijabla koja nije deklarirana itd. Ovakvu vrstu pogreške računalo automatski otkriva prilikom kompajliranja programskoga koda. Posebno označi liniju koda koja nije prošla sintaktičku validaciju te navede poruku u kojoj programeru dodatno opisuje pogrešku (neki alati znaju predložiti rješenje – primjera radi alat Eclipse).

Semantička pogreška je pogreška koja se pojavljuje u računalnome programu koji je prošao sintaktičku validaciju i koji se može izvesti. Nju računalo ne može automatski uočiti. Semantičku pogrešku uočava korisnik računalnoga programa. Primjera radi, ako računalni program računa površinu pravokutnika, te nakon korisničkoga unosa stranica u iznosu 5 i 6 kao rezultat prikaže 2 (ili cijeli računalni program prestane s izvođenjem) iskazala se semantička pogreška (jer je očekivani rezultat trebao biti 30). Semantičke se pogreške, odnosno njihovo iskazivanje, otkrivaju u fazi testiranja računalnoga programa. Nameće se kako pitanje kako pronaći njihov uzrok. U svrhu pronalaska i ispravljanja ovakvih pogreški koriste se posebni alati za pronalazak i ispravljanje pogreški (*engl. debugger*). Ovi alati pružaju mogućnost programeru da kontrolira korake (naredbe, linije koda) koje računalo izvodi prilikom pokretanja računalnoga programa te da promatra kako se mijenjaju vrijednosti varijabli. Pri promatranju linija koda koje se izvode i vrijednosti varijabli, programer uspoređuje očekivano izvođenje naredbi i vrijednosti varijabli s promatranom situacijom. Kada promatrana situacija odstupa od očekivane, programer izvodi dodatnu analizu točke u kojoj je došlo do promjene jer se tu krije semantička pogreška.

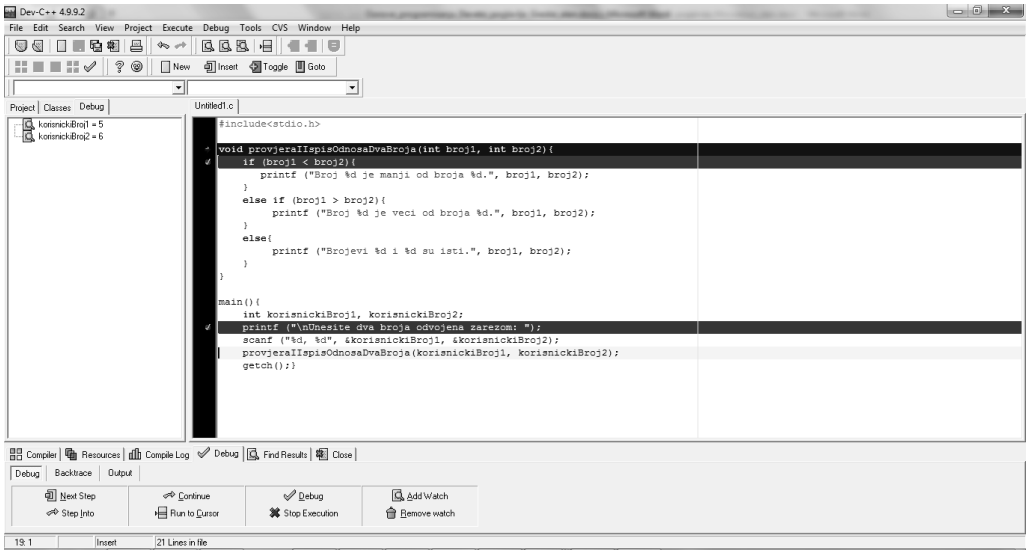
Alat za pronalaženje i ispravak pogreški omogućava programeru označavanje linije koda (*engl. break point*) do koje računalo može izvesti program i u kojoj treba izvođenje zaustaviti i čekati daljnja uputstva programera. Moguće je označiti više različitih linija koda.

Nakon što računalo stane s izvođenjem računalnoga programa u označenoj liniji koda, programer može naređivati računalu da dalje izvodi naredbe korak po korak (*engl. next step*), da izvede sve naredbe unutar neke procedure i po završetku se opet zaustavi (*engl. step into*) ili da nastavi s izvođenjem naredbi sve dok ne dođe do sljedeće označene linije koda i dok ne dođe do kraja programa (*engl. continue*).

Pored navedenoga, programer može navesti nazive varijabli čije vrijednosti želi promatrati tijekom izvođenja računalnoga programa (*engl. add watch*).

9.2. Alat za pronalazak i ispravljanje pogreški u računalnome programu Dev-C++

U računalnome programu Dev-C++ postoji posebni izbornik kojim se programski kod pokreće u posebnom obliku (*engl. debug*). Točke prekida se u programskome kodu postavljaju tako da se pokazivačem miša klikne na tamno područje ispred željene linije koda. Nakon toga, linija koda biti će crveno označena (na slici linije sa svjetlijom oznakom). Unutar izbornika *Debug* postoji još jedan izbornik *Debug* kojim se pokreće računalni program u "debug" načinu izvođenja. Liniju koda koju treba izvesti računalo označava plavom bojom (na slici linija s tamnijom oznakom). Nakon pokretanja računalnoga programa, na donjem dijelu Dev-C++ se otvara područje *Debug* u čijem se sklopu nalaze kontrole izvođenja programa (*Next Step, Step Into, Continue*). Ovdje se nalazi i operacija za dodavanje varijabli čija vrijednost će se promatrati (*Add Watch*). Na slici se mogu primijetiti dvije varijable (`korisnickiBroj1` i `korisnickiBroj2`) koje su stavljene na promatranje (lijevo područje na slici). Može se uočiti da trenutno ove varijable imaju vrijednost 5 i 6, jer su im te vrijednosti unesene preko naredbe `scanf`.



LITERATURA

Svaka knjiga obično završava popisom izvora koje su njezini autori koristili u pisanju i na koje se referiraju. Ovaj udžbenik nema popisa izvora budući da su se autori u pisanju oslanjali na svoja praktična programerska i nastavnička iskustva. Objašnjenja i praktični primjeri koje su autori koristili tijekom višegodišnjega održavanja nastave iz kolegija Osnove programiranja skupljeni su i uobličeni u tekst pogodan za udžbenik.

Ipak, kao dodatne izvore koje čitatelji mogu iskoristiti u dostizanju zacrtanih ishoda učenja, autori predlažu sljedeće internetske izvore:

1. Programmiz, http://www.programiz.com/, 24.02.2014.
2. Programming Simplified, http://www.programmingsimplified.com/, 24.02.2014.
3. C Programming Tutorial, http://www.tutorialspoint.com/cprogramming/index.htm, 24.02.2014.

PRILOG – PODSJETNIK

1. Varijable

1.1. Osnovni tipovi podataka

Tip	C jezik (ključna riječ)	Interval	Zauzeće memorije
znakovni tip	`char` `unsigned char`	-127 .. 128 0 .. 255	1 byte 1 byte

cjelobrojni tip	`int`
	`short`
	`long`
		-2147483648.. 2147483647
		-32768 .. 32767
		-2147483648.. 2147483647
			4 byte
			2 byte
			4 byte

kardinalni tip	`unsigned`
	`unsigned short`
	`unsigned long`
		0 .. 4294967295
		0 .. 65535
		0 .. 4294967295
			4 byte
			2 byte
			4 byte

realni tip	`float`	min ± 1.175494351e-38 maks ± 3.402823466e+38	4 byte
realni tip dvostruke preciznosti	`double`	min ± 2.2250738585072014e-308 maks ± 1.7976931348623158e+308	8 byte

1.2. Aritmetički operatori

Ako je `var_a` varijabla koja sadrži cjelobrojnu vrijednost 10 i `var_b` varijabla koja sadrži cjelobrojnu vrijednost 20 tada vrijedi:

Operator	Opis	Primjer	
`+`	zbraja dva operanda	`var_a + var_b`	(rezultat je 30)
`-`	oduzima drugi operand od prvoga	`var_a - var_b`	(rezultat je -10)
`*`	množi dva operanda	`var_a * var_b`	(rezultat je 200)
`/`	dijeli prvi operand s drugim	`var_b / var_a`	(rezultat je 2)
`%`	ostatak pri cjelobrojnome dijeljenju prvoga operanda s drugim (modulo)	`var_b % var_a`	(rezultat je 0)
`++`	povećava operand za jedan	`var_a++`	(rezultat je 11)
`--`	smanjuje operand za 1	`var_a--`	(rezultat je 9)
`-`	mijenja predznak operanda	`-var_a`	(rezultat je -10)

1.3. Relacijski operatori

Relacijskim se operatorima grade jednostavni logički izrazi koji mogu poprimiti dvije vrijednosti – istina i neistina. Ako se u logičkim izrazima koriste konkretne vrijednosti tipa podatka `char`, tada se one trebaju navesti unutar jednostrukih navodnika (`'`) –

npr. `znak !='a'`. Konkretni se brojčani tipovi podataka navode izravno – npr. `broj >= 5.56`.
Neka `var_a` sadrži cjelobrojnu vrijednost 10 i `var_b` cjelobrojnu vrijednost 20. Tada vrijedi:

Operator	Opis	Primjer
==	provjerava jesu li jednake vrijednosti dva operanda	`(var_a == var_b)` uvjet nije istinit
!=	provjerava jesu li vrijednosti dva operanda različite	`(var_a != var_b)` uvjet je istinit
>	provjerava je li vrijednost prvoga operanda strogo veća od vrijednosti drugoga operanda	`(var_a > var_b)` uvjet nije istinit
<	provjerava je li vrijednost prvoga operanda strogo manja od vrijednosti drugoga operanda	`(var_a < var_b)` uvjet je istinit
>=	provjerava je li vrijednost prvoga operanda veća ili jednaka od vrijednosti drugoga operanda.	`(var_a >= var_b)` uvjet nije istinit
<=	provjerava je li vrijednost prvoga operanda manja ili jednaka od vrijednosti drugoga operanda	`(var_a <= var_b)` uvjet je istinit

1.4. Logički operatori

Logički operatori povezuje jednostavne logičke izraze u složenije koji mogu poprimiti dvije vrijednosti – istina i neistina.
Ako je `izraz_a` jednostavni logički izraz s vrijednošću istinit, a `izraz_b` jednostavni logički izraz s vrijednošću neistinit, tada vrijedi:

Operator	Opis	Primjer
&&	logičko I – ako oba operanda imaju vrijednost istina, tada je rezultat logičkoga I istina. Inače je rezultat neistina.	`(izraz_a && izraz_b)` rezultat je neistina
||	logičko ILI – ako barem jedan operand ima vrijednost istina, tada je rezultat logičkoga ILI istina. Inače je neistina.	`(izraz_a || izraz_b)` rezultat je istina
!	logička negacija – ako logički izraz ima vrijednost neistina, rada je rezultat logičke negacije istina. Inače je neistina.	`!izraz_a` rezultat je neistina `!(izraz_a && izraz_b)` rezultat je istina

1.5. Operator pridruživanja

Operator pridruživanja (=) pridružuje vrijednost na njegovoj desnoj strani varijabli koja se nalazi na njegovoj lijevoj strani. Ako je `var_a` varijabla koja sadrži cjelobrojnu vrijednost 10 i `var_b` varijabla koja sadrži cjelobrojnu vrijednost 20 tada ako je `var_c = var_a + var_b;` u `var_c` će se nalaziti vrijednost 30.

1.6. Skraćeni operatori

Skraćenim operatorima se mogu zapisati neki algebarski izrazi.

Operator	Opis	Primjer
`+=`	operator zbrajanja i pridruživanje – dodaje desni operand lijevome i rezultat pridružuje lijevom operandu	`var_b += var_a` ekvivalentno s `var_b = var_b + var_a`
`-=`	operator oduzimanja i pridruživanje – oduzima desni operand od lijevoga i rezultat pridružuje lijevom operandu	`var_b -= var_a` ekvivalentno s `var_b = var_b - var_a`
`*=`	operator množenja i pridruživanje – množi desni operand s lijevim i rezultat pridružuje lijevom operandu	`var_b *= var_a` ekvivalentno s `var_b = var_b * var_a`
`/=`	operator dijeljenja i pridruživanje – dijeli lijevi operand s desnim i rezultat pridružuje lijevom operandu	`var_b /= var_a` ekvivalentno s `var_b = var_b / var_a`
`%=`	operator modulo i pridruživanje – ostatak pri cjelobrojnome dijeljenju lijevoga operanda s desnim pridružuje lijevom operandu	`var_b %= var_a` ekvivalentno s `var_b = var_b % var_a`

1.7. Zapis

Definicija zapisa:

```
typedef struct {
   Tip_Podatka1 ime_Komponente1;
   Tip_Podatka2 ime_Komponente2;
   ...
   Tip_PodatkaN ime_KomponenteN;
} Identifikator_Zapisa;
```

Deklaracija varijable novim složenim tipom podatka (zapisom):

```
Identifikator_Zapisa ime_Varijable;
```

Dolazak do neke komponente zapisa:

```
ime_Varijable.ime_Komponente;
```

1.7. Polje

Deklaracija jednodimenzionalnoga polja:

```
Tip_podatka ime_Varijable[Broj_elemenata];
```

Inicijalizacija jednodimenzionalnoga polja:

```
char znakovi[] = "Rijec";
ili
char znakovi[10] = "Rijec";

float brojevi[5] = {327.44, 6.52, 12.0};
ili
float brojevi[] = {327.44, 6.52, 12.0, 21.3, 10.0};
```

Pridruživanje vrijednosti jednodimenzionalnome polju:

```
int brojevi[3];
char znakovi[5];

brojevi[0]=-56;
znakovi[0]='a';
```

Deklaracija dvodimenzionalnoga polja:

```
tip_podatka ime_varijable[broj_redaka][broj_stupaca];
```

Pridruživanje vrijednosti dvodimenzionalnome polju:

```
int matrica[3][2];
matrica[2][1]=21;
```

1.8. Pokazivači

Operator	Opis
*	operator dereferenciranja ili indirekcije. Vraća vrijednost koja se čuva na memorijskoj lokaciji na koju pokazuje pokazivač.
&	adresni operator. Vraća adresu memorijske lokacije koja je rezervirana za varijablu.

Deklaracija pokazivača:

```
tip_podatka *ime_varijable;
```

Primjer:

```
int var_a = 5;
int *var_b;
var_b = &var_a;
printf ("%d", *var_b);
```

2. Neke funkcije u C

Funkcija	Primjer	Opis	Zaglavna datoteka
printf	`printf("Unesite broj:");` `printf("%d", naziv_varijable);`	ispis teksta i/ili vrijednosti varijable	stdio.h
scanf	`scanf("%d",&naziv_varijable);`	unos vrijednosti varijable	stdio.h
getch	`getch();`	unos jednoga znaka (tip podatka `char`)	conio.h
puts	`puts("Unesite broj:");` `puts(naziv_varijable);`	ispis teksta ili vrijednosti varijable (znakovnoga niza)	stdio.h
gets	`gets(naziv_varijable);`	unos vrijednosti varijable (znakovnoga niza)	stdio.h
strcpy	`strcpy(var_odrediste, "Tekst");` `strcpy(var_odrediste, var_izvor);`	kopiranje niza znakova u odredišnu varijablu	string.h
strcat	`strcat(var_odrediste, "Tekst");` `strcat(var_odrediste, var_izvor);`	dodavanje niza znakova odredišnoj varijabli	string.h
strlen	`strlen("Tekst");` `strlen(naziv_varijable);`	vraća vrijednost koja predstavlja duljinu niza znakova (broj znakova)	string.h

pow	`pow(5.4,2);`
	`pow(var_baza,-4.5);`
	`pow(var_baza,var_eksponent);`
		vraća vrijednost potenciranja baze na eksponent	math.h

sqrt	`sqrt(5);` `sqrt(var_vrijednost);`	vraća vrijednost kvadratnoga (drugoga) korijena nekog realnog broja	math.h
fabs	`fabs(-5.67)` `fabs(var_vrijednost);`	vraća apsolutnu vrijednost nekoga realnog broja. Za cijele brojeve se koristi `abs`.	math.h
sin	`sin(var_vrijednost)`	vraća sinus kuta u radijanima	math.h
cos	`cos(var_vrijednost)`	vraća kosinus kuta u radijanima	math.h
tan	`tan(var_vrijednost)`	vraća tanges kuta u radijanima	math.h
log10	`log10(var_vrijednost)`	vraća logaritam broja po bazi 10	math.h
log	`log(var_vrijednost)`	vraća logaritam broja po bazi e	math.h
exp	`exp(var_vrijednost)`	vraća potenciranje baze e na eksponent	math.h
rand	`rand();`	generira slučajne cijele brojeve do limita koji je definiran u stdlib.h	stdlib.h
srand	`srand(time(0))`	inicijalizira generator slučajnih brojeva	stdlib.h
system	`system("cls");`	omogućava brisanje ekrana	stdlib.h

Posebni znakovi kod ispisa i unosa podataka:

Znak	Opis
`\n`	prijelaz u novi red
`\b`	pomak za jedno mjesto u desno
`\r`	povratak na početak reda
`\t`	horizontalni tabulator
`\a`	oglašavanje zvuka upozorenja
`%p`	ispis broja u heksadecimalnome zapisu
`%e`	ispis broja u znanstvenoj notaciji (e-notaciji)
`%s`	ispis/unos niza znakova
`%f`	ispis/unos realnoga broja
`%c`	ispis/unos jednoga znaka
`%d`	ispis/unos cijeloga broja

3. Razgranata algoritamska struktura

3.1. Sintaksa naredbe `if-else if-else`

```
 1. ...
 2. naredbaX
 3. if(logički_izraz_1)
 4. {
 5.   blok_naredbi_1
 6. }
 7. else if (logički_izraz_2)
 8. {
 9.   blok_naredbi_2
10. }
11. ...
12. else if (logički_izraz_n)
13. {
14.   blok_naredbi_n
15. }
16. else
17. {
18.   blok_naredbi_x
19. }
20. naredbaY
21. ...
```

3.2. Sintaksa naredbe `switch-case`

```
...
naredbaX
switch (izraz){
case konstanta_1:
    blok_naredbi_1
    break
case konstanta_2:
    blok_naredbi_2
...
case konstanta_n:
    blok_naredbi_n
default:
    blok_naredbi_x
}
naredbaY
...
```

4. Ciklička algoritamska struktura

4.1. Naredba `for`

```
...
naredbaX
for(inicijalizacija; logički_izraz; korak)
{
   blok_naredbi;
}
naredbaY
...
```

4.2. Naredba `while`

```
...
naredbaX
while(logički_izraz)
{
   blok_naredbi;
}
naredbaY
...
```

4.3. Naredba `do-while`

```
1.  ...
2.  naredbaX
3.  do{
4.     blok_naredbi;
5.  }
6.  while(logički_izraz);
7.  naredbaY
8.  ...
```

5. Algoritamska struktura bezuvjetnoga skoka

Naredba	Opis
`break`	izlazak iz bloka naredbi (npr. kod `switch-case`, `for`, `while`, `do-while`)
`continue`	izvođenje narednoga ciklusa petlje
`return`	povratak iz procedure
`exit`	završetak izvođenja programa
`goto`	skok na simboličko ime linije programskoga koda

6. Deklaracija i definicija procedura

Definicija procedure može se nalaziti ispred ili iza `main` funkcije. Ako se nalazi iza, tada se procedura mora deklarirati prije početka `main` funkcije.
Opći oblik definicije i njezin poziv procedure u C:

```
 1.  povratni_tip naziv_procedure(const tip_argumenta1 naziv_argumenta1,
                                  tip_argumenta2 naziv_argumenta2,
                                  ...
                                  tip_argumentan naziv_argumentan){
 2.     naredba1
 3.     return povratna_vrijednost;
 4.  }
 5.  neka_procedura(){
 6.     naredbaX
 7.     vrijednost = naziv_procedure (konkretna_vrijednost,
                                       varijabla1,
                                      ...
                                      varijablan);
 8.     naredbaY
 9.     ...
10.  }
```

7. Rad s datotekama

Otvaranje i zatvaranje datoteke:

```
 1. ...
 2. FILE *datoteka;
 3. datoteka = fopen(ime_datoteke, vrsta_otvaranja);
 4. if (datoteka == NULL){
 5.   blok_naredbi_za_gresku;
 6. }
 7. else {
 8.   blok_naredbi_za_rad_s_datotekom;
 9. }
10. fclose(datoteka);
11. ...
```

Vrsta otvaranja tekstualne datoteke	Značenje
r	postojeća se datoteka otvara samo za čitanje
w	stvara se nova datoteka samo za pisanje
a	postojeća se datoteka otvara samo za dodavanje teksta
r+	postojeća se datoteka otvara za čitanje i pisanje
w+	stvara se nova datoteka za pisanje i čitanje
a+	postojeća se datoteka otvara za čitanje i dodavanje teksta

Posebno za vrstu otvaranja `w`, `w+`, `a` i `a+` vrijedi:

- ako se postojeća datoteka otvara s `w` ili `w+`, njezin će sadržaj biti prebrisan novim sadržajem
- ako datoteka koja se otvara s `a` ili `a+` ne postoji, bit će stvorena nova datoteka.
- ako se datoteka otvara s `a` ili `a+`, novi će tekst biti dodan na kraj datoteke, odnosno na kraj teksta koji se nalazi u datoteci

Vrsta otvaranja binarne datoteke	Značenje
rb	postojeća se binarna datoteka otvara samo za čitanje
wb	stvara se nova binarna datoteka samo za pisanje
ab	postojeća se binarna datoteka otvara samo za dodavanje
rb+	postojeća se binarna datoteka otvara za čitanje i pisanje
wb+	stvara se nova binarna datoteka za pisanje i čitanje
ab+	postojeća se binarna datoteka otvara za čitanje i dodavanje

I za vrstu otvaranja `wb`, `wb+`, `ab` i `ab+` vrijedi ono što je napisano za `w`, `w+`, `a` i `a+`.

Formatirano čitanje i pisanje u tekstualnu datoteku:

```
1. ...
2. fprintf(datoteka, format, podatak1, ...);
3. ...
```

```
1.  ...
2.  int broj_Objekata;
3.  broj_Objekata = fscanf(datoteka, format, podatak1, ...);
4.  ...
```

Čitanje i pisanje u binarnu datoteku:

```
1.  ...
2.  fread(podatak, velicina_podatka, broj_podataka, datoteka);
3.  ...
```

```
1.  ...
2.  fwrite(podatak, velicina_podatka, broj_podataka, datoteka);
3.  ...
```

Primjer formatiranoga čitanja i pisanja:

```
 1.  ...
 2.  datoteka = fopen("Studenti.txt", "w");
 3.  if (datoteka==NULL){
 4.    printf ("\nGRESKA U STVARANJU DATOTEKE.");
 5.  }
 6.  else{
 7.    fprintf(datoteka, "%s\t%s\t%f\n", ime, prezime, ocjena);
 8.  }
 9.  fclose (datoteka);
10.  ...
```

```
 1.  ...
 2.  datoteka = fopen("Studenti.txt", "r");
 3.  if (datoteka==NULL){
 4.     printf ("\nGRESKA U OTVARANJU DATOTEKE.");
 5.  }
 6.  else{
 7.     while (fscanf(datoteka, "%s\t%s\t%f", ime, prezime, ocjena) != EOF){
 8.           brojStudenata++;
 9.     }
10.  }
11.  fclose (datoteka);
12.  ...
```

Primjer čitanja i pisanja u binarnu datoteku:

```
 1.  ...
 2.  datoteka = fopen("Studenti.bin", "w");
 3.  if (datoteka==NULL){
 4.    printf ("\nGRESKA U STVARANJU DATOTEKE.");
 5.  }
 6.  else{
 7.    fwrite (studenti, sizeof(studenti), 1, datoteka);
 8.  }
 9.  fclose (datoteka);
10.  ...
```

```
 1.  ...
 2.  datoteka = fopen("Studenti.bin", "r");
 3.  if (datoteka==NULL){
 4.    printf ("\nGRESKA U OTVARANJU DATOTEKE.");
 5.  }
 6.  else{
 7.    fread(studenti, sizeof(studenti), 1, datoteka);
 8.  }
 9.  fclose (datoteka);
10.  ...
```

O AUTORIMA

Izv. prof. dr. sc. Alen Jakupović, prof. v. š., diplomirao je na Pedagoškom fakultetu Sveučilišta u Rijeci te stekao stručni naziv profesora matematike i informatike. Magistrirao je na fakultetu Organizacije i informatike u Varaždinu, Sveučilišta u zagrebu na polju informacijskih i komunikacijskih znanosti. Doktorirao je na Filozofskom fakultetu Sveučilišta u Zagrebu na polju informacijskih znanosti. U nastavnom zvanju profesora visoke škole u trajnom zvanju izabran je na Veleučilištu u Rijeci, dok je u znanstveno-nastavnom zvanju izvanrednog profesora izabran na Odjelu za informatiku Sveučilišta u Rijeci. Nositelj je niza kolegija na Stručnim studijima informatike i telematike, te Specijalističkoga studija informatike na Veleučilištu u Rijeci. Sudjeluje u nastvi doktorskog studija iz polja informacijskih i komunikacijskih znanosti na Odjelu za informatiku Sveučilišta u Rijeci. Osim u nastavi, aktivan je i u gospodarstvu gdje je sudjelovao i sudjeluje u više informatičkih projekata vezanih za razvoj programske podrške u poslovnim procesima. Zanstveni i stručni rad provodi u sklopu projekata u koje je uključen kao istraživač ii voditelj. Područje njegovog interesa uključuje: umjetnu inteligenciju (posebno načine prikaza znanja), metode i metrike u razvoju informacijskih sustava, modeliranje, simulaciju i emulaciju sustava, Internet -Of-Things te primjenu IKT-a u obrazovanju. Objavio je preko 30 znanstvenih članaka u međunarodnim časopisima i na međunarodnim konferencijama.

Dr. sc. Sabrina Šuman, viši predavač, diplomirala je na Filozofskom fakultetu Sveučilišta u Rijeci te stekla stručni naziv profesora matematike i informatike. Doktorirala je na Odjelu za Informatiku Sveučilišta u Rijeci na polju informacijskih i komunikacijskih znanosti. U nastavnom zvanju višeg predavača izabrana je na Veleučilištu u Rijeci. Nositeljica je niza kolegija na Stručnim studijima informatike i telematike, te Specijalističkih studija informatike i poduzetništva na Veleučilištu u Rijeci. Zanstveni i stručni rad provodi u sklopu projekata u koje je uključena kao

istraživač ili suradnik. Područja njezinog znanstvenog interesa uključuju: primjena metoda umjetne inteligencije, procesiranje prirodnih i formalnih jezika, upravljanje kvalitetom, poslovno izvješćivanje i dubinske analize podataka. Objavila je preko 25 znanstvenih i sručnih članaka u međunarodnim časopisima i na međunarodnim konferencijama, te 5 recenziranih nastavnih materijala.