Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Dražen Bajer

Bilješke s auditornih vježbi iz Programiranja 1

[verzija 0.8279]

Osijek, 2018.

Ova skripta pokriva gradivo obrađeno na auditornim vježbama iz Programiranja 1. Poznavanje navedenog gradiva nužno je za samostalni rad na laboratorijskim vježbama te za rješavanje zadataka na pismenom ispitu. Sadržaj skripte je podijeljen na 14 poglavlja, gdje je zadanje poglavlje dodatak. Uz to su na kraju dana četiri posebna poglavlja sa zadacima za vježbu.

Na početku svakog poglavlja je sažeto prikazano gradivo koje obuhvaća, dok su na kraju dani riješeni programski primjeri kroz koje je ilustrirana primjena prethodno opisanog. Stoga je za potpuno svladavanje gradiva potrebno posvetiti pažnju navedenim primjerima uz koje su osim mogućih rješena dana i kratka obrazloženja. Mnogi od zadataka u poglavljima sa zadacima za vježbu bili su na nekim od prijašnjih pismenih ispita. Stoga, manje ili više ogledaju što se može očekivati na pismenom ispitu.

Sadržaj

POGLAVLJE 1

Uvod u programski jezik C

Programski jezik C je proceduralni jezik opće namjene. Stvoren je ranih 70-tih godina 20. stoljeća u Bellovim laboratorijima (*Bell Labs*) od strane Dennisa M. Ritchiea. Standardiziran je 1989. godine od ANSI (neformalno nazivan C89 ili ANSI C), a godinu dana kasnije i od ISO (neformalno nazivan C90 i identičan C89)[1]. Bez obzira na njegovu starost i danas je vrlo opsežno korišten u industriji i znanosti. Ovo se posebno odnosi na primjene gdje je nužno učinkovito i brzo izvođenje programa. Primjerice, ključni elementi gotovo svih operacijskih sustava pisani su u programskom jeziku C (originalno razvijen za potrebe sistemskog programiranja). Također, brojni prevoditelji ili interpreteri za druge programske jezike pisani su u njemu, a iznimno je popularan i u razvoju programske podrške za ugrađene računalne sustave (engl. *embedded computer systems*). Na koncu, o značaju ovog jezika i njegovom utjecaju u računarstvu dovoljno govori postojanje pojma "C-oliki programski jezici" (engl. *C-based/C-family programming languages*).

1.1 Zašto programski jezik C?

Programski jezik C ima mnoga poželjna svojstva i značajke kojima se ističe u odnosu na druge programske jezike. Iako spada u više programske jezike, on pruža pristup konceptima na strojnoj razini [primjerice, operacije nad (memorijskim) adresama i bitovima], što omogućuje pisanje programa koji se vrlo brzo izvode te učinkovito koriste dostupne resurse odnosno memoriju. S obzirom da postoje prevoditelji za gotovo sve moguće računalne platforme, programi pisani u C-u portabilni su. To znači kako ih je potrebno samo prevesti na odgovarajućoj platformi, često bez potrebe značajnijih izmjena teksta programa. Uz to, radi se o malenom jeziku koji u odnosu na mnoge druge, posebno modernije, ima manje značajki. Na ovo upućuje i relativno mali broj ključnih riječi[2] čiji je pregled dan je u tablici 2.1. Međutim, jezik se uvelike oslanja na funkcije standardne biblioteke [3].

Programski jezik C pruža veliku slobodu programeru (podrazumijeva se da programer zna što radi). Osim toga, ne provode se detaljne provjere grešaka kao što je slučaj u

[1] Noviji standard je uslijedio 1999. godine (neformalno nazivan C99), a zadnji 2011. godine (neformalno nazivan C11). Ovi standardi su uveli neke nove značajke u jezik koje se mogu pronaći u modernijim programskim jezicima.

[2] Navedene ključne riječi odnose se na standard jezika C89/C90. Noviji standardi su uveli neke dodatne.

[3] Standardna biblioteka se isporučuje uz prevoditelje i sadrži ogroman broj funkcija koje obavljaju različite zadatke, a stoje programeru na raspolaganju.

Tablica 1.1: Ključne riječi programskog jezika C

`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`	`while`

nekim drugim jezicima. Ovo s jedne strane omogućuje pisanje kôda koji ne bi bio ispravan u drugim jezicima te uz brojne tipove podataka i operatore koje nudi omogućuje pisanje sažetog kôda. S druge strane, ovo daje prostor greškama koje je teško pronaći i ispraviti te nudi pisanje kôda koji je teško razumljiv[4]. Još jedan oblik slobode koji jezik pruža je proizvoljna uporaba praznina – razmaka, tabulatora i novih redaka. Na taj način, programer ima slobodu oblikovanja teksta programa što primjerice nije slučaj u programskom jeziku Python.

Na koncu, programski jezik C predstavlja izvrsnu podlogu pri učenju brojnih drugih i modernijih jezika kao što su C++, C# i Java. Naime, mnogi jezici su sintaksno (gramatički) vrlo slični i koriste iste konstrukcije za upravljanje tijekom izvođenja.

1.2 Put od teksta programa do izvršne datoteke programa

Izgradnja izvršne datoteke[5] (engl. *executable file*) iz teksta programa[6] ili izvornog kôda (engl. *source code*) provodi se pri uporabi programskog jezika C u dva koraka. Jednostavnije rečeno, programer piše tekst programa koji predstavlja običnu tekstualnu datoteku, a kako bi se dobila izvršna datoteka istog, nužno je da on prođe kroz dva koraka. Ta dva koraka su prevođenje (engl. *compiling*) i povezivanje (engl. *linking*). Shodno tome, programi koji ih obavljaju nazivaju se redom prevoditelj (engl. *compiler*) te povezivač ili kolektor (engl. *linker*).

Prevođenjem se tekst programa pretvara u strojni kôd (engl. *machine language code*) razumljiv računalu. Međutim, rezultat prevođenja nije program kojeg je moguće pokrenuti, nego objektna datoteka[7] (engl. *object file*). Nedostaje kôd za pokretanje (engl. *start-up code*) koji služi kao spona između napisanog programa i operacijskog sustava (za različite operacijske sustave u pravilu je potreban različiti kôd za pokretanje). Uz to, (gotovo) svi programi pisani u programskom jeziku C koriste se kôdom dostupnim u različitim bibliotekama (primjerice, za ispis na ekran) kojeg je također potrebno dodati. Dodavanje kôda za pokretanje i kôda iz biblioteka ili drugih prevedenih datoteka na objektnu datoteku (prevedeni tekst programa) je posao koji se obavlja tijekom povezivanja. Konačno, rezultat dobiven nakon povezivanja je izvršna datoteka koju je moguće i pokrenuti.

Opisani pristup izgradnji izvršne datoteke programa omogućuje modularni razvoj. Naime, programi se mogu pisati u više datoteka, što je uobičajeno za složenije programe, jer pojednostavljuje/olakšava razvoj. Izmjena jedne ili više datoteka zahtjeva samo njihovo

[4] Postoji međunardno natjecanje u pisanju što teže čitljivog kôda – The International Obfuscated C Code Contest, http://www.ioccc.org/.

[5] Izvršne datoteke na Windows operacijskim sustavima imaju proširenje .exe, a na sustavima zasnovanim na UNIX-u .out.

[6] Datoteke teksta programa imaju proširenje .c.

[7] Objektne datoteke u pravilu imaju proširenje .o ili .obj.

ponovno prevođenje, a ne svih datoteka, dok se na kraju povezivanjem objedinjuju/spajaju sve datoteke.

1.3 Osnovna struktura programa pisanih u programskom jeziku C

U osnovi programi pisani u programskom jeziku C sastoje se od (pretprocesorskih) direktiva, imenovanih blokova i naredbi. Navedeni blokovi se u jeziku nazivaju funkcijama[8] (engl. *functions*) unutar kojih se nalaze naredbe (engl. *statements*). Uz to, ti blokovi/funkcije mogu sadržavati i neimenovane blokove. Ukratko, blokovi su omeđeni parom vitičastih zagrada (`{` i `}`), a naredbe završavaju točkom-zarez (`;`). Direktive provodi pretprocesor prije prevođenja. Jednostavno rečeno, pretprocesor mijenja izvorni kôd iz jednog u drugi oblik.

Svaki program mora sadržavati funkciju imena `main` (zadano ime koje mora biti korišteno) koja predstavlja ulaznu/početnu točku izvođenja programa. Zapravo, više nije ni potrebno za cjelovit program. Ovo je ilustrirano primjerom danim u izlistanju 1.1.

```
int main(void)
{
  return 0;
}
```

Izlistanje 1.1: Elementarni primjer programa pisanog u programskom jeziku C

U prvom retku/liniji `int main(void)` predstavlja ime funkcije, gdje okrugle zagrade [`main()`] naznačavaju da je `main` funkcija, ključna riječ `int` ispred imena funkcije naznačava da ona vraća cjelobrojnu vrijednost (tip funkcije), a `void` unutar zagrada poslije imena naznačava da funkcija nema parametre, odnosno ne prima argumente. Tijelo funkcije počinje s otvorenom vitičastom zagradom ({) i završava sa zatvorenom (}). Unutar njega nalazi se samo jedna naredba `return 0;` koja određuje koju vrijednost funkcija vraća (vrijednost poslije ključne riječi `return`). Prema opisanom, program samo vraća vrijednost nula njegovom pozivatelju kojeg predstavlja operacijski sustav.

U izlistanju 1.2 je prikazan blago složeniji primjer. Za razliku od prethodnog sadrži jednu direktivu te dodatnu naredbu. Navedeni program ispisuje poruku *Programiranje 1* na standardni izlaz kojeg u pravilu predstavlja ekran.

```
#include <stdio.h>

int main(void)
{
  printf("Programiranje 1");
  return 0;
}
```

Izlistanje 1.2: Nešto složeniji primjer programa pisanog u programskom jeziku C

Tekst programa izlistanja 1.2 započinje s `#include <stdio.h>`, gdje je naredba `#include` pretprocesorska direktiva (nije naredba jezika C). Rezultat navedene direktive je uključi-

[8] U nekim drugim programskim jezicima imenovani blokovi se nazivaju podrutinama ili procedurama.

vanje cjelokupnog sadržaja datoteke `stdio.h`[9] na početak teksta programa. Podsjetnika radi, ovaj korak obavlja pretprocesor na zahtjev prevoditelja prije samog prevođenja. Prva naredba u tijelu funkcije je `printf("Programiranje 1");` koja predstavlja poziv (korištenje) funkcije (imena `printf`) iz standardne biblioteke. Funkcija `printf()` opisana je u datoteci `stdio.h` koju je je zbog toga bilo nužno uključiti u tekst programa. U konkretnom primjeru ona ispisuje niz/slijed znakova između dvostrukih navodnika na ekran. Kao i u prethodnom primjeru (izlistanje 1.1) program završava s naredbom `return 0;` što je pravilno. Važno je napomenuti kako program završava s izvođenjem čim se izvrši naredba `return 0;` pa eventualne naredbe napisane nakon nje nikada ne bi bile izvršene.

1.4 Korištenje standardnog ulaza i izlaza

Programski jezik C ne uključuje ugrađene mehanizme za ispis (na izlaz) i unos/čitanje (s ulaza) podataka, nego se za to oslanja na funkcije dostupne u standardnoj biblioteci. Postoje brojne funkcije za rukovanje ulazom i izlazom (ulazno/izlazne funkcije koje su temelj za komunikaciju s programom). Ulaz (standardni ulaz) uobičajeno predstavlja tipkovnicu, dok izlaz (standardni izlaz) uobičajeno predstavlja ekran. Dvije često korištene i svestrane funkcije sa širokim rasponom mogućnosti prikazane su u nastavku.

Funkcija `printf()` omogućuje formatiranje samog ispisa. Ona nije ograničena na ispis samo jednostavnih poruka. U izlistanju 1.3 dani primjer programa daje uvid tek u jedan mali dio njenih mogućnosti.

```
#include <stdio.h>

int main(void)
{
  int broj;
  broj = 7;

  printf("Prosti brojevi su prirodni brojevi veci od 1 ");
  printf("koji nemaju pozitivnih djelitelja osim broja 1 i samog sebe.\n");
  printf("Broj %d je prost broj!\n", broj);

  return 0;
}
```

Izlistanje 1.3: Primjer uporabe funkcije za ispis na izlaz/ekran

Prvu uočiljivu razliku u odnosu na prethodne primjere čine naredbe `int broj;` i `broj = 7;` koje se odnose na varijablu korištenu u programu. Prva je naredba deklaracije (engl. *declaration statement*) kojom se najavljuje varijabla[10] imena `broj` koja je cjelobrojnog tipa podatka, a što je naznačeno ključnom riječi `int` ispred imena. Druga je naredba pridruživanja/dodjele (engl. *assignment statement*) kojom se pridružuje vrijednost 7 varijabli `broj`. To je ostvareno operatorom pridruživanja = kojim se vrijednost s desne strane dodjeljuje lijevoj. Nadalje, može se primijetiti u drugom pozivu funkcije `printf()` da se na kraju poruke nalazi `\n` što je poseban znak koji označava prelazak pokaznika (engl. *cursor*) na početak novog retka/linije. Zadnji poziv funkcije `printf()` izgleda značajno drukčije

[9] Datoteke s proširenjem .h predstavljaju zaglavne datoteke (engl. *header files*). Postoji veliki broj zaglavnih datoteka, gdje svaka sadrži opise/informacije jednog dijela standardne biblioteke. Tako datoteka stdio.h sadrži informacije o dijelu koji se odnosi na standardni ulaz/izlaz.

[10] Grubo rečeno, varijabla je spremnik za podatak.

u odnosu na prethodne. Unutar poruke (omeđena dvostrukim navodnicima) nalazi se oznaka `%d`, nakon poruke je ime varijable, a odvojeni su zarezom. U ovom slučaju bit će ispisana zadana poruka kao i ranije, ali će umjesto `%d` biti postavljena/ispisana vrijednost varijable `broj` odnosno 7.

Za unos ili čitanje podataka s ulaza/tipkovnice uobičajno se koristi funkcija `scanf()`. Ona ima brojne mogućnosti za interpretiranje ulaza – unos s tipkovnice je samo niz znakova. U primjeru danom u izlistanju 1.4 prikazan je samo trivijalan slučaj uporabe navedene funkcije.

```
#include <stdio.h>

int main(void)
{
  int broj1, broj2;

  printf("Unesite dva cijela broja s tipkovnice, odvojena razmakom. ");
  printf("Nakon unosa pritisnite tipku ENTER.\n");

  scanf("%d", &broj1);
  scanf("%d", &broj2);
  printf("\nBrojevi %d i %d su uneseni.\n", broj1, broj2);

  return 0;
}
```

Izlistanje 1.4: Primjer uporabe funkcije za unos s ulaza/tipkovnice

Slično kao u prethodnom primjeru u izlistanju 1.3, na početku tijela funkcije `main()` nalazi se naredba deklaracije koja redom najavljuje dvije varijable imena `broj1` i `broj2`. Obje su istog cjelobrojnog tipa podatka (na koji ukazuje ključna riječ `int` ispred njihovih imena). Nakon dvije naredbe koje su pozivi funkcije `printf()`, slijede dva poziva funkcije `scanf()` koji se blago razlikuju. Prvo, oznaka `%d` između dvostrukih navodnika određuje da se unos s tipkovnice interpretira kao cijeli broj. Drugo, nakon zareza slijedi ime varijable koje prethodni operator `&` (adresni operator) koji obavezan i koji ukazuje funkciji `scanf()` gdje se nalazi ta varijabla. Pri korištenju, funkcija `scanf()` interpretira unos s tipkovnice i takvog ga sprema zadanu varijablu. Prema opisanom, u konkretnom primjeru, prvi unesni broj bit će spremljen u varijablu imena `broj1`, a drugi u varijablu imena `broj2`.

1.5 Komentari

Komentari su dio teksta programa koji služe samo za dokumentiranje napisanog. Prema tome, oni služe samo čovjeku koji čita tekst programa, a prije prevođenja ih se uklanja iz teksta programa. Dobra je praksa koristiti ih kako bi se pojasnio napisani programski kôd. Potreba za opsežnom uporabom komentara može pak upućivati na nejasno napisan programski kôd što treba izbjegavati. Čitljivo i jasno napisan programski kôd stoga značajno smanjuje potrebu za komentarima.

Jednostavno rečeno, komentari su tekst omeđen s parom `/*`, `*/`. Sve između `/*` i `*/` je komentar. Navedene oznake mogu se protezati kroz više linija/redaka teksta programa što ih čini prikladnim za komentiranje veće količine teksta. Nekoliko primjera dano je izlistanjem 1.5. Nadalje, standard jezika C99 uveo je novu vrstu komentara, linijski komentar (može se reći kako je preuzet iz programskog jezika C++). Oznaka koja se

koristi za tu vrstu komentara je `//`. Sve što slijedi iza te oznake, u istom retku, je komentar. U izlistanju 1.6 je navedeno nekoliko primjera linijskih komentara.

```
/* Ovo je komentar u jednoj liniji */

/*
  Ovaj komentar
  proteze se kroz
  vise linija
*/
```

Izlistanje 1.5: Primjeri komentara

```
// Ovo je komentar u jednoj liniji

// Oznaka se odnosi samo
// na liniju u kojoj se nalazi
// odnosno na ono sto slijedi iza nje
```

Izlistanje 1.6: Primjeri linijskih komentara

1.6 Riješeni primjeri

U nastavku je navedeno nekoliko primjera zadataka u kojima je potrebno koristiti funkcije `printf()` i `scanf()`. Dana rješenja predstavljaju potpune tekstove programa na temelju kojih je moguće izgraditi izvršne datoteke programa. Svi primjeri uključuju u većoj ili manjoj mjeri komentare primarno kako bi se prikazala njihova uporaba, a ne zbog potrebe za njima. Treba napomenuti kako se u primjerima nigdje ne navodi teskt kao što je "Napisati tekst programa u programskom jeziku C koji...", nego se podrazumijeva.

Primjer 1. *Ispisati na ekran poruku "Dobar dan i dobro dosli!".*

```
#include <stdio.h>

int main(void)
{
  printf("Dobar dan i dobro dosli!"); //poruka se navodi izmedu dvostrukih navodnika
  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Razlikuje od drugog primjera s početka samo po poruci koju se ispisuje.

Primjer 2. *Deklarirati jednu cjelobrojnu varijablu i pridružiti joj proizvoljnu vrijednost. Potom ispisati vrijednost varijable na ekran.*

```
#include <stdio.h>

int main(void)
{
  int a;  /* deklaracija cjelobrojne varijable   */

  a = 12; /* pridruzivanje vrijednosti varijabli */

  printf("%d", a); //ispis vrijednosti varijable na ekran

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Rudimentarno rješenje bez suvišnih poruka.

Primjer 3. *Omogućiti korisniku unos jednog cijelog broj, a nakon unosa ispisati poruku koja uključuje taj broj.*

```
#include <stdio.h>

int main(void)
{
  int b; //deklaracija varijable u koju ce biti spremljen uneseni broj

  scanf("%d", &b); //unos podatka/cijelog broja
  printf("Broj koji ste unijeli je: %d", b); //ispis unesenog broj uz poruku

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Moguće je dodati ispis prikladne poruke prije poziva funkcije `scanf()` kako bi se postigla bolja komunikacija s korisnikom. Poruka koja se ispisuje na ekran proizvoljno je odabrana, jer nije definirana primjerom/zadatkom.

Primjer 4. *Omogućiti korisniku unos dva cijela broja. Prije unosa ispisati prikladnu poruku sa zahtjevom za unos. Na kraju ispisati svaki broj zasebno uz odgovarajuću poruku.*

```
#include <stdio.h>

int main(void)
{
  int a, b; //deklaracija varijabli

  printf("Unesite dva cijela broja:\n");

  //Moguce je obaviti trazeni unos sa samo jednim pozivom funkcije scanf()
```

```
  scanf("%d %d", &a, &b);
  /*
  Ekvivalentno dva zasebna poziva:
      scanf("%d", &a);
      scanf("%d", &b);
  */

  printf("Drugi broj je %d\n", b); //ispis vrijednosti druge varijable
  printf("Prvi broj je %d\n", a);  //ispis vrijednosti prve varijable

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Traženi unos dva cijela broja ostvaren je sa samo jednim pozivom funkcije `scanf()`. Treba primijetiti da su u pozivu navedene obje varijable i za svaku po oznaka `%d`. Važno je ne zaboraviti da imenima varijabli prethodi operator `&` kod korištenja funkcije `scanf()`.

POGLAVLJE 2

Osnovni tipovi podataka

Računalni programi rukuju podacima koji mogu biti u različitim oblicima. U osnovi, ti podaci su numeričke prirode. Programski jezik C ima podršku za dvije kategorije podataka – cjelobrojni (engl. *integer*) i realni (engl. *floating point*). Obje kategorije dolaze u nekoliko varijanti ili tipova, a razlikuju se u rasponu vrijednosti koje podržavaju. Više različitih tipova za istu kategoriju omogućuju prilagodbu programa računalnim okolinama ili zahtjevima na učinkovitost, jer općenito podatkovni objekti različitih tipova imaju različitu potrebu za memorijom. Sve navedeno je zadano s nekoliko ključnih riječi u jeziku čiji je pregled dan u tablici 2.1.

Tablica 2.1: Ključne riječi vezane za osnovne tipove podatka

`char`	`double`	`float`
`int`	`long`	`short`
`signed`	`unsigned`	`void`

2.1 Varijable i konstante

Podaci s kojima program radi mogu biti unaprijed zadani i nepromjenjivi/fiksni ili se mogu mijenjati tijekom njegovog izvođenja. Zadani i nepromjenjivi podaci predstavljaju konstante (engl. *constants*) u jeziku. Ugrubo, konstante su vrijednosti zapisane u tekstu programa i prema tome nisu izmjenjive tijekom izvođenja programa. Suprotno konstantama, varijablama (engl. *variables*) se vrijednost može mijenjati ili pridruživati. Jednostavno rečeno, varijable su mjesta za spremanje/pohranu podataka za vrijeme izvođenja programa. I dok prevoditelj može odrediti tip podatka konstante na temelju njenog izgleda/zapisa, za varijable je nužno zadati njen tip podatka.

Varijable je obavezno deklarirati (engl. *declaration*) prije njihovog korištenja. Deklaracija varijable obuhvaća navođenje željenog tipa podatka i imena[1], redom. Općenito, varijable se deklariraju na način:

```
tip_podatka ime_varijable;
```

[1]Imena varijabli mora započinjati slovom ili podvlakom (_), a što može slijediti kombinacija slova, znamenki i podvlaka. Važno je napomenuti kako je jezik osjetljiv na mala i velika slova pa primjerice, `broj`, `Broj` i `bRoj` predstavljaju tri različita imena.

Sve varijable moraju biti deklarirane na početku bloka (imenovanog ili neimenovanog). To znači kako se ispred naredbi deklaracije ne mogu nalaziti druge vrste naredbi. Međutim, od standarda C99 to ograničenje više ne postoji pa je dovoljno varijable deklarirati prije prvog korištenja. To je uobičajena praksa u jezicima kao primjerice, C++ i Java te može učiniti tekst programa jednostavnijim za pratiti.

Deklaracijom varijable nalaže se zauzimanje mjesta u memoriji koje se asocira s njenim imenom. Međutim, ne dodjeljuje joj se nikakva vrijednost. Zapravo, zadržava se raspored bitova u djelu memorije koji je zauzet. Takve varijable se često nazivaju neinicijaliziranim varijablama. Varijablama je moguće dodijeliti početnu vrijednost proširenjem deklaracije što predstavlja inicijalizaciju (engl. *initialization*). Općenito, varijabla se inicijalizira na način:

```
tip_podatka ime_varijable = vrijednost;
```

Početno zadana vrijednost može biti konstanta, varijabla koja ima vrijednost ili izraz.

Treba spomenuti kako je moguće deklarirati više od jedne varijable jednom naredbom deklaracije. Pristup je isti kao ranije, ali uz navođenje više imena varijabli nakon zadanog tipa podatka. Imena se razdvajaju zarezima. Jasno, sve varijable pri deklaraciji su istog tipa podatka. Štoviše, i u ovom slučaju, dijelu ili svim navedenim varijablama može biti zadana i početna vrijednost (inicijalizacija) što pak nije preporučljivo zbog otežanog čitanja teksta programa.

Uporaba varijabli i konstanti ilustrirana je tekstom programa koji slijedi. U primjeru navedenom u izlistanju 2.1 postoje dvije varijable i dvije konstante. Varijable su imena `starost` i `visina`, gdje je `starost` cjelobrojnog tipa (`int`), a `visina` realnog tipa podatka (`double`). Konstante predstavljaju brojevi 19 i 1.87 koji su redom cjelobrojnog i realnog tipa podatka. Štoviše, istog su tipa kao i varijable kojima su pridružene.

```
#include <stdio.h>

int main(void)
{
  int starost = 19; //incijalizacija varijable starost
  double visina;    //deklaracija varijable visina

  visina = 1.87;    //pridruzivanje vrijednosti varijabli visina

  return 0;
}
```

Izlistanje 2.1: Primjeri varijabli i konstanti u programu

2.2 Tipovi podataka

Kao što je spomenuto na početku, u programskom jeziku C postoje dvije kategorije podataka. One se značajno razlikuju u zapisu u računalu što je ilustrirano slikom 2.1. Stoga bi isti raspored bitova predstavljao potpuno različite vrijednosti u jednom odnosno drugom slučaju.

2.2.1 Cjelobrojni tipovi podataka

Temeljni cjelobrojni tip podataka u jeziku dan je ključnom riječi `int`. Ključne riječi (pridjevi) `long`, `short` i `unsigned` omogućuju varijacije. Primjeri takvih varijacija su `short`

Slika 2.1: Predstavljanje (a) realnih i (b) cjelobrojnih vrijednosti u računalu.

`int` i `unsigned long int`. Ključna riječ `int` je pri varijacijama podrazumijevana pa ju je moguće izostaviti.

U većini slučajeva tip podatka `int` je dovoljan za rukovanje cjelobrojnim vrijednostima u programu. Standard jezika nalaže kako je raspon vrijednosti koji mora barem podržavati od −32768 do 32767 što odgovara 16-bitnom zapisu, gdje se jedan bit koristi za predznak. Međutim, duljina zapisa ili, jednostavno, veličina ovog tipa ovisi o računalnoj platformi i prevoditelju pa je za tip `int` uobičajen 32-bitni zapis čime je podržan raspon vrijednosti od −2147483647 do 2147483647. U određenim slučajevima tip `int` može biti nedostatan ako se radi o 16-bitnom zapisu podatka. U takvim slučajevima, rješenje može predstavljati ključna riječ `long`, odnosno tip podatka `long int` ili kraće `long`. Prema standardu jezika, tip podatka `long` mora podržavati raspon vrijednosti od −2147483647 do 2147483647. S druge strane, tip podatka `short int` ili samo `short` mora prema standardu podržavatiti najmanje raspon od −32768 do 32767 (isti zahtjev kao na tip `int`) što najčešće i jest slučaj pa je u pravilu manji od tipa `int`.

Kao što se može lako zaključiti iz prethodnog, svi navedeni tipovi podataka su s predznakom, što znači kako obuhvaćaju pozitivne i negativne cijele brojeve te nulu. Stoga, ključna riječ `signed` u tim slučajevima služi kako bi se to eksplicitno naznačilo. Tako primjerice, `short int`, `short`, `signed short int` i `signed short` označavaju isti tip podatka. S druge strane, dodatkom ključne riječi `unsigned` dobivaju se cjelobrojni tipovi bez predznaka. Naime, bit za predznak se također koristi za predstavljanje vrijednosti/-broja. To znači kako `unsigned` varijacijama nije moguće predstaviti negativne vrijednosti, odnosno podržani raspon vrijednosti za pojedini tip podatka se pomiče u nenegativno područje (primjerice, s [−32768 , 32767] pomiče se na [0, 65535]).

U izlistanju 2.2 su dani primjeri deklaracije i inicijalizacije varijabli cjelobrojnih tipova podataka. Komentarima su prikazani i kraći zapisi korištenih tipova podataka.

```
int broj;              //deklaracija - isto kao: signed int broj;
long int a = 160872;   //inicijalizacija - isto kao: long a = 16008672;
unsigned int np, mp;   //deklaracija - isto kao: unsigned np, mp;
short int m = -1245;   //inicijalizacija - isto kao: short m = -1245;
unsigned short int h; //deklaracija - isto kao: unsigned short h;
unsigned long int l;   //deklaracija - isto kao: unsigned long l;
```

Izlistanje 2.2: Primjeri deklaracije i inicijalizacije varijabli cjelobrojnih tipova podataka

Cjelobrojne konstante

Konstante u tekstu programa, kao primjerice `-128`, `34` i `9011`, su cjelobrojne i tipa podatka `int`. Ukoliko tip `int` nije dostatan za pohranu neke konstante pokušava se tip podatka `long`, a ukoliko nije ni on, onda se pokušava s tipom podatka `unsigned long`. Treba spomenuti kako je moguće dobiti pohranu cjelobrojnih konstanti kao tipa `long` umjesto `int` (pod pretpostavkom da je vrijednost moguće pohraniti u tip `int`). To se postiže dodavanje sufiksa `l` ili `L` poslije vrijednosti pa primjerice, konstanta `34` je tipa `int`, dok

je konstanta `34L` (ili `34l`) tipa `long`. Isto tako je moguće dodati i `u` ili `U` prije navedenog sufiksa kako bi se konstanta tretirala kao tipa `unsigned long` (primjerice, konstanta `45UL` je tipa `unsigned long`).

Programski jezik C pruža mogućnost zapisa cjelobrojnih konstanti u oktalnom i heksadecimalnom brojevnom sustavu. To se postiže uporabom prefiksa `0` (nula) za oktalni zapis, te prefiksa `0x` ili `0X` za heksadecimalni zapis. Primjerice, konstante `34`, `042` i `0x22` (ili `0X22`) predstavljaju istu vrijednost, ali u dekadskom, oktalnom i heksadecimalnom zapisu, redom. Kao i pri dekadskom zapisu, moguće je koristiti sufiks `l` ili `L` i uz dodatak `u` ili `U`.

2.2.2 Realni tipovi podataka

Cjelobrojni tipovi podataka često nisu dovoljni ukoliko se radi o nekakvim izračunima, jer rezultati tih izračuna mogu biti realne vrijednosti. Prema tome, potrebni su realni tipovi podataka. U jeziku postoje tri takva tipa, a to su `float`, `double` i `long double`.

Osnovni realni tip u jeziku je dan ključnom riječi `float`. Uobičajeno se radi o 32-bitnom tipu podatka. Prema standardu jezika mora najmanje podržavati raspon vrijednosti od 10^{-37} do 10^{37} uz mogućnost predstavljanja barem šest značajnih znamenki (jednostavno rečeno, mora moći prikazati barem šest znamenki ispravno). Ukoliko je potrebna veća preciznost i eventualno veći raspon podržanih vrijednosti, nudi se tip `double`. Za njega se uobičajeno koristi 64-bitni zapis, a prema standardu jezika mora podržavati najmanje isti raspon kao tip podatka `float`, ali uz mogućnost predstavljanja najmanje 10 značajnih znamenki (najčešće se radi o 15 značajnih znamenki i o rasponu od 10^{-307} do 10^{308}, ali to ovisi o implementaciji prevoditelja). Nadalje, ukoliko ni tip podatka `double` ne zadovoljava zahtjeve (iznimno malo vjerojatno), potencijalno se nudi tip `long double` koji je prema standardu jezika barem kao tip `double`, ali može biti precizniji i podržavati veći raspon vrijednosti (opet, ovisi o implementaciji prevoditelja). U izlistanju 2.3 slijede primjeri deklaracije i inicijalizacije varijabli realnih tipova podataka.

```
float broj;         //deklaracija
double a = 16.02;   //inicijalizacija
double np, mp;      //deklaracija
float m = -1.245E3; //inicijalizacija
long double h;      //deklaracija
```

Izlistanje 2.3: Primjeri deklaracije i inicijalizacije varijabli realnih tipova podataka

Realne konstante

Konstante u tekstu programa kao primjerice, `7.5`, `-1024.125` i `101.0` su realne i tipa podatka `double`. Važno je primijetiti uporabu točke umjesto zareza za odvajanje cijelog i decimalnog dijela broja. Kao što je spomenuto, realne konstante se tretiraju kao tipa `double`, međutim, moguće je to zaobići. Naime, uporabom sufiksa `f` ili `F` te sufiksa `l` ili `L` prevoditelj tretira konstantu kao tipa `float` odnosno tipa `long double`. Tako je primjerice, konstanta `7.5F` (ili `7.5f`) tipa podatka `float`, a konstanta `7.5L` (ili `7.5l`) tipa podatka `long double`.

Osim na navedeni način zapisa, realne konstante je moguće zapisati i uporabom znanstvene notacije (zapis pomoću potencije broja 10 kao primjerice, $123.456 = 1.23456 \cdot 10^2$). To se postiže sufiksom `e` ili `E` nakon kojeg slijedi potencija. Primjerice, konstantu `101.0`

može se zapisati kao `1.01E2` (ili `1.01e2`), a konstantu `0.00012` kao `1.2E-4` (ili `1.2e-4`). Kao i pri ranije navedenom zapisu, moguće je koristiti sufiks `f` ili `F` te sufiks `l` ili `L` (oba dolaze nakon eksponenta, primjerice, `1.2E-4F`) kako bi se konstanta tretirala kao tipa `float`, odnosno kao tipa `long double`.

2.2.3 Korištenje znakova

Programski jezik C omogućuje rukovanje znakovima (engl. *characters*) kao što su primjerice slova, znamenke i znakovi interpunkcije. Za to služi tip podatka `char` koji spada u kategoriju cjelobrojnih tipova. Naime, interno se koristi mapiranje cjelobrojnih vrijednosti i znakova. Postoji više različitih tablica za to mapiranje, a često je korištena ASCII[2]. Tablicom je svaki znak povezan s odgovarajućom cjelobrojnom vrijednosti. Tako primjerice, broj 43 predstavlja znak +, broj 64 znak @, broj 65 slovo A, a broj 98 slovo b.

Tip podatka `char` u pravilu koristi 8-bitni zapis (jezik jamči da je dovoljno velik za tablicu mapiranja koju koristi sustav). Može biti sa ili bez predznaka što ovisi o implementaciji prevoditelja. Stoga, ukoliko se želi osigurati da je na različitim sustavima s predznakom, može se koristiti ključna riječ `signed` (`signed char`). Suprotno, ukoliko se želi osigurati da je bez predznaka, može se koristiti ključna riječ `unsigned` (`unsigned char`). Prema tome i uz pretpostavku 8-bitne veličine, tip `signed char` podržava raspon vrijednosti od −128 do 127, a tip `unsigned char` raspon vrijednosti od 0 do 255 pa mogu biti korišteni i za rukovanje malim cjelobrojnim vrijednostima u programu.

Nije potrebno parktički ništa znati o ASCII tablici (ili nekoj drugoj tablici koja se koristi) kako bi se koristili znakovi u tekstu programa. Tako je svaki znak između jednostrukih navodnika znakovna konstanta. Primjerice, `'a'`, `'!'`, `'P'` i `'*'` su znakovne konstante koje zapravo predstavljaju cjelobrojne vrijednosti 97, 33, 80 i 42, redom. Jednostruki navodnici su obavezni, jer se u suprotnom smatra da znak predstavlja ime varijable ili sličnog, točinije, da predstavlja identifikator[3] (engl. *identifier*).

U izlistanju 2.4 su dani primjeri deklaracije i inicijalizacije varijabli tipa podatka `char`. Nije preporučeno koristiti ASCII vrijednosti znakova umjesto znakovnih konstanti iz očiglednog razloga – nerazumljivost.

```
char c;          //deklaracija
char znak = 'a'; //inicijalizacija - isto kao: char znak = 97;
unsigned char;   //deklaracija
signed char;     //deklaracija
```

Izlistanje 2.4: Primjeri deklaracije i inicijalizacije varijabli tipa podatka `char`

2.3 Operatori = i `sizeof`

Jedan od načina davanja vrijednosti varijablama je korištenjem operatora pridruživanja (engl. *assignment operator*) =. On ne predstavlja jednakost, nego služi da dodjelu/pridruživanje vrijednosti. Kao što je navedeno ranije, ovaj operator se koristiti i pri inicijalizaciji

[2]ASCII je akronim za *American Standard Code for Information Interchange*. Osnovna tablica obuhvaća cjelobrojne vrijednosti od 0 do 127. Pregled tablice moguće je naći na pozvenici http://www.asciitable.com/.

[3]Ugrubo, identifikator je ime "nečega" kao što je varijabla ili funkcija.

varijabli. Jednostavno rečeno, vrijednost s desne strane (desni operand) pridružuje se varijabli koja je s lijeve strane operatora (lijevi operand). Asocijativnost operatora pridruživanja je s desna na lijevo. Stoga, važno je napomenuti kako obratno navođenje, odnosno navođenje varijable s desne i konstantne vrijednosti s lijeve strane operatora, nije valjano. Prema tome, naredba `broj = 127;` je valjana, dok naredba `127 = broj;` nije valjana. Desni operand ne mora biti konstanta, nego može biti primjerice varijabla koja ima vrijednost, poziv funkcije koja vraća vrijednost ili neki izraz.

Nadalje, operator `sizeof` daje veličinu operanda (količinu memorije koju zauzima njegova pohrana) desno od njega u bajtima. Operand može biti podatkovni objekt kao primjerice varijabla (odnosno njeno ime) ili tip podatka. Ukoliko je operand tip podatka, nužno ga je navesti u zagradama. Treba spomenuti kako je veličina tipa podatka `char`, odnosno podatkovnog objekta tog tipa uvijek 1 bajt. Naime, jezik definira jedan bajt kao veličinu tipa `char`.

Primjer teksta programa dan u izlistanju 2.5 ilustrira uporabu navedena dva operatora. Sažeto obrazloženje dano je komentarima.

```
#include <stdio.h>

int main(void)
{
  unsigned long a, b; //deklaracija varijabli a i b
  double r = 0.1;     //inicijalizacija varijable r
  double t;           //deklaracija varijable t

  t = 172.25;         //pridruzivanje konstantne vrijednosti varijabli t
  a = sizeof(double); //pridruzivanje rezultata operatora sizeof varijabli a
                      //isti rezultat bi se dobio sa sizeof(r) s obzirom da je r tipa double

  b = sizeof a; //pridruzivanje rezultata operatora sizeof varijabli b
  r = t;        //pridruzivanje vrijednosti varijable t varijabli r

  return 0;
}
```

Izlistanje 2.5: Primjeri uporabe operatora pridruživanja = i operatora `sizeof` u programu

2.4 Korištenje funkcija `printf()` i `scanf()` pri različitim tipovima podataka

Ispis na izlaz i čitanje s ulaza može se u programskom jeziku C ostvariti na različite načine, odnosno pomoću različitih funkcija ovisno o potrebama i zahtjevima. Izlaz i ulaz ne moraju nužno biti samo ekran i tipkovnica nego mogu biti primjerice datoteke ili neki drugi uređaj kao printer. Funkcije za rukovanje ulazom odnosno izlazom pruža standardna biblioteka. Ulazno-izlazne funkcije su opisane u zaglavnoj datoteci `stdio.h` i stoga programi koji se koriste tim funkcijama trebaju uključiti navedenu zaglavnu datoteku:

```
#include <stdio.h>
```

Dvije svestrane i često korištene funkcije za rukovanje standardnim izlazom i ulazom su funkcije `printf()` i `scanf()`, redom. Standardni izlaz je u pravilu ekran, dok je standardni ulaz u pravilu tipkovnica. To je omogućuje temeljnu interakciju korisnika s programom. Navedene funkcije su svestrane u smislu podrške za različite tipove

podataka te mogućnostima oblikovanja ispisa na izlaz i interpretiranja ulaza. Način korištenja jedne i druge funkcije je sličan te obje zahtijevaju string formata i eventualno druge argumente.

Neformalno i općenito, funkciju `printf()` koristi se na način:

```
printf(string_formata, stavka1, stavka2, ...);
```

String `string_formata` je slijed znakova omeđen dvostrukim navodnicima koji nalaže kako će stavke (`stavka1, stavka2, ...`) biti ispisane. Za svaku stavku je nužno navesti oznaku pretvorbe (engl. *conversion specifier*) koja nalaže kako će vrijednost koju predstavlja pojedina stavka biti prikazana. Neke od dostupnih i često korištenih oznaka pretvorbi dane su tablicom 2.2 (a). Može se primijetiti kao za obje kategorije tipova podataka (cjelobrojne i realne) postoje odgovarajuće oznake pretvorbe. Za moguće varijacije tipova koriste se dopune oznaka pretvorbe (engl. *conversion specifier modifiers*), a neke od njih su dane tablicom 2.2 (b). Važno je uskladiti tip podatka svake stavke i oznake pretvorbe kako bi se dobio željeni/očekivani ispis. Podsjetnika radi, string formata može biti i jednostavna poruka, a stavke ne moraju biti varijable, nego mogu biti primjerice konstante i izrazi. Primjena funkcije `printf()` ilustrirana je tekstom programa u izlistanju 2.6.

```
#include <stdio.h>

int main(void)
{
  int a = -59;
  unsigned long b = 1029;
  float c = 45.123F;
  double d = 1.234E-2;

  printf("Vrijednosti varijabli:\n");
  printf("a = %d, b = %lu", a, b);
  printf("c = %f, d = %f", c, d);
  //moze se ispisati i jednom pozivom funkcije printf():
  //printf("a = %d, b = %lu, c = %f, d = %f", a, b, c, d);

  printf("%d %f %c", 124, 10.58, 'R');

  return 0;
}
```

Izlistanje 2.6: Primjeri uporabe funkcije `printf()`

Kao što je spomenuto, korištenje funkcije `scanf()` je slično korištenju funkcije `printf()`. Neformalno, funkciju `scanf()` se koristi na način:

```
scanf(string_formata, &varijabla1, &varijabla2, ...);
```

String `string_formata` nalaže kako će unos s ulaza biti interpretiran. Unos s tipkovnice je samo slijed znakova pa je bitna njegova interpretacija. I u ovom slučaju se koriste oznake pretvorbe i njihove dopune (vidi tablicu 2.2). Važno je zapamtiti kako ispred imena varijabli ide operator `&` te je važno uskladiti oznake pretvorbe s tipom podatka svake varijable. Primjena funkcije `scanf()` ilustrirana je tekstom programa u izlistanju 2.7.

2.5 Još malo o konstantama

Uporaba konstanti u različitim programima je praktički neizbježna. Štoviše, uporaba iste konstante na više mjesta u tekstu programa je uobičajena. Nedostatak korištenja zapisa

```c
#include <stdio.h>

int main(void)
{
  int a;
  unsigned long b;
  float c;
  double d;

  scanf("%d %lu", &a, &b);
  scanf("%f %lf", &c, &d);
  //ucitavanje se moze ostvariti i jednim pozivom funkcije scanf():
  //scanf("%d %lu %f %lf", &a, &b, &c, &d);

  return 0;
}
```

Izlistanje 2.7: Primjeri uporabe funkcije `scanf()`

Tablica 2.2: Neke (a) oznake pretvorbe i (b) njihove dopune za funkcije `printf()` i `scanf()`

(a)

Oznaka pretvorbe	opis
`%d` ili `%i`	cijeli broj s predznakom
`%u`	cijeli broj bez predznaka
`%o`	cijeli broj bez predznaka u oktalnom brojevnom sustavu
`%x` ili `%X`	cijeli broj bez predznaka u heksadecimalnom brojevnom sustavu
`%f`	realni broj
`%e` ili `%E`	realni broj u znanstvenoj notaciji
`%c`	jedan znak

(b)

Dopuna	opis
`h`	za cjelobrojnu pretvorbu – naznačavanje tipa `short int` ili `unsigned short int`
`hh`	za cjelobrojnu pretvorbu – naznačavanje tipa `signed char` ili `unsigned char`
`l`[1]	za cjelobrojnu pretvorbu – naznačavanje tipa `long int` ili `unsigned long int`
`L`	za realnu pretvorbu – naznačavanje tipa `long double`

[1] Pri funkciji `scanf()` dodatno služi i za realnu pretvorbu za naznačavanje tipa `double`.

vrijednosti je što nekome tko čita tekst programa sama vrijednost ne mora ništa značiti. Također, ako je korištena na više mjesta u tekstu programa, promjena vrijednosti zahtjeva ručne izmjene na svim tim mjestima.

Uvođenje imena za konstante moglo bi riješiti navedene nedostatke. U programskom jeziku C postoje dva načina na koje je moguće to ostvariti, odnosno uvesti simboličke konstante (engl. *symbolic constants*). Jedan od mogućih načina je korištenje pretprocesora kao:

```c
#define simbolicko_ime vrijednost
```

Ovakve konstante se još nazivaju manifest konstantama. Treba primijetiti kako ne ide točka-zarez nakon vrijednosti s obzirom da se radi o pretprocesorskoj direktivi, a ne o naredbi jezika C. Isto tako ne koristi se operator pridruživanja. Također, s obzirom da se radi o pretprocesorskoj naredbi sve pojave `simbolicko_ime` u tekstu programa bit će zamijenjene sa `vrijednost` prije prevođenja.

Drugi način je korištenje ključne riječi `const` kao pridjeva/kvalifikatora tipu podatka

(engl. *type qualifier*). Općenito, to se postiže na način:

```
const tip_podatka simbolicko_ime = vrijednost;
```

Navedeno odgovara inicijalizaciji varijable uz dodatak ključne riječi `const` na početku. Ukoliko se izostavi pridruživanje vrijednosti pri deklaraciji, naknadno to neće biti moguće kao što nije naknadno moguće promijeniti vrijednost.

Imenovanje simboličkih konstanti podliježe istim pravilima kao imenovanje varijabli. Najčešće, odnosno tradicionalno se za imena koriste samo velika slova kako bi se napravila razlika u odnosu na varijable. U izlistanju 2.8 je dan primjer uvođenja simboličkih konstanti.

```
#include <stdio.h>

#define PI 3.1415 //direktiva
#define G  9.81   //direktiva

int main(void)
{
  const float E = 2.718;       //deklaracija konstante
  const unsigned int PDV = 25; //deklaracija konstante

  return 0;
}
```

Izlistanje 2.8: Primjeri uvođenja simboličkih konstanti u programu

2.6 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je potrebno koristiti varijable različitih tipova podataka te ih ispisati na ekran ili im učitati vrijednost s ulaza, odnosno tipkovnice. Kada tipovi podataka nisu eksplicitno navedeni, proizvoljno su odabrani.

Primjer 1. *Inicijalizirati nekoliko varijabli različitih tipova podataka te im ispisati vrijednosti na ekran.*

```
#include <stdio.h>

int main(void)
{
  int i = -5;
  unsigned int u = 501;
  float f = .525;  //isto kao: float f = 0.525;
  double d = 1E-6; //isto kao: double d = 1e-6;
  long int li = -123456;
  unsigned short int us = 771;

  printf("%d\n", i);
  printf("%u\n", u);
  printf("%f\n", f);
  printf("%f\n", d);
  printf("%ld\n", li);
  printf("%hu\n", us);

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Kao što je naznačeno prvim komentarom, ako je cijeli dio realne konstante 0 (nula), nije ju obavezno pisati. U svim pozivima je korišten posebni znak `\n` nakon kojeg se pokaznik pomiče na početak novog retka. Taj znak je zapravo znakovna konstanta koja prema ASCII tablici ima dekadsku vrijednost 10.

Primjer 2. *Ispisati na ekran veličine nekoliko različitih tipova podataka.*

```
#include <stdio.h>

int main(void)
{
  printf("Tip int zauzima %lu bajta.\n", sizeof(int));
  printf("Tip long int zauzima %lu bajta.\n", sizeof(long int));
  printf("Tip short int zauzima %lu bajta.\n", sizeof(short int));
  printf("Tip float zauzima %lu bajta.\n", sizeof(float));
  printf("Tip double zauzima %lu bajta.\n", sizeof(double));
  printf("Tip long double zauzima %lu bajta.\n", sizeof(long double));
  printf("Tip char zauzima %lu bajta.\n", sizeof(char));

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Ukoliko program ne ispisuje smislene vrijednosti pokušati zamijeniti `%lu` sa `%u` ili ukoliko korišteni prevoditelj podržava standard jezika C99/C11 sa `%zd`. Umjesto `short int` i `long int` može pisati samo `short` i `long`, redom.

Primjer 3. *Deklarirati jednu varijablu tipa `int`, jednu tipa `double` i jednu tipa `short`. Potom omogućiti korisniku unos vrijednosti za svaku od njih.*

```
#include <stdio.h>

int main(void)
{
  int a;
  double b;
  short d;

  scanf("%d", &a);   //
  scanf("%lf", &b);  // isto kao: scanf("%d %lf %hd", &a, &b, &c);
  scanf("%hd", &c);  //
```

```
  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Kao što je napomenuto komentarom, korištena tri poziva funkcije `scanf()` može se zamijeniti jednim pozivom. Važno je ne zaboraviti operator & ispred imena varijabli u pozivu (ili pozivima) funkcije `scanf()` kako bi znala gdje se varijable nalaze u memoriji i time im mogla dodijeliti vrijednost.

Primjer 4. *Omogućiti korisniku jednog prirodnog broja. Ispisati na ekran unesenu vrijednost u oktalnom, dekadskom i heksadecimalnom brojevnom sustavu.*

```
#include <stdio.h>

int main(void)
{
  unsigned broj; //isto kao: unsigned int broj;

  printf("Unesite prirodan broj: ");
  scanf("%u", &broj);

  printf("OKT: %o, DEK: %u, HEKS: %x\n", broj, broj, broj);

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: S obzirom da se očekuje unos prirodnog broja odabran je cjelobrojni tip podatka bez predznaka.

Primjer 5. *Omogućiti korisniku unos jednog znaka. Ispisati ga potom na ekran te njegovu ASCII vrijednost u oktalnom, dekadskom i heksadecimalnom brojevnom sustavu.*

```
#include <stdio.h>

int main(void)
{
  char znak;

  printf("Unesite jedan znak: ");
  scanf("%c", &znak);

  printf("Unijeli ste znak %c koji ima ASCII vrijednost: \
%o (OKT), %d (DEK), %x (HEKS)\n", znak, znak, znak, znak);

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Kako je poruka u drugom pozivu funkcije `printf()` dugačka, razlomljena je u dva retka što je postignuto sa znakom \ koji neće biti ispisan, a označava da se string (slijed znakova omeđen dvostrukim navodnicima) nastavlja u novom retku teksta programa. Ostatak poruke u novom retku mora započeti od samog početka, odnosno ne treba biti praznina ispred. Ukoliko postoje praznine, one će biti ispisane na ekran. Važno je napomenuti kako se tip `char` pri predaji funkciji `printf()` pretvara u tip `int` pa je stoga moguće koristiti oznake pretvorbe `%o`, `%d` i `%x` umjesto `%hho`, `%hhd` i `%hhx`, redom. Uporaba jednih i drugih oznaka pretvorbe je valjana.

Primjer 6. *Omogućiti korisniku unos dva realna broja. Ispisati ih poravnate udesno uz preciznost od 4 te u znanstvenoj notaciji.*

```
#include <stdio.h>

int main(void)
{
  double a, b;

  scanf("%lf %lf", &a, &b);

  //poravnanje udesno sa sirinom polja od 20 i preciznosti od 4
  printf("%20.4f\n%20.4f\n\n", a, b);
  //ispis u znanstvenoj notaciji bez dodatnog formatiranja
  printf("%E\n%E", a, b);

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Uz pozitivnu širinu polja poravnanje je udesno, dok je uz negativnu širinu polja poravnanje uljevo. Ukoliko decimalni dio broja ima više znamenki od zadane preciznosti, dolazi do zaokruživanja, odnosno do dodavanja nula ukoliko ima manje znamenki. Navedene je moguće koristiti i pri ispisu u znanstvenoj notaciji.

POGLAVLJE 3

Operatori i izrazi

Podatke s kojima program radi najčešće je potrebno mijenjati, uspoređivati i koristiti u izračunima. Programski jezik C ima mnoštvo operatora (oko 40 operatora, a za neke se čak koriste iste oznake – vidi tablicu 3.9) koji, između ostalog, omogućuju obradu i rukovanje podacima. U tom pogledu, vrlo često su korišteni aritmetički, logički i relacijski operatori (dostupni uglavnom u svim programskim jezicima). Operatori su temelj za izgradnju različitih izraza (jednostavno rečeno, formule koje nalažu kako izračunati vrijednost) od kojih svaki ima vrijednost. Izrazi vrlo jednostavno postaju naredbe.

3.1 Operatori

Operatori rade nad operandima kako bi dali vrijednost. Ukoliko operator zahtijeva dva operanda onda se naziva binarnim operatorom, a ukoliko zahtijeva jedan operand onda ga se naziva unarnim (u jeziku postoji jedan ternarni operator što znači da zahtijeva tri operanda). Kao što je ranije navedeno, programski jezik C ima veliki broj operatora, a neki vrlo često korišteni prikazani su i sažeto opisani u nastavku.

3.1.1 Aritmetički operatori

Aritmetički operatori[1] (engl. *arithmetic operators*) dostupni u jeziku su dani tablicom 3.1. Navedeni operatori su jednostavni za korištenje i uglavnom se ponašaju očekivano. Operatori `*`, `/` i `%` imaju jednako prvenstvo i veće od operatora `+` i `-`. Uz to su lijevo asocijativni. Naravno, moguće je koristiti (okrugle) zagrade kako bi se promijenio redoslijed primjene operatora u izrazu. Važno je napomenuti kako modulo operator (`%`) zahtijeva cjelobrojne operande, odnosno kako ne radi s realnim operandima. To ima smisla s obzirom da je rezultat primjene operatora ostatak pri cjelobrojnom dijeljenju. Osim toga, operator dijeljenja (`/`) može dati neočekivane rezultate. Naime, ako su oba operanda cjelobrojnog tipa podatka, rezultat je isto cjelobrojan, odnosno odbacuje se decimalni dio. Pri korištenju oba spomenuta operatora, desni operand ne bi trebao biti nula, jer je u tom slučaju ponašanje programa nedefinirano.

[1] Treba primijetiti kako programski jezik C nema operator potenciranja kao neki drugi programski jezici. Međutim, standardna biblioteka nudi funkciju za izračun potencija. Funkcija `pow()` računa i vraća rezultat a^b, gdje se a i b predaju funkciji kao argumenti, redom. Navedena funkcija je opisana u zaglavnoj datoteci `math.h` u kojoj su opisane i mnoge druge matematičke funkcije.

Tablica 3.1: Aritmetički operatori

oznaka	opis
+	operator zbrajanja
-	operator oduzimanja
*	operator množenja
/	operator dijeljenja
%	modulo operator

Operatori prikazani tablicom 3.1 su binarni, a postoje još unarni operatori plus (+) i minus (-), odnosno predznaci. Unarni operator plus ne čini ništa (može se primjerice koristiti kako bi se eksplicitno ukazalo da je konstanta pozitivna), dok unarni operator minus mijenja predznak operanda. Oba operatora imaju prednost ili prvenstvo primjene u odnosu na ranije navedene binarne operatore i desno su asocijativni. Izlistanjem 3.1 prikazani su primjeri uporabe aritmetičkih operatora.

```
a = 2 + 2;      //pridruzuje varijabli vrijednost 4
r = 1.0 / 10.0; //pridruzuje varijabli vrijednost 0.1
z = b % 10;     //pridruzuje varijabli ostatak pri cjelobrojnom dijeljenju s 10
a = -a;         //mijenja predznak vrijednosti varijable
z = z + 10;     //dodaje 10 na trenutnu vrijednost varijable
y = x / 2 - x / 3 + 1.0 / 4 * x;
y = x * x - 2 * x * z + z * z;
```

Izlistanje 3.1: Primjeri korištenja aritmetičkih operatora

3.1.2 Relacijski operatori

Usporedbu dviju vrijednosti obavlja se relacijskim operatorima (engl. *relational operators*) koji su prikazani tablicom 3.2. Ti operatori su binarni i lijevo asocijativni. Relacijski operatori tvore relacijske izraze (engl. *relational expressions*) koji mogu imati samo jednu od dvije vrijednosti – istina ili neistina. Drugačije rečeno, relacijskim izrazima se provjerava istinitost zadanog odnosa operanada. Istina (logička istina) je predstavljena s vrijednosti jedan (1), dok je neistina predstavljena s vrijednosti nula (0). Prema tome, ukoliko je relacijski izraz istinit, on ima vrijednost 1, a u suprotnom ima vrijednost 0. Međutim, treba spomenuti kako u programskom jeziku C sve vrijednosti različite od nula (0) predstavljaju istinu, dok neistinu predstavlja samo vrijednost nula (0). Ipak, relacijski izrazi uvijek imaju vrijednost 1 (istina) ili 0 (neistina).

Relacijski operatori imaju manje prvenstvo u odnosu na aritmetičke operatore. Uz to, operatori <, >, <= i >= imaju jednako i veće prvenstvo naspram operatora == i != (oba imaju jednako prvenstvo). Operandi relacijskih operatora mogu biti cjelobrojnih (uključujući i znakove) i realnih tipova podataka. Međutim, treba biti oprezan kada su operandi realnih tipova podataka zbog mogućih gubitaka pri zaokruživanju.

3.1.3 Logički operatori

Logički operatori (engl. *logical operators*) koji su na raspolaganju u jeziku dani su tablicom 3.3. Logički operatori se vrlo često koriste za kombiniranje dva ili više relacijskih

Tablica 3.2: Relacijski operatori

oznaka	opis
<	operator manje od
>	operator veće od
<=	operator manje od ili jednako
>=	operator veće od ili jednako
==	operator jednakosti
!=	operator nejednakosti

Tablica 3.3: Logički operatori

oznaka	opis
&&	logički I operator
||	logički ILI operator
!	logička negacija

izraza. S obzirom da logički operatori I i ILI imaju manju razinu prvenstva u odnosu na relacijske operatore, nije nužno koristiti zagrade. Negacija ima vrlo visoku razinu prvenstva – višu od aritmetičkih operatora, a logički operator I ima veće prvenstvo u odnosu na logički operator ILI.

Logički operatori I i ILI su lijevo asocijativni, dok je logička negacija desno asocijativna. Operandi logičkih operatora mogu biti cjelobrojnih ili realnih tipova podataka (ne moraju imati vrijednost jednaku 1 ili 0). Logički izrazi (engl. *logical expressions*) mogu imati samo dvije vrijednosti – istina (1) ili neistina (0). Njih se uvijek vrednuje s lijeva na desno. Vrednovanje se prekida čim se ustanovi da je logički izraz neistinit u slučaju logičkog operatora I (lijevi operand neistinit) te istinit u slučaju logičkog operatora ILI (lijevi operand istinit).

U izlistanju 3.2 dano je nekoliko primjera korištenja logičkih operatora. Kao što je već spomenuto, vrlo često se koriste za kombiniranje relacijskih izraza.

```
a || b            //izraz je istinit ako je a ili b istinit
a && b            //izraz je istinit ako su a i b istiniti
!a                //izraz je istinit ako je a neistinit i obratno
x >= 5 && x <= 10 //izraz je istinit samo ako je x u intervalu [5,10]
x < 5 || x > 10   //izraz je istinit samo ako je x izvan intervala [5,10]
a + 5 > b - 2 || c * 10 //isto kao: ((a + 5) > (b - 2)) || (c * 10)
a = !6.825; //pridruzuje vrijednost 0 varijabli a, jer je negacija istine (6.825), neistina
```

Izlistanje 3.2: Primjeri korištenja logičkih operatora

3.1.4 Složeni operatori pridruživanja

Programski jezik C posjeduje nekoliko operatora pridruživanja. Složeni operatori pridruživanja (engl. *compound assignment operators*) predstavljaju kombinaciju aritmetičkih operatora i osnovnog operatora pridruživanja (=)[2]. Oni su prikazani tablicom 3.4, a služe

[2]Postoje još složeni operatori pridruživanja koji kombiniraju bitovne operatore (engl. *bitwise operators*) i operator pridruživanja.

Tablica 3.4: Složeni operatori pridruživanja

oznaka	opis
+=	pridruživanje zbroja
-=	pridruživanje razlike
*=	pridruživanje produkta/umnoška
/=	pridruživanje kvocijenta
%=	pridruživanje ostatka pri cjelobrojnom dijeljenju

za ažuriranje vrijednosti operanda. To znači, kako se lijevom operandu pridružuje vrijednost koja je njegova stara vrijednost ažurirana prema vrijednosti desnog operanda. Način ažuriranja ovisi o operatoru uz pridruživanje. U izlistanju 3.3 je dano nekoliko primjera korištenja složenih operatora pridruživanja. Oni u osnovi omogućuju samo sažeti zapis ažuriranja vrijednosti varijabli.

```
a += 1;  //isto kao: a = a + 1;
a -= b;  //isto kao: a = a - b;
a *= 2;  //isto kao: a = a * 2;
a /= 10; //isto kao: a = a / 10;
a %= 2;  //isto kao: a = a % 2;
b += c * 10;    //isto kao: b = b + (c * 10);
b *= 2 * c - 5; //isto kao: b = b * (2 * c - 5);
```

Izlistanje 3.3: Primjeri korištenja složenih operatora pridruživanja

Treba spomenuti kako su navedeni operatori isto kao operator pridruživanja binarni i desno asocijativni. Osim toga, imaju jednako i najmanje prvenstvo primjene (jedino operator zarez[3] ima manje prvenstvo). Važno je podsjetiti kako i kod ovih operatora, kao i kod osnovnog operatora pridruživanja, lijevi operand mora biti varijabla ili općenito, podatkovni objekt čiju je vrijednost moguće promijeniti[4]. Također, valja obratiti pozornost pri uporabi operatora /= i %=, iz istih razloga spomenutih ranije kod opisa aritmetičkih operatora.

3.1.5 Operatori povećanja i umanjivanja

Operator povećanja (engl. *increment operator*) i operator umanjivanja (engl. *decrement operator*) prikazani su tablicom 3.5. Radi se o unarnim operatorima koji postoje u dva oblika – prefiks i sufiks oblici/varijante. Operand za ove operatore mora biti varijabla, odnosno objekt kojem je moguće promijeniti vrijednost. Naime, navedeni operatori povećavaju, odnosno umanjuju vrijednost operanda za jedan.

Lako se može zaključiti kako prefiks oblik ide ispred operanda, dok sufiks oblik ide poslije operanda. Iako oba oblika čine isto, razlikuju se po točnom vremenu kada dolazi od izmijene vrijednosti operanda. Jednostavno rečeno, prefiks oblik se može tumačiti kao:

[3] Zarez (,) nije uvijek operator pa tako primjerice u naredbama deklaracije više varijabli ima ulogu razdjelnika.

[4] Formalno se podatkovni objekti čiju je vrijednost moguće promijeniti nazivaju promjenjivim l-vrijednostima (engl. *modifiable lvalues*). Naziv ukazuje kako se objekt može nalaziti s lijeve strane operatora pridruživanja. Analogijom, postoji i objekti koji se nazivaju r-vrijednostima (engl. *rvalues*), a odnosi se objekte koje je moguće pridružiti promjenjivim l-vrijednostima, ali koji to nisu. Podrazumijeva se, promjenjive l-vrijednosti mogu se nalaziti s obje strane operatora pridruživanja.

Tablica 3.5: Složeni operatori pridruživanja

oznaka	opis
++	povećanje
--	umanjivanje

"povećaj (ili umanji) vrijednost operanda pa ga potom iskoristi", dok se sufiks oblik može tumačiti kao: "iskoristi vrijednost operanda pa ga potom povećaj (ili umanji)".

U izlistanju 3.4 je prikazano nekoliko primjera korištenja operatora povećanja i umanjivanja. Može se primijetiti da u trivijalnom slučaju (izraz koji ima samo operator povećanja ili umanjivanja) nema razlike u prefiks i sufiks oblicima. Međutim, u složenijim izrazima vrlo često postoji razlika pa nije svejedno koji oblik se koristi.

```
a++;     //isto kao: a += 1; ili a = a + 1; ili ++a;
--b;     //isto kao: b -= 1; ili b = b - 1; ili b--;
c = a++; //a ce biti pridruzen c i zatim povecan pa nije isto kao: c = ++a;
c = --b; //b ce biti umanjen i zatim pridruzen c pa nije isto kao: c = b--;
```

Izlistanje 3.4: Primjeri korištenja operatora povećanja i umanjivanja

Oba operatora imaju vrlo visoku razinu prvenstva (samo zagrade imaju veću). Prefiks oblik je desno asocijativan, dok je sufiks oblik lijevo asocijativan. Važno je naglasiti kako treba izbjegavati uporabu spomenutih operatora na varijablama koje se unutar nekog izraza pojavljuju više od jednom, jer redoslijed vrednovanja podizraza nije definiran standardom i prepušten je implementaciji prevoditelja.

3.2 Izrazi i naredbe

Izraz (engl. *expression*) je kombinacija operatora i operanda. Najjednostavniji izraz je samo jedan operand. Izrazi mogu sadržavati manje izraze, odnosno podizraze (engl. *subexpressions*). Ključno je naglasiti kako svaki izraz ima (numeričku) vrijednost. U nastavku je navedeno nekoliko primjera izraza. Iako jezik dopušta ili omogućuje pisanje "čudnih" izraza, to nije nikako preporučljivo (i nije dobra praksa) zbog nečitljivosti i nerazumljivosti istih. Izlistanje 3.5 dani su primjeri izraza.

```
5                          //samo operand
1 + 17                     //ima vrijednost 18
t * 2 - 1 < 0              //istinit (vrijednost 1) samo ako t nije pozitivan
a = 13                     //ima vrijednost 13
a = 10 / 5 + (c = 19) - 8 //ima vrijednost 13, ali nije preporucljivo pisati ovake izraze
```

Izlistanje 3.5: Primjeri izraza

Naredba (engl. *statement*) je potpuna instrukcija računalu. U jeziku su naredbe naznačene s točkom-zarez na kraju. Između ostalog, dodavanjem točke-zarez na kraja izraza čini naredbu. Valja primijetiti da nisu sve naredbe bez točke-zarez izrazi pa primjerice, naredba deklaracije (kao što je deklaracija varijable) bez točke-zarez nije izraz i nema vrijednost. Potpuna instrukcija nije sama po sebi naredba, ali može biti dio naredbe – točka-zarez je nužna kako bi potpuna instrukcija predstavljala naredbu. Treba još spomenuti složene naredbe (engl. *compound statements*) koje su bezimeni/neimenovani blokovi,

odnosno jedna ili više naredbi omeđene vitičastim zagradama. Blok naredbi (ili složena naredba ili blok) je sintaktički jedna naredba. U izlistanju 3.6 dano je nekoliko primjera naredbi.

```
;       //nul-naredba - najjednostavnija moguca naredba
5;      //naredba koja ne cini nista
2 - 5;  //naredba koja ne cini nista
int a;  //naredba deklaracije
a = 13; //naredba pridruzivanja
a = 10 / 5 + (c = 19) - 8; // c = 19 je potpuna instrukcija racunalu
printf("PROG1");           //poziv funkcije
```

Izlistanje 3.6: Primjeri naredbi

3.3 Pretvaranje tipova

U izrazima, odnosno naredbama bi trebalo koristiti samo varijable i konstante koje su istog tipa podatka. Iako je to poželjno, programski jezik C omogućuje i uporabu objekata različitih tipova podataka unutar istog izraza (to nije slučaj sa svim programskim jezicima). Kako bi navedeno radilo, vrše se automatske pretvorbe tipova podataka prema definiranim pravilima. Osnovna pravila su:

- Tip `char` i `short` (`signed` i `unsigned` varijante) unutar izraza se pretvaraju u tip `int` (ili po potrebi u tip `unsigned int`, odnosno kada je tip `short` iste veličine kao tip `int`).
- Pri operacijama (primjena operatora) koje uključuju dva različita tipa, vrijednosti oba operanda se pretvaraju u onaj tip koji je višeg ranga.
- U naredbama pridruživanja, rezultat s desna se pretvara u tip podataka varijable kojoj se pridružuje.
- [5]Kada se tipovi `char` i `short` predaju nekoj funkciji kao argumenti, pretvara ih se u tip `int`. Isto tako, tip `float` se pretvara u tip `double`.

Tipovi su rangirani na sljedeći način, od višeg prema nižem rangu: `long double`, `double`, `float`, `unsigned long`, `long`, `unsigned int` i `int`. Ukoliko dolazi do automatske pretvorbe u tip višeg ranga, to se naziva promocija. Suprotno tome, democija označava pretvaranje u tip nižeg ranga (može se dogoditi kod pridruživanja). Ovdje treba napomenuti da pri cjelobrojnom dijeljenju dolazi do eventualne promocije (primjerice, ako se rezultat pridružuje varijabli realnog tipa) tek nakon izračuna što ne sprječava odbacivanje decimalnog dijela rezultata.

Pretvorbu tipa u drugi željeni tip je moguće i zahtijevati pomoću operatora pretvaranja (engl. *cast operator*). On je označen okruglim zagradama unutar kojih se navodi željeni tip podatka:

```
(tip_podatka) operand
```

[5]Primjerice, zbog ovog pravila se kod uporabe funkcije `printf()` s tipovom podatka `float` i `double` koristi ista oznaka pretvorbe (`%f`). Međutim, pravilo je moguće zaobići prototipovima kod pisanja vlastitih funkcija.

Tablica 3.6: Neke dopune oznaka pretvorbe za funkciju `printf()`

Dopuna	opis
zastavica	zastavice su: `-`, `+`, `#`, `0` i razmak
znamenka/e[1]	najmanja širina polja; šire polje će biti korišteno ako broj ili string kojeg se ispisuje nestane u njega. Polje čine mjesta za ispis znakova.
.znamenka/e	preciznost; za oznake pretvorbe `%f`, `%e` i `%E` predstavlja koliko će znamenki decimalnog dijela broja biti ispisano. Za cjelobrojne oznake pretvorbe predstavlja najmanji broj znamenki koje se ispisuje; koriste se nule sprijeda po potrebi.

[1] Pri funkciji `scanf()` predstavlja najveću širinu polja; čitanje se zaustavlja nakon što je dosegnuto ili prije ako je se pojavi praznina.

Operand može biti konstanta ili varijabla. Navedeni operator ima veće prvenstvo od aritmetičkih operatora i desno je asocijativan. Korištenje ovog operatora je ilustrirano primjerom teksta programa u izlistanju 3.7.

```
#include <stdio.h>

int main(void)
{
  float rez;
  int br = 7;

  rez = br / 2; //cjelobrojno dijeljenje
  printf("Rezultat je: %f\n", rez); //rezultat ima vrijednost 3

  rez = (float) br / 2; //nije vise cjelobrojno dijeljenje
  printf("Rezultat je: %f\n", rez); //rezultat ima vrijednost 3.5

  return 0;
}
```

Izlistanje 3.7: Primjer korištenja operatora pretvaranja u programu

3.4 Još malo o korištenju funkcija `printf()` i `scanf()`

Kao što je već opisano, pri uporabi funkcija `printf()` i `scanf()` koriste se oznake pretvorbe kako bi se naznačio način ispisa stavki na izlaz, odnosno kako bi se naznačila interpretacija unosa s ulaza. Oznake pretvorbe je moguće nadopuniti, a neke dodatne dopune prikazane su tablicom 3.6 (podsjetnika radi, dopune se navode odmah nakon `%` u oznaci pretvorbe). Dopune koje predstavljaju zastavice posebno su izdvojene i prikazane tablicom 3.7.

Nadalje, ASCII tablica sadrži znakove kao primjerice znak za novi redak, znak za tabulator (horizontalni/vertikalni) i slično. Neki od tih znakova su predstavljeni u jeziku posebnim sekvencama[6] (engl. *escape sequences*) koje zapravo čine dva znaka, a prikazane su tablicom 3.8. Kada se navedene sekvence pridružuju varijablama navodi ih se unutar jednostrukih navodnika kao i uobičajene znakove. One kao i drugi uobičajeni znakovi mogu biti (bilo gdje) unutar stringa (slijed znakova omeđen dvosturkim navodnicma). Može se dodatno spomenuti kako je za ispis znaka za postotak (%) funkcijom `printf()` potrebno

[6] Postoji još posebnih sekvenci, ali ne rade sve s ispisom na ekran.

Tablica 3.7: Zastavice za funkciju `printf()`

Zastavica	opis
`-`	lijevo poravnanje; stavka se ispisuje s lijeve strane unutar polja.
`+`	ispisuje predznak stavke (plus (+) ako je pozitivna te minus (-) ako je negativna).
`#`	ispisuje početnu `0` za oznaku pretvorbe `%o` te početno `0x` ili `0X` za oznaku pretvorbe `%x`, odnosno `%X`.
`0`	ispis početnih nula umjesto razmaka; ignorira se ako je korištena zastavica - ili, u slučaju cjelobrojnih pretvorbi, ako je korištena dopuna za preciznost.
razmak	ispisuje predznak samo za negativne vrijednosti, a u slučaju pozitivnih razmak; dodatna uporaba i zastavice + prepisuje razmak.

Tablica 3.8: Neke posebne sekvence

Sekvenca	značenje
`\a`	kratak zvučni signal
`\n`	novi redak
`\t`	horizontalni tabulator
`\\`	obrnuta kosa crta (\)
`\'`	jednostruki navodnik (')
`\"`	dvostruki navodnik (")
`\?`	upitnik (?)

koristiti `%%` (jer oznake pretvorbe počinju s tim znakom), ali je znakovna konstanta dana na očekivan način s `'%'`.

3.5 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je potrebno koristiti različite operatore i njihove kombinacije za rješavanje zadataka. Tipovi podataka, odnosno kategorija tipova nije uvijek navedena te je odabrana prema potrebama zadataka. Korištena su vrlo kratka imena varijabli, često samo jedan znak. Međutim, to nije preporučljivo u slučaju složenijih programa. Dobra je praksa koristiti opisna imena iz kojih se odmah razumije što predstavljaju.

Primjer 1. *Omogućiti korisniku unos dva cijela pozitivna broja a i b. Izračunati izraz* $a^2 + b^{-2} - 1$ *i ispisati njegovu vrijednost na ekran.*

```
#include <stdio.h>

int main(void)
{
  unsigned int a, b;
  float r;

  printf("Unesite dva cijela pozitivna broja:\n");
  scanf("%u %u", &a, &b);

  r = a * a + 1.0f / (b * b) - 1; //isto kao: r = ((a * a) + (1.0f / (b * b))) - 1;

  printf("Rezultat izracuna je: %.4f\n", r); //ispisuje 4 znamenke decimalnog dijela vrijednosti

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Iako su svi operandi cjelobrojni, vrijednost izraza je realna zbog $b^{-2} = \frac{1}{b^2}$. Kvadrati brojeva su jednostavno riješeni množenjem broja sa samim sobom. U slučaju manjih cjelobrojnih potencija nije potrebno koristiti funkciju `pow()`, a brže se i izvršava. Važno je primijetiti realnu konstantu `1.0f` (tipa podatka `float`) pa je stoga cijeli izraz na kraju realne vrijednosti. Nadalje, može se primijetiti uporaba zastavice `.4` s oznakom `%f` čime se postiže ispis samo četiri znamenke decimalnog dijela rezultata.

Primjer 2. *Omogućiti korisniku unos dva realna broja. Ispisati na ekran vrijednosti relacija ta dva broja.*

```
#include <stdio.h>

int main(void)
{
  double a, b;

  printf("Unesite dva realna broja:\n");
  scanf("%lf %lf", &a, &b);

  printf("Izraz %f < %f ima vrijednost %2d\n", a, b, a < b);
  printf("Izraz %f <= %f ima vrijednost %d\n", a, b, a <= b);
  printf("Izraz %f > %f ima vrijednost %2d\n", a, b, a > b);
  printf("Izraz %f >= %f ima vrijednost %d\n", a, b, a >= b);
  printf("Izraz %f == %f ima vrijednost %d\n", a, b, a == b);
  printf("Izraz %f != %f ima vrijednost %d\n", a, b, a != b);

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Može se primijetiti kako su relacijski izrazi predani kao zadnji argument funkciji `printf()`. Izraz će biti prvo vrednovan, a potom će njegova vrijednost biti predana funkciji. Na taj način je izbjegnuto korištenje posebne varijable za spremanje vrijednosti relacijskih izraza. Kod relacijskih operatora < i > korištena je oznaka pretvorbe `%2d` kako bi sve vrijednosti izraza bile ispisane u istom stupcu.

Primjer 3. *Omogućiti korisniku unos jednog realnog broja. Napisati izraz koji provjerava nalazi li se uneseni broj unutar intervala* $[-10, 100]$ *te ispisati njegovu vrijednost na ekran.*

```
#include <stdio.h>

int main(void)
```

```
{
  double broj;
  unsigned rez; //isto kao: unsigned int rez;

  printf("Unesite realan broj: ");
  scanf("%lf", &broj);

  rez = broj >= -10.0 && broj <= 100.0; //isto kao: rez = (broj >= -10.0) && (broj <= 100.0);
                                        //isto kao: rez = !(broj < 10.0 || broj > 100.0);
                                        //nije ispravno: -10.0 <= broj <= 100.0;
  printf("Vrijednost izraza je %u", rez);

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Važno je primijetiti kako je potrebno koristiti dva relacijska izraza kombinirana logičkim operatorom. Izraz `-10.0 <= broj <= 100.0`, iako sintaktički ispravan, nije valjan, jer je jednak `(-10.0 <= broj) <= 100.0`.

Primjer 4. *Omogućiti korisniku unos realnog broja d. Ispisati na ekran 1 ako je* $0 < d \leq 10$*, a u suprotnom ispisati 0.*

```
#include <stdio.h>

int main(void)
{
  double d;
  unsigned r; //isto kao: unsigned int r;
  printf("Unesite realan broj: ");
  scanf("%lf", &d);

  r = d > 0 && d <= 10;
  printf("%u", r);

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Vidi obrazloženje za prethodni zadatak.

Primjer 5. *Omogućiti korisniku unos duljine stranice kvadrata. Izračunati te ispisati na ekran opseg i površinu kvadrata.*

```
#include <stdio.h>

int main(void)
{
  double a;        //duljina stranice kvadrata
  double o_k, p_k; //opseg, odnosno povrsina kvadrata

  printf("Unesite duljinu stranice kvadrata: ");
  scanf("%lf", &a);

  o_k = 4 * a;
  p_k = a * a;
  printf("Opseg kvadrata je %f\nPovrsina kvadrata je %f\n", o_k, p_k);

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Korišten je realni tip podatka za duljinu stranice, jer je općenitije od korištenja cjelobrojnih tipova u ovakvim situacijama.

Primjer 6. *Omogućiti korisniku unos polumjer kruga r i središnji kut α u stupnjevima. Izračunati te ispisati na ekran duljinu kružnog luka i kružni isječak.*

```
#include <stdio.h>

#define PI 3.141592 //manifest konstanta

int main(void)
{
  double r;        //polumjer kruga
  unsigned a;      //sredisnji kut
  double dkl, pki; //duljina kruznog luka, odnosno povrsina kruznog isjecka

  printf("Unesite polumjer kruga i sredisnji kut (u stupnjevima):\n");
  scanf("%lf %u", &r, &a);

  dkl = r * PI * a / 180.0;
  pki = r * r * PI * a / 360.0;
  printf("Duljina kruznog luka je %.2f\nPovrsina kruznog isjecka je %.2f\n", dkl, pki);

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Duljina kružnog luka je $l = \frac{r \cdot \pi \cdot \alpha}{180^\circ}$, a površina kružnog isječka je $p = \frac{r^2 \cdot \pi \cdot \alpha}{360^\circ}$. Dolazi do promocije vrijednosti varijable `a` jer je prvi operand tipa višeg ranga (konstanta 3.141592 je tipa podatka `double`).

Dodatak: Tablica operatora u jeziku C

Standard programskog jezika C propisuje gramatiku jezika iz koje je moguće izvesti tablicu razina prvenstava primjene operatora. Operatori dostupni u jeziku kao i njihove razine prvenstava te asocijativnosti prikazane su tablicom 3.9. Treba napomenuti kako se navedeni operatori odnose na standard jezik C90 te kako su standardi C99 i C11 uveli po jedan dodatni operator.

Tablica 3.9: Prvenstvo i asocijativnost operatora u programskom jeziku C

Razina prvenstva	operator/i	oznaka/e	asocijativnost

1	indeksiranje polja
	poziv funkcije
	član strukture i unije
	povećanje (sufiks)
	umanjivanje (sufiks)
		`[]`
		`()`
		`. ->`
		`++`
		`--`
			L

2	povećanje (prefiks)
	umanjivanje (prefiks)
	adresa
	indirekcija/dereferenciranje
	plus (unarni)
	minus (unarni)
	bitovni komplement
	logička negacija
	veličina
		`++`
		`--`
		`&`
		`*`
		`+`
		`-`
		`~`
		`!`
		`sizeof`
			D

3	pretvaranje	`()`	D
4	množenje, dijeljenje i ostatak pri cjelobrojnom dijeljenju	`* / %`	L
5	zbrajanje i oduzimanje	`+ -`	L
6	bitovno pomicanje	`« »`	L
7	relacije [manje od (ili jednako) i veće od (ili jednako)]	`< <= > >=`	L
8	relacije (jednakost i nejednakost)	`== !=`	L
9	bitovno I	`&`	L
10	bitovno isključivo ILI	`^`	L
11	bitovno ILI	`|`	L
12	logičko I	`&&`	L
13	logičko ILI	`||`	L
14	uvjet	`?:`	D
15	(složeno) pridruživanje	`= *= /= %= += -= «= »= &= ^= |=`	D
16	zarez	`,`	L

POGLAVLJE 4

Upravljanje tijekom izvođenja programa: Petlje

Kod razvoja programske podrške, često se nameće potreba za višestrukim izvođenjem neke radnje koja se može sastojati od jedne ili više naredbi. Ispis niza podataka ili pružanje korisniku unosa više različitih podataka predstavljaju primjere navedene potrebe. Broj ponavljanja ne mora uvijek biti unaprijed poznat. Navedeno zahtijeva odgovarajuće konstrukcije za upravljanje tijekom izvođenja programa. U programskom jeziku C postoje tri konstrukcije ili naredbe koje to omogućuju. Takve naredbe se uobičajeno nazivaju petljama (engl. *loops*).

4.1 Petlja `for`

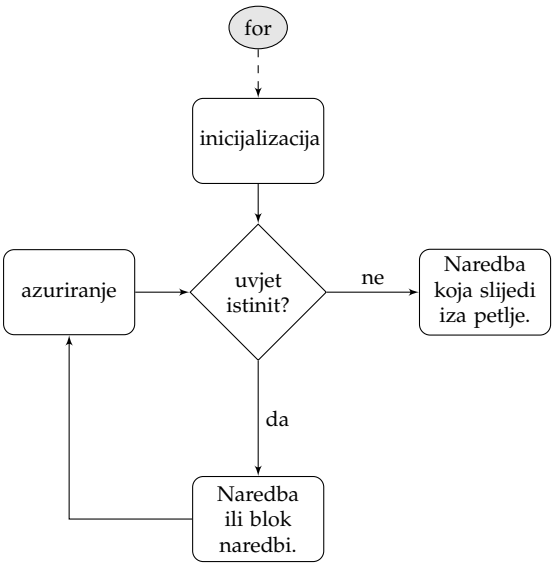
Petlja `for` vrlo često je korištena za postizanje višestrukog izvođenje jedne ili niza naredbi (blok naredbi). Općeniti oblik petlje `for` je prikazan u nastavku, a način rada je ilustriran slikom 4.1.

```
for (inicijalizacija; uvjet; azuriranje)
  naredba
```

Ključna riječ `for`, izrazi[1] `inicijalizacija`, `uvjet` i `azuriranje` unutar zagrada te `naredba` (ili blok naredbi) predstavljaju jednu cjelinu. Kao što se može vidjeti na slici 4.1, prvo se vrednuje izraz `inicijalizacija` (uglavnom predstavlja dodjelu ili pridruživanje nekoj varijabli). Potom, utvrđuje se hoće li se izvršiti naredba ili blok naredbi pod petljom. To se čini vrednovanjem izraza `uvjet` (uglavnom predstavlja relaciju). Naime, ukoliko je njegova vrijednost različita od nule (istina) naredba se izvršava, u suprotnom (vrijednost izaraz jednaka je nula – neistina) nastavlja se izvođenje s prvom naredbom koja slijedi nakon petlje. Po izvršenju naredbi, obavlja se vrednovanje izraza `azuriranje` (često predstavlja promjenu vrijednosti varijable koja se nalazi u izrazima `inicijalizacija` i `uvjet`). Potom, ponavlja se postupak od vrednovanja izraza `uvjet`.

Koraci vrednovanje izraza `uvjet`, izvršavanje naredbe ili bloka naredbi te vrednovanje izraza `azuriranje` predstavljaju jednu iteraciju petlje. Važno je primijetiti kako se izraz `inicijalizacija` vrednuje samo jednom i to na početku. Isto tako, važno je primijetiti

[1] [Potsjetnik] Izrazi se sastoje od operatora i operanada (ono nad čime operatori rade). Svaki izraz ima vrijednost. Najjednostavniji izrazi sastoje se samo od jednog operanda. Primjerice, `7` predstavlja izraz koji ima vrijednost 7. Nadalje, `a = 5 + 4 - 10 / 2` je izraz koji ima vrijednost 4, dok izraz `b < 5` ima vrijednost 1 ukoliko je istinit, odnosno 0 u suprotnom. Svaki izraz se može pretvoriti u naredbu dodavanjem točke-zarez na kraj.



Slika 4.1: Način rada petlje **for**

točke-zarez koje razdvajaju navedene izraze i koje su obavezne. U tom pogledu, točke-zarez su obavezne, ali izrazi `inicijalizacija`, `uvjet` i `azuriranje` nisu te ih je moguće izostaviti (jedan ili više njih[2]).

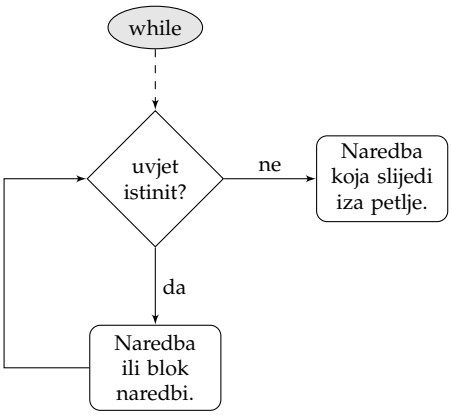
S obzirom na način rada petlje `for`, radi se o petlji s provjerom uvjeta pri ulazu (engl. *entry-condition loop*). To znači kako se na početku svake iteracije provjera uvjet ulaska, odnosno izvršavanja naredbe ili bloka naredbi. U izlistanju 4.1 je prikazano nekoliko jednostavnih primjera uporabe petlje `for`.

```
for (i = 0; i < 5; i++) //jedna naredba u sklopu petlje
  printf("FERIT\n");    //bit ce ispisano pet puta

for (i = 2; i <= 100; i += 2) { //blok naredbi u sklopu petlje
  printf("i = %d\n", i);
  s += i;
}

for (c = 5; c > 0; ) { //izostavljen zadnji izraz
  p *= c;
  printf("%d\t", p);
  --c;
}

for (x = 'a'; x <= 'z'; x++)
  printf("%c\n", x); //ispis slova od a do z

for ( ; ; ) //beskonacna petlja
  printf("Gdje je kraj?\n");
```

Izlistanje 4.1: Primjeri petlje `for`

Prilikom uporabe petlje `for`, važno je osigurati da izraz `uvjet` u jednom trenutku postane jednak nula (neistina) kako bi se petlja prekinula. Ukoliko to nije osigurano, radi se o beskonačnoj petlji. Navedeno vrijedi i za petlje čiji opisi slijede.

[2]Ukoliko je izostavljen izraz `uvjet` smatra se uvjet izvršavanja naredbe ili bloka naredbi istinitim. Stoga je nužno unutar bloka naredbi ugraditi način zaustavljanja, odnosno prekidanja petlje.



Slika 4.2: Način rada petlje **`while`**

4.2 Petlja `while`

Općeniti oblik petlje `while` je prikazan u nastavku, a način rada je ilustriran slikom 4.2.

```
while (uvjet)
  naredba
```

Ključna riječ `while` te izraz `uvjet` unutar zagrada i `naredba` (ili blok naredbi) predstavljaju jednu cjelinu. Kako se radi, kao i u slučaju `for` petlje, o petlji s provjerom uvjeta pri ulazu, prvo se vrednuje izraz unutar zagrada (`uvjet`). Ukoliko je vrijednost izraza različita od nule (istina) izvršava se naredba u sklopu petlje `while`, dok se u suprotnom, odnosno ako je vrijednost izraza jednaka nula (neistina) prelazi na prvu naredbu koja slijedi nakon petlje. Kao i u prethodnom slučaju s petljom `for`, pod petljom `while` može biti jedna naredba ili pak blok naredbi (niz naredbi unutar vitičastih zagrada). Provjera uvjeta i izvršavanje naredbe ili bloka naredbi predstavlja jednu iteraciju petlje. Izlistanjem 4.2 je prikazano nekoliko primjera uporabe petlje `while`.

```
while (a++ < 10) //jedna naredba u sklopu petlje
  printf("ETFOS\n");

while (d != 0) { //blok naredbi u sklopu petlje
  p *= d % 10;
  d /= 10;
}

while (1) //beskonacna petlja
  printf("Gdje je ovdje kraj?\n");
```

Izlistanje 4.2: Primjeri petlje `while`

4.3 Petlja do while

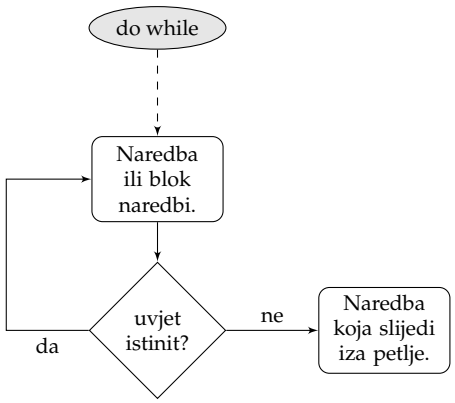
Općeniti oblik ili struktura petlje `do while` prikazana je u nastavku, a način rada je ilustriran slikom 4.3. Za razliku od dvije prethodno opisane petlje, `do while` je petlja s izlaznom provjerom uvjeta (engl. *exit-condition loop*). To znači kako se provjera obavlja na kraju svake iteracije, a ne početku kao što je slučaj s petljama `for` i `while`.

```
do
  naredba
while (uvjet);
```

Ključna riječ `do while`, naredba i izraz `uvjet` unutar zagrada koje slijedi točka-zarez čine jednu cjelinu. U sklopu petlje se može nalaziti jedna ili blok naredbi. Shodno slici 4.3, prvo se izvršava naredba ili blok naredbi i potom se provjera uvjet ponovnog izvođenja ili ponovnog ulaska u petlju. Ukoliko je izraz `uvjet` istinit ponovo se ulazi u petlju, dok se u suprotnom prekida ista te se izvršava prva naredba koja slijedi nakon petlje. Prema navedenom, naredba ili blok naredbi biti će uvijek barem jednom izvršen bez obzira na vrijednost izraza `uvjet`. Na koncu, važno je primijetiti točku-zarez na kraju (iza zagrada poslije `while`) koja je obavezna. U izlistanju 4.3 je prikazano nekoliko primjera uporabe petlje `do while`.

```
do  //jedna naredba u sklopu petlje
  printf("Neka poruka.\n");
while (++c <  20);

do { //blok naredbi u sklopu petlje
  p *= 1.1;
  printf("%f\n", p);
} while (p < 1);

do //beskonacna petlja
  printf("Nema kraja?\n");
while (1);
```

Izlistanje 4.3: Primjeri petlje `do while`



Slika 4.3: Način rada petlje **`do while`**

4.4 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je potrebno koristiti petlje za njihovo rješavanje. Treba napomenuti kako se često vrši ispis poruka na ekran iako nisu eksplicitno zahtijevane u zadacima. To je korišteno kako zbog povećanja jasnoće teksta programa tako i zbog estetskih razloga.

Primjer 1a. *Ispisati na ekran po 20 puta proizvoljnu poruku (svaku u posebnom retku) uporabom petlji* `for`, `while` *i* `do while`, *redom.*

```
#include <stdio.h>

int main(void)
{
  int i; //deklaracija varijable koja predstavlja brojac

  for (i = 0; i < 20; i++)
    printf("Programiranje 1.\n"); //ispis poruke na ekran; \n za prijelaz u novi red

  //nakon zavrsetka for petlje, vrijednost brojaca jednaka je 20 pa ga je nuzno vratiti na nula
  i = 0;
  while (i < 20) {
    printf("Programiranje 1.\n");
    i++; //ili i += 1 ili i = i + 1
  }

  //isti razlog za vracanje brojaca na nula
  i = 0;
  do {
    printf("Programiranje 1.\n");
    i++;
  } while (i < 20);

  return 0;
}
```

Kratko obrazloženje [Primjer 1a]: U svim slučajevima brojač (`i`) je postavljen na nula (`i=0`), a uvećava se (`i++`) po ispisu poruke. Tako, vrijednost brojača odgovara broju ispisanih poruka. Kada se ispiše dvadeseti puta poruka, uvjeti (`i<20`) neće biti zadovoljeni i prekidaju se petlje, odnosno izlazi se iz njih i nastavlja se izvođenje s prvom naredbom koja slijedi.

Primjer 1b. *Ispisati na ekran po 20 puta proizvoljnu poruku (svaku u posebnom retku) uporabom petlji* `for`, `while` *i* `do while`, *redom.*

```
#include <stdio.h>

int main(void)
{
  int i; //deklaracija varijable koja predstavlja brojac

  for (i = 0; i < 20; i++)
    printf("Programiranje 1.\n"); //ispis poruke na ekran; \n za prijelaz u novi red

  //nakon zavrsetka for petlje, vrijednost brojaca jednaka je 20 pa ga je nuzno vratiti na nula
  i = 0;
  while (i++ < 20)
    printf("Programiranje 1.\n");

  //isti razlog za vracanje brojaca na nula
  i = 0;
  do
```

```
    printf("Programiranje 1.\n");
  while (++i < 20);

  return 0;
}
```

Kratko obrazloženje [Primjer 1b]: Isti zadatak kao u Primjeru 1a, ali je važno primijetiti razlike u slučaju petlji `while` i `do while`. Sažetiji zapis je postignut uporabom inkrement (uvećavanja) operatora, postfiks i prefiks varijantama, redom. Navedeno je vrlo često korišteno u takvim ili sličnim situacijama.

Primjer 2. *Ispisati na ekran sve neparne brojeve između 5 i 100.*

```
#include <stdio.h>

int main(void)
{
  int b; //deklaracija varijable koja predstavlja trenutni neparni broj

  for (b = 7; b < 100; b += 2) // 7 je prvi neparni broj kojeg treba ispisati
    printf("%d\t", b); //ispis neparnog broja; razdvojeni tabulatorom zbog \t

  b = 7;
  while (b < 100) {
    printf("%d\t", b);
    b += 2; //ili b = b + 2
  }

  b = 7;
  do {
    printf("%d\t", b);
    b += 2; //ili b = b + 2
  } while (b < 100);

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Ilustracije radi, zadatak je riješen sa sve tri petlje i vrlo je sličan prethodnom. S obzirom da nije eksplicitno navedeno koju petlju (naredbu) treba koristiti izbor je proizvoljan.

Primjer 3. *Omogućiti korisniku unos cijelog broja. Osigurati da bude u intervalu* $[10, 50]$.

```
#include <stdio.h>

int main(void)
{
  int b; //deklaracija varijable koja predstavlja broj

  //viticaste zagrade ({}) nisu obavezne u ovom slucaju, jer je unutar njih samo jedna naredba
  do {
    scanf("%d", &b); //trazenje ulaza od strane korisnika
  } while (b < 10 || b > 50);

  scanf("%d", &b);
  while (b < 10 || b > 50)
    scanf("%d", &b);

  scanf("%d", &b);
  for ( ; b < 10 || b > 50; )
    scanf("%d", &b);

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Kao i prethodno, ilustracije radi, zadatak je riješen sa sve tri petlje. S obzirom da nije eksplicitno navedeno koju naredbu (petlju) treba koristiti izbor je proizvoljan, ali najjasniji i najčišći oblik je prvi. Kako je `do while` petlja s izlaznom provjerom uvjeta, barem jednom će biti pozvana funkcija `scanf()`, odnosno barem jednom će se tražiti ulaz od strane korisnika. Stoga, nema potrebe za dvostruku uporabu funkcije `scanf()`. Iako je moguće izbjeći dvostruki `scanf()` i u preostalim slučajevima, primjerice postavljanjem varijable `b` na vrijednost izvan zadanog intervala, time se samo gubi na jasnoći teksta programa koja je od iznimnog značaja i treba biti na prvom mjestu kod pisanja programa.

Primjer 4. *Omogućiti korisniku unos dva realna broja. Osigurati da prvi bude manji od drugog.*

```
#include <stdio.h>

int main(void)
{
  float a, b; //deklaracija varijabli koje predstavljaju brojeve

  printf("Unesite prvi broj: ");
  scanf ("%f", &a);
  do {
    printf("Unesite drugi broj koji mora biti veci od prvog (%f): ", a);
    scanf("%f", &b);
  } while (b <= a); //ili (a >= b)

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Zadatak je vrlo sličan prethodnom pa vidi pripadajuće komentare. Ispis poruka pomoću funkcije `printf()` uveden je samo iz estetskih razlog, jer nije tražen u zadatku.

Primjer 5. *Tražiti od korisnika unos 10 realnih brojeva. Potom, ispisati na ekran njihov zbroj, produkt i srednju vrijednost.*

```
#include <stdio.h>

int main(void)
{
  float broj;
  float zbroj, produkt, sred_vrijed;

  zbroj = 0;
  produkt = 1;

  printf("Unesite 10 realnih brojeva.\n");
  for (i = 0; i < 10; i++) {
    printf("%d. broj: ", i + 1); //ispisuje redni broj trazenog unosa
    scanf("%f", &broj);
    zbroj += broj;   //ili zbroj = zbroj + broj
    produkt *= broj; //ili produkt = produkt * broj
  }
  sred_vrijed = zbroj / 10; //izracun srednje vrijednosti

  printf("Zboj unesenih brojeva je: %f\n", zbroj);
  printf("Produkt unesenih brojeva je: %f\n", produkt);
  printf("Srednja vrijednost unesenih brojeva je: %f\n", sred_vrijed);

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Treba primijetiti kako su, za razliku od prethodnih primjera, korištena opisna imena za varijable što može u slučaju blago složenijih tekstova programa značajno unaprijediti čitljivost i jasnoću. Ključno je postaviti varijable koje predstavljaju zbroj i produkt na odgovarajuće (neutralne) vrijednosti prije uporabe, jer im se trenutne vrijednosti mijenjaju. Potsjetnika radi, naredbe deklaracije ne postavljaju vrijednosti varijabli. U primjeru je korištena `for` petlja, ali se mogao riješiti i s nekom od preostalih.

Primjer 6. *Omogućiti korisniku unos cijelog broja n iz intervala* $[5, 20\rangle$*. Potom, tražiti ga unos n parnih brojeva. Izračunati srednju vrijednost svih unesenih n parnih brojeva.*

```
#include <stdio.h>

int main(void)
{
  int n, broj;
  float zbroj, sredVrijed;

  zbroj = 0;

  do {
    printf("Unesite cijeli broj iz [5,20): ");
    scanf("%d", &n);
  } while (n < 5 || n >= 20);

  printf("Unesite %d parnih brojeva.\n", n);
  for (i = 0; i < n; i++) {
    //trazi se unos sve dok uneseni broj nije paran
    do {
      printf("%d. parni broj: ", i + 1);
      scanf("%d", &broj);
    } while (broj % 2 != 0);
    //tek nakon unosa parne vrijednosti nastavlja se trenutna iteracije for petlje
    zbroj += broj;
  }

  sredVrijed = zbroj / n;
  printf("Srednja vrijednost unesenih brojeva je: %f\n", sredVrijed);

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Prvi dio zadatka je isti kao Primjer 3. Drugi dio je sličan prethodnom. Slično kao u Primjeru 3, traži se unos sve dok ne zadovoljava uvjet. Nakon što se unese paran broj, dodaje se njegova vrijednost zbroju i prelazi se u narednu iteraciju petlje `for`. S obzirom da će srednja vrijednost općenito biti realna, varijabla `sredVrijed` je tipa podatka `float`. Slično, varijabla koja predstavlja zbroj je također tipa podatka `float`, a ne `int` kako ne bi došlo do cjelobrojnog dijeljenja kod izračuna srednje vrijednosti.

Primjer 7. *Omogućiti korisniku unos cijelog broja različitog od nula. Potom, ispisati svaku znamenku tog broja u posebnom retku te izračunati srednju vrijednost svih znamenki.*

```
#include <stdio.h>

int main(void)
{
  int b, t, bz; // t je pomocna varijabla
  float z, sv;

  printf("Unesite cijeli broj razlicit od nula: ");
```

```
  do {
    scanf("%d", &b);
  } while (b == 0);

  z = 0;  //zbroj znamenki unesenog broja
  bz = 0; //brojac znamenki unesenog broja
  while (b != 0) {
    t = b % 10; //varijabla t ce poprimiti vrijednost zadnje po redu znamenke (krajnja desna)
    b /= 10;    //"odbacivanje" zadnje znamenke; isto sto i b = b / 10
    printf("%d\n", t); //ispis znamenke
    z += t; //dodavanje vrijednosti znamenke zbroju
    bz++;   //trenutni broj znamenki
  }

  sv = z / bz; //izracun srednje vrijednosti
  printf("Srednja vrijednost znamenki je: %f", sv);

  return 0;
}
```

Kratko obrazloženje [Primjer 7]: Važno je primijetiti kako nema ograničenja na unos broja osim što mora biti različit od nule. Prema tome, broj znamenki nije unaprijed poznat. Kako bi se rastavilo broj na znamenke u svakoj iteraciji petlje prvo se računa ostatak pri cjelobrojnom dijeljenju s 10 (modulo 10 — odgovara bazi dekadskog brojevnog sustava) što daje zadnju znamenku (krajnju desnu). Potom, broj se dijeli s 10 kako bi se navedenu izgubilo. Osim što je varijabla `b` cjelobrojna, radi se o cjelobrojnom dijeljenju pa bi ju (krajnju desnu znamenku) se izgubilo bez obzira. Nakon što je rezultat dijeljenja (`b/10`) bio nula, završava se izvođenje petlje, jer se prošlo kroz sve znamenke unesenog broja.

Primjer 8. *Omogućiti korisniku unos cijelog broja iz intervala* $[0, 10]$. *Potom, izračunati i na ekran ispisati faktorijel tog broja.*

```
#include <stdio.h>

int main(void)
{
  int b, i;
  int f = 1; //varijabla koja predstavlja rezultat izracuna

  do {
    printf("Unesite cijeli broj iz [0, 10]: ");
    scanf("%d", &b);
  } while (b < 0 || b > 10);

  //izracuna faktorijela broja ako je veci ili jednak 2, u suprotnom se nece uci u petlju
  for (i = 2; i <= b; i++)
    f *= i;

  printf("Faktorijel broja %d (%d!) jednak je: %d", f);

  return 0;
}
```

Kratko obrazloženje [Primjer 8]: Faktorijel je definiran za nenegativne cijele brojeve (≥ 0). Faktorijel broja b označava se s $b!$ i računa se kao produkt svih pozitivnih cijelih brojeva (> 0) koji su manji ili jednaki b, odnosno $b! = \prod_{i=1}^{b} i$. Po definiciji je $0! = 1$. Primjerice, $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$. Prema navedenom, varijablu `i` u petlji `for` prvotno se postavlja na dva (`i=2`), jer u slučaju da je uneseni broj (`b`) manji od dva, varijabla `f` već predstavlja rješenje i nema potreba za daljnjim računanjem i može se preskočiti petlja (uvjet `i<=b` neće biti zadovoljen).

Primjer 9. *Omogućiti korisniku unos realnih brojeva sve dok ne unese nenumeričku vrijednost. Potom, izračunati i na ekran ispisati srednju vrijednost svih unesenih brojeva*

```
#include <stdio.h>

int main(void)
{
  double b;
  double z, sv;
  int u;

  z = u = 0; //zbroj i broj unesenih vrijednosti
  printf("Unosite realne brojeve. Nenumericki znak za kraj.");
  while (scanf("%lf", &b) == 1) {
    z += b;
    u++;
  }

  sv = z / u; //doci ce do dijeljenja s nulom ukoliko je prva unesena vrijednost nenumericka
  printf("Srednja vrijednost unesenih brojeva je: %f\n", sv);

  return 0;
}
```

Kratko obrazloženje [Primjer 9]: U zadatku se oslanja na povratnu vrijednost funkcije `scanf()` koja vraća broj uspješno učitanih podataka (u konkretnom slučaju je jedna u pitanju) ili posebnu vrijednost `EOF` (često jednaka -1) u slučaju greške. Prema tome, ukoliko je unesena numerička vrijednost, odnosno broj, funkcija `scanf()` vratit će vrijednost 1 te će uvjet biti istinit. Tako, prije svake nove iteracije petlje poziva se funkcija `scanf()` i potom se provjerava uvjet.

Primjer 10. *Za svaki cijeli broj* $1 \leq x \leq 20$ *ispisati na ekran, u posebnom retku, sve cijele brojeve od trenutne vrijednosti x do 1.*

```
#include <stdio.h>

int main(void)
{
  int x, y;

  for (x = 1; x <= 20; x++) { //vanjska petlja
    for (y = x; y >= 1; y--)  //unutarnja petlja
      printf("%d ", y); //ispis brojeva od x do 1
    printf("\n"); //prelazak u novi red
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 10]: Blok naredbi pod prvom petljom sadrži petlju. Ovakve konstrukcije nisu neuobičajene. Prva se petlja često naziva vanjskom, a druga, ona unutar nje, unutarnjom petljom. Isto se može primijetiti u Primjeru 6, gdje je unutarnja petlja `do while`.

Primjer 11. *Izračunati i ispisati na ekran n-ti član Fibonaccijevog niza, gdje je* $2 \leq n \leq 200$*, a zadaje ga korisnik.*

```
#include <stdio.h>

int main(void)
{
  int p1, p2, fn;
```

```
  int i, n;

  do {
    printf("Unesite cijeli broj iz intervala [3,200]:\n");
    scanf("%d", &n);
  } while (n < 2 || n > 200);

  p1 = 0; //prvi clan niza; prva prethodna vrijednost
  p2 = 1; //drugi clan niza; druga prethodna vrijednost
  for (i = 2; i <= n; i++) {
    fn = p1 + p2; //izracun i-tog clana niza
    //azuriranje prethodna dva clana
    p1 = p2;
    p2 = fn;
  }

  printf("%d-ti (%d. po redu) clan Fibonaccijevog niza je: %d.\n", n, n + 1, fn);

  return 0;
}
```

Kratko obrazloženje [Primjer 11]: Fibonaccijev niz brojeva je niz čiji su članovi, od trećeg pa nadalje, jednaki zbroju prethodna dva člana. Prva dva člana su 1 i 1 ili 0 i 1. Primjerice, prvih 10 članova niza su: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34$. Formalno, n-ti član niz je $F_n = F_{n-1} + F_{n-2}$, gdje je $n = 0, 1, 2, \ldots$, a $F_0 = 0$ i $F_1 = 1$. Prema tome, ključno je uvijek imati pretodna dva člana niza u odnosu na trenutni (varijable `p1`, `p2` i `fn`, redom). U konkretnom primjeru, kreće se od nultog člana koji je jednak 0 pa je stoga n-ti zapravo $(n+1)$-ti po redu. Međutim, kao što je navedeno, moguće je izostaviti nulti član i krenuti od $n = 1$ ($F_1 = 1$ i $F_2 = 1$ su prva dva člana u tom slučaju).

POGLAVLJE 5

Upravljanje tijekom izvođenja programa: Grananje

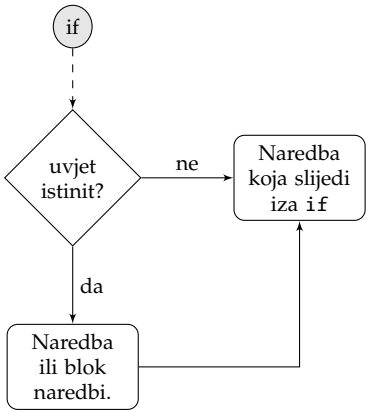
U pravilu se naredbe koje čine neki program ne izvode slijedno (sve redom jedna iza druge), nego je vrlo često potrebno neke preskočiti ili izvršiti ovisno o, primjerice, trenutnom stanju nekih varijabli ili rezultatu nekog izračuna. Za postizanje uvjetnog izvršavanja naredbi ili blokova istih potrebne su odgovarajuće konstrukcije. Programski jezik C raspolaže s nekoliko konstrukcija ili naredbi za ostvarivanje navedenog. Njima se mogu postići različiti tokovi izvođenja programa, odnosno grananja tijeka izvođenja. Naredbe za grananje vrlo često se koriste u kombinaciji s petljama.

5.1 Naredba `if`

Naredba `if` je fleksibilna i najčešće korištena konstrukcija za uvjetno izvođenje jedne ili više naredbi. Općeniti oblik ili struktura naredbe `if` prikazana je u nastavku. Slikom 5.1 prikazan je njen način rada.

```
if (uvjet)
  naredba
```

Izvođenje naredbe ili bloka naredbi ovisi o vrijednosti izraza `uvjet`. Ukoliko je istinit, naredba ili blok naredbi pod `if` se izvršavaju, dok se u suprotnom ne izvršavaju (preskače ih se). U oba slučaja daljnje izvođenje programa se nastavlja s prvom naredbom koja slijedi



Slika 5.1: Način rada naredbe **`if`**

iza `if` kao što se može vidjeti na slici 5.1. Navedeni prikaz naredbe može se čitati kao: *ako je* `uvjet` *istinit onda čini* `naredba`. Ključna riječ `if` s pripadajućim uvjetom u zagradama i naredbom ili blokom naredbi, smatra se jednom cjelinom, odnosno naredbom. U izlistanju 5.1 je navedeno nekoliko primjera primjene naredbe `if`.

```
if (a == 0)
  printf("Nula.");

if (a < 1 || a > 10)
  printf("Izvan granica.");

if (broj % 2 == 0) {
  bp++;
  zbroj += broj;
}
```

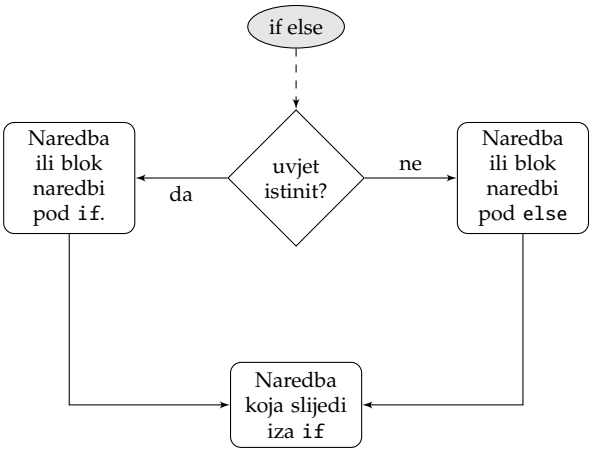
Izlistanje 5.1: Primjeri naredbe `if`

5.1.1 Proširenje `if` s `else`

Uporabom naredbe `if` izvršava se naredba ili blok naredbi ako je uvjet ispunjen, dok se u suprotnom preskaču isti. Ukoliko je nužno izvršiti posebnu naredbu ili blok naredbi ako uvjet nije ispunjen, može se `if` proširiti s ključnom riječi `else`, što je prikazano općenitom strukturom u nastavku. Slikom 5.2 prikazan je način rada `if else` konstrukcije.

```
if (uvjet)
  naredba1
else
  naredba2
```

Kao i u prethodnom slučaju izvršavanje naredbe ili bloka naredbi pod `if` (predstavljeno s `naredba1`) ovisi o vrijednosti izraza `uvjet`. Ukoliko je istinit, izvršava se, dok se u suprotnom (ili inače) izvršava naredba ili blok naredbi pod `else` (predstavljeno s `naredba2`). Navedeni prikaz `if else` može se čitati kao: *ako je* `uvjet` *istinit onda čini* `naredba1`, a u suprotnom čini `naredba2`. Treba istaknuti kako se `if` i `else` s pripadajućim naredbama ili blokovima naredbi smatraju jednom cjelinom ili naredbom. Važno je primijetiti kako se uz `else` ne može navoditi poseban uvjet, jer je `else` vezan za `if` i njemu



Slika 5.2: Način rada naredbe **`if`** u kombinaciji s **`else`**

pripadajući uvjet. Izlistanjem 5.2 je prikazano nekoliko primjera uporabe `if` u kombinaciji s `else`.

```
if (a > 0)
  printf("Pozitivno.");
else
  printf("Negativno");

if (broj % 2 == 0) {
  bp++;
  bp += broj;
}
else {
  bnp++;
  znp += broj;
}

if (br > 0 && br <= 1)
  t1++;
else {
  to++;
  printf("%d\t%d\n", t1, to);
}
```

Izlistanje 5.2: Primjeri kombinacije `if` i `else`

Ukoliko je potrebno ostvariti višestruko grananje ili odabir između više različitih naredbi, moguće je ugnijezditi više `if else` konstrukcija kao što je pokazano primjerom u nastavku. Važno je primijetiti kako `else if` kao ključna riječ ne postoji, nego da se radi o `else` pod kojim se nalazi `if`. Također, zadnji dio ne mora biti `else`, nego može biti i `if`, odnosno pod krajnjim `else` može biti samo `if` (bez naknadnog `else`). Navedeno je ilustrirano izlistanjem 5.3.

```
if (br > 0 && br < 1)
  t1++;
else {
  if (br > 1 && br < 2)
    t2++;
  else {
    if (br > 2 && br < 3)
      t3++;
    else {
      to++;
      printf("%d\t%d\t%d\%d\n", t1, t2, t3, to);
    }
  }
}
//sazeti i uobicajeni zapis istog kao i iznad
if (br > 0 && br < 1)
  t1++;
else if (br > 1 && br < 2)
  t2++;
else if (br > 2 && br < 3)
  t3++;
else {
  to++;
  printf("%d\t%d\t%d\%d\n", t1, t2, t3, to);
}
```

Izlistanje 5.3: Primjeri višestruke kombinacije `if` i `else`

5.2 Naredba `switch`

Jednostavno rečeno, naredba `switch` omogućuje izbor između niza radnji (mogu se sastojati od jedne ili više naredbi). Općenita struktura naredbe `switch` prikazan je u nastavku (u ograničenom obliku).

```
switch (test) {
  case konst1:
    naredbe1
  case konst2:
    naredbe2
  default:
    naredbe3
}
```

Svaka radnja ima svoju oznaku (engl. *label*) i predstavlja jedna od slučajeva (engl. *cases*) za izvođenje. U prikazanoj strukturi, oznake su predstavljene s `konst1` i `konst2`, a radnje s `naredba1`, `naredba2` i `naredba3`. Koja radnja će biti izvršena ovisi o vrijednosti izraza `test`. Izvođenje se započinje od oznake koja je jednaka vrijednosti izraza `test` i nastavlja se do samog kraja (zatvorenih uglatih zagrada). Ukoliko vrijednost izraza `test` nije jednak niti jednoj oznaci izvodi se, ako je navedena (nije obavezno), podrazumijevana radnja/slučaj (naredbe pod `default` — ne ide oznaka poslije te ključne riječi). Treba istaknuti kako se pod `switch` može nalazi jedan ili više slučajeva s različitim oznakama za svaki.

Ključno je napomenuti kako izraz `test` mora biti cjelobrojan (izraz koji se sastoji isključivo od cjelobrojnih operanada). Slično tome, pojedine oznake (`konst1`, `konst2`, ...) moraju biti cjelobrojne konstante, ili općenito, cjelobrojni konstantni izrazi (izraz koji se sastoji isključivo od cjelobrojnih i konstantnih operanada). Ako se radnja sastoji od više naredbi, one se ne moraju nalaziti unutar para vitičastih zagrada. U izlistanju 5.4 je prikazano par primjera uporabe naredbe `switch`.

```
swtich (a) {
  case 1:  printf("Jedan!"); break;
  case 2:  printf("Dva!");   break;
  case 3:  printf("Tri!");   break;
  default: printf("Ne ide tako daleko.");
}

switch (c)  {
  case 'A':
  case 'a': brA++; printf("Malo ili veliko slovo a.\n"); break;
  case 'B':
  case 'b': brB++; printf("Malo ili veliko slovo b.\n"); break;
  case 'C':
  case 'c': brC++; printf("Malo ili veliko slovo c.\n"); break;
}
```

Izlistanje 5.4: Primjeri naredbe `switch`

Izraz `test` najčešće predstavlja samo varijablu cjelobrojnog tipa[1], čiju se vrijednost traži među oznakama navedenima pod `switch`. Kao što je prethodno spomenuto, kada se pronađe odgovarajuća oznaka izvršavaju se sve naredbe od tog mjesta do kraja. Stoga je iznimno važno primijetiti naredbu `break` koja se pojavljuje u prikazanim primjerima. Njenim izvođenjem se prekida `switch`, a nakon čega se prelazi na prvu naredbu koja slijedi iza `switch`. Prema opisanom, oznake se mogu promatrati kao početak niza naredbi, a

[1] Podrazumijeva i tip podatka `char`.

`break` kao kraj tog niza. Naredba `break` nije nužna ako nema daljnjih oznaka, ali ju je dobro ostaviti radi konzistentnosti.

5.3 Operator ?:

Operator `?:` je jedini ternarni operator u jeziku. To znači kako ima tri operanda. Naziva ga se i uvjetnim operatorom (engl. *conditional operator*). U nastavku je prikazan općeniti oblik tog operatora s pripadajućim operandima.

```
uvjet ? stavka1 : stavka2
```

Prikazani izraz poprima vrijednost koja odgovara `stavka1` ili `stavka2` ovisno o izrazu `uvjet`. Ako je izraz `uvjet` istinit (općenito, vrijednost različita od nula) cijeli izraz vrednuje se kao `stavka1`, dok se u suprotnom vrednuje kao `stavka2`. Treba napomenuti kako `stavka1` i `stavka2` općenito predstavljaju izraze (tako da između ostalog može biti konstanta ili pak poziv neke funkcije). Isto tako, valja napomenuti da ukoliko je izraz `uvjet` istinit bit će vrednovan samo izraz predstavljen sa `stavka1`, dok `stavka2` neće i obratno[2]. Izlistanjem 5.5 je prikazano nekoliko primjera uporabe uvjetnog operatora.

```
abs = x < 0 ? -x : x;

max = a > b ? a : b;

printf("var %s djeljivo s pet.", a % 5 == 0 ? "je" : "nije");
```

Izlistanje 5.5: Primjeri uvjetnog operatora `?:`

Iz prikazanog se da zaključiti kako se isto može postići odgovarajućom `if else` naredbom. S tog gledišta, prvi izraz poslije `?` i prije `:` je pod `if`, a drugi je pod `else`. Međutim, treba biti svjestan razlika. S obzirom da se radi o izrazu [uvjetni izraz (engl. *conditional expression*)], moguće ga je ugraditi u druge izraze ili predati funkcijama kao argument.

5.4 Ključne riječi break i continue

Ključna riječ `break` (ukoliko stoji točka-zarez nakon nje predstavlja naredbu) osim što se koristi za izlazak iz konstrukcije `switch`, može se još koristiti za izlazak ili prekid petlji (`for`, `while` i `do while`). Prema tome, naredbu `break` moguće je isključivo koristiti unutar `switch` i unutar petlji. U oba slučaja, nakon izvršavanja naredbe `break` dolazi do prekida (time i preskakanje svih eventualnih naredbi koje slijede iza) i prelaska na izvršavanje prve naredbe koja slijedi iza `switch` ili petlje.

Primjena ključne riječi, odnosno naredbe `continue` je ograničena isključivo na petlje. Za razliku od `break`, naredba `continue` ne prekida izvođenje petlje, nego trenutnu iteraciju uz izravni prelazak na iduću. To znači kako se nakon izvršavanje naredbe `continue` prelazi na provjeru uvjeta o ponovnom izvršavanju naredbi ili bloka naredbi u sklopu petlje. Eventualne naredbe koje slijede iza `continue` bit će preskočene. Bitno je napomenuti,

[2]To je slično logičkim operatorima && i ||. Tako u slučaju logičkog operatora I (&&), neće biti vrednovan drugi operand ukoliko je prvi neistinit. U slučaju logičkog operatora ILI (||), neće biti vrednovan drugi operand ukoliko je prvi istinit.

kako se u slučaju petlje `for` zapravo prvo vrednuje zadnji izraz (`azuriranje`[3]), a onda provjerava uvjet.

Naredbe `break` i `continue` unutar petlji se u pravilu uvijek nalaze pod nekim uvjetom (uglavnom, pod naredbom `if`). Te naredbe su korisne kada broj ponavljanja (iteracija) nije unaprijed poznat i kada ovisi od ispunjavanju određenog uvjeta. U izlistanju 5.6 je prikazano par primjera uporabe naredbi `break` i `continue` unutar petlji.

```
while (1) {
  scanf("%d", &b);
  if (b > 0 && b <= 10)
    break;
  else
    printf("Unos izvan granica.");
}

for (i = 0; i < 10; i++) {
  scanf("%d", &b);
  if (b % 2 != 0)
    continue;
  bp++;
  zp += b;
  printf("Parna vrijednost.");
}
```

Izlistanje 5.6: Primjeri naredbi `break` i `continue`

5.5 Naredba goto

U programskom jeziku C, kao i u nekim drugim starijim jezicima (primjerice, Basic i Fortran), ali čak kao i u nekim modernim jezicima (primjerice, C#) postoji naredba `goto` koja omogućuje skokove unutar tijeka izvođenja. Međutim, njena uporaba može učiniti tekst programa vrlo teškim za razumjeti i pratiti. Stoga se ne preporučuje njena uporaba. Višestruko ugniježđene petlje predstavljaju jednu od vrlo rijetkih situacija kada postoji donekle opravdan razlog za uporabu naredbe `goto` (preporuka je koristiti naredbe `break` i `continue` za prekide izvođenja petlji ili preskakanje iteracija). Zbog svega navedenog, toj naredbi neće biti posvećena daljnja pozornost.

5.6 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je potrebno koristiti neki oblik grananja za njihovo rješavanje. Uz to, često je nužno koristiti i petlje. U primjerima nije eksplicitno navođeno koji oblik grananja treba upotrijebiti.

[3] `for (inicijalizacija; uvjet; azuriranje)`

Primjer 1. *Omogućiti korisniku unos jednog cijelog broja. Ispisati na ekran odgovarajuću poruku ovisno o tome je li broj negativan, pozitivan ili jednak nula.*

```
#include <stdio.h>

int main(void)
{
  int x;

  scanf("%d", &x);

  if (x < 0)
    printf("Broj je negativan.");
  else if (x > 0)
    printf("Broj je pozitivan.");
  else
    printf("Unijeli ste nula.");

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: S obzirom da treba ispisati odgovarajuću poruku ovisno o unesenoj vrijednosti, naredba `if` s `else` nameće se kao prikladan izbor za postizanje navedenog. Uvjete i odgovarajuće poruke moguće je postaviti i drukčije, primjerice prvo ispitati je li unesena vrijednost jednaka nula itd. Također, primjer se mogao riješiti i bez uporabe `else`, odnosno s tri (odvojene) naredbe `if`. Međutim, u tom slučaju se uvijek obavljaju tri vrednovanja izraza unutar zagrada iza ključne riječi `if` (ispitivanja uvjeta), dok se u konkretnom slučaju obavlja jedno ili dva ovisno o unesenoj vrijednosti. Osim toga, uporaba `else` doprinosi i jasnoći onog što se želi ostvariti.

Primjer 2. *Omogućiti korisniku unos jednog cijelog broja. Izračunati i na ekran ispisati apsolutnu vrijednost unesenog broja.*

```
#include <stdio.h>

int main(void)
{
  int x, a;

  printf("Unesite cijeli broj: ");
  scanf("%d", &x);

  a = x < 0 ? -x : x;
  printf("Apsolutna vrijednost broja %d je %d.", x, a);

  //isto kao i prethodno s uvjetnim operatorom
  if (x < 0)
    a = -x;
  else
    a = x;
  printf("Apsolutna vrijednost broja %d je %d.", x, a);

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Apsolutnu vrijednost moguće je izračunati jednostavnom promjenom predznaka ako je broj negativan, dok u suprotnom već predstavlja apsolutnu vrijednost. Prema tome, naredba `if` u kombinaciji s `else` predstavlja valjan izbor. Međutim, to predstavlja i uvjetni operator `?:` koji uz to omogućuje sažetiji način postizanja traženog.

Primjer 3. *Tražiti od korisnika unos dva realna broja. Potom, omogućiti mu unos aritmetičke operacije kao znaka +, -, * ili /. Ovisno o unosu željene operacije, izvršiti odgovarajući izračun nad unesenim brojevima i na ekran ispisati rezultat.*

```
#include <stdio.h>

int main(void)
{
  double x, y;
  double rez;
  char op;
  int f = 1; //zastavica koja signalizira valjanost unosa operacije

  printf("Unesite dva realna broja:\n");
  scanf("%lf %lf", &x, &y);

  printf("Unesite operaciju koju zelite izvrsiti na unesenim brojevima (+, -, * ili /):\n");
  scanf(" %c", &op); //primijeti razmak ispred znaka %
  switch (op) {
    case '+':
      rez = x + y;
      break;
    case '-':
      rez = x - y;
      break;
    case '*':
      rez = x * y;
      break;
    case '/':
      rez = x / y;
      break;
    default:
      printf("Nepoznata operacija!\n");
      f = 0; //zastavica se postavlja na drugu vrijednost kako bi se izbjegao nevaljani ispis
  }

  if (f == 1) //rezultat se samo ispisuje ukoliko je bila unesena valjana operacija
    printf("\n%f %c %f = %f\n", x, op, y, rez);

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: S obzirom da je nužno ispitati varijablu koja može imati nekoliko različitih prihvatljivih vrijednosti, naredba `switch` je prikladan izbor koji čini tekst programa jasnijim nego uporaba naredbe `if`. Za svaku od prihvatljivih vrijednosti definirana je odgovarajuća radnja (pod `case`-ovima), a za neprihvatljive posebna (pod `default`). Iako općenito nebitno, uporaba naredbe `switch` može rezultirati blago bržim izvođenjem u odnosu na uporabu `if else`.

Primjer 4. *Tražiti od korisnika unos tri realna broja. Ispisati ih silaznim redoslijedom (od najvećeg prema najmanjem).*

```
#include <stdio.h>

int main(void)
{
  float a, b, c;
  float x; //pomocna varijabla

  printf("Unesite tri realna broja:\n");
  scanf("%f %f %f", &a, &b, &c);

  if (a < b) { //zamijena vrijednosti varijabli a i b, ako je b vece
    x = a;
    a = b;
    b = x;
```

```
  }
  if (a < c) { //zamijena vrijednosti varijabli a i c, ako je c vece
    x = a;
    a = c;
    c = x;
  }
  if (b < c) { //zamijena vrijednosti varijabli b i c, ako je c vece
    x = b;
    b = c;
    c = x;
  }

  printf("Sortirani unos: %f, %f, %f.", a, b, c);

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Ovdje je bitno imati (odvojene) naredbe `if` bez `else`, jer sva tri izraza mogu biti istinita pa ih se ne smije preskakati. Kako bi se zamijenile vrijednosti dvije varijable potrebna je pomoćna kao "međuspremnik" (može se zamisliti kao zamjena tekućina iz dvije posude). Nakon prve dvije naredbe `if`, varijabla `a` sigurno sadrži najveću vrijednost, te preostaje samo eventualna zamjena vrijednosti varijabli `b` i `c`.

Primjer 5. *Omogućiti korisniku unos jednog cijelog broja. Osigurati da bude unutar intervala* $[0, 10]$*. Koristiti beskonačnu petlju i naredbu* `break`*.*

```
#include <stdio.h>

int main(void)
{
  int x;

  while (1) { //beskonacna petlja
    printf("Unesite cijeli broj: ");
    scanf("%d", &x);
    if (x < 0 || x > 10) //provjera unesene vrijednosti
      printf("Broj mora biti unutar [0,10].\n"); //ispis upozorenja ako je unos izvan granica
    else
      break; //ako je unesena vrijednost unutar zahtjevanog intervala prekida se petlja
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Iako se mogu koristiti i preostale petlje, često je korištena `while` zbog jasnoće i sažetosti. Kako se radi o beskonačnoj petlji, njen prekid je nužno obaviti pomoću `break` koja je pod naredbom `if`, odnosno `else` u konkretnom slučaju.

Primjer 6. *Omogućiti korisniku unos n cijelih brojeva. Korisnik također unosi* $5 \leq n \leq 50$*. Odrediti i ispisati na ekran koliko je parnih brojeva uneseno.*

```
#include <stdio.h>

int main(void)
{
  int x, n, brp;
  int i;

  printf("Koliko brojeva zelite unijeti? ");
  for ( ; ; ) { //beskonacna petlja; manje uobicajena nego while (1)
    scanf("%d", &n);
    if (x >= 5 && x <= 50)
```

```
      break; //ako je unesena vrijednost unutar zahtijevanog intervala prekida se petlja
    else
      printf("\nBroj mora biti iz [5,50]!");
  }

  brp = 0;
  for (i = 0; i < n; i++) {
    printf("\nUnesite %d. broj: ", i + 1);
    scanf("%d", &x);
    if (x % 2 == 0)
     brp++;
  }
  printf("Unijeli ste %d parnih brojeva.\n", brp);

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Prvi dio zadatka je isti kao i prethodni, ali uz uporabu petlje `for` i obratnog ispitivanja uvjeta prekida petlje. Za svaki unos se provjerava parnost.

Primjer 7. *Omogućiti korisniku unos 15 znakova. Odrediti i ispisati na ekran koliko je kojih samoglasnika uneseno.*

```
#include <stdio.h>

int main(void)
{
  char z;
  int ba, be, bi, bo, bu;
  int i;

  ba = be = bi = bo = bu = 0; //pridruzivanje ide s desna na lijevo
  for (i = 0; i < 15; i++) {
    printf("\nUnesite %d. znak: ", i + 1);
    scanf(" %c", &z);
    switch (z) { //u obzir se uzimaju mala i velika slova
      case 'a':
      case 'A': ba++; break;
      case 'e':
      case 'E': be++; break;
      case 'i':
      case 'I': bi++; break;
      case 'o':
      case 'O': bo++; break;
      case 'u':
      case 'U': bu++;
    }
  }
  printf("Unijeli ste %d samoglasnika.\n", ba + be + bi + bo + bu);
  printf("Slovo \'a\' %d puta.\nSlovo \'e\' %d puta.\nSlovo \'i\' %d puta.\n"
  "Slovo \'o\' %d puta.\nSlovo \'u\' %d puta.\n", ba, be, bi, bo, bu);

  return 0;
}
```

Kratko obrazloženje [Primjer 7]: Kako bi se u obzir uzela mala i velika slova nužno je za sve napisati odgovarajući slučaj (`case`). Međutim, neovisno je li malo ili veliko slova istu radnju treba obaviti. Stoga, slučajevi su "grupirani" u parove malo i veliko slovo, gdje dolazi do "propadanja" ukoliko se radi o malom slovu (naravno, obratno navođenje je također moguće — veliko pa malo slovo). Pod zadnjim slučajem (samoglasnik u) izostavljena je naredba `break`, jer nema daljnjih slučajeva.

Primjer 8. *Omogućiti korisniku unos 10 realnih brojeva. Odrediti i ispisati na ekran najmanji i najveći uneseni broj.*

```
#include <stdio.h>

int main(void)
{
  float br, min, max;
  int i;

  for (i = 0; i < 10; i++) {
    printf("\nUnesite %d. broj: ", i + 1);
    scanf("%f", &br);

    if (i == 0) //nakon unosa prvog broja on predstavlja trenutno najmanji i najveci
      min = max = br;
    else if (min > br) //svaki naredni unos, azurira se trenutno najmanji
      min = br;
    else if (max < br) //isto tako se azurira trenutno najveci
      max = br;
  }

  printf("Najmanji i najveci uneseni broj su %f i %f, redom.\n", min, max);

  return 0;
}
```

Kratko obrazloženje [Primjer 8]: Nakon prvog unosa, varijable `min` i `max` se postavljaju na njegovu vrijednost. Svaki sljedeći unos obavlja se ažuriranje vrijednosti navedenih varijabli. Prema tome, varijable `min` i `max` u svakoj iteraciji održavaju do tada najmanju i najveću unesenu vrijednost, redom.

Primjer 9. *Omogućiti korisniku unos prirodnog broja većeg od 1. Odrediti i ispisati na ekran je li prost.*

```
#include <stdio.h>

int main(void)
{
  int b, i;
  int f = 1; //zastavica koja signalizira je li broj prost

  printf("Unesite prirodan broj veci od 1:\n");
  do {
    scanf("%d", &b);
    if (b < 1)
      printf("\nBroj mora biti veci od 1!");
    else
      break;
  } while (1);

  for (i = 2; i < b; i++)
    if (b % i == 0) {
      f = 0; //broj nije prost ako je djeljiv s bilo kojim brojem osim 1 i samog sebe
      break; //ako nije prost moze se prekinuti petlja
    }
  //uvjetni operator unutar printf()
  printf("Broj %d %s prost.\n", b, f == 0 ? "nije" : "je");

  return 0;
}
```

Kratko obrazloženje [Primjer 9]: Prirodan broj veći od 1 koji nema pozitivnih djelitelja osim 1 i samog sebe je prost broj. Prema tome, je li neki broj prost može se utvrditi dijeljenjem istog sa svim prirodnim brojevima između 1 i njega samog. Ako je djeljiv s

bilo kojim od njih, broj nije prost. Navedeno je postignuto petljom `for` kojom se provjera djeljivost sa svim brojevima između 1 i broja kojeg se provjera.

Primjer 10. *Izračunati zbroj svih prirodnih brojeva manjih od 1000 koji su djeljivi s 3 ili 5*[4]*. One brojeve koji su djeljivi s obje navedene vrijednosti uzeti samo jednom.*

```
int main(void)
{
  int zbroj, b;

  zbroj = 0;
  for (b = 3; b < 1000; b++) //prvi broj koji zadovoljava uvjet je 3
    if (b % 3 == 0 || b % 5 == 0)
      zbroj += b;

  return 0;
}
```

Kratko obrazloženje [Primjer 10]: Jednostavan zadatak, gdje je potrebno provjeriti djeljivost s tri i pet svih prirodnih brojeva manjih od 1000. Ako je broj (predstavljen varijablom b) djeljiv s tri ili pet, dodaje ga se zbroju.

Primjer 11. *Omogućiti korisniku unos jednog cijelog broja. Odrediti i ispisati na ekran koliko parnih znamenki sadrži.*

```
#include <stdio.h>

int main(void)
{
  int broj, bp;
  int z;

  printf("Unesite cijeli broj: ");
  scanf("%d", &broj);

  bp = 0; //broj parnih znamenki
  while (broj != 0) {
    z = broj % 10; //zadnja znamenka po redu (krajnja desna)
    if (z % 2 == 0)
      ++bp; //isto kao bp++ (u ovom slucaju)
    broj /= 10; //odbacivanje krajnje znamenke
  }
  printf("Uneseni broj sadrzi %d parnih znamenki.\n", bp);

  return 0;
}
```

Kratko obrazloženje [Primjer 11]: Svaku znamenku se provjerava je li parna.

Primjer 12. *Omogućiti korisniku unos jednog pozitivnog cijelog broja. Odrediti i ispisati na ekran je li broj savršen. Savršen broj predstavlja svaki pozitivni cijeli broj koji je jednak zbroju svojih pravih pozitivnih djelitelja (svi pozitivni djelitelji osim samog broja).*

```
#include <stdio.h>

int main(void)
{
  int broj, d;
  int i;
```

[4] Prvi zadatak s Project Euler, https://projecteuler.net

```
  while (1) {
    printf("Unesite pozitivan cijeli broj: ");
    scanf("%d", &broj);
    if (broj < 1)
      printf("Broj mora biti pozitivan.\n");
    else
      break;
  }

  d = 0;
  for (i = 1; i < broj; i++) {
    if (broj % i == 0)
      d += i; //zbrajanje svih djelitelja i
  }

  if (d == broj)
    printf("Uneseni broj je savrsen.\n");
  else
    printf("Uneseni broj nije savrsen.\n");

  return 0;
}
```

Kratko obrazloženje [Primjer 12]: Prema opisu savršenih brojeva, potrebno je za uneseni broj zbrojiti njegove prave djelitelje. Ukoliko je taj zbroj jednak unesenom broju, onda je broj savršen. Ispis odgovarajuće poruke se mogao riješiti i jednim pozivom funkcije `printf()` uz uporabu uvjetnog operatora kao u Primjeru 9.

POGLAVLJE 6

Polja: jednodimenzionalna polja

Spremanje veće količine srodnih podataka, kao što su primjerice ocjene grupe studenata iz nekog kolegija ili dnevne temperature zadnjih nekoliko dana, zahtjevalo bi isto toliko velik broj varijabli koje bi bile istog tipa podatka. Što na koncu predstavlja značajno opterećenje u pogledu imenovanja i rukovanja varijablama. Upravo polja (engl. *arrays*) omogućuju pohranu i rukovanje velikim brojem srodnih podataka na prikladan i zgodan način. Stoga, ona predstavljaju vrlo bitnu značajku gotovo svakog programskog jezika.

Polja se sastoje od slijeda elemenata istoga tipa podataka. Ona se još nekada nazivaju *nizovima* ili čak *vektorima*. Polja mogu biti jednodimenzionalna, dvodimenzionalna ili više-dimenzionalna, pri čemu polja više od dvije dimenzije rijetko imaju primjenu. Treba napomenuti kako polje samo po sebi nije varijabla, nego se to može reći samo za njegove elemente.

6.1 Jednodimenzionalna polja

Jednodimenzionalna (1-D) polja su vjerojatno polja načešće korištene dimenzionalnosti. Slikom 6.1 ilustrirana su 1-D polja čiji su elementi cijeli, odnosno realni brojevi. Kao što je navedeno, pojedine elemente moguće je promatrati kao spremnike ili varijable.

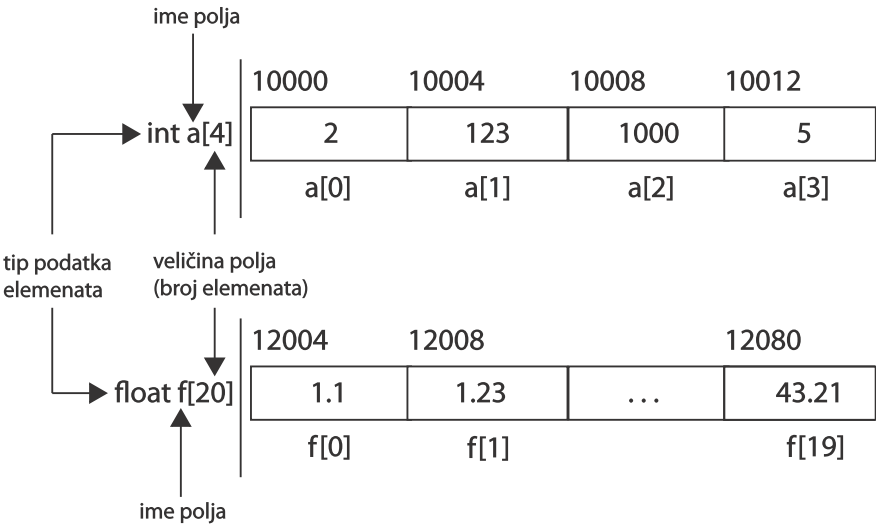
6.1.1 Deklaracija i inicijalizacija

Polja je kao i primjerice varijable potrebno deklarirati prije upotrebe. Deklaracija određuje ime, tip podatka elemenata i veličinu polja (broj elemenata):

```
tip_podatka ime_polje[VELICINA_POLJA];
```

Tip podatka elemenata polja može biti bilo koji, kao primjerice, `int`, `float`, `double` i `char`. Veličina polja mora biti veća od 0 i mora biti konstantan cjelobrojni izraz, odnosno cjelobrojni izraz čija je vrijednost poznata prije prevođenja[1]. Prilikom deklaracije polja, redom se navode, tip podatka (elemenata), ime i veličina unutar para uglatih zagrada. Imenovanje polja podliježe istim pravilima kao i varijabli osnovnih tipova podataka. U izlistanju 6.1 navedeno je nekoliko primjera deklaracije 1-D polja.

[1] Treba napomenuti kako se pod vrijendosti poznatom prije prevođenja ne smatraju deklaracije konstanti preko ključne riječi `const` (kao primjerice, `const int VEL = 50;`).



Slika 6.1: Primjeri jednodimenzionalnih (1-D) polja. Prvo predstavlja polje od 4 elementa, gdje je svaki tipa podatka `int`. Drugo predstavlja polje od 20 elemenata, gdje je svaki tipa podatka `float`.

```
int polje[5];   //deklaracija 1-D polja od 5 elementa, svaki tipa podatka int
float polje_a[12];  //deklaracija 1-D polja od 12 elementa, svaki tipa podatka float
char niz[20];   //deklaracija 1-D polja od 20 elementa, svaki tipa podatka char
double vektor[100]; //deklaracija 1-D polja od 100 elementa, svaki tipa podatka double
```

Izlistanje 6.1: Primjeri deklaracije 1-D polja

Polja je kao i varijable moguće inicijalizirati (zadati vrijednosti elemenata prilikom deklaracije). Za inicijalizaciju se koristi inicijalizator (engl. *array initializer*), gdje se pojedine vrijednosti elemenata navode, redom, unutar vitičastih zagrada, a razdvaja ih se zarezima. Vrijednosti moraju predstavljati konstantne izraze. To znači, ne mogu se koristiti varijable unutar inicijalizatora. Prilikom inicijalizacije moguće je izostaviti veličinu polja, te će u tom slučaju, prevoditelj sam odrediti veličinu na temelju inicijalizatora (koliko je vrijednosti navedeno, tolika će biti veličina polja). Inicijalizacija može biti potpuna i djelomična (u slučaju da se navodi veličina polja). Kod potpune, navode se vrijednosti za sve elemente. Kod djelomične, navodi se samo vrijednosti za dio elemenata, dok se ostali automatski postavljaju na vrijednost nula (0). Treba napomenuti kako prilikom inicijalizacije mora biti navedena barem jedna vrijednost, odnosno inicijalizator ne smije biti prazan. U izlistanju 6.2 navedeno je nekoliko primjera inicijalizacije jednodimenzionalnog polja.

```
int polje[7] = {1, 2, 5, 18, 123, 1, 5};      //potpuna inicijalizacija
char fakultet[5] = {'e', 't', 'f', 'O', 'S'}; //potpuna inicijalizacija
float v[] = {1.2, 2.3, 3.45, 6.78, 9.01, 121.125};    //inicijalizacija; prevoditelj
                                                      //odreduje velicinu (6)
unsigned u[] = {152, 56, 87, 12, 69, 101, 4, 8, 5, 12, 6, 7}; //inicijalizacija; prevoditelj
                                                               //odreduje velicinu (12)
double x[20] = {0.12345, 5.5, 100.005, 0, 12.125, 78.9012};   //djelomicna inicijalizacija;
                                                               //prvih 6 elem. poprima navedene
                                                               //vrijednosti, dok ostali 0
long p[50] = {0};           //djelomicna inicijalizacija;
                            //postavlja sve elemente na 0
```

Izlistanje 6.2: Primjeri inicijalizacije 1-D polja

6.1.2 Pristup elementima

Pojedinim elementima polja pristupa se preko imena tog polja i indeksa (engl. *index* ili *subscript*). Indeks se navodi unutar para uglatih zagrada koje, u ovom kontekstu, predstavljaju operator – operator indeksiranja `[]` koji je binaran (ime polja i indeks su operandi) i lijevo asocijativan. Vrijednost indeksa mora biti cjelobrojni izraz (ne smije biti realan i ne mora biti konstantan). Prvi element polja ima indeks s vrijednosti nula (0), a zadnji element polja, indeks s vrijednosti $n - 1$, gdje je n veličina polja (vidi Sl. 6.1). Svaki element polja se ponaša kao varijabla tipa navedenog pri deklaraciji polja.

Izuzetno je bitno voditi brigu o vrijednostima indeksa kako se ne bi izašlo izvan granica polja. Ponašanje programa pri pokušaju pristupa nepostojećim elementima nije definirano. Izlistanjem 6.3 dano je nekoliko primjera pristupa elementima 1-D polja.

```
float a[8] = {5.1, 3, 7, 1.1, 2.25, 17.125, 24.75, 0.8};
float x;
int i = 1;
x = a[0];   //pridruzivanje vrijednosti 5.1 varijabli x
x = a[7];   //pridruzivanje vrijednosti 0.8 varijabli x
x = a[i];   //pridruzivanje vrijednosti 3 varijabli x
x = a[i + 1];   //pridruzivanje vrijednosti 7 varijabli x
x = a[i - 1];   //pridruzivanje vrijednosti 5.1 varijabli x
a[5] = i;   //pridruzivanje vrijednosti 1 sestom elementu
a[4] = 111.5;   //pridruzivanje vrijednosti 111.5 petom elementu
a[i + 2] = 8.185; //pridruzivanje vrijednosti 8.185 cetvrtom elementu
a[7 - i] = 99.5;  //pridruzivanje vrijednosti 99.5 sedmom elementu
```

Izlistanje 6.3: Primjeri pristupa elementima 1-D polja

6.2 Ključna riječ `const` s poljima

Pri deklaraciji i inicijalizaciji polja moguće je koristiti ključnu riječ `const`. Uporabom kvalifikatora `const` na način prikazan u nastavku elementi polja postaju konstantni.

```
    const tip_podatka ime_polje[VELICINA_POLJA] = {V1, ..., VN};
```

Važno je napraviti inicijalizaciju, jer naknadne izmjene vrijednosti elemenata nisu moguće. S obzirom na inicijalizaciju i kao što je ranije opisano, veličinu polja je moguće izostaviti te vrijednosti u inicijalizatoru moraju biti konstante, odnosno konstatni izrazi.

6.3 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je potrebno koristiti jednodimenzionalna polja za njihovo rješavanje. U primjerima gdje nije posebno navedeno, veličina polja i tip podatka elemenata odabrani su proizvoljno. [Potsjetnik] Korištene su i pretprocesorske direktive (naredbe) za definiranje simboličnih konstanti (engl. *symbolic constants*): `#define` *makro* tijelo, gdje se svako pojavljivanje *makroa* u tekstu programa zamjenjuje s tijelom. Imenovanje makroa podliježe istim pravilima kao imenovanje varijabli te se tradicionalno koriste velika slova. Nadalje, tijelo može biti brojčana, znakovna ili string konstanta. Prednost uporabe simboličnih konstanti je što kada je potrebno zamijeniti njenu vrijednost, dovoljno je promjenu načiniti na jednom mjestu, a ne svugdje gdje se konstanta koristi u programu.

Primjer 1. *Deklarirati tri različita polja, različitih veličina i tipova podataka. Popuniti sva polja.*

```
int main(void)
{
  int   i;
  int   ip[10]; //deklaracija polja cijelih brojeva velicine 10
  float fp[15]; //deklaracija polja realnih brojeva velicine 15
  char  cp[26]; //deklaracija polja znakova velicine 26

  for (i = 0; i < 10; i++)
    ip[i] = i + 1;       //popunjava brojevima od 1 do 10
  for (i = 0; i < 15; i++)
    fp[i] = (i + 1) / 10.0f; //popunjava brojevima od 0.1 do 1.5
  for (i = 0; i < 26; i++)
    cp[i] = 'A' + i;       //popunjava slovima od A do Z (engl.)

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Uobičajeno se koriste petlje za slijedni pristup elementima polja. Tako se naredba `for` nameće kao najprirodniji izbor, jer objedinjuje postavljanje, provjeru i promjenu vrijednosti cjelobrojne varijable koju se koristi kao indeks polja. Jasno, moguće je koristiti i ostale petlje kako bi se isto postiglo.

Primjer 2. *Deklarirati i inicijalizirati dva različita polja, različitih veličina i tipova podataka te ih ispisati na ekran.*

```
#include <stdio.h>

#define N 5  //pretprocesorske direktive, zamjenjuju
#define M 15 //u tekstu programa N i M s 5 i 15, redom

int main(void)
{
  int i;
  //inicijalizacija polja - navedene su sve vrijednosti elem. polja
  int ip[N] = {2, 4, 6, 8, 10};

  //inicijalizacija polja - nisu navedene sve vrijednosti elem. polja
  //ostatak se automatski postavlja na 0
  float fp[M] = {1.1, 2.2, 3.3};

  for (i = 0; i < N; i++)
    printf("%d ", ip[i]); //ispisuje: 2 4 6 8 10

  for (i = 0; i < M; i++)
    printf("%f ", fp[i]); //ispisuje: 1.1 2.2 3.3 0 0 ... 0

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Ispis elemenata polja treba obaviti element po element. Kao sto je slučaj s varijablama osnovnih tipova podataka treba voditi računa koja se oznaka pretvaranja (engl. *conversion specifier*) koristi u funkciji `printf()` (primjerice, `%d` ili `%i` ako su elementi tipa podatka `int`, ili `%f` ako su elementi tipa podatka `float` ili `double`).

Primjer 3. *Omogućiti korisniku unos vrijednosti elemenata u polje od 20 realnih brojeva. Pronaći i ispisati na ekran najmanju vrijednost.*

```
#include <stdio.h>

int main(void)
{
  int i;
```

```
  float fp[20], min;

  printf("Unesite elemente polja:\n\n");
  for (i = 0; i < 20; i++) {
    printf("Unesite vrijednost %d. elem.: ", i + 1);
    scanf("%f", &fp[i]); //sprema unesenu vrijednost u elem. s indeksom i
  }

  min = fp[0]; //postavlja min na vrijednost prvog elem.
  for (i = 1; i < 20; i++) { //provjerava se sve ostale elemente polja
    if (fp[i] < min) //ako je elem. s indeksom i manji od trenutno najmanjeg
      min = fp[i];   //postavlja se min na tu vrijednost
  }
  //Ispisuje najmanju vrijednost nakon sto je odredena
  printf("Najmanja vrijednost je: %f\n", min);

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Kod traženja najmanje vrijednosti elemenata polja, postavlja se prva za trenutno najmanju, te se potom redom provjerava postoji li još manja vrijednost. Ukoliko se pronađe manja vrijednost, ona se postavlja za trenutno najmanju i nastavlja se prolazak kroz elemente polja. Svaki puta kada se pronađe vrijednost manja od trenutno najmanje ona se postavlja za trenutno najmanju.

Primjer 4. *Omogućiti korisniku unos* $0 < n \leq 50$ *vrijednosti elemenata u polje realnih brojeva. Pronaći i ispisati na ekran najveću vrijednost.*

```
#include <stdio.h>

#define VEL_POLJA 50

int main(void)
{
  int i, n;
  float max, fp[VEL_POLJA]; //zauzima se mem. za najveci dozvoljeni broj elem.

  printf("Unesite zeljeni broj elementa polja:\n\n");
  do {
    scanf("%d", &n);
  } while (n < 1 || n > VEL_POLJA); //trazi se unos zeljeno broja
              //elemenata sve dok se ne zadovolji uvjet

  for (i = 0; i < n; i++) //omogucava se unos n elem.
    scanf("%f", &fp[i]);  //sprema unesenu vrijednost u elem. s indeksom i

  max = fp[0]; //postavlja max na vrijednost prvog elem.
  for (i = 1; i < n; i++) { //provjerava se ostalih n-1 elemenata polja
    if (fp[i] > max) //ako je elem. s indeksom i veci od trenutno najveceg
      max = fp[i];   //postavlja se max na tu vrijednost
  }

  printf("Najveca vrijednost je: %f\n", max);
  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Kako bi se osiguralo da broj vrijednosti elemenata koje korisnik želi unijeti bude unutar postavljenih granica ($0 < n \leq 50$), traži se ponovni unos sve dok taj uvjet nije ispunjen. Unos vrijednosti elemenata od strane korisnika ostvaren je pomoću funkcije `scanf()`, gdje je bitno koristiti adresni operator `&` ispred imena polja. Traženje najveće vrijednosti u polju je ostvareno analogno kao traženje najmanje u primjeru 3.

Primjer 5. *Omogućiti korisniku unos vrijednosti elemenata u polje od 20 realnih brojeva. Sortirati polje od najmanjeg prema najvećem i ispisati ga takvog na ekran.*

```
#include <stdio.h>

#define VEL 20

int main(void)
{
  float a[VEL], priv;
  int i, j;

  for (i = 0; i < VEL; i++)
    scanf("%f", &a[i]);
  //selekcijsko sortiranje
  for (i = 0; i < VEL - 1; i++)   //svaki elem. (osim zadnjeg)
    for (j = i + 1; j < VEL; j++) //usporeduje se sa svim elem. poslije njega
      if (a[i] > a[j]) { //ako je elem. s indeksom i veci od elem. s indeksom j
        priv = a[i];     //zamjenjuju im se vrijednosti
        a[i] = a[j];
        a[j] = priv;
      }

  for (i = 0; i < VEL; i++)
    printf("%f\n", a[i]);

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: U primjeru je korišteno selekcijsko sortiranje (engl. *selection sort*), gdje sve uspoređuje vrijednost svakog elementa sa svakim poslije njega. Vrijednosti elemenata koje se uspoređuju zamjenjuju se ako vrijednost prvog veća ili manja od drugog, ovisno vrši li se sortiranje od najmanjeg prema najvećem ili obratno. Složenost ovog algoritma sortiranja je $O(n^2)$, gdje je n veličina odnosno broj elemenata u polju.

Primjer 6. *Deklarirati i inicijalizirati polje od 10 realnih brojeva. Sortirati polje od najvećeg prema najmanjem i ispisati ga takvog na ekran.*

```
#include <stdio.h>

#define VEL 10

int main(void)
{
  float a[VEL] = {1.2, 2.3, 0.36, -5.543, -2.76, -3.14, 3.1415, -2.718, 5.025, 10.2};
  float priv;
  int i, j, f;

  //mjehuricasto sortiranje
  do {
    f = 0; //ukljanja se signalizacija prije svakog prolaza
    for (i = 0; i < VEL - 1; i++)
      if (a[i] < a[i + 1]) {
        priv = a[i];
        a[i] = a[i + 1];
        a[i + 1] = priv;

        f = 1; //signalizira se da je doslo do zamijene
      }
  } while (f == 1); //ako je bilo zamijene opet se prolazi kroz polje, inace je sortirano

  for (i = 0; i < VEL; i++)
    printf("%f\n", a[i]);

  return 0;
```

```
}
```

Kratko obrazloženje [Primjer 6]: U primjeru je korišteno mjehuričasto sortiranje (engl. *bubble sort*), gdje se redom uspoređuju susjednih elemenata. Prednost ovog algoritma sortiranja u odnosu na algoritam selekcijskog sortiranja je što se može zaustaviti ukoliko nije u prethodnom prolazu bilo zamijene. Složenost ovog algoritma (u najgorem slučaju) je $O(n^2)$.

Primjer 7. *Omogućiti korisniku unos vrijednosti elemenata u polje od 25 realnih brojeva. Potom, omogućiti korisniku unos broja kojeg će se tražiti. Sekvencijalnim/slijednim pretraživanjem provjeriti postoji li traženi broj u polju te ispisati broj izvršenih provjera.*

```
#include <stdio.h>

#define N 25

int main(void)
{
  int a[N], t, p, f, i;

  printf("Unesite %d vrijednosti u polje:\n", N);
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);

  printf("Unesite trazeni broj: ");
  scanf("%d", &t);
  p = f = 0;
  for (i = 0; i < N; i++) {
    p++; //broje se izvrsene provjere
    if (a[i] == t) {
      f = 1; //ako je trazeni broj pronaden
      break; //zavrsava se pretraga
    }
  }

  if (f == 1)
    printf("Trazeni broj, %d, je pronaden nakon %d provjera.\n", t, p);
  else
    printf("Trazeni broj nije pronaden.\n");

  return 0;
}
```

Kratko obrazloženje [Primjer 7]: Kod slijednog pretraživanja (engl. *sequential search*) prolazi se redom kroz elemente polja. Svaki puta se provjera odgovara li vrijednost trenutnog elementa traženoj, te ako je tako pretraga se završava, a u suprotnom nastavlja. Složenost ovog algoritma pretraživanja je $O(n)$.

Primjer 8. *Omogućiti korisniku unos vrijednosti elemenata u polje od 30 realnih brojeva. Zatim, omogućiti korisniku unos broja kojeg će se tražiti. Binarnim pretraživanjem provjeriti postoji li traženi broj u polju te ispisati broj izvršenih provjera.*

```
#include <stdio.h>

#define VEL 30

int main(void)
{
  float a[VEL], priv, t;
  int i, j, dg, gg, s, f, p;

  printf("Unesite 30 vrijednosti u polje:\n");
```

```
  for (i = 0; i < VEL; i++)
    scanf("%f", &a[i]);

  //selekcijsko sortiranje
  for (i = 0; i < VEL - 1; i++)
    for (j = i + 1; j < VEL; j++)
      if (a[i] > a[j]) {
        priv = a[i];
        a[i] = a[j];
        a[j] = priv;
      }

  printf("Unesite trazeni broj: ");
  scanf("%f", &t);

  f = p = 0;
  dg = 0; gg = VEL; //postavljanje granica prostora pretrage
  while (dg <= gg) {
    p++; //broje se provjere
    s = (dg + gg) / 2;
    if (a[s] == t) {
      f = 1; //ako je trazeni broj pronaden
      break; //zavrsava se pretraga
    }
    else if (a[s] < t) //podesavanje granica prostora pretrage
      dg = s + 1;      //promjena donje granice, ako je vrijed. srednjeg elem. manja od trazene
    else gg = s - 1;   //promjena gornje granice, ako je vrijed. srednjeg elem. veca od trazene
  }

  if (f == 1)
    printf("Trazeni broj, %d, je pronaden nakon %d provjera.\n", t, p);
  else
    printf("Trazeni broj nije pronaden.\n");

  return 0;
}
```

Kratko obrazloženje [Primjer 8]: Binarno pretraživanje (engl. *binary search*) zahtjeva da su vrijednosti elemenata polja sortirana (u pravilu od najmanjeg prema najvećem). U svakom koraku prostor pretrage se dijeli na dva i pomiče prema manjim ili većima vrijednostima u polju, ovisno je li vrijednost srednjeg elementa prostora pretrage manja ili veća od tražene. Pretraživanje se prekida ukoliko je pronađena tražena vrijednost. Složenost ovog algoritma pretraživanja je $O(\log_2 n)$.

Primjer 9. *Omogućiti korisniku unos vrijednosti elemenata u polje od 20 realnih brojeva. Nakon toga izračunati standardnu devijaciju tih vrijednosti i ispisati ju na ekran. Standardnu devijaciju računati prema:*

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}\,,$$

gdje $\overline{x}$ predstavlja srednju vrijednost svih n vrijednosti.

```
#include <stdio.h>
#include <math.h> //sqrt()

#define N 20

int main(void)
{
  double a[N];
  int i, n;

  double svr = 0, zbroj = 0;
```

```
  double var, sd;

  printf("Unesite %d vrijednosti u polje:\n", N);
  for (i = 0; i < N; i++)
    scanf("%lf", &a[i]);

  //izracun srednje vrijednosti
  for (i = 0; i < N; i++)
    svr += p[i];
  svr /= n;

  //izracun zbroja kvadrata razlike elemenata od srednje vrijednosti
  for (i = 0; i < N; i++)
    zbroj += (p[i] - svr) * (p[i] - svr);

  var = zbroj / N; //varijanca; kvadrirana standardna devijacija
  sd = sqrt(var);  //standardna devijacija

  printf("Standardna devijacija vrijednosti elemenata polja iznosi: %f\n", sd);

  return 0;
}
```

Kratko obrazloženje [Primjer 9]: Prema formuli x_i odgovara vrijednosti elemenata polja s indeksom $i-1$, jer je u formuli početna vrijednost $i = 1$, a prvi element polja ima vrijednost indeksa 0. U primjeru se koristi funkcija za računanje kvadratnog korijena `sqrt()`, kojoj je potrebno predati vrijednosti, a ona kao rezultat vraća kvadratni korijen iste.

Primjer 10. *Omogućiti korisniku unos vrijednosti elemenata u polje od 20 realnih brojeva. Nakon toga izračunati medijan unesenih vrijednosti. Medijan je vrijednost koja dijeli popis brojeva na gornju i donju polovicu. Podrazumijeva se kako su brojevi sortirani uzlazno (od najmanjeg prema najvećem). Ako je neparan broj brojeva medijan je središnji član, a u slučaju parnog broja medijan je srednja vrijednost dva središnja člana. Primjerice, medijan popisa* $1, 2, 5, 6, 9$ *je 5, dok je medijan popisa* $3, 5, 7, 9, 12, 15$ *jednak* $(7 + 9)/2 = 8$.

```
#include <stdio.h>

#define N 20

int main(void)
{
  double a[N];
  int i, n, f;
  double med;

  printf("Unesite %d vrijednosti u polje:\n", N);
  for (i = 0; i < N; i++)
    scanf("%lf", &a[i]);

  //sortiranje elemenata polja - mjehuricasto sortiranje
  do {
    f = 0;
    for (i = 0; i < N - 1; i++)
      if (a[i] < a[i + 1]) {
        priv = a[i];
        a[i] = a[i + 1];
        a[i + 1] = priv;
        f = 1;
      }
  } while (f == 1);
  //Izracun medijana ovisno o parnosti broja elemenata polja
  if (N % 2 == 1)
    med = a[N / 2]; //ako je neparan broj elemenata polja
  else
    med = (a[N / 2 - 1] + a[N / 2]) / 2; //ako je paran broj elemenata polja
```

```
  printf("Medijan unesenih brojeva je: %f\n", med);
  return 0;
}
```

Kratko obrazloženje [Primjer 10]: S obzirom da nije za očekivati kako je korisnik unio brojeve ulaznim redoslijedom, nužno je elemente polja prvo sortirati što je napravljeno mjehuričastim sortiranjem. Nakon toga, jednostavno je odrediti, prema navedenom opisu, medijan tih brojeva. Važno je samo voditi računa je li broj elemenata polja paran ili neparan.

POGLAVLJE 7

Polja: dvodimenzionalna polja

U programskom jeziku C moguće je osim jednodimenzionalnih (1-D) polja deklarirati i polja više dimenzija. Tako se primjerice, osim 1-D polja mogu naći u primijeni i dvodimenzionalna (2-D) polja. Polja više od dvije dimenzije vrlo se rijetko koriste.

7.1 Dvodimenzionalna polja

Dvodimenzionalna su polja čiji je svaki element 1-D polje. Stoga, može se reći kako 2-D polje predstavlja polje polja. Svaki element (polje) istog je tipa podatka i iste veličine. Na slici 7.1 prikazan je primjer 2-D polja. Dvodimenzionalna polja često se prikazuje pravokutnom tablicom (matricom, ako se radi o brojevima), iako se elementi 2-D kao i kod 1-D polja pohranjuju slijedno u memoriji. U takvom slučaju svaki redak predstavlja jedan element 2-D polja, odnosno polje dane veličine (Sl. 7.2).

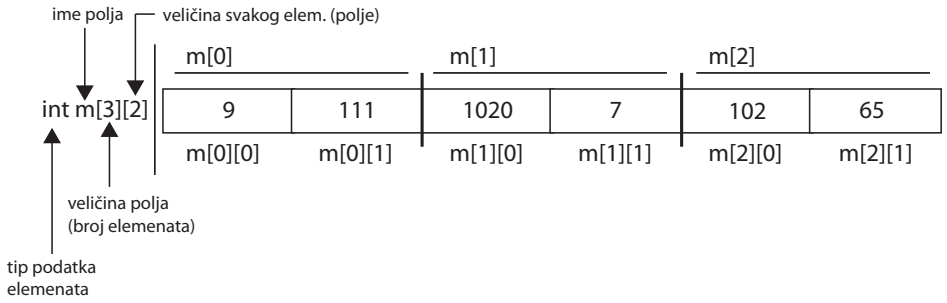
7.1.1 Deklaracija i inicijalizacija

Prije uporabe nužno je deklarirati 2-D polje. Dvodimenzionalna polja se deklariraju na sličan način kao i jednodimenzionalna polja, a deklaracija određuje ime, tip podatka, broj elemenata (veličinu) kao i veličinu svakog elementa koji zapravo predstavlja 1-D polje. Veličina 2-D polja kao i veličina elemenata mora biti konstantan cjelobrojni izraz veći od nule, a navode se unutar para uglatih zagrada, redom. U izlistanju 7.1 dano je nekoliko primjera deklaracije 2-D polja.

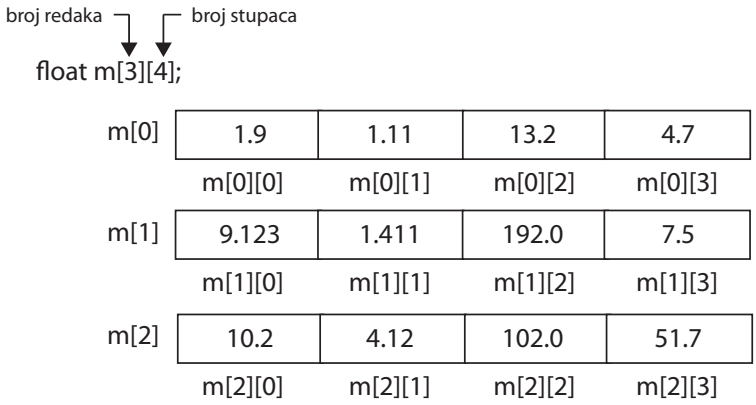
```
int polje_2d[2][2]; //deklaracija 2-D polja od 2 elementa
                    //gdje je svaki polje od 2 elementa tipa podatka int
float a[2][3];      //deklaracija 2-D polja od 2 elementa
                    //gdje je svaki polje od 3 elementa tipa podatka float
double m[8][4];     //deklaracija 2-D polja od 8 elementa
                    //gdje je svaki polje od 4 elementa tipa podatka double
char c2D[5][4 * 5]; //deklaracija 2-D polja od 5 elementa
                    //gdje je svaki polje od 20 elemenata tipa podatka char
long lp[2 + 6][13 - 1]; //deklaracija 2-D polja od 8 elementa
                        //gdje je svaki polje od 12 elemenata tipa podatka long int
```

Izlistanje 7.1: Primjeri deklaracije 2-D polja

Inicijalizacija 2-D polja je slična inicijalizaciji 1-D polja pa se tako unutar vitičastih zagrada navode pojedini elementi odvojeni zarezom – inicijalizator. Kako je svaki element



Slika 7.1: Dvodimenzionalno polje cijelih brojeva. Predstavlja polje od 3 elementa gdje je svaki polje od 2 podatka tipa `int`.



Slika 7.2: Tablični prikaz dvodimenzionalnog polja.

2-D polja i sam polje, tako se svaka vrijednost (konstantni izraz) elementa tog polja opet navode unutar vitičastih zagrada. Međutim, moguće je izostaviti vitičaste zagrade za pojedine elemente 2-D polja i navesti sve vrijednosti unutar jednog para zagrada (kao i kod 1-D polja). U tom slučaju slijedno se dodjeljuju vrijednosti. Prilikom inicijalizacije može se izostaviti velilčina 2-D polja, međutim, veličinu elemenata nužno je navesti. Ako se izostavi veličina polja, prevoditelj će sam odrediti istu na temelju inicijalizatora. Kao i kod 1-D polja moguća je samo djelomična inicijalizacija. U tom slučaju svi neinicijalizirani elementi postavljaju se na vrijednost nula (0). Izlistanjem 7.2 je navedeno nekoliko primjera deklaracije i popratne inicijalizacije 2-D polja.

```
int p2d[2][2] = { {1, 2}, {3, 4} };  //potpuna inicijalizacija
char c2d[2][3] = { {'a', 'b', 'c'}, {'e', 't', 'f'} }; //potpuna inicijalizacija
char c[3][6] = { "abcde", "etfos", "fghij" };          //potpuna inicijalizacija ->
                                                       //svaki element 2-D polja
                                                       //predstavlja string
float M[][2] = { {1.2, 1.3}, {2.3, 2.4}, {3.4, 3.5}, {4.5, 4.6} }; //potpuna inicijalizacija
                                                                   //2-D polja velicine 4
long A[2][4] = {1, 2, 3, 4, 5, 6, 7, 8};          //potpuna inicijalizacija
short B[][5] = {1, 2, 3, 4, 5, 6, 7};             //djelomicna inicijalizacija
                                                  //2-D polja velicine 2
double md[2][3] = { {1, 2} };                     //djelomicna inicijalizacija
float mf[2][4] = { {1, 2, 3}, {4} };              //djelomicna inicijalizacija
```

Izlistanje 7.2: Primjeri inicijalizacije 2-D polja

7.1.2 Pristup elementima

Pojedinim elementima 2-D polja pristupa se preko imena tog polja i dva indeksa, svaki unutar para uglatih zagrada (Sl. 7.1), odnosno putem dvostruke primjene operatora indeksiranja. Prvi indeks određuje element 2-D polja, dok drugi indeks određuje element unutar tog elementa (1-D polja). Ako se uzme u obzir tablični prikaz, prvim indeksom obilazi se po redcima, dok se drugim obilazi po stupcima (Sl. 7.2). Indeksi, kao i u slučaju 1-D polja, moraju predstavljati cjelobrojne izraze. Također, od izuzetne je važnosti voditi brigu o njihovim vrijednosti kako se ne bi napustile granice polja. Izlistanjem 7.3 je prikazano nekoliko primjera pristupa elementima dvodimenzionalnog polja.

```
double md[2][3] = { {1, 2, 3}, {4, 5, 6} };
double y;
int i = 1, j = 2;
y = md[0][1];   //pridruzivanje vrijednosti 2 varijabli y
y = md[i][j];   //pridruzivanje vrijednosti 6 varijabli y
y = md[1][i];   //pridruzivanje vrijednosti 5 varijabli y
y = md[j - 1][j - 2]; //pridruzivanje vrijednosti 4 varijabli y
y = md[i - 1][0]; //pridruzivanje vrijednosti 1 varijabli y
y = md[i - 1][j]; //pridruzivanje vrijednosti 3 varijabli y
md[1][1] = 5.6;   //pridruzivanje vrijednosti 5.6 drugom elementu u drugom retku
md[0][1 + 1] = 8.575; //pridruzivanje vrijednosti 8.575 trecem elementu u prvom retku
md[1][i + 1] = 7.5;   //pridruzivanje vrijednosti 7.5 trecem elementu u drugom retku
md[i - 1][j] = 10.125;  //pridruzivanje vrijednosti 10.125 trecem elementu u prvom retku
md[j - 2][i - 1] = 0;   //pridruzivanje vrijednosti 0 prvom elementu u prvom retku
```

Izlistanje 7.3: Primjeri pristupa elementima 2-D polja

7.2 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je potrebno koristiti 2-D polja. U nekoliko primjera umjesto 2-D polja navodi se *matrica* određenih dimenzija pa se tako, primjerice, pod matricom dimenzija $m \times n$ smatra 2-D polje veličine m čiji je svaki element veličine n. Također, u nekoliko primjera koristi se funkcija `putchar()` koja ispisuje predani joj znak na ekran, a njen prototip je: `int putchar(int znak);`

Primjer 1. *Deklarirati dva 2-D polja različitih veličina i tipova podataka. Oba 2-D polja popuniti vrijednostima i potom ih ispisati na ekran.*

```
#include <stdio.h>

int main(void)
{
  int i, j;
  //deklaracije 2-D polja
  int im[6][5];   //6 elem., svaki je polje velicine 5
  float fm[4][4]; //4 elem., svaki je polje velicine 4

  //popunjavanje 2-D im
  for (i = 0; i < 6; i++) //obilazak po redcima
    for (j = 0; j < 5; j++) //obilazak po stupcima
      im[i][j] = (i * 4) + j + 1; //popunjava brojevima od 1 do 30

  //popunjavanje 2-D fm
  for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
      fm[i][j] = ((i * 4) + j + 1) / 10.0; //popunjava brojevima od 0.1 do 1.6

  //ispis 2-D polja im
  //ispisuje redak po redak
  for (i = 0; i < 6; i++) {    //obilazak po redcima
    for (j = 0; j < 5; j++) {  //obilazak po stupcima
      printf("%d ", im[i][j]); //ispis elem. s indeksima i,j
    }
    putchar('\n'); //prelazak u novi redak
  }

  putchar('\n'); //prelazak u novi redak zbog preglednije ispisa
  //ispis 2-D polja fm
  //ispisuje redak po redak
  for (i = 0; i < 4; i++) {    //obilazak po redcima
    for (j = 0; j < 4; j++) {  //obilazak po stupcima
      printf("%f ", fm[i][j]); //ispis elem. s indeksima i,j
    }
    putchar('\n'); //prelazak u novi redak
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Kod pristupa elementima 2-D polja uglavnom se uzima apstrakcija tabličnog prikaza te se zbog toga, uobičajeno, koriste dvije `for` petlje. Vanjska petlja koristi se za obilazak po redcima, dok se unutarnja koristi za obilazak po stupcima. To znači da se vrijednost brojača vanjske petlje koristi kao prvi indeks, a vrijednost brojača unutarnje kao drugi indeks.

Primjer 2. *Deklarirati i inicijalizirati 2-D polje realnih brojeva te izračunati i na ekran ispisati srednju vrijednost svih elemenata.*

```
#include <stdio.h>

int main(void)
{
  int i, j;
  double srv, zbroj = 0;
  //deklaracije i inicijalizacija 2-D polja realnih brojeva
  double m[3][4] = {{1.2, 2.3, 3.4, 4.5},        //prvi redak
                    {0.2, 0.4, 0.6, 0.8},        //drugi redak
                    {12.12, 23.23, 34.34, 45.45} //treci redak
                   };

  for (i = 0; i < 3; i++)
    for (j = 0; j < 4; j++)
```

```
      zbroj += m[i][j]; //zbrajanje svih elemenata, redak po redak

  srv = zbroj / (3 * 4); //racunaje srednje vrijednosti - ukupan broj elem. je 12
  printf("Srednja vrijednost svih elemenata: %f\n", srv);

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Veličina 2-D polja kao i veličina elemenata su proizvoljno odabrane, jer nisu u primjeru definirane. Kod inicijalizacije 2-D polja korištene su proizvoljne vrijednosti s obzirom da nisu navedene u primjeru.

Primjer 3. *Omogućiti korisniku unos dimenzija matrice realnih brojeva $m \times n$ ($0 < m < 11$ i $0 < n \leq 5$) i njeno popunjavanje. Izračunati i ispisati na ekran zbroj elemenata svakog stupca zasebno.*

```
#include <stdio.h>

#define REDCI 10
#define STUPCI 5

int main(void)
{
  int   i, j, m, n;  //m - broj redaka; n - broj stupaca
  float z, a[REDCI][STUPCI]; //dimenzije matrice najvece su dopustene prema primjeru

  do {
    scanf("%d", &m); //unos zeljenog broja redaka
  } while (m <= 0 || m > REDCI); //mora biti unutar zadanih granica

  do {
    scanf("%d", &n); //unos zeljenog broja stupaca
  } while (n <= 0 || n > STUPCI); //mora biti unutar zadanih granica

  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
      scanf("%f", &a[i][j]); //sprema unesenu vrijednost u elem. s indeksima i,j

  for (i = 0; i < n; i++) { //obilazak po stupcima
    z = 0; //postavljanje na 0 prije racunanja zbroja svakog stupca
    for (j = 0; j < m; j++) //obilazak po redcima
      z += a[j][i];

    printf("Zbroj %d. stupca je %f\n", i + 1, z);
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Kako bi se obavio obilazak redom po stupcima, prvi indeks je vrijednost brojača unutarnje petlje (`j` u ovom slučaju), dok je drugi indeks vrijednost brojača vanjske (`i` u ovom slučaju). Sukladno tomu, vanjska petlja koristi se za obilazak po stupcima, a unutarnja za obilazak po redcima.

Primjer 4. *Omogućiti korisniku unos dimenzija matrice realnih brojeva $m \times n$ ($2 \leq m < 16$ i $3 < n \leq 8$) i njeno popunjavanje. Izračunati i ispisati na ekran produkt elemenata svakog retka zasebno.*

```
#include <stdio.h>

#define M 15
#define N 8
```

```
int main(void)
{
  int   i, j, m, n;  //m - broj redaka; n - broj stupaca
  float p, a[M][N]; //dimenzije matrice najvece su dopustene prema primjeru

  do {
    scanf("%d", &m); //unos zeljenog broja redaka
  } while (m < 2 || m > M); //mora biti unutar zadanih granica

  do {
    scanf("%d", &n); //unos zeljenog broja stupaca
  } while (n < 4 || n > N); //mora biti unutar zadanih granica


  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
      scanf("%f", &a[i][j]); //sprema unesenu vrijednost u elem. s indeksima i,j

  for (i = 0; i < m; i++) { //obilazak po redcima
    p = 1; //postavljanje na 1 prije racunanja produkta svakog retka
    for (j = 0; j < n; j++) //obilazak po stupcima
      p *= a[i][j];

    printf("Produkt %d. retka je %f\n", i + 1, p);
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Rješenje je vrlo slično kao za primjer 3.

Primjer 5. *Omogućiti korisniku unos elemenata kvadratne matrice dimenzija* 4×4*. Izračunati i na ekran ispisati zbroj elemenata glavne dijagonale i produkt elemenata sporedne dijagonale.*

```
#include <stdio.h>

#define VEL 4

int main(void)
{
  int   i;
  float zgd; //zbroj elemenata glavne dijagonale
  float psd; //produkt elemenata sporedne dijagonale
  float a[VEL][VEL];

  for (i = 0; i < VEL; i++)
    for (j = 0; j < VEL; j++)
      scanf("%f", &a[i][j]);

  zgd = 0;
  for (i = 0; i < VEL; i++)
    zgd += a[i][i]; //elementi glavne dijagonale

  psd = 1;
  for (i = 0; i < VEL; i++)
    psd *= a[i][VEL - 1 - i]; //elementi sporedne dijagonale

  printf("Zbroj elem. gl. dijag. je %f, a produkt sporedne dijag. %f\n", zgd, psd);

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Općenito, elementima 2-D polja `M`, velčine n čiji je svaki element također veličine n (kvadratna matrica), koji predstavljaju glavnu dijagonalu pristupa se na sljedeći način: `M[i][i]` za `i = 0, ..., n-1`. Elementima koji predstavljaju sporednu dijagonalu pristupa se na način: `M[i][n-1-i]` za `i = 0, ..., n-1`.

Primjer 6. *Omogućiti korisniku unos dimenzija matrica realnih brojeva,* $\mathbf{A}_{m\times n}$ *i* $\mathbf{B}_{n\times p}$*,* $(0 < m < 11,\ 0 < n \leq 10$ *i* $0 < p < 11)$ *i njihovo popunjavanje. Izračunati i ispisati na ekran produkt matrica* **A** *i* **B***.*

```
#include <stdio.h>

#define VEL 10

int main(void)
{
  int   i, j, k, m, n, p;
  float A[VEL][VEL], B[VEL][VEL], C[VEL][VEL];

  do {
    scanf("%d", &m);
  } while (m < 1 || m > VEL);
  do {
    scanf("%d", &n);
  } while (n < 1 || n > VEL);
  do {
    scanf("%d", &p);
  } while (p < 1 || p > VEL);

  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
      scanf("%f", &A[i][j]);

  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
      scanf("%f", &B[i][j]);

  //racunanje produkta matrica A i B -> rezultirajuca matrica C je dimenzija m * p
  for (i = 0; i < m; i++) {     //obilazak po redcima matrica A i C
    for (j = 0; j < p; j++) {   //obilazak po stupcima matrica B i C
      C[i][j] = 0; //postavlja C[i][j] na 0 prije racunanja vrijednosti
      for (k = 0; k < n; k++) { //obilazak po stupcima matrice A i redcima matrice B
        C[i][j] += A[i][k] * B[k][j];
      }
      printf("%f ", C[i][j]);
    }
    putchar('\n');
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Kako bi se mogle pomnožiti dvije matrice **A** i **B**, matrica **B** mora imati broj redaka jednak broju stupaca matrice **A**. Zbog toga je korisniku omogućen unos željenih dimenzija prve matrice, ali samo unos željenog broja stupaca za drugu matricu. [Produkt matrica **A** i **B**: $(\mathbf{AB})_{i,j} = \sum_{k=1}^{n} (\mathbf{A})_{i,k} \cdot (\mathbf{B})_{k,j}$]

POGLAVLJE 8

Polja: polja znakova i stringovi

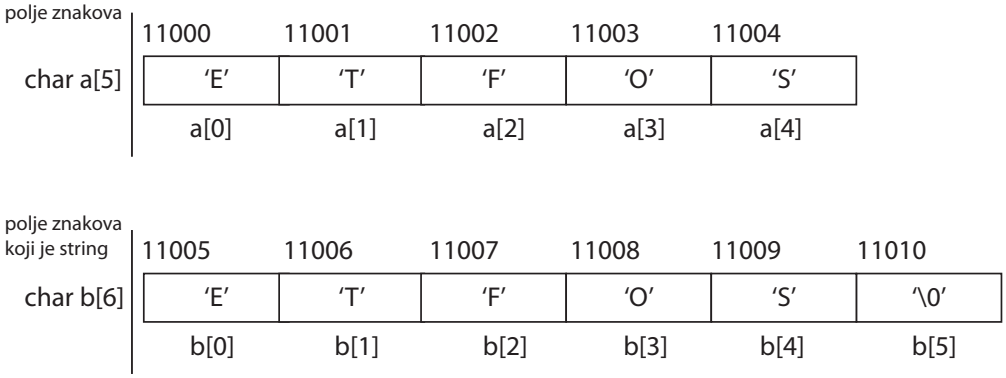
Posebnu ulogu u jezku igraju polja znakova koji predstavljaju stringove. Oni omogućuju jednostavno rukovanje nizovima znakova koji su prikladni za održavanje raznih podataka kao primjerice, poruka, imena, adresa, telefonskih brojeva i sličnog. S obzirom na njihovu široku uporabu, postoje razne funkcije za ulaz i izlaz (U/I) koje rade isključivo sa stringovima i funkcije koje omogućuju razne radnje nad njima.

8.1 Stringovi i njihova uporaba

Stringovi ili znakovni stringovi (engl. *character strings*) su (jednodimenzionalna) polja znakova (tip podataka elemenata je `char`) koji završavaju sa ili sadrže tzv. *null* znak (`\0`). Na slici 8.1, vidljiva je razlika između polja znakova i polja znakova koje je string. Navedeni *null* znak ne mora nužno biti vrijednost zadnjeg elementa polja. On se postavlja neposredno nakon zadnjeg "valjanog" znaka. Primjerice, ako je polje znakova veličine 20, a u njega se spremi riječ od 5 slova, *null* znak će biti šesti znak (element polja s indeksom 5). Kaže se kako je duljina tog stringa 5 znakova (*null* znak se ne broji). Prema opisanom, sadržaj stringa predstavljaju samo znakovi do *null* znaka bez obzira na veličinu polja.

8.1.1 Deklaracija i inicijalizacija

S obzirom da su stringovi jednodimenzionalna (1-D) polja znakova, njihova deklaracija odgovara deklaraciji 1-D polja čiji su elementi tipa podataka `char`. Prilikom deklaracije



Slika 8.1: Polje znakova i polje znakova koje je string.

treba voditi računa kako je potrebno dodatno mjesto za *null* znak pa shodno tome treba postaviti veličinu polja.

Stringove je moguće inicijalizirati uobičajenim načinom inicijalizacije polja—preko inicijalizatora. Međutim, tada treba dodati *null* znak poslije zadnjeg "valjanog" znaka kako bi polje bilo string. Premda to nije obavezno ako je zadana odgovarajuća veličina polja (barem za jedan veća od broja znakova). Naime, svi neinicijalizirani elementi biti automatski postavljeni na vrijednost 0 koja i odgovara *null* znaku prema ASCII tablici, ipak je dobro navesti ga. Nadalje, inicijalizaciju je moguće obaviti navođenjem niza znakova unutar para dvostrukih navodnika. Niti prilikom ovakve incijalizacije nije obavezno navođenje veličine polja. Prevoditelj će u tom slučaju sam odrediti veličinu koja će odgovarati broju znakova uz dodani *null* znak. U izlistanju 8.1 je navedeno nekoliko primjera inicijalizacije stringova.

```
char fakultet[6] = {'E', 'T', 'F', 'O', 'S', '\0'};
char poruka[11] = "dobar dan!";
char s[] = {'P', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'r', 'a', 'n', 'j', 'e', ' ', '1', '\0'};
char string[] = "Laboratorijske vjezbe iz Programiranja 1";
//svi neinicijalizirani elementi se postavljaju na vrijednost 0
char t[20] = {'E', 'T', 'F'};
//nije string, jer nema null znaka na kraju;
//zauzima se mjesta za tocno onoliko koliko ima znakova u inicijalizatoru
char r[] = {'N', 'i', 'j', 'e', '_', 's', 't', 'r', 'i', 'n', 'g'};
```

Izlistanje 8.1: Primjeri inicijalizacije stringova

Niz znakova između para dvostrukih navodnika (primjerice, `"Niz znakova."`) predstavlja string literal (engl. *string literal*) ili string konstantu koji ima *null* znak na kraju (*null* znak je automatski dan od strane prevoditelja). Navedeni je moguće samo pridružiti polju znakova u sklopu inicijalizacije. Naknadni pokušaj pridruživanja rezultirat će greškom prilikom prevođenja.

8.1.2 Pristup elementima

S obzirom da su stringovi samo polja znakova, pristup njenim elementima isti je kao pristup elementima bilo kojeg 1-D polja. Prema tome, pristup se obavlja preko imena polja i indeksnog operatora uz navođenje indeksa. Stoga, prvi element ima vrijednost indeksa nula (0), a zadnji $n - 1$, gdje je n duljina stringa. Treba primijetiti, da ne uključuje *null* znak te da eventualni sadržaj poslije njega nije relevantan. Nekoliko primjera pristupa elementima polju znakova koje je string prikazano je izlistanjem 8.2.

```
char faks[6] = {'E', 'T', 'F', 'O', 'S', '\0'};
char z;
int i = 0;
z = faks[0];     //dodjela znaka E varijabli z
z = faks[i + 3]; //dodjela znaka O varijabli z
z = faks[3 + 1]; //dodjela znaka S varijabli z
faks[2] = 'f';   //dodjela znaka f trecom elementu
faks[i] = 'e';   //dodjela znaka e prvom elementu
faks[4] = 115;   //dodjela znaka s (prema ASCII tablici) petom elementu
```

Izlistanje 8.2: Primjeri pristupa elementima stringa

8.2 Neke funkcije za rukovanje stringovima

Kao što je na početku spomenuto, postoje mnoge različite funkcije koje je moguće koristiti isključivo sa stringovima i koje olakšavaju rukovanje njima. Funkcije pruža standardna biblioteka. Tako primjerice, postoji nekoliko prilagođenih funkcija za U/I (opisanih u zaglavnoj datoteci stdio.h) koje omogućuju jednostavno učitavanje i ispis stringova. Štoviše, zaglavna datoteka string.h opisuje mnoge druge funkcije za obavljanje različitih radnji nad stringovima kao što su kopiranje, spajanje, uspoređivanje i slično.

8.2.1 Učitavanje i ispis stringova

Iako se radi o poljima, stringove nije nužno učitavati/ispisivati znak po znak (iako je i to moguće). Dovoljan je jedan poziv funkcije. Ne pružaju sve funkcije za U/I iste mogućnosti i ne ponašaju se jednako. To se posebno odnosi na funkcije za učitavanje s ulaza. Primjerice, neke funkcije omogućuju učitavanje stringa koji sadrži prazine, dok neke druge ne omogućuju. Slično, nekima je moguće zadati broj znakova koje će biti spremljen u polje, što je važno kako ne bi došlo do pokušaja spremanja većeg broja znakova nego je veličina polja.

Funkcije `scanf()` i `printf()`

Za učivanje i ispis stringova moguće je koristiti funkcije `scanf()` i `printf()` s oznakom pretvorbe `%s`. Dovoljan je jedan poziv navedenih funkcija za učitavanje, odnosno ispisivanje stringa. U slučaju funkcije `scanf()` svi znakovi [izuzev početnih praznina (engl. *whitespace*)] do pojave prve praznine se spremaju u polje, a funkcija dodaje i *null* znak na kraj. Stoga, ona je pogodna kada se želi učitati niz znakova koji ne sadrži praznine. Navođenjem širine polja kao dodatka oznaci pretvorbe (primjerice, `%20s`) moguće je ograničiti broj znakova koji će biti spremljeni u polje. Funkcija `printf()` ispisuje sve znakove u stringu do pojave *null* znaka. Kao što je slučaj s drugim oznakama pretvorbe i `%s` se može nalaziti u sklopu složenijeg string formata i ne mora biti samostalan. Primjer u izlistanju 8.3 ilustrira uporabu spomenutih funkcija za učitavanje/ispisivanje strinogva.

```
char str[20];

scanf("%s", str); //ne ide operator & ispred imena polja
printf("Unesen je string: %s\n", str);
```

Izlistanje 8.3: Primjeri uporabe funkcija `scanf()` i `printf()`

Funkcije `gets()` i `puts()`

Jednostavno učitavanje odnosno ispisivanje stringova pružaju funkcije `gets()` i `puts()`. Obje samo traže da im se da polje kao argument. U slučaju funkcije `gets()` svi se znakovi do pojave znaka za novi red (`\n`, koji se generira pritiskom tipke ENTER na tipkovnici) spremaju u polje, a na kraj se dodaje *null* znak. Ona ne pruža mogućnost ograničavanja najvećeg broja znakova kojeg će se pokušati spremiti u polje, što može predstavljati problem. Funkcija `puts()` ispisuje string i sama ispisuje dodatni znak za prelazak u novi red. Izlistanjem 8.4 dan je primjer uporabe opisanih funkcija.

```
char str[20];

gets(str);
puts(str);
```

Izlistanje 8.4: Primjeri uporabe funkcija `gets()` i `puts()`

Funkcije `fgets()` i `fputs()`

Funkcije `fgets()` i `fputs()` značajno se razlikuju od prethodno opisanih `gets()` i `puts()`. One nisu ograničene na standardni ulaz i izlaz, nego mogu čitati iz i pisati u druge tokove, odnosno datoteke. Stoga, između ostalog, nužno im je zadati tok iz kojeg će čitati, odnosno u koji će pisati. Postoje tri standardna toka: `stdin` je standardni ulaz (u pravilu, tipkovnica), `stdout` je standardni izlaz (u pravilu, ekran) i `stderr` je standardni izlaz za greške (najčešće ekran). Navedeni su deklarirani u zaglavnoj datoteci stdio.h i bivaju automatski otvoreni prilikom pokretanja programa.

Funkcija `fgets()` omogućuje učitavanje niza znakova (`stdin` za učitavanje s tipkovnice odnosno standardnog ulaza), ali uz kontrolu najvećeg broja znakova koji će biti spremljen u polje. Stoga je ona sigurnija za uporabu u odnosu na funkciju `gets()` koju se preporučuje izbjegavti. Funkcija učitava sve znakove do pojave znaka za novi red kojeg također sprema u polje ili do $n - 1$ znakova, gdje je n maksimalan broj znakova (uključujući *null* znak), a na kraj dodaje *null* znak. Funkcija `fputs()` omogućuje ispis stringa (`stdout` za ispis na standardni izlaz), ali za razliku od `puts()` ne dodaje automatski znak za prelazak u novi red. Primjer uporabe opisanih funkcija je dan izlistanjem 8.5.

```
char str[20];

fgets(str, 20, stdin);
puts(str, stdout);
```

Izlistanje 8.5: Primjeri uporabe funkcija `fgets()` i `fputs()`

8.2.2 Duljina stringa, kopiranje i usporedba

Osim različitih funkcija za potrebe U/I, postoje mnoge za obavljanje različitih radnji nad stringovima. Zaglavna datoteka string.h opisuje mnoge od njih.

Funkcije `strlen()`, `strcpy()` i `strcmp()`

Nekada je potrebno znati duljinu stringa pa je za to moguće iskoristiti funkciju `strlen()`. Njoj se predaje string čiju se duljinu želi znati, a vraća broj znakova do pojave *null* znaka. Uporaba te funkcije i razlike u odnosu na uporabu operatora `sizeof` prikazana je primjerom u izlistanju 8.6.

```
char str[] = "FERIT";
int d;

d = strlen(str); //funkcija vraca vrijednost 5
d = sizeof str;  //operator daje vrijednost 6 (5 znakova + null znak = velicina polja)
```

Izlistanje 8.6: Primjeri uporabe funkcija `strlen()`

Funkcija `strcpy()` omogućuje kopiranje sadržaja jednog stringa u drugi. Ona prima dva argumenta, prvi je odredišno polje, dok je drugi izvorno polje. Izvorno polje može biti i string literal, ali on ne može biti odredišno polje. Važno je voditi brigu o tome da je odredišno polje dovoljno veliko, odnosno da u njega stanu svi znakovi izvornog polja, uključujući *null* znak. Primjer uporabe navedene funkcije dan je u izlistanju 8.7.

```
char o_str[20] = "Odredisno polje" ;
char i_str[] = "Izvorno polje...";

strcpy(o_str, i_str);
strcpy(o_str, "novi izvorni string");
```

Izlistanje 8.7: Primjeri uporabe funkcija `strcpy()`

Dva stringa je moguće usporediti pomoću funkcije `strcmp()`. Funkcija prima dva stringa kao argumente. Funkcija uspoređuje stringove znak po znak do pojave prvog *null* znaka ili do para znakova koji se ne poklapaju (nisu isti u oba stringa). Ako dođe do para znakova koji se ne poklapaju i znak u prvom stringu je po ASCII vrijednosti prije znaka u drugom, funkcija vraća negativnu vrijednost. U obratnom slučaju ne poklapanja, funkcija vraća pozitivnu vrijednost. Samo u slučaju da su oba stringa jednaka, funkcija vraća vrijednost nula. Primjer u izlistanju 8.8 ilustrira uporabu opisane funkcije.

```
char str1[] = "abc" ;
char str2[] = "abd";
char str3[] = "abb";
char str4[] = "abc";
char str5[] = "Abc";

int r;

r = strcmp(str1, str2); //vraca negativnu vrijednost
r = strcmp(str1, str3); //vraca pozitivnu vrijednost
r = strcmp(str1, str4); //vraca nulu
r = strcmp(str1, str5); //vraca pozitivnu vrijednost, jer je, prema ASCII, 'A' prije 'a'
```

Izlistanje 8.8: Primjeri uporabe funkcija `strcmp()`

8.3 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima potrebno koristiti stringove za njihovo rješavanje. U primjerima je prikazana, između ostalog, uporaba prethodno opisanih funkcija za učitavanje i ispisivanje stringova. Uz to su prikazane dvije jednostavne funkcije za učitavanje i ispis pojedničanih znakova, odnosno funkcije `getchar()` i `putchar()`.

Primjer 1. *Deklarirati i inicijalizirati polje znakova koje je string. Potom ga ispisati* 5 *puta na ekran.*

```
#include <stdio.h>

int main(void)
{
  char s[] = "Dobar dan!\n"; //inicijalizacija; null znak se dodaje automatski
  int i;

  for (i = 1; i <= 5; i++)
    printf("%s", s); //%s za ispis stringa

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Stringove je moguće jednostavno ispisati na ekran, što znači da ih nije potrebno ispisivati element po element. To se pomoću funkcije `printf()` postiže oznakom pretvorbe `%s`.

Primjer 2. *Deklarirati i inicijalizirati polje znakova koje je string. Potom ispisati svaki drugi znak na ekran.*

```
#include <stdio.h>

int main(void)
{
  char string[] = "Ovo je inicijalizacija stringa";
  int i;

  for (i = 0; string[i] != '\0'; i++)
    if ((i + 1) % 2 == 0)
      putchar(string[i]); //ispis znaka; umjesto: printf("%c", string[i]);

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Funkcija `putchar()` služi za ispisivanje jednog znaka pa ju je moguće koristiti umjesto funkcije `printf()` (uz oznaku pretvorbe `%c`) kada nije potrebno dodatno formatiranje ispisa. Njoj se predaje samo znak kojeg se želi ispisati. S obzirom da stringovi završavaju *null* znakom tu činjenicu moguće je iskoristiti kako se odredilo gdje je zadnji znak u polju, što je i učinjeno u naredbi `for`.

Primjer 3. *Tražiti od korisnika unos jedne riječi od maksimalno 20 znakova. Izbrojiti i na ekran ispisati koliko se puta u toj riječi nalazi znak* `a` *ili* `A`*.*

```
#include <stdio.h>

int main(void)
{
  char s[21]; //dodatno mjesto za null znak
  int i, c = 0;

  scanf("%20s", s); //unos rijeci; spremaju se znakovi do prve praznine
          //ili maksimalno 20 znakova te na kraj,
                    //poslije zadnjeg znaka, dodaje se null znak
  for (i = 0; s[i] != '\0'; i++) //provjeravaju se svi znakovi do kraja stringa
    if (s[i] == 'a' || s[i] == 'A') //ako je znak u polju s indeksom i jednak znaku a ili A
      c++;              //uvecava se brojac

  printf("Slovo a se pojavljuje %d puta.", c);
  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Za učitavanje stringa sa standardnog ulaza (u pravilu tipkovnice) moguće je koristiti funkciju `scanf()` uz oznaku pretvorbe `%s`.

Primjer 4. *Tražiti od korisnika unos jedne rečenice u polje znakova veličine 101. Izbrojiti i na ekran ispisati koliko se u toj rečenici nalazi samoglasnika.*

```
#include <stdio.h>

int main(void)
{
  char s[101]; //maksimalno stane 100 znakova jer je potrebno mjesto za null znak
  int i, c = 0;

  gets(s); //unos recenice; spremaju se znakovi do pojave znaka za novi red
  for (i = 0; s[i] != '\0'; i++) {
    //provjera samo malih slova, lako se moze prosiriti i za velika slova
    if (s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u')
      c++;
  }

  printf("Ima %d samoglasnika u recenici.\n", c);

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Uporabom funkcije `gets()` moguće je učitati niz znakova odnosno stringa koji sadrži praznine, što je bitno, jer se traži da korisnik ima mogućnost unosa rečenice.

Primjer 5. *Deklarirati i inicijalizirati string proizvoljne veličine. Brojanjem znakova odrediti njegovu duljinu.*

```
#include <stdio.h>

int main(void)
{
  char s[] = "Primjer inicijalizacije stringa"; //null znak se dodaje automatski
  int c;

  c = 0;
  while (s[c] != '\0') //broje se svi znakovi od pojave null znaka
    c++;

  printf("Ima %d znakova u stringu.\n", c);

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Duljina stringa je broj znakova do pojave *null* znaka. Brojanjem svih znakova do *null* znaka dobiva se duljina stringa. Duljina stringa i veličina polja ne moraju se podudarati uz brojanje i *null* znaka, jer se on može nalaziti bilo gdje u polju.

Primjer 6. *Deklarirati i inicijalizirati string proizvoljne veličine. Zamijeniti svaki samoglasnik znakom za podcrtavanje te ga potom ispisati na ekran.*

```
#include <stdio.h>

int main(void)
{
  char s[] = "Neki string proizvoljnje velicine...";
```

```
  int i;

  for (i = 0; s[i] != '\0'; i++)
    switch (s[i]) {
      case 'a':  case 'A':
      case 'e':  case 'E':
      case 'i':  case 'I':
      case 'o':  case 'O':
      case 'u':  case 'U':
      s[i] = '_'; break;
    }

  puts(s); //ispis stringa; umjesto: printf("%s\n", s);

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Može se koristiti `puts()` umjesto funkcije `printf()` kada nije potrebno dodatno formatiranje ispisa. Njoj se predaje samo polje znakova koje je string.

Primjer 7. *Omogućiti korisniku unos jednog stringa od maksimalno 80 znakova. Potom, dodatno mu omogućiti unos jednog znaka. Provjeriti i na ekran ispisati nalazi li se taj znak u stringu.*

```
#include <stdio.h>

int main(void)
{
  char s[81];
  char tz;
  int i, f;

  printf("Unesite neki niz znakova:\n");
  fgets(s, 81, stdin); //funkciji se predaju polje, maksimalan broj znakova i tok

  printf("Unesite trazeni znak: ");
  tz = getchar(); //ucitavanje jednog znaka; umjesto: scanf("%c", &tz);

  f = 0;
  for (i = 0; s[i] != '\0'; i++)
    if (s[i] == tz) {
      f = 1; //signalizira da je pronaden trazeni znak
      break; //pretraga se prekida ukoliko je pronaden trazeni znak
    }

  //uporaba uvjetnog operatora ? : koji je jedini ternarni operator u jeziku
  //cijeli izraz poprima vrijednost izraza lijevo od : ako je uvjet ispred ? istinit,
  //u suprotnom vrijednost izraza desno od :
  printf("Trazeni znak%spostoji u nizu\n", f == 0 ? " ne " : " ");

  return 0;
}
```

Kratko obrazloženje [Primjer 7]: Funkcija `fgets()` omogućuje učitavanje niza znakova, ali kontrolu maksimalnog broja znakova koji će biti spremljen u polje. Nadalje, funkcija `getchar()` služi za učitavanje jednog znaka.

Primjer 8. *Omogućiti korisniku unos jednog stringa od maksimalno 100 znakova. Potom, ispitati i na ekran ispisati odgovarajuću poruku je li string palindrom.*

```
#include <stdio.h>
#include <string.h> //strlen()
```

```
int main(void)
{
  char s[101];
  int i, j, m, f;

  printf("Unesite neki niz znakova:\n");
  fgets(s, 101, stdin);

  m = strlen(s); //vraca duljinu stringa (ne ukljucuje null znak)

  f = 1;     //pocetna pretpostavka kako je string palindrom
  i = 0;     //pocetak, vrijednost indeksa prvog znaka
  j = m - 1; //kraj, vrijednost indeksa zadnjeg znaka
  while (i <= j) {
    if (s[i] != s[j]) { //ako nisu jednaki
      f = 0;    //string ne moze biti palindrom
      break;    //pa se moze prekinuti daljna provjera
    }
    i++; //pomicanje u desno, znak po znak
    j--; //pomicanje u lijevo, znak po znak
  }

  printf("Niz znakova%spalindrom.\n", f == 0 ? " nije " : " je ");

  return 0;
}
```

Kratko obrazloženje [Primjer 8]: Početna pretpostavka je kako je string palindrom, a provjerom se to pokušava opovrgnuti. Funkcija `strlen()` vraća duljinu predanog joj polja znakova koje je string. Kako bi ju se moglo koristiti potrebno je uključiti zaglavnu datoteku string.h. Duljina predstavlja broj znakova do pojave *null* znaka.

POGLAVLJE 9

Funkcije: uvod

Svaki program pisan u programskom jeziku C sastoji se od jedne ili više funkcija. One predstavljaju imenovane blokove (naredbi). Izvođenje programa kreće od funkcije `main()` koja je obavezna. Kako je C vrlo malen programski jezik, tako u pravilu svi programi koriste, između ostalog, neke od funkcija dostupnih iz standardne biblioteke (primjerice, funkcije `printf()` i `scanf()`). Funkcije (gotove) možemo promatrati kao "crne kutije" koje imaju ulaz (ono što joj je potrebno predati) i izlaz (rezultat), a korisnik ne mora brinuti o tome što se događa unutar funkcije. Međutim, C programi nisu ograničeni na uporabu samo funkcija standardne biblioteke, nego jezik omogućuje pisanje vlastitih funkcija.

Funkcije predstavljaju samostalne cjeline programa koje obavljaju dani zadatak. Pisanje vlastitih funkcija ima niz prednosti u odnosu na pisanje cijelog programa unutar funkcije `main()`. One omogućuju raspodjelu programa u manje cjeline, što u konačnici može učiniti program značajno čitljivijim, lakše razumljivim i jednostavnim za održavanje. Također, one omogućuju izbjegavanje višestrukog pisanja određenog slijeda naredbi, odnosno izbjegavanje dupliciranja kôda koji uvelike otežava izmjene programa.

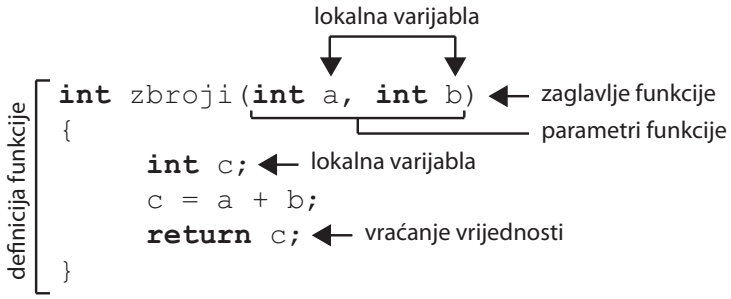
9.1 Pisanje i uporaba funkcija

Kao što je prethodno navedeno, funkcija obavlja određeni zadatak pa je stoga potrebno, uz kako točno to obavlja zadatak, odrediti i njen ulaz i izlaz. To znači, mora se odrediti što je nužno predati funkciji kako bi obavila zadatak te što je njen rezultat nakon njegovog obavljanja. Navedeno je nužno odrediti prilikom definiranja i deklariranja funkcije.

9.1.1 Definiranje funkcija

Funkcije je nužno definirati, što znači da je potrebno točno odrediti što će ona raditi, odnosno koji zadatak obavlja. Primjer definicije jednostavne funkcije prikazan je na slici 9.1. Prilikom definicije, potrebno je navesti tip funkcije (izlaz), ime funkcije, parametre funkcije (ulaz) te tijelo funkcije:

- *Tip funkcije*. Određuje tip podatka kojeg funkcija vraća.
- *Ime funkcije*. Vrijede ista pravila kao i kod imenovanja varijabli.
- *Parametri funkcije*. Navode se unutar zagrada nakon imena funkcije, gdje je za svaki parametar nužno posebno navesti njegov tip podatka, a razdvajaju se zarezima.



Slika 9.1: Definicija funkcije koja ima kao parametre dva cijela broja i kao rezultat vraća zbroj njihovih vrijednosti.

- *Tijelo funkcije*. Slijed naredbi unutar vitičastih zagrada koje opisuju što funkcija radi.

Prema slici 9.1, prve tri stavke predstavljaju zaglavlje funkcije koje označava tip, ime i parametre funkcije. Nakon zaglavlja se nalazi tijelo funkcije koje točno opisuje što ona radi. Shodno tome, unutar vitičastih zagrada ({ }) ide slijed naredbi koje opisuju zadatak kojeg obavlja funkcija i koje će se izvoditi prilikom svakog poziva ili uporabe funkcije.

Jednostavno rečeno, ovisno o zadatku ili radnji koju funkcija obavlja ona treba dati vrijednost na izlaz ili ne. Ukoliko zadatak koju obavlja funkcija ne proizvodi neku vrijednost onda se za njen tip navodi `void`, što označava da funkcija ne vraća ništa. U suprotnom slučaju kada zadatak koji obavlja funkcija proizvodi neku vrijednost, onda se za njen tip navodi tip podatka koji odgovora toj vrijednosti. Nadalje, ako su funkciji potrebni podaci izvana kako bi obavila svoj zadatak ona ima ulaz predstavljen parametrima. Parametere se navodi unutar zagrada razdvojene zarezima, gdje je za svaki pojedini parametar nužno navesti tip podatka i ime. Suprotno tome, ako funkcija nema parametre (nema ulaza) onda se unutar zagrada, poslije imena funkcije, navodi samo `void`.

Parametri funkcije predstavljaju lokalne varijable za tu funkciju. To znači da su te varijable vidljive (dostupne) samo unutar definicije, odnosno tijela funkcije i gdje ih je kao takve (kao varijable) moguće koristiti. Prema tome, deklaracija varijabli unutar tijela funkcije istih imena kao parametara rezultirala bi greškom prilikom prevođenja. Dodatno, one će imati vrijednosti koje su zadane pri pozivu ili uporabi funkcije. Dalje, kada funkcija treba vratiti neku vrijednost koristi se, unutar tijela funkcije, ključna riječ `return` nakon koje ide odgovarajući podatak ili općenito, izraz koji će biti vrednovan prije vraćanja. Važno je napomenuti kako funkcija može vratiti isključivo jednu vrijednost. Također, izvršavanjem naredbe `return` završava se izvođenje funkcije bez obzira ima li naredbi nakon nje. Osim toga, tip podatka čiju se vrijednost uzima kao povratnu vrijednost treba odgovarati tipu funkcije. Ukoliko pak ne odgovara, bit će pretvorena u tip podatka koji je naveden za tip funkcije.

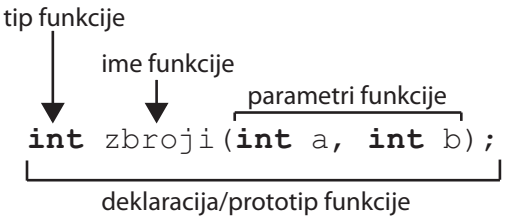
U izlistanju 9.1 navedeni su primjeri definicije funkcije. Funkciju nije moguće definirati unutar definicije neke druge funkcije. To znači kako se definicija funkcije mora nalaziti izvan svih funkcija. Definicija mora prethoditi prvom pozivu ili korištenju funkcije te je vidljiva (moguće ju je koristiti) od mjesta definicija do kraja datoteke.

9.1.2 Deklariranje funkcija

Funkcije je moguće i deklarirati, ali ih je nužno naknadno definirati. Deklaracija najavljuje da će se u programu koristiti funkcija (koju se deklarira). Deklaracija se obavlja pomoću

```
//funkcija tipa void koja nema parametara
void poruka(void)
{
  printf("Poruka!"); //ispis poruke na ekran
}

//funkcija tipa double koja ima parametar tipa double
double kvadrat(double x)
{
  double y;
  y = x * x; //racunanje kvadrata vrijednosti parametra
  return y; //vracanje rezultata izracuna
}

//funkcija tipa double koja ima parametar tipa int
double srednja_vrijednost(int n)
{
  double z;
  z = n * (n + 1) / 2; //racunanje zbroja prvih n prirodnih brojeva
  return z / n;        //racunanje i vracanje srednije vrijednosti tih brojeva
}
```

Izlistanje 9.1: Primjeri definicije funkcije



Slika 9.2: Prototip funkcije cjelobrojnog tipa koja ima dva cjelobrojna parametra.

prototipa funkcije (engl. *function prototype*) koji određuje tip funkcije te broj i tipove podataka njenih parametara. Uz to, na kraj ide točka-zarez. Opet je važno napomenuti da ukoliko funkcija nema parametre ide `void` unutar zagrada poslije imena (nije isto značenje ostaviti prazne zagrade i staviti `void` unutar njih). Deklaraciju je potrebno navesti prije prve uporabe (poziva) funkcije, a njena definicija može onda nalaziti na drugom mjestu unutar teksta programa. Za razliku od definicije funkcije, deklaraciju je moguće navesti unutar drugih funkcija. To znači, funkcije je moguće deklarirati izvan svih funkcija i unutar neke druge funkcije, dok ih je nužno definirati izvan svih funkcija. Ako je funkcija deklarirana unutar neke druge funkcije nju je samo unutar iste moguće pozvati. Također, za razliku od definicije, prilikom deklaracije moguće je ispustiti imena njenih parametara, odnosno moguće je navesti samo tipove podataka pojedenih parametara (to nije preporučeno, jer imena doprinose dokumentaciji). Primjer prototipa jednostavne funkcije prikazan je na slici 9.2, a u izlistanju 9.2 je navedeno nekoliko dodatnih primjera prototipa različitih funkcija.

Ukoliko funkcija nije deklarirana, zaglavlje njene definicije služi kao prototip. Uobičajeno se prototipovi funkcija pišu ispred funkcije `main()`, a definicije ispod nje. To olakšava čitanje teksta programa. Štoviše, u složenijim programima se organiziraju, odnosno odvajaju prototipovi i definicije u posebne datoteke.

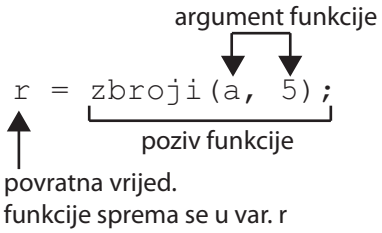
```
void funk1(void);                  //prototip funkcije koja ne vraca nista (funkcija tipa void)
                                   //i nema parametara

void funk2(int x);                 //prototip funkcije koja ne vraca nista (funkcija tipa void)
                                   //i ima jedan parametar tip int

double kvadrat(double n);          //prototip funkcije koja vraca vrijednost tipa double
                                   //(funkcija tipa double) i ima parametar tipa double

double sredina(float x, float y); //prototip funkcije koja vraca vrijednost tipa double
                                   //(funkcija tipa double) i ima dva parametra tipa float

int f(int x, float y, double z);  //prototip funkcije koja vraca vrijednost tipa int
                                   //(funkcija tipa int) i ima parametre tipa int, float i double
```

Izlistanje 9.2: Primjeri prototipa funkcije

9.1.3 Pozivanje funkcija

Nakon što je funkcija definirana (i deklarirana) može ju se koristiti, odnosno pozivati. Funkcije se poziva preko njihovog imena iza kojeg ide par zagrada. One su unarni operator koji je lijevo asocijativan. Ako funkcije ima parametre, prilikom poziva predaju joj se argumenti koji, jednostavno rečeno, predstavljaju ulazne vrijednosti. Argumenti mogu biti varijable, konstante, izrazi i slično. Argumenti se navode unutar zagrada poslije imena i razdvaja ih se zarezima. Vrijednosti argumenata dodjeljuju se redom parametrima funkcije. Drugačije rečeno, argumenti se predaju po vrijednosti te se može reći, ako su argumenti varijable, kako funkcija radi s kopijama argumenata pa promjene tih kopija ne utječu na stvarne argumente u pozivajućoj funkciji. Kako su parametri funkcije loklane varijable za nju, one postoje samo dok se funkcija izvodi nakon čega se oslobađa memorija zauzeta za njih. Na koncu, ukoliko funkcija nema parametre, zagrade poslije imena se ostavljaju praznima pri pozivu. Primjer poziva funkcije dan je na slici 9.3.

Svaka funkcija može pozivati druge funkcije. Ako se prilikom izvođenja programa dođe na naredbu koja predstavlja poziv neke funkcije, prekida se izvođenje naredbi u trenutnoj i izvode se naredbe u pozvanoj funkciji. Po završetku izvršavanja pozvane funkcije, nastavlja se s izvođenjem na mjestu poziva u pozivajućoj funkciji. Kao što je vidljivo sa slike 9.3, funkcije koje vraćaju vrijednost mogu se nalaziti u sklopu nekog izraza i shodno tome na mjestima gdje se očekuje izraz (primjerice, u naredbi `if` kao uvjet). To znači da se prilikom povratka u pozivajućuju funkciju, na mjesto poziva umeće povratna vrijednost i koristi. Funkcije koje su tipa `void`, prema tome, nije moguće koristiti u sklopu izraza.



Slika 9.3: Poziv funkcije. Funkciji su predana dva argumenta, gdje `a` predstavlja neku varijablu, a `5` konstantu. Rezultat funkcije, odnosno njena povratna vrijednost sprema se u varijablu `r`.

9.2 Neke smjernice za pisanje funkcija

Raspodjela programa u manje cjeline pomoću funkcija uvelike olakšava čitanje, razumijevanje i doradu teksta programa. Kako bi se to ostvarilo na što višoj razini, treba pažljivo pristupiti rasporedjeli programa. Između ostalog, funkcije bi trebale raditi samo jedan zadatak, biti malene i imati opisna imena.

Svaka funkcija obavlja samo jedan zadatak ako ju nije moguće razlomiti u manje smislene funkcije. Primjerice, ne treba miješati računanje i vraćanje vrijednosti te ispis na ekran. Računanje i vraćanje vrijednosti je jedan zadatak, dok je ispis na ekran drugi zadatak. Povratna vrijednost je ono čime program može raditi, dok ispis na ekran služi samo za potrebe korisnika, a beskoristan je za program. Ispis na ekran se uobičajeno obavlja u glavnoj, odnosno funkciji `main()`. Pisanje funkcija koje rade samo jedan posao/zadatak, uglavnom rezultira malenim funkcijama koje je jednostavno razumjeti i koje je moguće koristiti na različitim mjestima u programu. Međutim, pisanje malenih funkcije nije uvijek jednostavno te je nekada lakše prvo napisati veću funkciju pa ju onda razlomiti na manje. Na kraju, jednostavnosti uporabe doprinosi davanje opisnih imena iz kojih bi trebalo biti odmah jasno koji to zadatak funkcija obavlja, bez potrebe dodatnih komentara ili potrebe za proučavanjem samog tijela funkcije.

9.3 Riješeni primjeri

U nastavku slijedi nekoliko primjera u kojima je nužno napisati odgovarajuću funkciju koja rješava dani zadatak. Treba napomenuti kako su imena svih funkcija kao i pokazni primjeri njihove uporabe proizvoljno odabrani s obzirom da nisu u primjerima eksplicitno definirani, a tipovi podataka odabrani su shodno potrebama zadataka koje funkcije trebaju obaviti.

Primjer 1. *Napisati funkciju koja će na ekran ispisati sve parne brojeve između 2 i 20. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
//f-ja koja nista ne vraca niti prima
void parni(void); //deklaracija/prototip f-je parni()

int main(void)
{
  //Primjer uporabe napisane f-je
  parni(); //poziv funkcije parni()
  return 0;
}

//definicija funkcije parni()
void parni(void)
{
  int i; //lokalna varijabla; vidljiva samo unutar f-je parni()
  for (i = 2; i <= 20; i += 2)
    printf("%d ", i); //ispis na ekran
}
```

Kratko obrazloženje [Primjer 1]: S obzirom da funkcija treba samo ispisati parne brojeve između 2 i 20 ona ništa ne vraća i nema parametara, jer su granice (2 i 20) konstantne te funkcija ne treba ništa računati nego samo obaviti ispis na ekran.

Primjer 2. *Napisati funkciju koja će na ekran ispisati sve parne brojeve između 2 i n. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
//f-ja koja nista ne vraca, a prima cijeli broj
void parni(int n); //prototip funkcije parni()

int main(void)
{
  int a = 100;
  parni(a);  //poziv funkcije parni()
  printf("\n");
  parni(50); //poziv funkcije parni()
  return 0;
}

//definicija funkcije parni()
void parni(int n)
{//varijabla n je loklna za f-ju parni() te je vidljiva samo unutar te f-je
 //var. n ce poprimiti vrijed. koja je predana kao argument pri pozivu f-je
  int i; //lokalna varijable; vidljiva samo unutar f-je parni()
  for (i = 2; i <= n; i += 2)
    printf("%d ", i); //ispis na std. izlaz
}
```

Kratko obrazloženje [Primjer 2]: Kako funkcija treba samo obaviti ispis na ekran, tako ona ništa ne vraća pa je njen tip, kao i u primjeru 1, `void`. Funkcija ima samo jedan parametar, jer je donja granica (2) konstantna, dok je gornja (n) promjenjiva pa shodno tomu predstavlja cjelobrojni parametar funkcije.

Primjer 3. *Napisati funkciju koja će izračunati i vratiti zbroj svih parnih brojeva između 2 i n. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
//f-ja koja vraca i prima cijeli broj
int parni(int n); //prototip funkcije parni()

int main(void)
```

```
{
  int z;
  //povratnu vrijed. f-je dodjeljujemo var. z
  z = parni(12024); //poziv funkcije parni()
  printf("Zbroj prvih n parnih brojeva je: %d", z);
  return 0;
}

//definicija funkcije parni()
int parni(int n)
{//varijabla n je loklna za f-ju parni()
  int i, zbroj; //lokalne varijable; vidljive samo unutar f-je parni()
  zbroj = 0;
  for (i = 2; i <= n; i += 2)
    zbroj += i;

  return zbroj; //vraca izracuanti zbroj
}
```

Kratko obrazloženje [Primjer 3]: S obzirom da funkcija treba izračunati i vratiti zbroj svih parnih brojeva između 2 i n ona je cjelobrojnog tipa (`int` u ovom slučaju). Također, funkcija ima samo jedan cjelobrojni parametar iz istog razloga kao u primjeru 2.

Primjer 4. *Napisati funkciju koja će izračunati i vratiti zbroj znamenki predanog joj cijelog broja. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int zbrojzn(int n)
{
  int zbroj = 0;
  n = n < 0 ? -n : n; //uzima se apsolutna vrijednost broja kako bi zbroj uvijek bio pozitivan
  //zbraja znamenke od zadnje prema prvoj
  while (n != 0) {
    zbroj += n % 10; //dodaje trenutno zadnju znamenku zbroju
    n /= 10; //nakon sto trenutno zadnju znamenku doda zbroju, uklanja ju
  }
  return zbroj;
}

int main(void)
{
  int z, a = -12345;
  //povratnu vrijed. f-je dodjeljujemo var. z
  z = zbrojzn(a);
  printf("Zbroj znamenki broja %d je: %d", a, z);
  return 0;
}
```

Kratko obrazloženje [Primjer 4]: S obzirom da funkcija treba izračunati i vratiti zbroj znamenki predanog joj cijelog broja onda je cjelobrojnog tipa. Funkcija ima jedan cjelobrojni parametar koji predstavlja broj za koji treba izračunati zbroj znamenki.

Primjer 5. *Napisati funkciju koja će izračunati i vratiti zbroj znamenki predanog joj nenegativnog cijelog broja po bazi danog brojevnog sustava. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
//f-ja koja vraca cijeli broj i prima dva cijela broja
unsigned int zbrojzn(unsigned int n, unsigned short b);

int main(void)
{
  //u unsigned mogu se spremiti samo nenegativne brojeve
```

```
  unsigned z, a = 12345; //unsigned isto kao unsigned int
  z = zbrojzn(a, 2);
  printf("Zbroj znamenki broja %u je: %u", a, z);
  return 0;
}

unsigned int zbrojzn(unsigned int n, unsigned short b)
{//var. b odreduje bazu brojnog sustava
  unsigned int zbroj = 0;
  //zbraja znamenke (baze b) od zadnje prema prvoj
  while (n > 0) {
    zbroj += n % b; //dodaje trenutno zadnju znamenku zbroju
    n /= b; //nakon sto trenutno zadnju znamenku doda zbroju, uklanja ju
  }

  return zbroj;
}
```

Kratko obrazloženje [Primjer 5]: Funkcija je cjelobrojnog tipa iz istog razloga kao u primjeru 4. Međutim, za razliku od funkcije u primjeru 4, ona ima dodatni cjelobrojni parametar koji predstavlja bazu brojnog sustava prema kojem se računa zbroj znamenki. Za tip funkcije i tipove podataka parametara funkcije odabrani su cjelobrojni tipovi bez predznaka (`unsigned int` i `unsigned short`), jer funkcija treba raditi samo s nenegativnim brojevima.

Primjer 6. *Napisati funkciju koja će provjeriti je li dani nenegativni cijeli broj prost te vratiti 1 u slučaju da jest, a u suprotnom vratiti 0. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int prost(unsigned int); //izostavljen je naziv parametra u prototipu f-je

int main(void)
{
  unsigned int a[5] = {13, 45, 17, 31, 123};
  int i;
  for (i = 0; i < 5; i++) {
    if (prost(a[i])) printf("Broj %u je prost\n", a[i]); //poziv f-je unutar if naredbe
    else printf("Broj %u nije prost\n", a[i]);
  }
  return 0;
}

int prost(unsigned int n)
{
  unsigned int i;

  if (n < 2)
    return -1; // najmanji prost broj je 2
  //ako je ostatak pri dijeljenju s bilo kojim brojem u intervalu (2, n) 0 broj n nije prost
  for (i = 2; i < n; i++)
    if (n % i == 0)
      return 0; //vraca 0 i zavrsava izvodenje f-je

  return 1;
}
```

Kratko obrazloženje [Primjer 6]: U slučaju da ostatak pri dijeljenju danog broj s jednim od brojeva iz intervala $(2, n)$ bude nula isti ne može biti prost te nije potrebna dalja provjera i može se prekinuti izvođenje funkcije. Navedeno je postignuto s naredbom `return 0;` koja se izvodi u tom slučaju. Ako ostatak pri dijeljenju nije bio niti jednom 0 broj mora biti prost te se sukladno tomu vraća vrijednost 1. Kako je najmanji prosti broj 2, tako funkcija

vraća vrijednost -1, koju se može uzeti kao signal za pogrešku, u slučaju da joj se preda manji broj.

Primjer 7. *Napisati funkciju koja će izračunati i vratiti faktorijel danog cijelog broja iz intervala* $[0, 12]$. *Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int faktorijel(int n);

int main(void)
{
  int i, f;
  for (i = 0; i <= 12; i++) {
    f = faktorijel(i);
    printf("%d! = %d\n", i, f);
  }
  return 0;
}

int faktorijel(int n)
{
  int f = 1, i;
  if (n < 0)   //faktorijel je definiran samo za nenegativne cijele brojeve
    return -1; //prekida daljnje izvodenje funkcije i vraca -1
  if (n > 12)  //ako se prekoraci gornja granica zadanog intervala
    return -2; //prekida daljnje izvodenje funkcije i vraca -2

  for (i = 2; i <= n; i++) f *= i;

  return f;
}
```

Kratko obrazloženje [Primjer 7]: Kako je faktorijel definiran samo za nenegativne brojeve tako u slučaju da se funkciji preda negativan broj ona vraća vrijednost -1 što se jednostavno može protumačiti kao pogreška. Također, ako se funkciji preda broj veći od 12 (prekoračenje gornje granice) ona vraća vrijednost -2 što opet ne može biti faktorijel nekog broja te predstavlja pogrešku.

Primjer 8. *Napisati funkciju koja će izračunati i vratiti cjelobrojnu potenciju danog realnog broja. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

double potencija(double x, int n);

int main(void)
{
  double b = 2.5;
  double rez = potencija(b, 2); //inicijalizacija var. s povratnom vrijednoscu f-je

  printf("%f\n", rez); //ispisuje: 6.25

  return 0;
}

double potencija(double x, int n)
{
  int i;
  int m = n < 0 ? -n : n; //inicijalizacija var. rezultatom uvjetnog operatora
  double p = 1;

  for (i = 0; i < m; i++)
    p *= x;
```

```
  if (n < 0)
    return 1 / p;

  return p;
}
```

Kratko obrazloženje [Primjer 8]: S obzirom da potencija može biti negativna, potrebno je odrediti njenu apsolutnu vrijednost, za što je iskorišten uvjetni operator. U slučaju da je potencija negativna vraća se vrijednost $\frac{1}{x^n}$, jer je $x^{-n} = \frac{1}{x^n}$.

Primjer 9. *Napisati funkciju koja određuje i vraća zrcalnu vrijednost predanog joj cijelog broja. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
#include <math.h> //zbog funkcije pow()

int reverse(int num);    //racuna i vraca zrcalnu vrijednost danog broja
int digitCount(int num); //racuna i vraca broj znamenki danog broja

int main(void)
{
  int a = -12345;
  printf("%d\n", reverse(a)); //ispisuje: -54321

  return 0;
}

int reverse(int num)
{
  int reversed = 0;               //zrcalna vrijednost danog broja
  int digits = digitCount(num);  //broj znamenki danog broja
  int exp = pow(10, digits - 1); //potencija broja 10 za prvu znamenku zrcalnog broja

  while (num != 0) {
    int digit = num % 10;
    reversed += digit * exp;
    exp /= 10;
    num /= 10;
  }
  return reversed;
}

int digitCount(int num)
{
  int digits = 1;
  while (1) {
    num /= 10;
    if (num != 0)
      digits++;
    else
      break;
  }
  return digits;
}
```

Kratko obrazloženje [Primjer 9]: Prvo je potrebno znati koliki je broj znameki danog broja pa je taj posao izdvojen u posebnu (pomoćnu) funkciju. To doprinosi čitljivosti funkcije koja računa i vraća zrcalnu vrijednost. Na temelju broja znamenki se određuje najveća potrebna potencija broja 10, što će biti težina prve znamenke zrcalnog broja. Drugačiji pristup mogao bi predstavljati da se unutar iste funkcije odredi najveću potrebnu potenciju, čime bi se izbjegla potreba za funkcijom `pow()` ili pak bolje da se to izdvoji u posebnu funkciju. Funkcija `pow()` opisana je u zaglavnoj datoteci `math.h`, dok joj je prototip: `double pow(double baza, double pot)`.

Primjer 10. *Napisati funkciju koja određuje i vraća broj iz intervala* $[a, b] \subset \mathbb{N}$ *koji ima najviše dijelitelja. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int mostDivsibleIn(int a, int b);
int divCount(int num);

int main(void)
{
  int r;

  r = mostDivsibleIn(1, 12345);
  printf("%d\n", r);

  return 0;
}

int mostDivsibleIn(int a, int b)
{
  int max, i, num;
  num = a;
  max = divCount(a);
  for (i = a + 1; i <= b; i++) {
    if (divCount(i) > max) {
      max = divCount(i);
      num = i;
    }
  }
  return num;
}

int divCount(int num)
{
  int i, dc;
  dc = 0;
  for (i = 1; i <= num; i++) {
    if (num % i == 0) {
      dc++;
    }
  }
  return dc;
}
```

Kratko obrazloženje [Primjer 10]: Kako bi se pronašao traženi broj, potrebno je za svaki iz zadanog intervala odrediti koliko ima dijelitelja te spremiti onaj koji je imao najviše. Određivanje koliko dani broj ima dijelitelja stoga je izdvojeno u posebnu funkciju. Kao i u prethodnom primjeru, svaka funkcija obavlja samo jedan zadatak. Iako u tekstu programa nema komentara trebalo bi biti razumljivo što i kako se radi.

POGLAVLJE 10

Uvod u pokazivače

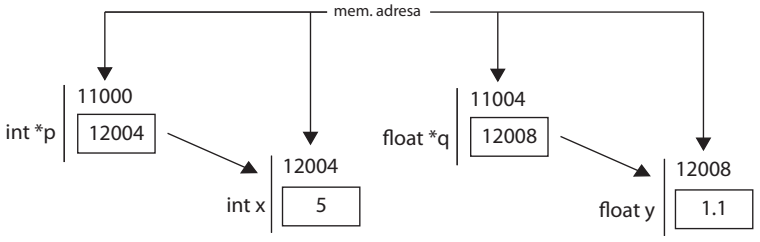
Pokazivači (engl. *pointers*) predstavljaju jednu od najbitnijih značajki programskog jezika C. Oni su u velikoj mjeri zaslužni za fleksibilnost i mogućnosti koje pruža jezik. Jednostavno rečeno, pokazivač je varijabla čija je vrijednost memorijska adresa (Sl. 10.1). Oni pružaju učinkovit, simbolički način rukovanja memorijom, odnosno memorijskim adresama. Može se reći kako pokazivači predstavljaju poseban tip. Pokazivači imaju široku primjenu u jeziku pa tako primjerice omogućuju učinkovito rukovanje poljima, promjenu vrijednosti argumenata predanih funkcijama, stvaranje složenih struktura podataka, rad s dinamički zauzetom memorijom itd.

10.1 Pokazivači i njihova uporaba

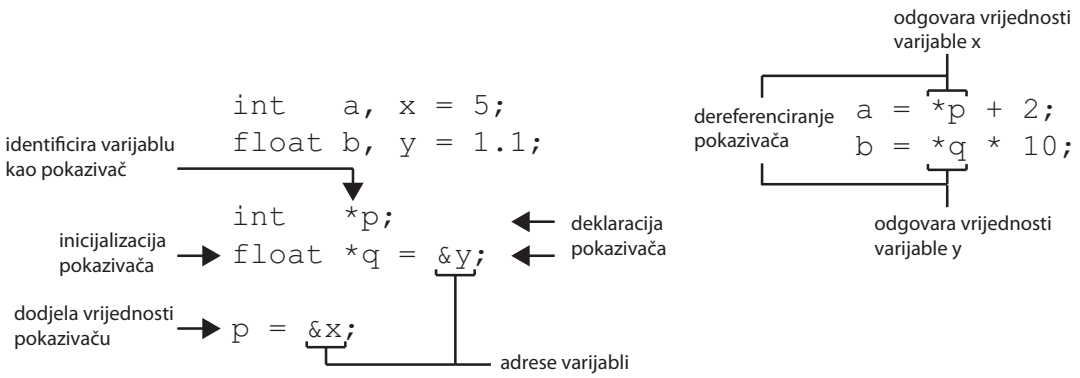
Pokazivači su varijable koje drže memorijske adrese drugih objekata (objekata u općem smislu) kao što su varijable osnovnih tipova podataka, funkcija i slično. Oni omogućuju neizravan (indirektan) pristup vrijednosti objekta na koje su usmjereni, odnosno na koje "pokazuju" i to preko adrese koju drže.

10.1.1 Deklaracija pokazivača

Pokazivače ili pokazivačke varijable je potrebno prije uporabe deklarirati. Deklaracija određuje tip podatka na koji je moguće usmjeriti pokazivač. S obzirom da oni pružaju neizravan pristup, navedeno je nužno, jer različiti tipovi podataka zauzimaju različitu količinu memorije i na različite načine predstavljaju vrijednosti u memoriji. Pokazivače se deklarira na sličan način kao i varijable osnovnih tipova podatka, s time da se ispred identifikatora (imena varijable) stavlja zvjezdica (*). Ona identificira varijablu kao poka-



Slika 10.1: Pokazivači.



Slika 10.2: Deklaracija, inicijalizacija i pridruživanje vrijednosti pokazivaču te dereferenciranje pokazivača.

zivač, a njen položaj nije bitan i ona nije dio imena pokazivača. U izlistanju 10.1 dano je nekoliko primjera deklaracije pokazivača.

```
int *int_pok; //deklaracija pokazivaca na tip podatka int     -> ime pokazivaca: int_pok
char * r;     //deklaracija pokazivaca na tip podatka char    -> ime pokazivaca: r
float *p;     //deklaracija pokazivaca na tip podatka float   -> ime pokazivaca: p
double* q;    //deklaracija pokazivaca na tip podatka double  -> ime pokazivaca: q
```

Izlistanje 10.1: Primjeri deklaracije pokazivča

S obzirom na opisano, može se govoriti o pokazivačima kao tipovima "pokazivač na `tip_podatka`", gdje je `tip_podatka` valjani tip u jeziku C kao primjerice, `char`, `int`, `double` i drugi. Stoga, moguće ga je korisiti za rukovanje samo adresama objekata tipa koji odgovara tipu u deklaraciji pokazivača.

10.1.2 Adresni operator i operator dereferenciranja

Nakon deklaracije moguće je koristiti pokazivač. To znači da mu je moguće dodijeliti/pridružiti vrijednost. Kako pokazivači pohranjuju adrese, tako im je potrebno istu pridružiti. Adresu dane varijable moguće je dohvatiti putem adresnog operatora `&` (engl. *address operator* ili *address-of operator*). On je unaran i desno asocijativan operator te daje adresu svog operanda[1]. Tako dohvaćenu adresu moguće je, osim za pridruživanje, koristiti i za inicijalizaciju pokazivača što je prikazano na slici 10.2. U izlistanju 10.2 prikazni su primjeri inicijalizacije i pridruživanja vrijednosti pokazivačima.

```
int a = 5;
double b = 17.25;
int *p = &a;      //deklaracija pokazivaca na tip podatka int i njegova incijalizacija
double *q;        //deklaracija pokazivaca na tip podatka double
q = &b;           //pridruzivanje vrijednosti pokazivacu
```

Izlistanje 10.2: Primjeri inicijalizacije i pridruživanje vrijednosti pokazivaču

[1]Operand adresnog operatora mora biti l-vrijednost (engl. *lvalue*). Jednostavno rečeno, l-vrijednost predstavlja objekt koji zauzima mjesto u memoriji koje je moguće idendificirati. Suprotno od l-vrijednosti su r-vrijednosti (engl. *rvalue*) koje, analogijom, ne zauzimaju mjesto u memoriji koje je moguće identificirati.

Naravno, jednom pokazivaču može se pridružiti vrijednost (adresa) drugog pokazivača na isti tip podatka. Važno je zapamiti kako je pokazivaču na dani tip (određen deklaracijom pokazivača) moguće samo pridružiti adresu objekta tog tipa.

Nakon što je pokazivač usmjeren (pokazuje) na neku varijablu, može joj se pristupiti posredstvom pokazivača. Vrijednosti varijable preko pokazivača pristupa se putem operatora indirekcije (engl. *indirection operator*) ili dereferenciranja * (označen isto kao aritmetički operator množenja). To znači da se tim operatorom dohvaća vrijednost zapisana na danoj adresi, odnosno na adresi koja je spremljena u pokazivač (Sl. 10.2). Taj operator je unaran i desno asocijativan. Osim što je moguće dohvatiti vrijednost, moguće ju je i promijeniti pridruživanjem nove vrijednosti dereferenciranom pokazivaču. Važno je za napomenuti kako ne treba dereferencirati neinicijalizirani pokazivač, odnosno pokazivač kojem nije pridružena vrijednost, jer kao i u slučaju varijabli osnovnih tipova podataka, pokazivači nakon deklaracije sadrže neku slučajnu/neodređenu vrijednost. Dereferenciranje pokazivača prikazano je primjerima u izlistanju 10.3.

```
int a = 4;
double b = 5.8;
int *p = &a;    //deklaracija i incijalizacija pokazivaca p
double *q = &b; //deklaracija i incijalizacija pokazivaca q
*p = 6;   //dereferenciranje pokazivaca i
    //zapisivanje nove vrijednosti na adresu var. a (isto kao a = 6)
*q = 1.2; //dereferenciranje pokazivaca i
    //zapisivanje nove vrijednosti na adresu var. b (isto kao b = 1.2)
```

Izlistanje 10.3: Primjeri dereferenciranja pokazivača

Na temelju prethodno opisanog, može se reći kako je u slučaju pokazivačkih varijabli primarno adresa, a sekundarno vrijednost (na toj adresi), dok je obratno s "običnim" varijablama. Drugačije rečeno, ime pokazivača predstavlja adresu, dok je za vrijednost potrebno koristiti operator (dereferenciranja).

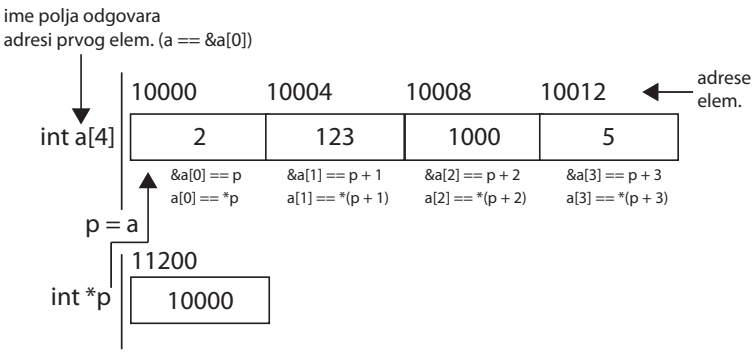
10.2 Pokazivači i polja

Postoji bliska veza između pokazivača i polja. Naime, ime polja predstavlja adresu prvog elementa tog polja (Sl. 10.3). Navedena vrijednost je konstantna te ju nije moguće mijenjati. Moguće ju je pridružiti, kao vrijednost, pokazivaču, a pokazivač je na tip podatka elemenata polja, a ne pokazivač na polje. Može se reći kako je ime polja pokazivač na prvi element toga polja, ali pokazivač kojeg nije moguće preusmjeriti. Primjer usmjeravanja pokazivača na polje dan je izlistanjem 10.4.

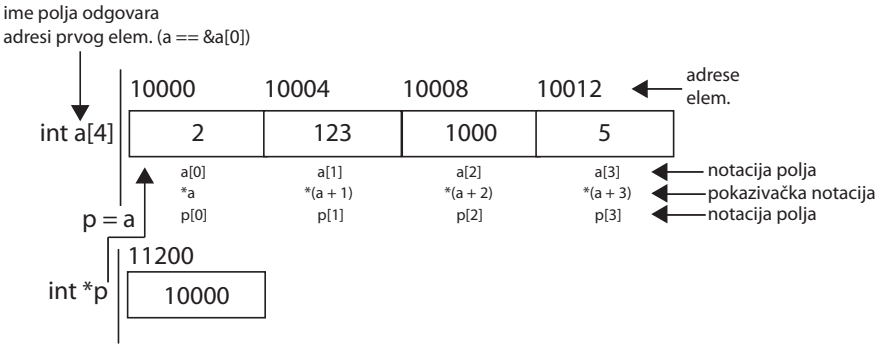
```
float a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
float *p;
p = a; //isto kao: p = &a[0], jer je a == &a[0]
```

Izlistanje 10.4: Primjer pridruživanja adrese prvog elementa polja pokazivaču

Dodavanje cjelobrojne vrijednosti pokazivaču koji pokazuje na neki element polja, ali i imenu polja (pokazivač na prvi element) rezultira izrazom čija je vrijednost adresa elementa koji se nalazi onoliko mjesta (svako mjesto je veliko koliko zauzima memorije tip podatka elemenata) dalje koliko je dodano pokazivaču. Vrijednostima elemenata polja pristupa se preko pokazivača, njegovim dereferenciranjem. Općenito, elementima polja



Slika 10.3: Pokazivač usmjeren na jednodimenzionalno polje.



Slika 10.4: Pristup elementima polja putem pokazivačke i notacije polja.

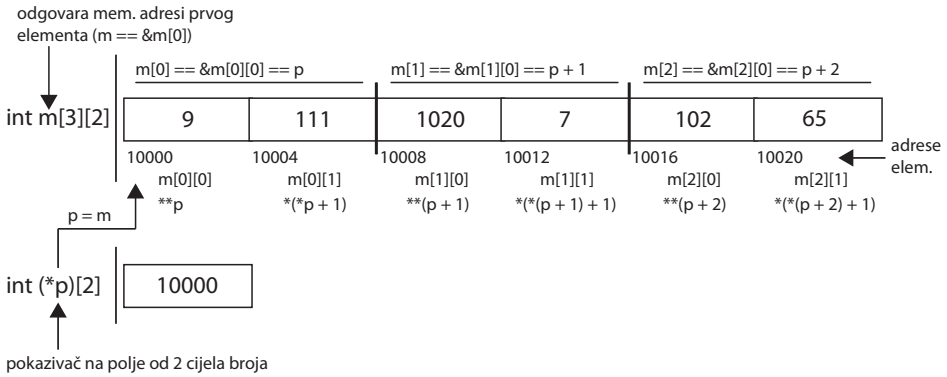
može se pristupati uporabom pokazivačke notacije (engl. *pointer notation*) ili notacije polja (engl. *array notation*), što je prikazano na slici 10.4. Notacija polja odnosi se na korištenje operatora indeksiranja `[]`, dok se pokazivačka notacija odnosi na korištenje operatora dereferenciranja `*`. Notaciju polja moguće je koristiti i s pokazivačima, jer je ona i definirana u sklopu pokazivača te predstavlja "ljepši" zapis kada se radi s poljima. Operator indeksiranja obavlja dereferenciranje uz prethodno pomicanje za zadanu cjelobrojnu vrijednosti (unutar uglatih zagrada) u odnosu na adresu na koju ga se primjenjuje [stoga se indeks još naziva i pomakom (engl. *offset*)]. U izlistanju 10.5 je prikazan primjer pristupa elementima polja preko pokazivača putem pokazivačke i notacije polja.

```
float a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
float x;
float *p;
p = a;
x = *(p + 1); //pokazivaca notacija -> *(p+1) == a[1] -> (p+1) == &a[1]
              //pridruzuje se vrijednost 2 varijabli x, jer je a[1] == 2
x = p[5];     //notacija polja -> p[5] == a[5] == *(p+5) == *(a+5) -> &p[5] == &a[5] == p+5
              //pridruzuje se vrijednost 6 varijabli x, jer je a[5] == 6
```

Izlistanje 10.5: Primjer pristupa elementima polja putem pokazivačke i notacije polja

10.2.1 Pokazivači na polja

Pokazivače je, naravno, moguće koristiti i za pristup elementima dvodimenzionalnih (2-D) polja. Kako je svaki element 2-D polja i sam polje, tako je potrebno koristiti pokazivače na polja (Sl. 10.5). Prilikom deklaracije pokazivača na polje potrebno je navesti veličinu



Slika 10.5: Pokazivač (na polje) usmjeren na dvodimenzionalno polje.

polja na kojeg će ga se moći usmjeriti i tip podatka elemenata. Izlistanjem 10.6 su dani primjeri deklaracije pokazivača na polje i usmjeravanja na 2-D polje. Treba naglasiti kako su pri deklaraciji zagrade oko zvjezdice i imena obavezne. Naime, bez njih bi se radilo o deklaraciji polja pokazivača na dani tip.

```
int a[2][3] = { {1, 2, 3}, {4, 5, 6} };
double b[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
int (*p)[3];    //deklaracija pokazivaca na polje od 3 elementa tipa int
double (*q)[4]; //deklaracija pokazivaca na polje od 4 elemenata tipa double
p = a;    //isto kao: p = &a[0]
q = b;    //isto kao: q = &b[0]
```

Izlistanje 10.6: Primjeri deklaracije pokazivača na polje i pridruživanja adrese prvog elementa 2-D polja pokazivaču

Kako bi se pristupilo nekoj vrijednosti 2-D polja preko pokazivača na polje, potrebno je takav pokazivač dva puta dereferencirati. Prvim dereferenciranjem dobiva se element 2-D polja koji je opet polje (ime polja je adresa prvog elementa). Ponovnim dereferenciranjem pokazivača moguće je dobiti vrijednost (Sl. 10.5). U izlistanju 10.7 je naveden primjer usmjeravanja pokazivača na 2-D polje te pristup elementima putem pokazivačke i notacije polja.

```
int a[2][3] = { {1, 2, 3}, {4, 5, 6} };
int (*p)[3] = a;
int x;
x = *(*p + 1); //pokazivacka notacija -> *(*p+1) == a[0][1] -> (*p+1) == &a[0][1]
               //pridruzuje se vrijednost 2 varijabli x, jer je a[0][1] == 2
x = p[1][2];   //notacija polja -> p[1][2] == a[1][2] -> &p[1][2] == &a[1][2]
               //pridruzuje se vrijednost 6 varijabli x, jer je a[1][2] == 6
```

Izlistanje 10.7: Primjer pristupa elementima 2-D polja putem pokazivačke i notacije polja

10.3 Riješeni primjeri

U nastavku slijedi nekoliko primjera za čije je rješavanje potrebno koristiti pokazivače, odnosno pokazivačku notaciju. Treba napomenuti kako su vrijednosti korištene za inicijalizaciju ili pridruživanje proizvoljno odabrane s obzirom da nisu eksplicitno zadane u primjerima.

Primjer 1. *Deklarirati i inicijalizirati dvije cjelobrojne varijable i deklarirati jedan pokazivač na cjelobrojni tip. Pokazivač usmjeriti na prvu varijablu ispisati njenu vrijednost na ekran te joj vrijednost promijeniti putem pokazivača i ponovo je ispisati. Nakon toga preusmjeriti pokazivač na drugu varijablu i ispisati njenu vrijednost izravno i preko pokazivača.*

```
#include <stdio.h>

int main(void)
{
  int a = 1, b = 23;
  int *p; //deklaracija pokazivaca p kojeg se moze usmjeriti na varijable tipa int

  p = &a; //usmjeravanje p na a [& - adresni operator] (pridruzivanje adrese var. a var. p)
  printf("%d", a); //ispisuje: 1
  *p = 5; //neizravni pristup var. a [* - operator dereferenciranja ili indirekcije]
  printf("%d", a); //ispisuje: 5

  p = &b;
  printf("%d %d", b, *p); //ispisuje: 23 23

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Kako je pokazivač varijabla, tako je moguće mijenjati vrijednost koju čuva. To znači, moguće ga je usmjeriti na jednu varijablu pa zatim na drugu itd.

Primjer 2. *Deklarirati i inicijalizirati dvije realne varijable te deklarirati i inicijalizirati dva pokazivača na njih. Ispisati adrese obje varijable i vrijednosti oba pokazivača.*

```
#include <stdio.h>

int main(void)
{
  float a = 1, b = 23;
  float *p = &a; //deklaracija i inicijalizacija pokazivaca p
  float *q = &b; //deklaracija i inicijalizacija pokazivaca q

  //ispis memorijskih adresa varijabli
  printf("%p, %p\n", &a, &b); //ispisuje memorijske adrese varijabli a i b
  //ispis vrijednosti na koju pokazivaca pokazuje
  printf("%p, %p\n", p, q);   //ispisuje isto

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Za ispis podatka kojeg čuva pokazivač, odnosno adrese, u pravilu u heksadecimalnom obliku, koristi se oznaka pretvorbe `%p`.

Primjer 3. *Deklarirati i inicijalizirati dvije cjelobrojne varijable te dva pokazivača na njih. Ispisati vrijednosti varijabli, zamijeniti im vrijednosti preko pokazivača te im opet ispisati vrijednosti.*

```
#include <stdio.h>

int main(void)
{
  int a = 7;
  int b = 12;
  int *p = &a;
  int *q = &b;
  int priv;  //predstavlja pomocnu varijablu

  printf("%d, %d\n", a, b); //ispisuje: 7, 12
  //zamjena
```

```
  priv = *p; //cuva vrijednost na koju pokazuje p
  *p = *q;
  *q = priv;

  printf("%d, %d\n", a, b); //ispisuje: 12, 7

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Kako bi se napravila zamjena vrijednosti varijabli potrebno je koristiti pomoćnu varijablu/spremnik u koju će se spremiti vrijednost one koju se prvo mijenja, odnosno one čiju se vrijednost prvo prepisuje. Na taj način ona neće biti izgubljena.

Primjer 4a. *Deklarirati polje realnih brojeva veličine 10. Deklarirati i usmjeriti pokazivač na to polje. Tražiti od korisnika unos elemenata polja te ga potom ispisati na ekran. Unos elemenata i njihov ispis ostvariti preko pokazivačke notacije.*

```
#include <stdio.h>

#define VEL 10

int main(void)
{
  float a[VEL];
  float *p;
  int i;

  p = a; //ili p == &a[0], jer je a == &a[0]
  for (i = 0; i < VEL; i++)
    scanf("%f", p + i); //p+i == a+i == &p[i] == &a[i]

  for (i = 0; i < VEL; i++)
    printf("%f\n", *(p + i)); //*(p+i) == *(a+i) == p[i] == a[i]

  return 0;
}
```

Kratko obrazloženje [Primjer 4a]: Kako operator dereferenciranja ima veću razinu prvenstva (prioritet) u odnosu na operator zbrajanja, tako je nužno koristiti zagrade. Tako primjerice, izraz `*(p + 1)` ima vrijednost elementa `a[1]`, dok izraz `*p + 1` ima vrijednost elementa `a[0]` uvećanu za jedan, odnosno `a[0] + 1`.

Primjer 4b. *Deklarirati polje realnih brojeva veličine 10. Deklarirati i usmjeriti pokazivač na to polje. Tražiti od korisnika unos elemenata polja te ga potom ispisati na ekran. Unos elemenata i njihov ispis ostvariti preko pokazivačke notacije.*

```
#include <stdio.h>

#define VEL 10

int main(void)
{
  float a[VEL], *p;

  //pomice pokazivac redom s jednog na drugi elem. polja
  //povecavanjem vrijed. pokazivaca dobiva se adresa iduceg elem. polja
  for (p = a; p < a + VEL; p++)
    scanf("%f", p);

  //nakon prve for petlje pokazivac je usmjeren na mem. adresu odmah poslije
  //zadnjeg elementa polja (na tome mjestu ga nije preporucljivo dereferencirati)
```

```
  for (p = a; p < a + VEL; p++)
    printf("%f\n", *p);

  return 0;
}
```

Kratko obrazloženje [Primjer 4b]: Primjeri je isti kao 4a, međutim riješen je na drukčiji način. Takav, sekvencijalni, pristup elementima polja preko pokazivača (uporabom operatora uvećavanja nad pokazivačem) može rezultirati bržim izvođenjem u odnosu na uporabu indeksa. Iako je dozvoljeno usmjeravanje pokazivača na adresu odmah nakon zadnjeg elementa polja, njegovo dereferenciranje nije preporučljivo na tom mjestu, jer nije zajamčeno da će biti uspješno.

Primjer 5. *Deklarirati polje od 50 cjelobrojnih elemenata te omogućiti korisniku njegovo popunjavanje. Izračunati i na ekran ispisati zbroj parnih te broj neparnih elemenata polja. Koristiti pokazivačku notaciju za pristup elementima polja. Pretpostaviti da nula nije ni parna ni neparna.*

```
#include <stdio.h>

#define N 50

int main(void)
{
  int a[N];
  int i, *p;
  int zbroj = 0, broj = 0;

  p = a;
  //popunjavanje polja preko standardnog ulaza
  for (i = 0; i < N; i++)
    scanf("%d", p + i);

  for (i = 0; i < N; i++) {
    if (*(p + i) == 0) //nula nije ni parna ni neparna
      continue; //preskace se ostatak i prelazi odmah na iduci korak u petlji
    else if (*(p + i) % 2 == 0)
      zbroj += *(p + i); //ako je vrijed. na mem. adresi p+i parna dodaje ju se zbroju
    else
      broj++;
  }

  printf("Zbroj parnih elem. je: %d\nBroj neparnih elem. je: %d", zbroj, broj);
  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Ako je neka vrijednost elementa polja jednaka nula, preskače se odmah na iduću iteraciju, jer nula nije ni parna ni neparna.

Primjer 6. *Omogućiti korisniku unos* $0 < n \leq 50$ *elemenata u polje cijelih brojeva. Ispisati na ekran elemente čija je vrijednost djeljiva s 5. Koristiti pokazivačku notaciju za pristup elementima polja.*

```
#include <stdio.h>

int main(void)
{
  int a[50];
  int i, n;

  do {
    scanf("%d", &n);
  } while (n <= 0 || n > 50); //korisnik odreduje koliko zeli unijeti elemenata
```

```
  //koristi se ime polja (adresa prvog elem.) umjesto pokazivaca
  for (i = 0; i < n; i++)
    scanf("%d", a + i);

  for (i = 0; i < n; i++)
    if (*(a + i) % 5 == 0)      //ako je neki elem. polja djeljiv s 5
      printf("%d\n", *(a + i)); //ispisuje ga

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Koristi se ime polja kao pokazivač na prvi element tog polja te se koristi pokazivačka notacija umjesto uobičajene notacije polja. Treba napomenuti kako u ovom primjeru nije moguće koristi operator uvećavanja, kao u primjeru 4b, jer ime polja ne predstavlja podatak čiju je vrijednost moguće promijeniti.

Primjer 7. *Deklarirati polje od 20 realnih brojeva te omogućiti korisniku njegovo popunjavanje. Izračunati i ispisati na ekran zbroj onih elemenata polja čija je vrijednost veća od* 2 *te manja od* 6*. Koristiti pokazivačku notaciju za pristup elementima polja.*

```
#include <stdio.h>

#define M  20
#define DG 2
#define GG 6

int main(void)
{
  float a[M], zbroj = 0;
  int i;

  for (i = 0; i < M; i++)
    scanf("%f", a + i);

  for (i = 0; i < M; i++)
    if (*(a + i) > DG && *(a + i) < GG)
      zbroj += *(a + i); //zbrajanje brojeva koji su u intervalu (2, 6)

  printf("Zbroj je: %f", zbroj);
  return 0;
}
```

Kratko obrazloženje [Primjer 7]: Kao i u primjeru 6, koristi se ime polja u sklopu pokazivačke notacije.

Primjer 8. *Omogućiti korisniku unos elemenata 2-D polja (matrice) realnih brojeva dimenzija* 4×4*. Ispisati na ekran elemente glavne dijagonale. Koristiti pokazivačku notaciju za pristup elementima 2-D polja.*

```
#include <stdio.h>

#define N 4

int main(void)
{
  float m[N][N];
  float (*p)[N]; //deklaracija pokazivaca na polje od 4 elementa tipa float
  int i, j;

  p = m; //usmjeravanje pokazivac na 2-D polje
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
```

```
      scanf("%f", *(p + i) + j);    //isto kao: &m[i][j] ili &p[i][j]

  for (i = 0; i < N; i++)
    printf("%f ", *(*(p + i) + i)); //isto kao: m[i][i] ili p[i][i]

  return 0;
}
```

Kratko obrazloženje [Primjer 8]: Kod deklaracije pokazivača na polje zagrade su nužne, jer uglate zagrade imaju veću razinu prvenstva u odnosu na zvjezdicu. Tako onda, `float *p[4]` predstavlja polje od 4 pokazivača na tip podatka `float`, a ne pokazivač na polje od 4 elementa tipa `float`.

POGLAVLJE 11

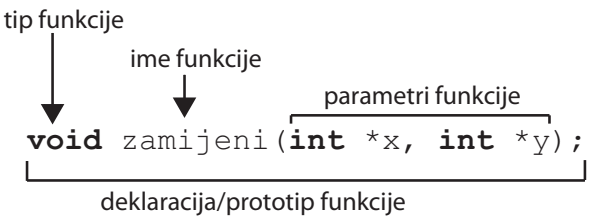
Funkcije: pokazivači i polja kao argumenti funkcija

Kako se prilikom poziva funkcije argumenti predaju po vrijednosti, tako ih nije moguće promijeniti unutar te funkcije. Međutim, u određenim situacijama potrebno je da funkcija promijeni vrijednost varijable ili varijabli koje joj se predaju (primjerice, na taj način moguće je zaobići ograničenje jedne povratne vrijednosti funkcije). Kako bi se navedeno postiglo nužno je koristiti pokazivače. Osim toga, ukoliko se funkciji želi predati polje kao argument, utoliko je opet nužno koristiti pokazivače. Primjer funkcije koja mijenja vrijednosti predanih joj varijabli predstavlja funkcija `scanf()` kojoj je nužno predati adrese varijabli u koje će biti spremljen unos sa standardnog ulaza, odnosno tipkovnice.

11.1 Pisanje i uporaba funkcija koje rade s pokazivačima i poljima

Prilikom poziva funkcija argumenti iz pozivajuće funkcije koriste se za inicijalizaciju parametara pozvane funkcije što znači da se samo prepisuju vrijednosti argumenata. Isto tako, pokazivači omogućuju neizravan (indirektan) pristup vrijednostima varijabli na koje su usmjereni. Prema tome, kako bi se omogućila promjena vrijednost varijable neke funkcije putem druge funkcije, takvoj se funkciji mora kao argument predati ne vrijednost nego adresa te varijable (ili pokazivač na nju). Stoga, parametar funkcije mora biti pokazivač na tip podataka te varijable. Na slici 11.1 prikazan je primjer prototipa funkcije koja ima pokazivače kao parametre. U izlistanju 11.1 je navedeno nekoliko primjera deklaracija, odnosno prototipa funkcija koje kao parametre imaju pokazivače.

Kako je ime nekog polja adresa prvog elementa, odnosno pokazivač na prvi element,



Slika 11.1: Prototip funkcije koja ništa ne vraća i ima kao parametre dva pokazivača na cjelobrojni tip podatka.

```
void funkcija(int *x);   //prototip funkcije tipa void koja ima pokazivac na tip podatka int

int f(double *t, int n); //prototip funkcije tipa int koja ima dva parametra;
                         //prvi, pokazivac na tip podatka double, a drugi tipa int
float funk(float *);     //prototip funkcije tipa float koja ima pokazivac na tip podatka float
                         //(ime parametra je izostavljeno)
```

Izlistanje 11.1: Primjeri deklaracije/prototipa funkcija koje imaju pokazivače kao parametre

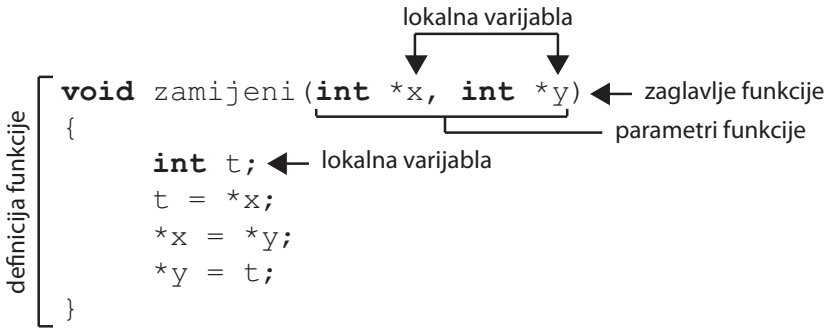
tako parametar funkcije kojoj se želi predati polje mora biti pokazivač na tip podataka elemenata danog polja. Kada se želi naglasiti da funkcija očekuje polje, a ne samo pokazivač na varijablu može se kao parametar funkcije navesti parametar odgovarajućeg tipa podatka kojeg slijedi par praznih uglatih zagrada. Samo u tom kontekstu takva konstrukcija ima isto značenje kao pokazivač. Jasno, takav oblik moguće je koristiti, kako prilikom deklaracije, tako i prilikom definicije u zaglavlju funkcije. Navedeno je prikazano primjerima u izlistanju 11.2.

```
double srv(double p[], int n);  //prototip funkcije tipa double koja ima dva parametra;
                                //prvi, pokazivac na tip podatka double, a drugi tipa int
float medijan(float [], int l); //prototip funkcije tipa float koja ima dva parametra;
                                //prvi, pokazivac na tip podatka float, a drugi tipa int
                                //(ime parametra pokazivaca je izostavljeno)
int duljina(char s[]);    //prototip funkcije tipa int koja za parametar ima pokazivac
                          //na tip podatka char
int usporedi(int v[], int w[], int n); //prototip funkcije tipa int koja ima tri parametra; prva
                                       //dva, pokazivaci na tip podatka int, a zadnji, tipa int
```

Izlistanje 11.2: Primjeri drugačijeg oblika prototipa funkcija koje imaju pokazivače kao parametre

Prilikom pisanja definicije funkcije koja ima pokazivače kao parametre, vrijednostima na koje su isti usmjereni, pristupa se na uobičajeni način, dereferenciranjem pokazivača (putem operatora dereferenciranja/indirekcije *). U slučaju polja, moguće je koristiti pokazivačku notaciju ili notaciju polja. Kako će takvi parametri sadržavati adrese predanih argumenata, sve promijene vrijednosti odraziti će se na argumente. Na slici 11.2 prikazan je primjer definicije funkcije koja ima pokazivače kao parametre. Izlistanjem 11.3 je prikazano nekoliko primjera definicija funkcija koje imaju pokazivače kao parametre.

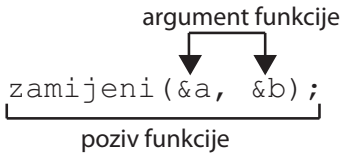
Funkcijama koje imaju pokazivače kao parametre treba kao argumente predati odgovarajuće pokazivače ili pak adrese. Najčešće nema potrebe za uvođenjem posebnog pokazivača na varijablu kojeg će se predati kao argument funkciji nego se funkciji predaje adresa te varijable (dobivenu adresnim operatorom &). Navedeno je prikazano primjerom



Slika 11.2: Definicija funkcije koja kao parametre ima dva pokazivačka na cjelobrojni tip podatka i koja im zamjenjuje vrijednosti.

```
void povecaj(int *x) //funkcija tipa void koja ima za parametar pokazivac na tip int
{
  *x += 5; //povecava vrijednost za 5 na adresi koja je spremljena u pokazivac x
}

void krug(float r, float *o, float *p) //funkcija tipa void koja ima tri parametra;
                                       //prvi, tipa float, a ostali pokazivaci na tip float
{
  *o = 2 * 3.14f * r; //racuna opseg kruga i rezultat sprema na
                      //adresu koja je spremljena u pokazivac o
  *p = r * r * 3.14f; //racuna povrsinu kruga i rezultat sprema na
                      //adresu koja je spremljena u pokazivac p
}

void uvecaj(double *a, int n, double v) //funkcija tipa void koja ima tri parametra;
                                        //prvi, pokazivac na tip double, drugi, tipa int
                                        //i zadnji tipa double
{
  int i;
  for (i = 0; i < n; i++)
    a[i] += v; //uvecava vrijednosti elemenata za vrijednost parametra v, redom
}

void zbroj(int v[], int w[], int r[], int n) //funkcija tipa void koja ima cetiri parametra;
                                             //prva, tri pokazivaci na tip int, a
                                             //zadnji, tipa int
{
  int i = 0;
  for (i = 0; i < n; i++)
    r[i] = v[i] + w[i]; //rezultat zbrajanja odgovarajucih parova vrijednosti
                        //polja v i w sprema se u polje r, redom
}
```

Izlistanje 11.3: Primjeri definicija funkcija koje imaju pokazivače kao parametre



Slika 11.3: Poziv funkcije. Funkciji su predana dva argumenta, oba predstavljaju adrese varijabli.

na slici 11.3. Ako funkcija očekuje polje kao argument, dovoljno joj je predati ime polja, jer ono predstavlja adresu prvog elementa tog polja.

Iako funkcije kojima se predaju pokazivači mogu mijenjati vrijednosti objekata na koje su usmjereni ti pokazivači, unutar tih funkcija nije moguće promijeniti vrijednost predanih pokazivača, jer se argumenti predaju po vrijednosti. To znači da se samo vrijednosti pokazivača, odnosno adrese, kopiraju/prepisuju u parametre funkcije.

11.2 Ključna riječ `const` s pokazivačima

Ključnu riječ `const` kao kvalifikatora može se koristiti i pri deklaraciji pokazivačkih varijabli. Tako je, primjerice, moguće deklarirati pokazivač na konstantan tip i konstantan pokazivač na neki tip. U oba slučaja je deklaracija slična, a razlika je u položaju ključne riječi `const`. Opći oblik deklaracije pokazivača na konstantan tip je:

```
    const tip_podatka *ime_pokazivaca;
```

U navedenom slučaju, kvalifikator `const` se odnosi na tip podatka, a ne na pokazivač.

Prema tome, radi se o "pokazivač na `const tip_podatka`". To znači kako je tako deklariran pokazivač moguće preusmjeriti, ali ga nije moguće koristiti za izmjenu vrijednosti objekta na koji pokazuje ili na koji je usmjeren. Treba spomenuti kako se `const` može navesti i nakon tipa podatka te kako u oba slučaja ima isto značenje. Može se još zaključiti kako je jedino takav pokazivač moguće usmjeriti na objekt, primjerice varijablu, kvalificiran s `const`.

Ukoliko se želi učiniti pokazivač konstantnim, ključnu riječ `const` treba navesti poslije zvjezdice i prije imena:

```
tip_podatka *const ime_pokazivaca = vrijednost;
```

Sada se ona odnosi na pokazivač. Shodno tome, radi se o "konstantni pokazivač na `tip_podatka`". Važno je naglasiti da je konstantni pokazivač nužno inicijalizirati, jer naknadno pridruživanje vrijednosti (adrese) nije moguće, odnosno rezultirati će greškom prilikom prevođenja. Opisana deklaracija čini pokazivač konstantnim, ali ga se može koristiti za izmjenu vrijednosti objekta na koji je usmjeren.

11.3 Zaštita elemenata polja od izmjena u funkciji

S obzirom da je jedini način predaje polja funkciji, preko pokazivača, funkcija može mijenjati vrijednosti elemenata tog polja. Međutim, za obavljanje nekih zadataka dovoljno je da funkcija samo dohvaća vrijednosti elemenata, a ne i mijenja ih. Kako bi se naglasilo da funkcija ne mijenja vrijednosti elemenata polja te kako bi se izbjegle slučajne promijene vrijednosti prilikom pisanja (definicije) funkcije moguće je koristiti kao parametar pokazivač na konstantan tip. Shodno tome će se u funkciji prema predanom polju postupati kao prema polju konstantnih podataka i svaki pokušaj promijene vrijednosti elemenata polja rezultirati će greškom. Uporaba ključne riječi `const` u navedenom kontekstu prikazana je primjerom teksta programa u izlistanju 11.4.

```
void funk(const double a[], int n); //deklaracija/prototip funkcije

int main(void)
{
  double polje[6] = {1.2, 2.3, 3.4, 4.5, 5.6, 6.7};
  funk(polje, 6);

  return 0;
}

void funk(const double a[], int n)  //definicija funkcije
{
  int i;
  for (i = 0; i < n; i++)
    a[i] *= 1.2; //GRESKA! nije dozvoljeno mijenjanje vrijednosti elemenata
}
```

Izlistanje 11.4: Primjeri uporabe ključne riječi `const` u svrhu zaštite elemenata polja od izmjena

Osim što navedeni pristup onemoguće izmjenu elemenata, on pruža mogućnost rada s poljima konstantnih elemenata. Takva polja nije moguće predati funkcijama koje nemaju parametar koji je pokazivač na konstantan tip podatka.

11.4 Riješeni primjeri

U nastavku je navedeno nekoliko primjera u kojima je nužno napisati odgovarajuću funkciju koja rješava dani zadatak. Imena svih funkcija kao i pokazni primjeri njihove uporabe proizvoljno su odabrani s obzirom da nisu u primjerima eksplicitno definirani, a tipovi podataka su odabrani shodno potrebama zadataka koje funkcije trebaju obaviti.

Primjer 1. *Napisati funkciju koja će kvadrirati vrijednost dane cjelobrojne varijable. Funkcija ne treba ništa vratiti nego treba promjenu načiniti na danoj varijabli. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

//definicija f-je koja ne vraca nista i prima pokazivac na tip int
void kvadriraj(int *p)
{//pokazivacka varijabla p je lokalna za f-ju kvadriraj()
  *p *= *p; //kvadrat vrijednosti na adresi koja je spremljena u pokazivac p
}

int main(void)
{
  int a = 2;
  int *q = &a;
  kvadriraj(q); //predajemo pokazivac q (adresu varijable a) kao argument f-je
      //f-ja radi s kopijom vrijednosti pokazivaca q pa ga ne moze promijeniti

  printf("%d\n", a); //ispisuje: 4

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Kako funkcija ne treba ništa vratiti, ona je tipa `void`. Funkcija kao parametar ima pokazivač pa ga je potrebno dereferencirati kako bi se pristupilo vrijednosti na adresi koju sadrži.

Primjer 2. *Napisati funkciju koja će izračunati opseg i površinu trokuta. Funkciji se predaju duljine stranica kao i varijable, odnosno adrese na koje će biti spremljene izračunate vrijednosti. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
#include <math.h> //sqrt()
//deklaracija f-je koja ne vraca nista i koja ima pet parametara;
//prva tri su tipa float, dok su preostala dva pokazivaci na tip float
void trokut(float a, float b, float c, float *o, float *p);

int main(void)
{
  float ta = 2, tb = 4, tc = 5;
  float opseg, povrsina;
  //funkciji se predaju, uz duljine stranica, i adrese varijabli opseg i povrsina
  trokut(ta, tb, tc, &opseg, &povrsina);
  printf("Opseg trokuta: %f\nPovrsina trokuta: %f", opseg, povrsina);

  return 0;
}

void trokut(float a, float b, float c, float *o, float *p)
{
  float s;

  *o = a + b + c; //racunanje opsega trokuta
  s = *o / 2;     //racunanje poluopsega
```

```
  *p = sqrt(s * (s - a) * (s - b) * (s - c)); //racunanje povrsine trokuta - Heronova formula
}
```

Kratko obrazloženje [Primjer 2]: Funkcija treba izračunati dvije vrijednosti, koje treba dostaviti pozivajućoj funkciji (funkcija `main()` u ovom slučaju). Funkcije mogu imati samo jednu povratnu vrijednosti pa je jedan od načina zaobilaska tog ograničenja spremanje izračunatih vrijednosti u varijable odnosno adrese koje joj se predaju.

Primjer 3. *Napisati funkciju koja će izračunati i vratiti zbroj vrijednosti elemenata predanog joj polja realnih brojeva. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
//ime polja je adresa 1. elementa pa stoga f-ja ima pokazivac kao parametar,
//a i cijeli broj koji predstavlja velicinu polja
float zbrojelem(float *a, int n); //ili: float zbrojelem(const float *a, int n);

int main(void)
{
  float fa[10] = {2.3, 1.43, 6.4, 8.09, 1.24, 5.5, 6.2, 3.1, 9.9, 1.2};
  float z, *p;

  p = fa; //fa == &fa[0]
  //funkciji se predaje pokazivac na prvi element polja (adresu 1. elementa) i velicinu polja
  z = zbrojelem(p, 10);  //ili: z = zbrojelem(a, 10);
  printf("Zbroj elemenata polja je: %f\n", z);

  return 0;
}

float zbrojelem(float *a, int n) //ili: float zbrojelem(const float *a, int n)
{
  float z = 0;
  int i;
  for (i = 0; i < n; i++)
    z += a[i]; //ili: z += *(a + i);

  return z; //vraca zbroj elemenata polja
}
```

Kratko obrazloženje [Primjer 3]: Funkcija ima dva parametra, jer polje odnosno pokazivač ne sadrži nikakvu informaciju o broju elemenata pa je prema tome potrebno funkciji prosljediti i informaciju o veličini polja. Za pristup elementima polja, unutar definicije funkcije, koristi se notacija polja iako je moguće koristiti pokazivačku notaciju.

Primjer 4. *Napisati funkciju koja će pronaći i vratiti vrijednost najmanjeg elementa predanog joj polja realnih brojeva. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

float minelem(const float *a, int n);

int main(void)
{
  float fa[10] = {2.3, 1.43, 6.4, 8.09, 1.24, 5.5, 6.2, 3.1, 9.9, 1.2};
  float min;
  //funkciji se predaje polje (adresu 1. elementa) i velicinu polja
  min = minelem(fa, 10);
  printf("Najmanja vrijednost u polju je: %f\n", min);

  return 0;
}

float minelem(const float *a, int n)
```

```
{
  float min = a[0];
  int i;
  for (i = 1; i < n; i++)
    if (a[i] < min)
      min = a[i];

  return min; //vraca vrijednost najmanjeg elementa polja
}
```

Kratko obrazloženje [Primjer 4]: Ključnom riječi `const` je naglašeno kako funkcija ne mijenja elemente polja s obzirom da funkcija treba samo pronaći i vratiti element s najmanjom vrijednosti. Svaki pokušaj promijene vrijednosti nekog od elemenata polja, unutar definicije funkcije, rezultirao bi greškom prilikom prevođenja.

Primjer 5. *Napisati funkciju koja će vratiti broj elemenata predanog joj polja cijelih brojeva koji imaju parne vrijednosti. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int parnielem(const int a[], int n);

int main(void)
{
  int ia[] = {2, 3, 6, 9, 4, 5, 6, 3, 9, 1, 0, 7};
  int brparni, v;

  v = sizeof (ia) / sizeof (int); //racunanje velicine odnosno broja elemenata polja
  brparni = parnielem(ia, v);
  printf("Broj parnih elemenata u polju je: %d\n", brparni);

  return 0;
}

int parnielem(const int a[], int n)
{
  int i, br = 0;
  for (i = 0; i < n; i++)
    if (a[i] % 2 == 0 && a[i] != 0) //nula nije ni parna ni neparna
      br++;

  return br; //vraca broj elemenata koji imaju parnu vrijednost
}
```

Kratko obrazloženje [Primjer 5]: Izračun veličine polja odnosno broja elemenata ostvaren je pomoću operatora `sizeof`. On daje veličinu svog operanda u bajtima. U primjeru se koristi dva puta, u prvom slučaju operand je polje (uporaba zagrada nije nužna), a u drugom, tip podatka, `int` (uporaba zagrada je nužna).

Primjer 6. *Napisati funkciju koja će slijedno pretražiti i vratiti vrijednost 1 ako u danom polju cijelih brojeva postoji traženi broj, inače će vratiti vrijednost 0. Traženi broj također se predaje kao argument funkciji. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
//f-ja zahtjeva osim polja i njegove velicine i broj kojeg se trazi
int pretrazi(int a[], int n, int t); //ili: int pretrazi(const int a[], int n, int t);

int main(void)
{
  int ia[12] = {2, 3, 6, 9, 4, 5, 6, 3, 9, 1, 0, 7};
  int f;

  f = pretrazi(ia, 12, 5);
```

```
  printf("%sostoji trazeni broj", f == 0 ? "Ne p" : "P");

  return 0;
}

int pretrazi(int a[], int n, int t) //ili: int pretrazi(const int a[], int n, int t)
{//linearno pretrazivanje polja - trazi se vrijednost t
  int i;
  for (i = 0; i < n; i++)
    if (a[i] == t)
      return 1; //vraca 1 u slucaju da je pronadena trazena vrijednost

  return 0; //inace vraca 0
}
```

Kratko obrazloženje [Primjer 6]: Kod slijednog pretraživanja, redom se provjerava vrijednost svakog elementa polja te ako ona odgovara traženoj vrijednosti pretraga se prekida. Navedeno je u funkciji postignuto tako što u slučaju kada vrijednost nekog elementa polja odgovara traženoj, ona vraća vrijednost jedan, čime se ujedno završava izvođenje funkcije.

Primjer 7. *Napisati funkciju koja će prepisati vrijednosti zadanog broja elemenata n, jednog polja u drugo polje (oba su polja cijelih brojeva). Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

void kopiraj(const int izv[], int odr[], int n);

int main(void)
{
  int ia1[12] = {2, 3, 6, 9, 4, 5, 6, 3, 9, 1, 0, 7};
  int ia2[12] = {1, 7, 6, 9, 8, 3, 3, 1, 0, 1};
  int i;

  kopiraj(ia1, ia2, 5); //kopira prvih 5 brojeva iz prvog u drugo polje
  for (i = 0; i < 12; i++)
    printf("ia1[%d] = %d \t ia2[%d] = %d\n", i, ia1[i], i, ia2[i]);

  return 0;
}

void kopiraj(const int izv[], int odr[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    odr[i] = izv[i];
}
```

Kratko obrazloženje [Primjer 7]: Prvo polje je zaštićeno, jer se iz njega samo prepisuju vrijednosti elemenata u drugo polje koje ne smije biti zaštićeno.

Primjer 8. *Napisati funkciju koja će vratiti duljinu predanog joj stringa. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int brznak(const char s[]);

int main(void)
{
  char str[100] = "Koliko li ima ovdje znakova?";
  //povratnu vrijednost f-je brznak() koristimo kao argument f-je printf()
```

```
  //prije izvrsavanja f-je printf() biti ce pozvana i izvrsena f-ja brznak()
  printf("Ima %d znakova.", brznak(str));

  return 0;
}

int brznak(const char s[])
{
  int i, br = 0;
  for (i = 0; s[i] != '\0'; i++) //ne broji null znak
    br++;

  return br; //br == i
}
```

Kratko obrazloženje [Primjer 8]: Duljinu stringa predstavlja broj znakova do *null* znaka (`'\0'`). Prema tome, funkcija redom broji sve znakove do pojave *null* znaka. Treba napomenuti kako istu funkcionalnost ima funkcija `strlen()` iz standardne biblioteke, za koju je potrebno uključiti zaglavlje string.h.

Primjer 9. *Napisati funkciju koja će provjeriti postoji li dani string unutar drugog danog stringa. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

int podstring(const char t[], const char s[]);
int brznak(const char s[]); //pomocna funkcija za odredjivanje broja znakova danog stringa

int main(void)
{
  char a[] = "Programiranje 1 - laboratorijske vjezbe";
  char b[] = "bor";

  int o = podstring(a, b);
  printf("Trazeni string %sostoji.", o == 1 ? "p" : "ne p");

  return 0;
}

int podstring(const char s[], const char t[])
{
  int p;
  int dt, ds;
  int i, j;

  dt = brznak(t); //duljina trazenog stringa
  ds = brznak(s); //duljina stringa kojeg se pretrazuje
  //Pretrazuje string segment po segment s time da se segmenti preklapaju
  for (i = 0; i < ds - dt + 1; i++) {
    p = 1; //signalizira da je trazeni string pronadjen
    for (j = 0; j < dt; j++)
      if (s[i + j] != t[j]) {
        p = 0;
  break;
      }

    if (p == 1)
      break; //ako je trazeni string pronadjen prekida se pretraga
  }

  return p;
}

int brznak(const char s[])
{
  int br = 0;
  while (*s++ != '\0') //isto kao: *(s++) != '\0'
    br++;
```

```
  return br;
}
```

Kratko obrazloženje [Primjer 9]: Pretraga se odvija dio po dio. String kojeg se pretražuje obilazi se znak po znak te se svaki puta provjerava slijedećih m znakova (uključujući i trenutni znak), gdje je m duljina traženog stringa. Provjera se opet obavlja znak po znak i ako neki od odgovarajućih znakova nisu jednaki, pretraga se odmah nastavlja za jedan znak dalje u stringu kojeg se pretražuje. U primjeru se koristi i pomoćna funkcija koja vraća duljinu predanog joj stringa, ali je drukčije napisana u odnosu na primjer 8. Treba primijetiti kako ključna riječ `const` ne čini pokazivač konstantnim pa je moguće na njega primijeniti operator uvećavanja. Na taj način se pokazivač usmjerava sa znaka na znak.

Primjer 10a. *Napisati funkciju koja će izračunati i vratiti standardnu devijaciju vrijednosti elemenata predanog joj polja realnih brojeva. Standardnu devijaciju računati prema:*

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}\,,$$

gdje μ predstavlja aritmetičku sredinu. Na primjeru u funkciji `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
#include <math.h> //sqrt()

double stddev(const double p[], int n);

int main(void)
{
  double a[20];
  int i, n;

  printf("Unesite vrijednost, koliko zelite unijeti elemenata.");
  while (1) {
    scanf("%d", &n);
    if (n < 5 || n > 20)
      printf("Unesite cjelobrojnu vrijednost iz intervala [5, 20]");
    else
      break;
  }

  printf("Unesite redom vrijednosti elemenata:\n");
  for (i = 0; i < n; i++)
    scanf("%lf", &a[i]);

  printf("Standardna devijacija = %f\n", stddev(a, n));

  return 0;
}

double stddev(const double p[], int n)
{
  double svr = 0;
  double var = 0
  int i;

  //izracun srednje vrijednosti
  for (i = 0; i < n; i++)
    svr += p[i];
  svr /= n;

  //izracun zbroja kvadrata razlike elemenata od srednje vrijednosti
```

```
  for (i = 0; i < n; i++)
    var += (p[i] - svr) * (p[i] - svr);
  var /= n; //varijanca; kvadrirana standardna devijacija

  return sd = sqrt(var);  //standardna devijacija
}
```

Kratko obrazloženje [Primjer 10a]: Prema formuli x_i odgovara vrijednosti elemenata polja s indeksom $i-1$, jer je u formuli početna vrijednost $i=1$. Prototip funkcije `sqrt()` je `double sqrt(double)`, a ona računa i vraća vrijednost kvadratnog korijena prednog joj argumenta.

Primjer 10b. *Napisati funkciju koja će izračunati i vratiti standardnu devijaciju vrijednosti elemenata predanog joj polja realnih brojeva. Standardnu devijaciju računati prema:*

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}\,,$$

gdje $\overline{x}$ predstavlja aritmetičku sredinu. Na primjeru u funkciji `main()` pokazati uporabu napisane funkcije.

```
#include <stdio.h>
#include <math.h> //sqrt(), pow()

double stddev(const double seq[], int n); //standardna devijacija
double var(const double seq[], int n);    //varijanca
double mean(const double seq[], int n);   //aritmeticka sredina

int main(void)
{
  double sample[6] = {1.1, 1.2, 1.1, 1.4, 0.9, 0.8};
  printf("Standard deviation = %f\n", stddev(sample, 6));

  return 0;
}

double stddev(const double seq[], int n)
{
  double v = var(seq, n);
  double sd = sqrt(v);
  return sd;
}

double var(const double seq[], int n)
{
  double m = mean(seq, n);

  double v = 0;
  int i;
  for (i = 0; i < n; i++)
    v += pow(seq[i] - m, 2);
  v /= n - 1;

  return v;
}

double mean(const double seq[], int n)
{
  double m = 0;
  int i;
  for (i = 0; i < n; i++)
    m += seq[i];
  m /= n;
```

```
  return m;
}
```

Kratko obrazloženje [Primjer 10b]: Treba primijetiti blagu razliku u formuli za izračun standradne devijacije. U ovom slučaju se radi o ispravljenoj standardnoj devijaciji za uzorak populacije (engl. *unbiased/corrected sample standard deviation*). Međutim, bitnije je primijetiti razlike u pristupu ostvarivanje izračuna standardne devijacije. U odnosu na prethodni primjer, postoje tri funkcije koje su relativno malene. Svaka obavlja jedan zadatak i moguće je ih po potrebi koristiti dalje. Uz to, tekst programa je razumljiviji. Za razliku od prethodnih primjera, korišteni su identifikatori (imena varijabli, funkcija i sličnog) na engleskom jeziku.

Primjer 11. *Napisati funkciju koja kritptira/dekriptira predani joj string. Funkcija koristi Cezarovu šifru*[1] *uz zadani ključ. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
#include <ctype.h> //tolower()

void encrypt(char str[], int key);

int main(void)
{
  char str[] = "FERIT + Osijek";

  encrypt(str, 15);  //kriptiranje
  puts(str); //ispis sifrata
  encrypt(str, -15); //dekriptiranje
  puts(str); //ispis otvorenog teksta

  return 0;
}

void encrypt(char str[], int key)
{
  //deklaracija unutar funkcije, jer dalje nije potrebna
  char codedChar(char c, int key);
  //korekcija kljuca ukoliko je njegova apsolutna vrijednost veca od ili jednaka 26
  int adjustedKey = key % 26;

  int i;
  for (i = 0; str[i] != '\0'; i++)
    if (tolower(str[i]) >= 'a' && tolower(str[i]) <= 'z')
      str[i] = codedChar(str[i], adjustedKey); //vrsi se kriptiranje samo slova
}

char codedChar(char c, int key)
{
  int pos = tolower(c) - 'a'; //pozicija slova u abecedi
  int offset = pos + key;     //pozicija nakon primjene kljuca

  //prilagodba kljuca
  if (offset < 0)
    key += 26;
  else if (offset >= 26)
    key -= 26;

  return c + key;
}
```

Kratko obrazloženje [Primjer 11]: Cezarova šifra je jednostavna supstitucijska šifra, gdje se svako slovo zamijenjuje sa slovom tri mjesta dalje. Pretpostavka je kako se abeceda ciklički nastavlja. Uobičajeno se Cezarovom šifrom nazivaju i iste šifre s pomakom

[1] Vidi https://web.math.pmf.unizg.hr/~duje/kript/supst.html

različitim od tri. U primjeru, prije kriptiranja/dekriptiranja provjerava se predstavlja li trenutni znak slovo. Ukoliko je slovo, poziva se funkcija koja računa njegovu supstituciju/zamjenu na temelju zadanog ključa. Ključ predstavlja pomak. Kriptirani tekst (šifrat) jednostavno se dekriptira (dobiva se originalni ili otvoreni tekst) istim ključem sa suprotnim predznakom. Može se primijetiti uporaba funkcije `tolower()` (opisana u ctype.h) koja vraća malo slovo predanog joj argumenta. Time je izbjegnuta potreba za posebnim rukovanjem malim i velikim slovima.

Primjer 12. *Napisati funkciju koja računa medijan vrijednosti elemenata predanog joj polja realnih brojeva. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>

double median(double seq[], int n);
void bubbleSort(double seq[], int n);

int main(void)
{
  double sequence[6] = {1.1, 1.2, 1.1, 1.4, 0.9, 0.8};
  printf("Median = %f\n", median(sequence, 6));
  return 0;
}

double median(double seq[], int n)
{
  double med;
  bubbleSort(seq, n);

  if (n % 2 != 0)
    med = seq[n/2];
  else
    med = (seq[n/2 - 1] + seq[n/2]) / 2;
  return med;
}

void bubbleSort(double seq[], int n)
{
  void swap(double *a, double *b);
  int f, i;
  do {
    f = 0;
    for (i = 0; i < n - 1; i++)
      if (seq[i] < seq[i + 1]) {
        swap(&seq[i], &seq[i + 1]);
        f = 1;
      }
  } while (f == 1);
}

void swap(double *a, double *b)
{
  double temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
```

Kratko obrazloženje [Primjer 12]: Medijan je vrijednost koja dijeli popis/slijed brojeva na gornju i donju polovicu. Podrazumijeva se kako su brojevi sortirani uzlazno (od najmanjeg prema najvećem), iako mogu biti i silazno sortirani. Ako je neparan broj brojeva medijan je središnji član, a u slučaju parnog broja medijan je srednja vrijednost dva središnja člana. Primjerice, medijan popisa 1; 2; 5; 6; 9 je 5, dok je medijan popisa 3; 5; 7; 9; 12; 15 jednak (7 + 9)/2 = 8.

Primjer 13. *Napisati funkciju koja računa i vraća*

$$f(\mathbf{x}) = nx_n^{-3} + \prod_{i=1}^{n-1}\left[ix_i^{-2} + \cos\left(\frac{\pi}{x_i + 1}\right)\right], \quad \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n \ .$$

Na primjeru u funkciji `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdio.h>
#include <math.h>

#define PI 3.1415
#define N 10

double f(const double x[], int n);

int main(void)
{
  double a[N];
  int i;

  for (i = 0; i < N; i++) {
    do {
      scanf("%lf", &a[i]);
    } while (a[i] <= 0 || a[i] >= 5);
  }

  printf("%f\n", f(a, N));
  return 0;
}

double f(const double x[], int n)
{
  double a, t1, t2, p = 1;
  int i;

  a = n * pow(x[n-1], -3);
  for (i = 0; i < n-1; i++) {
    t1 = (i+1) * pow(x[i], -2);
    t2 = cos(PI / (x[i]-1);
    p *= t1 + t2;
  }

  return a + p;

}
```

Kratko obrazloženje [Primjer 13]: Formula u potpunosti opisuje definiciju funkcije koju treba napisati. Kao i u primjeru 10, treba biti pažljiv s indeksima polja. Za izračun kosinusa korištena je funkcija opisana u zaglavnoj datoteci math.h čiji je prototip: `double cos(double x)`, dok je za izračun potencija korištena funkcija opisana u istoj zaglavnoj datoteci čiji je prototip: `double pow(double x, double p)`.

POGLAVLJE 12

Stvaranje pseudo-slučajnih brojeva

Slučajni (engl. *random*) brojevi predstavljaju slijed/niz koji ne pokazuje nikakav uzorak u pojavljivanju tih brojeva. Računalno (algoritamski) stvoreni slučajni brojevi samo naizgled ne pokazuju uzorak, te su stoga pseudo-slučajni. Oni imaju vrlo opsežnu primjenu koja seže od kriptografije, metaheurističkih algoritama te računalnih igara i simulacija pa do nekih statističkih metoda analize.

12.1 Pseudo-slučajni brojevi u programskom jeziku C

Za dobivanje pseudo-slučajnih (u nastavku slučajni) brojeva u programskom jeziku C (također i u C++) uobičajeno se koristi funkcija `rand()`. Ona vraća cjelobrojnu vrijednost između 0 i vrijednosti koja odgovara konstanti `RAND_MAX` (prema standardu jezika je jednaka barem 32767), uključujući granice.

Kako bi se postavila početna vrijednost na temelju koje će se stvarati slučajni brojevi, koristi se funkcija `srand()` koja prima cijeli broj (bez predznaka). Opisi navedenih funkcija nalaze se u zaglavnoj datoteci stdlib.h. Prototipovi funkcija `rand()` i `srand()` su:

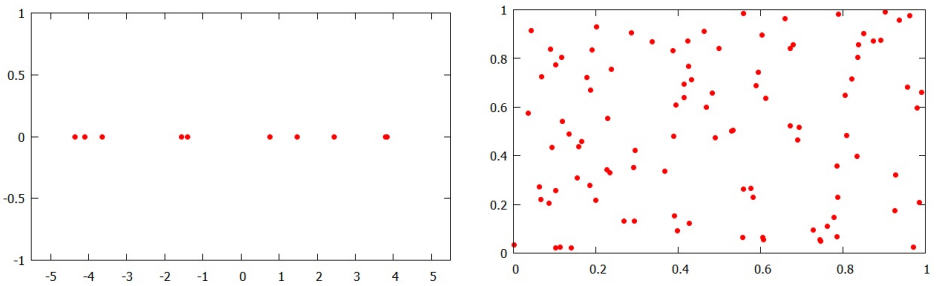
```
int rand(void);
void srand(unsigned int s);
```

Prije poziva funkcije `rand()` u pravilu se inicijalizira generator pseudo-slučajnih brojeva (engl. *pseudo random number generator*). Zapravo, postavlja se vrijednosti na temelju koje će se u funkciji `rand()` izračunati (upravo zbog toga su pseudo-slučajni) "slučajni" broj kojeg će na koncu vratiti. Kao što je navedeno, ta početna vrijednost se postavlja putem funkcije `srand()`. Poželjno je da vrijednost koja se predaje funkciji `srand()` bude, također, slučajna. Stoga se često koristi trenutno vrijeme sustava kojeg je moguće dobiti pozivom funkcije `time()`. Općenito, ona vraća trenutno kalendarsko vrijeme, a uglavnom se radi o vrijednost koja predstavlja broj sekundi koje su protekle od (0:00h) 1.1.1970. godine. Opis funkcije `time()` se nalazi u zaglavnoj datoteci time.h, a njen prototip je:

```
time_t time(time_t *t);
```

Povratni tip podatka funkcije `time()`, `time_t`, u pravilu nije ništa više nego drugo ime za neki cjelobrojni tip podatka. Argument te funkcije može biti pokazivač na neku varijablu tipa `time_t` ili tzv. *null* pokazivač (`NULL`). Ako nije `NULL`, ista vrijednost kao povratna sprema se na adresu varijable na koju pokazuje predani pokazivač.

Nakon inicijalizacije (nije nužna, ali je uobičajena), odnosno poziva funkcije `srand()` s odgovarajućim argumentom, može se pozivati funkcija `rand()` proizvoljan broj puta te



(a) (b)

Slika 12.1: Grafički prikaz generiranih pseudo-slučajnih točaka (a) u $[-5, 5]$ i (b) u $[0, 1]^2$.

se dobiva slijed slučajnih brojeva. Ako bi se ponovila inicijalizacija s istom vrijednosti te ponovo pozivala funkcija `rand()`, dobio bi se isti slijed slučajnih brojeva.

Na slici 12.1 grafički su prikazani skupovi od 10 i 100 slučajno generiranih točaka u $[-5, 5]$ i $[0, 1]^2$, redom. Navedeni su dobiveni na temelju primjera koji slijede.

Generiranje točaka u jednoj dimenziji

```
#include<stdlib.h>
#include<time.h>

#define N 10

int main(void)
{
  double slubr[N];
  int i;

  srand((unsigned)time(NULL));
  for (i = 0; i < N; i++)
    slubr[i] = -5 + (double)rand() /
      RAND_MAX * 10;

  return 0;
}
```

Generiranje točaka u dvije dimenzije

```
#include<stdlib.h>
#include<time.h>

#define N 100
#define D 2
#define X 0
#define Y 1

int main(void)
{
  double slubr[N][D];
  int i, j;

  srand((unsigned)time(NULL));
  for (i = 0; i < N; i++) {
    slubr[i][X] = (double)rand() / RAND_MAX;
    slubr[i][Y] = (double)rand() / RAND_MAX;
  }

  return 0;
}
```

Ukoliko se ne pozove funkcija `srand()`, dobiva se slijed brojeva kao da je pozvana navedena funkcija s argumentom 1. To znači kako je vrijednost na temelju koje se računaju brojevi početno zadana kao 1, a funkcijom `srand()` ju je moguće promijeniti. Najčešće se jednom pozove funkcija `srand()` i to unutar funkcije `main()` bez obzira u kojoj se funkciji (ili funkcijama) koristi `rand()`.

12.2 Riješeni primjeri

Slijedi nekoliko primjera za čije je rješavanje potrebno koristiti funkciju `rand()`. Naravno, korištene su i funkcije `srand()` te `time()` za potrebe inicijalizacije generatora pseudo-slučajnih brojeva. U primjerima u kojima je potrebno prikazati uporabu napisanih funkcija nisu eksplicitno navedene granice unutar kojih je potrebno generirati pseudo-slučajne brojeve pa su one postavljene na proizvoljne vrijednosti.

Primjer 1. *Deklarirati polje od 200 cijelih brojeva i popuniti ga pseudo-slučajnim brojevima.*

```
#include <stdlib.h> //rand() i srand()
#include <time.h>   //time()


int main(void)
{
  int a[200];
  int i;

  //inicijalizacija generatora pseudo-slucajnih brojeva
  srand((unsigned)time(NULL));

  //pozivanje funkcije rand() 200 puta
  for (i = 0; i < 200; i++)
    a[i] = rand();

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Polje će biti popunjeno vrijednostima iz $[0, \alpha] \subset \mathbb{Z}$, gdje α odgovara konstanti `RAND_MAX`. Inicijalizacija generatora pseudo-slučajnih brojeva vrši se prije prvog poziva funkcije `rand()`, i to samo jednom (u pravilu nikada nema potrebe za ponovnim incijaliziranjem).

Primjer 2. *Deklarirati polje od 350 cijelih brojeva i popuniti ga pseudo-slučajnim brojevima iz* $[0, 100] \subset \mathbb{Z}$.

```
#include <stdlib.h>
#include <time.h>

#define N 350

int main(void)
{
  int a[N], i;

  srand((unsigned)time(NULL));
  //vrijednost izraz desno od operatora pridruzivanja je u intervalu [0, 100]
  for (i = 0; i < N; i++)
    a[i] = rand() % 101;

  //moguce je isto postici na sljedeci nacin:
  for (i = 0; i < N; i++)
    a[i] = (float)rand() / RAND_MAX * 100;

  //takoder i na ovaj nacin:
  for (i = 0; i < N; i++)
    do {
      a[i] = rand();
    } while (a[i] > 100);

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Generiranje pseudo-slučajnih brojeva unutar zadanog intervala moguće je postići na različite načine. Od navedenih svaki ima svoje prednosti i nedostatke. Prva je najjednostavnija za napisati, ali ovisno o tome je li gornja granica djeljiva s `RAND_MAX` moguće je da se neki ostaci pri dijeljenju mogu pojaviti više puta od drugih. Glavni nedostatak drugog pristupa je što je vrlo mala vjerojatnost da će vrijednost izraza desno od operatora pridruživanja biti jednaka gornjoj granici. Treba napomenuti kako je pretvaranje (engl. *casting*) u `float` (ili `double`) nužno kako se ne bi obavila aritmetika (dijeljenje) nad cjelobrojnim vrijednostima. Treći pristup može zahtijevati značajno

veći broj poziva funkcije `rand()` nego što je potrebno generirati brojeva.

Primjer 3. *Deklarirati polje od 175 realnih brojeva i popuniti ga pseudo-slučajnim brojevima iz* $[5.5, 50.7] \subset \mathbb{R}$.

```
#include <stdlib.h>
#include <time.h>

#define N  175
#define DG 5.5
#define GG 50.7

int main(void)
{
  float a[N];
  int i;

  srand((unsigned)time(NULL));
  //vrijednost izraz desno od operatora
  //dodijele je unutar intervala [5.5, 50.7]
  for (i = 0; i < N; i++)
    a[i] = DG + (float)rand() / RAND_MAX * (GG - DG);

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Općenito, generiranje realnih vrijednosti [DG, GG] svodi se na $DG + \mathcal{U}[0,1] \cdot (GG - DG)$, gdje je $\mathcal{U}[0,1]$ slučajna varijabla prema uniformnoj razdiobi u $[0,1]$. Dobivanje realnog pseudo-slučajnog broja u $[0,1]$ ostvareno je s `(float)rand() / RAND_MAX`.

Primjer 4. *Deklarirati matricu (2-D polje) realnih brojeva dimenzija* 5×7 *i popuniti ju pseudo-slučajnim brojevima iz* $[0, 10.5] \subset \mathbb{R}$.

```
#include <stdlib.h>
#include <time.h>

int main(void)
{
  float m[5][7];
  int i, j;

  srand((unsigned)time(NULL));
  //vrijednost izraz desno od operatora
  //dodijele je unutar intervala [0, 10.5]
  for (i = 0; i < 5; i++)
    for (j = 0; j < 7; j++)
      m[i][j] = (float)rand() / RAND_MAX * 10.5;

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Popunjavanje 2-D polja je analogno popunjavanju 1-D polja, samo treba proći kroz sve elemente polja.

Primjer 5. *Deklarirati matricu (2-D polje) realnih brojeva dimenzija* 10×10 *i popuniti ju pseudo-slučajnim brojevima iz* $[-100, 100] \subset \mathbb{R}$.

```
#include <stdlib.h>
#include <time.h>

#define VEL 10
```

```
int main(void)
{
  double m[VEL][VEL];
  int i, j;

  srand((unsigned)time(NULL));
  //vrijednost izraz desno od operatora
  //dodijele je unutar intervala [-100, 100]
  for (i = 0; i < VEL; i++)
    for (j = 0; j < VEL; j++)
      m[i][j] = -100 + (double)rand() / RAND_MAX * 200; //200, jer je 100 - -100

  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Pogledati obrazloženja za primjere 3 i 4.

Primjer 6. *Napisati funkciju koja vraća cijeli pseudo-slučajni broj iz* $[0, x]$*, gdje je x cjelobrojni parametar funkcije. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdlib.h>
#include <time.h>

int rand_x(int x)
{
  return rand() % (x + 1);
}

int main(void)
{
  int a[100];
  int i;

  srand((unsigned)time(NULL));
  //popunjavanje polja slucajnim vrijednostima u [0, 50]
  for (i = 0; i < 100; i++)
    *(a + i) = rand_x(50);

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: S obzirom da je donja granica fiksna, funkcija ima samo jedan parametar—gornju granicu. Generiranje pseudo-slučajnog broja unutar zadanog intervala se moglo ostvariti i na neki drugi način (pogledati obrazloženje za primjer 2).

Primjer 7. *Napisati funkciju koja vraća realni pseudo-slučajni broj iz* $[a, b]$*, gdje su a i b realni parametri funkcije. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdlib.h>
#include <time.h>

#define VP 500

double rand_ab(double a, double b)
{
  return a + (double)rand() / RAND_MAX * (b - a);
}

int main(void)
{
  double p[VP];
  int i;

  srand((unsigned)time(NULL));
```

```
  //popunjavanje polja slucajnim vrijednostima u [-5.5, 10]
  for (i = 0; i < VP; i++)
    p[i] = rand_ab(-5.5, 10);

  return 0;
}
```

Kratko obrazloženje [Primjer 7]: S obzirom da se donja i gornja granica zadaju, funkcija ima dva parametra. Generiranje slučajnog broja ostvareno je kao u primjeru 3.

Primjer 8. *Napisati funkciju koja popunjava realno polje pseudo-slučajnim brojevima. Generirani brojevi trebaju biti unutar zadanog intervala. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```
#include <stdlib.h>
#include <time.h>

#define N 400

void rnd_array(double a[], int m, double dg, double gg);
double rand_ab(double a, double b); //pomocna funkcija

int main(void)
{
  double p[N];

  srand((unsigned)time(NULL));
  rnd_array(p, N, -100, 50);

  return 0;
}

double rand_ab(double a, double b)
{
  return a + (double)rand() / RAND_MAX * (b - a);
}

void rnd_array(double a[], int m, double dg, double gg)
{
  int i;
  for (i = 0; i < m; i++)
    a[i] = rand_ab(dg, gg);
}
```

Kratko obrazloženje [Primjer 8]: Korištena je funkcija iz prethodnog primjera kao pomoćna. Kako polje treba biti popunjeno vrijednostima iz zadanog intervala, tako su donja i gornja granica, također, parametri funkcije.

Primjer 9. *Generirati 500 prirodnih pseudo-slučajnih brojeva u* $[1, 10]$. *Ispisivanjem odgovarajućeg broja znaka* | *prikazati koliko je kojih brojeva bilo generirano.*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N  500
#define DG 1
#define GG 10

int main(void)
{
  int h[GG] = {0}; //postavljanje vrijed. svih elem. na 0
  int i, j, x;
```

```
  srand((unsigned)time(NULL));
  for (i = 0; i < N; i++) {
    x = DG + rand() % GG; //generiranje prirodnih brojeva u [1,10]
    h[x - 1]++; //uvecavanje vrijed. elem. s odgovarajucim indeksom
  }
  for (i = 0; i < GG; i++) {
    printf("[%2d] ", i + 1);
    for (j = 1; j <= h[i]; j++) //ispis odgovarajuceg broja znaka |
      putchar('|'); //eventualno iskoristiti: putchar(254); za estetski ljepsi ispis

    putchar('\n');
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 9]: Broji se koliko se puta koji broj pojavio odnosno koliko je puta bio generiran. Vrijednosti elementa polja s indeksom $0, \ldots, 9$, odgovara broju pojavljivanja broja $1, \ldots, 10$, redom.

Primjer 10. *Simulirati bacanje novćića 1000 puta. Ispisati koliko je puta bila bačena glava, odnosno pismo.*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
  int head, tail;
  int i;

  srand((unsigned)time(NULL));
  head = tail = 0;
  for (i = 1; i <= 1000; i++) {
    if ((double)rand() / RAND_MAX < 0.5) {
      head++;
    }
    else {
      tail++;
    }
  }

  printf("Head: %d\tTail: %d", head, tail);
  return 0;
}
```

Kratko obrazloženje [Primjer 10]: Ukoliko je generirani pseudo-slučajni broj iz $[0, 1]$ manji od 0.5 smatra se da je bačena glava, dok se u suprotnom smatra da je bačeno pismo.

Primjer 11. *Napisati funkciju koja predano joj polje znakova popunjava pseudo-slučajno generiranim velikim slovima. Funkcija postavlja* null *znak na kraju polja. U funkciji* `main()` *prikazati uporabu napisane funkcije.*

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void rndLetters(char s[], int n);

int main(void)
{
  char str[30];
```

```
  srand((unsigned)time(NULL));

  rndLetters(str, 30);
  puts(str);

  return 0;
}

void rndLetters(char s[], int n)
{
  int i;
  for (i = 0; i < n - 1; i++) {
    s[i] = 'A' + rand() % ('Z' - 'A' + 1);
  }
  s[n-1] = '\0';
}
```

Kratko obrazloženje [Primjer 11]: Kako se velika slova nalaze jedno iza drugog, moguće je uzeti slova A kao donju granicu, a slovo Z kao gornju granicu pri generiranju pseudo-slučajnih vrijednosti. Funkcija popunjava privh $n - 1$ elemenata polja, a zadanji element postavlja na *null* znak kako bi se dobio string.

POGLAVLJE 13

Dinamičko zauzimanje memorije

Dinamičko zauzimanje memorije (engl. *dynamic memory allocation*) omogućuje zauzimanje prostora u memoriji za vrijeme izvođenja programa. Ono ima važnu ulogu u jeziku jer omogućuje, primjerice, stvaranje složenih i fleksibilnih struktura podataka kao što su povezani popisi, stabla, grafovi i slično. Također, ono je mehanizam koji rješava ograničenja "običnih polja" – veličina polja mora biti prije prevođenja poznata te veličinu polja nije moguće promijeniti za vrijeme izvođenja.

Konceptualno, podatkovni objekti (varijable, polja i slično) koje neki program koristi spremaju se u različite dijelove memorije. Statičke varijable zauzimaju memoriju koja je konstantne veličine. One postoje od početka izvođenja programa do njegovog završetka. Automatske varijable se spremaju u dio memorije koji se uobičajeno naziva stog (engl. *stack*), a količina zauzete memorije se mijenja za vrijeme izvođenja programa. Memorija se za automatske (lokalne) varijable zauzima tek kada se dosegne blok naredbi unutar kojeg su deklarirane i oslobađa se odmah nakon izlaska iz istog. Prostor koji se koristi pri dinamičkom zauzimanju memorije, uobičajeno se naziva hrpa (engl. *heap*), a količina se zauzete memorije se također mijenja za vrijeme izvođenja programa. Ona nastaje pozivom odgovarajuće funkcije za zauzimanje i nestaje pozivom funkcije za oslobađanje.

Važno je primijetiti kako je odgovornost pri rukovanju memorijom (zauzimanje i oslobađanje memorije) na hrpi u potpunosti prebačena na programera. To je cijena fleksibilnosti koju se plaća. Tako, osim što je moguće po potrebi tražiti memorijski prostor za podatke, hrpa je u pravilu daleko veća od stoga te prema tome nudi prostor nužan za spremanje velikih količina podataka.

13.1 Dinamičko rukovanje memorijom

Dinamičko zauzimanje i oslobađanje memorije omogućeno je putem funkcija standardne biblioteke. One omogućuju zauzimanje bloka memorije, promjenu njegove veličine i njegovo oslobađanje. Navedene funkcije opisane su u zaglavnoj datoteci stdlib.h.

13.1.1 Zauzimanje memorije

Osnovna funkcija za dinamičko zauzimanje memorije je `malloc()`, dok se rjeđe koristi funkcija `calloc()`. Navedene funkcije zauzimanju memoriju, ali joj ne dodjeljuju ime. Prototipovi tih funkcija prikazani su:

```
void *malloc(size_t vel);
void *calloc(size_t n, size_t vel);
```

Obje funkcije zauzimaju zadanu količinu memorije i vraćaju pokazivač na njen početak, odnosno adresu njenog početka (prvi bajt). Povratnu vrijednosti tih funkcija treba pridružiti nekoj pokazivačkoj varijabli (Sl. 13.1). U slučaju neuspješnog zauzimanja memorije vraćaju tzv. *null* pokazivač. Može se reći kako je *null* pokazivač "pokazivač na ništa", a predstavljen je posebnom vrijednošću `NULL` koja je makro definiran u nekoliko različitih zaglavnih datoteka (uključujući stdio.h, stdlib.h, string.h i time.h).

Argument funkcije `malloc()` predstavlja veličinu željenog bloka memorije u bajtima. Prvi argument funkcije `calloc()` predstavlja broj elementa, a drugi veličinu elementa u bajtima. Funkcija `calloc()` postavlja sve bitove zauzete memorije na 0, za razliku od funkcije `malloc()` koja ne obavlja nikakav vid inicijalizacije.

Parametri tih funkcija su tipa `size_t` koji predstavlja neki cjelobrojni tip podatka bez predznaka definiran unutar standardne biblioteke. Treba primijetiti kako su obje funkcije tipa `void*` odnosno pokazivač na `void` što je generički pokazivač u programskom jeziku C, a njegova vrijednost u je osnovi samo memorijska adresa. To znači da takav pokazivač nije moguće koristiti za vrijednosti na adresi koju drži. Stoga je potrebno povratnu vrijednost tih funkcija pridružiti nekoj pokazivačkoj varijabli na "konkretan" tip. Nju je zatim moguće koristiti kao da se radi o imenu polja. Izlistanjem 13.1 su prikazni općeniti primjeri poziva i pridruživanja povratne vrijednosti funkcija `malloc()` i `calloc()`.

```
pokazivac = (tip_podatka *)malloc(n * sizeof(tip_podatka));
pokazivac = (tip_podatka *)calloc(n, sizeof(tip_podatka));
```

Izlistanje 13.1: Generički primjeri dinamičkog zauzimanje memorije funkcijama `malloc()` i `calloc()`

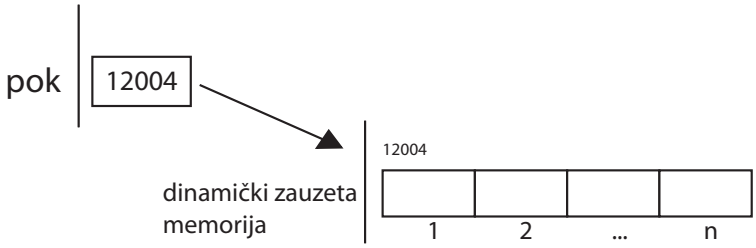
U prethodnim primjerima, prije pridruživanja, vrši se pretvaranje (engl. *casting*) povratne vrijednosti na pokazivač na tip podatka pokazivačke varijable (`pokazivac` u primjerima). Pretvaranje nije obavezno, jer je moguće pridružiti `void*` vrijednost pokazivačkoj varijabli bilo kojeg tipa (obrat isto vrijedi), ali je ipak uobičajeno (primjerice, u programskom jeziku C++ je nužno obaviti pretvaranje). Osim toga, koristi se operator `sizeof` koji daje veličinu svog operanda u bajtima. Taj operator nije nužno koristiti ukoliko se zauzima memorija za tip podatka `char`, jer je `char` uvijek veličine jedan bajt[1].

Nadalje, veličinu bloka prethodno dinamički zauzete memorije moguće je promijeniti (smanjiti ili povećati) uporabom funkcije `realloc()`, čiji je prototip:

```
void *realloc(void *pok, size_t vel);
```

Prvi argument funkcije `realloc()` mora biti pokazivač, odnosno adresa dobivena od funkcije `malloc()`, `calloc()` ili `realloc()`. Također, može biti *null* pokazivač (`NULL`) pa se u tom slučaju ponaša kao funkcija `malloc()`. Drugi argument predstavlja novu veličinu bloka te ona može biti veća ili manja u odnosu na originalnu. Ako je nova veličina 0 onda funkcija `realloc()` oslobađa memoriju. Funkcija `realloc()` vraća pokazivač na početak zauzete memorije, a u slučaju neuspjeha vraća *null* pokazivač te se podaci u prethodno zauzetoj memoriji ne mijenjaju.

[1] Uvijek vrijedi, `sizeof(char) je 1`.



Slika 13.1: Dinamičko zauzimanje memorije.

13.1.2 Provjera uspješnosti zauzimanja memorije

Kako pozivi prethodno navedenih standardnih funkcija za dinamičko zauzimanje memorije ne moraju uvijek biti uspješni, pravilo je provjeriti povratnu vrijednost tih funkcija. Što znači, treba provjeriti je li neka od navedenih funkcija vratila `NULL`, ukoliko jeste, zauzimanje memorije nije uspjelo pa je potrebno postupiti na odgovarajući način kao primjerice, završiti izvođenje programa. Navedeno je ilustrirano primjerom u izlistanju 13.2.

```
if (pok == NULL)
  exit(EXIT_FAILURE);
```

Izlistanje 13.2: Primjer provjere uspješnosti zauzimanja memorije

U prikazanom primjeru korištena je funkcija `exit()` koja dovodi do završetka izvođenja programa iz kojegod funkcije je pozvana. Opisana u zaglavnoj datoteci stdlib.h, a prototip joj je:

```
void exit(int status);
```

Definirane su dvije konstante koje joj je moguće predati kao argument. Predaje joj se konstanta `EXIT_SUCCESS` ili `EXIT_FAILURE` kako bi se ukazalo na uspješno ili neuspješno završavanje izvođenja. Ukoliko se memorija zauzima u funkciji `main()`, može se, primjerice, umjesto poziva funkcije `exit()` samo vratiti vrijednost 1 (`return 1;`).

13.1.3 Oslobađanje memorije

U pravilu, memoriju zauzetu na hrpi treba osloboditi čim ona više nije potrebna. Naime, ona se ne oslobađa automatski, primjerice po izlasku iz bloka naredbi u kojoj je zauzeta ili po završetku izvođenja funkcije, kao što je slučaj s automatskim varijablama. Oslobađanjem memorije se ona stavlja na raspolaganje sustavu za eventualnu ponovnu uporabu. Dinamički zauzetu memoriju potrebno je osloboditi nakon što ona više nije potrebna, a to se čini pomoću funkcije `free()`. Prema tome, pravilo je upariti svaki poziv funkcije `malloc()` (ili `calloc()` ili `realloc()`) s naknadnim pozivom funkcije `free()` čiji je prototip:

```
void free(void *pok);
```

Argument funkcije `free()` mora biti pokazivač, odnosno adresa dobivena od funkcije `malloc()`, `calloc()` ili `realloc()` (može biti i *null* pokazivač, ali u tom slučaju poziv funkcije `free()` nema nikakav učinak). Dodatno se nakon poziva funkcije `free()`, iz

sigurnosnih razloga, može postaviti predani joj pokazivač na `NULL`. Naime, poziv funkcije `free()` ne mijenja vrijednost pokazivača koji joj je predan[2].

13.2 Aritmetika nad pokazivačima

Iako su u pravilu adrese u memoriji predstavljane interno nekim od cjelobrojnih tipova bez predznaka, nikako ih ne treba kao takve promatrati. Tako je artimetika nad adresma, odnosno pokazivačima ograničena. U tom pogledu, jednino je moguće dodati ili oduzeti cjelobrojnu vrijednosti od pokazivača te je još moguće oduzeti jedan pokazivač od drugog. Uz to, još je moguće koristiti relacijske operatore na pokazivačima te operatore povećanja i umanjivanja.

Važno je napomenuti kako su opisane operacije jedino smislne za polja. Točnije, ukoliko pokazivač pokazuje na element polja ili ukoliko pokazivači pokazuju na elemente istog polja. U suprotnom ponašanje (programa) nije definirano.

Ako što je ranije opisano, dodavanje cjelobrojne vrijednosti pokazivaču ili adresi rezultira izrazom čija je vrijednost adresa koja je toliko mjesta dalje, pri čemu je svako mjesto veliko koliko zauzima tip na koji je pokazivač. Primjerice, dodavanjem vrijednosti dva pokazivaču na tip `int` rezultirat će adresom koja je osam bajta dalje (uz pretpostavku da je veličina tog tipa četiri bajta). Analogno, oduzimanjem cjelobrojne vrijednosti od pokazivača, rezultirati će adresom koja je toliko mjesta prije. Na kraju, oduzimanje dva pokazivača rezultira cjelobrojnom vrijednosti koja odgovara njihovoj udaljenosti u smislu mjesta, a ne bajta. Vrijednosti relacijskih izraza s pokazivačima kao operandima ovise o međusobnom položaju pokazivača (oba moraju pokazivati na elemente istog polja). Uporaba operatora povećanja i umanjivanja (moguće je koristiti prefiks ili sufiks varijante) mijenja vrijednost pokazivača na adresu koja slijedi ili prethodi originalnoj za jedno mjesto. Treba primijetiti kako je te operatore, za razliku od prethodnih, moguće koristiti samo s pokazivačima, ali ne i imenom polja. Primjena nekih od opisanih operatora ilustrirana je tekstom programa u izlistanju 13.3.

```
int main(void)
{
  float a[5] = {1, 2, 3, 4, 5};
  float *p = &a[1], *q = &a[4];

  float *r; int t;
  r = a + 3; // == &a[3]
  r = q - 2; // == &a[2]

  t = p - q; // == -3
  t = p > q; // == 0

  return 0;
}
```

Izlistanje 13.3: Primjer aritmetike nad pokazivačima

[2]Nakon poziva funkcije `free()` predani joj pokazivač pokazuje na oslobođenu memoriju. Takvi pokazivači se nazivaju "visećima" (engl. *dangling pointers*). Pokušaj promjene oslobođenog memorijskog bloka može uzrokovati različite greške, dok čitanje vjerojatno neće prouzročiti probleme.

13.3 Riješeni primjeri

Slijedi nekoliko primjera za čije je rješavanje potrebno upotrijebiti funkcije `malloc()`, `calloc()` ili `realloc()` te `free()`. Osim toga, u nekoliko navrata potrebno je generirati pseudo-slučajne brojeve. U primjerima gdje nije eksplicitno naveden tip podatka, on je odabran proizvoljno prema zahtjevu.

Primjer 1. *Dinamički zauzeti memoriju za 26 podatka tipa* `char` *(polje znakova veličine 26 elemenata). Dobiveno polje popuniti slovima* `A` *do* `Z` *te ga ispisati na ekran.*

```
#include <stdio.h>
#include <stdlib.h> //malloc(), free()

int main(void)
{
  char i;
  char *s;

  //dinamicko zauzimanje memorije pomocu funkcije malloc()
  //pretvaranje u povratne vrijednosti funkcije nije obavezno, ali je uobicajeno
  s = (char *)malloc(26); //zauzima se mjesto za 26 podatka tipa char
  if (s == NULL) //provjera uspjesnosti zauzimanja memorije
    return 1;


  //popunjavanje polja
  for (i = 0; i < 26; i++)
    s[i] = 'A' + i; //koristi se notacija polja s pokazivacem

  for (i = 0; i < 26; i++)
    printf("%c\n", s[i]); //ispisuje slova A - Z redom, svako slovo u novom retku

  free(s); //oslobadanje zauzete memorije
  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Odmah nakon poziva funkcije za dinamičko zauzimanje memorije, slijedi provjera uspješnosti. Ukoliko je bilo neuspješno završava se izvođenje programa, dok se u suprotnom nastavlja izvođenje. Povratna vrijednost funkcije `malloc()` pretvara se u tip podatka pokazivačke varijable kojoj se i pridružuje. Na taj način se daje "značenje" zauzetoj memoriji. Ako se na kraju zaboravi pozvati funkciju `free()`, zauzeta memorija će biti oslobođena po završetku izvođenja programa. Ipak je pravilo uvijek upariti zauzimanje memorije s njenim oslobađanjem.

Primjer 2. *Dinamički zauzeti memoriju za 1000 podatka tipa* `int`*. Dobiveno polje popuniti pseudo-slučajnim brojevima. Slijednim pretraživanjem u polju tražiti broj koji je također slučajan.*

```
#include <stdio.h>
#include <stdlib.h> //malloc(), free(), rand(), srand()
#include <time.h>   //time()

int main(void)
{
  int i, t, *p;
  int f = 0;

  srand((unsigned)time(NULL)); //inicijaliziranje generatora pseudo-slucajnih brojeva
  //zauzimanje mem. bloka velicine 1000 puta velicina tipa podatka int
  p = (int *)malloc(1000 * sizeof(int)); //mjesto 1000 podataka tipa int
  if (p == NULL)
    return 1;
```

```
  //popunjavanje polja
  for (i = 0; i < 1000; i++)
    p[i] = rand();

  t = rand(); //trazeni broj
  for (i = 0; i < 1000; i++) //slijedno pretrazivanje
    if (p[i] == t)
      f = 1;

  printf("%sostoji trazeni broj", f == 1 ? "P" : "Ne");

  free(p); //oslobadanje zauzete mem.
  return 0;
}
```

Kratko obrazloženje [Primjer 2]: Kod dinamičkog zauzimanja memorije za tipove podataka različite od `char` potrebno je koristiti `sizeof` operator kako bi se dobila veličina danog tipa podatka. Što znači kako se ne treba oslanjati da neki tip podatka zauzima istu količinu memorije na svakom sustavu, jer to općenito nije slučaj. Primjerice, stoga se nikada preporučuje pisati 4 umjesto `sizeof(int)`, iako je najčešće to veličina (u bajtima) tipa podatka `int`.

Primjer 3. *Omogućiti korisniku unos $n > 0$ elemenata u polje realnih brojeva. Dinamički zauzeti potrebnu memoriju. Izračunati i na ekran ispisati srednju vrijednost elemenata polja.*

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  float zbroj = 0, *p;
  int n, i;

  puts("Koliko brojeva zelite unijeti?");
  do { scanf("%d", &n); } while (n < 1);

  //zauzimanje tocne kolicine potrebne memorije
  //zauzimanje mem. bloka velicine n puta velicina tipa podatka float
  p = (float *)malloc(n * sizeof(float)); //mjesto n podataka tipa float
  if (p == NULL)
    return 1;

  for (i = 0; i < n; i++)
    scanf("%f", &p[i]); //ili scanf("%f", p + i);

  for (i = 0; i < n; i++)
    zbroj += p[i];

  printf("Srednja vrijednost unesenih elem.: %.2f", zbroj / n);

  free(p); //oslobadanje zauzete mem.
  return 0;
}
```

Kratko obrazloženje [Primjer 3]: Memorija za polje se zauzima tek nakon što je korisnik unio željenu veličinu polja. Prema tome, dinamičkim zauzimanjem memorije se izbjegava ograničenje fiksne veličine polja koja mora biti poznata prije prevođenja.

Primjer 4. *Omogućiti korisniku unos željene veličine cjelobrojnog polja. Dinamički zauzeti potrebnu memoriju. Popuniti dobiveno polje te nakon toga popuniti drugo polje svakim drugim brojem iz prvog polja.*

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int *p, *q;
  int n, m, i, j;

  puts("Unesite velicinu polja.");
  scanf("%d", &n);

  //zauzimanje mem. bloka velicine n puta velicina tipa podatka int
  p = (int *)calloc(n, sizeof(int));
  if (p == NULL)
    return 1;

  for (i = 0; i < n; i++)
    p[i] = (i + 1) * (i + 2) / 2; //zbroj prvih (i+1) prirodnih brojeva

  m = n / 2;
  j = 1;
  q = (int *)calloc(m, sizeof(int));
  if (q == NULL)
    return 1;

  for (i = 0; i < m; i++) { //prepisuje svaki drugi broj iz prvog polja u drugo
    g[i] = p[j];
    j += 2;
  }

  free(p); //oslobadanje zauzete mem.
  free(q); //oslobadanje zauzete mem.
  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Kao što je već bilo navedeno, za razliku od `malloc()`, funkcija `calloc()` inicijalizira sve bitove zauzete memorije na 0. Ipak treba biti na oprezu, jer na nekim sustavima u slučaju realnih tipova podataka, svi bitovi postavljeni na 0 ne moraju ujedno predstavljati i vrijednost 0.

Primjer 5. *Napisati funkciju koja će spojiti dva stringa u jedan. Funkcija treba vratiti pokazivač na dobiveni string. Na primjeru u funkciji* `main()` *pokazati uporabu napisane funkcije.*

```c
#include <stdio.h>  //sprintf()
#include <stdlib.h>
#include <string.h> //strlen()

char *spoji(char *s1, char *s2)
{
  int n = strlen(s1) + strlen(s2) + 1; //zbroj duljina stringova i mjesto za nul znak

  char *str = (char *)malloc(n * sizeof(char)); //zauzimanje memorije
  if (str == NULL)
    return NULL; //ako zauzimanje mem. nije uspjelo vraca NULL pokazivac

  //spajanje stringova
  sprintf(str, "%s%s", s1, s2);

  return str; //vraca pokazivac na dobiveni string
}

int main(void)
{
  char sa[] = "Prvi dio-";  //prvi string
  char sb[] = "drugi dio."; //drugi string
  char *sc;
```

```
  sc = spoji(sa, sb); //poziv funkcije i dodjela povratne vrijednosti
  //ako f-ja spoji() nije vratila NULL pokazivac, ispisuje: Prvi dio-drugi dio.
  if (sc != NULL)
    puts(sc);

  //oslobadanje zauzete memorije
  //(iako je zauzeta u drugoj f-ji, mem. nece biti automatski oslobodena)
  free(sc);
  return 0;
}
```

Kratko obrazloženje [Primjer 5]: Funkcija je tipa `char*` odnosno pokazivač na tip podatka `char`, jer treba vratiti string (podsjećanja radi: ime polja je pokazivač na prvi element). Analogno, parametri funkcije su istog tipa. Za određivanje duljine predanih stringova koristi se funkcija `strlen()`, a prilikom izračuna duljine, odnosno veličine polja za spojeni string treba uzeti u obzir i *null* znak. Spajanje se obavlja pomoću funkcije `sprintf()` koja umjesto na standardni izlaz (ekran) ispisuje u predani joj string (prvi argument) te dodaje *null* znak na kraj. Memorija za novi spojeni string se zauzima dinamički, a u slučaju neuspjeha funkcija vraća *null* pokazivač što se može koristiti za provjeru uspješnosti poziva napisane funkcije.

Primjer 6. *Generirati pseudo-slučajni cijeli broj iz* $[0, 1000]$ *koji predstavlja traženi. Nakon toga, 250 puta generirati pseudo-slučajni cijeli broj iz* $[100, 500]$ *koji predstavlja veličinu polja i dinamički zauzeti memoriju za polje generirane veličine. Potom ga popuniti pseudo-slučajnim brojevima iz* $[0, 1000]$*. Svaki puta provjeriti nalazi li se traženi broj u polju. Ispisati na ekran prosječnu veličinu polja i koliko je ukupno puta bio pronađen traženi broj.*

```
#include<stdlib.h>
#include<stdio.h>
#include<time.h>

#define P_DG 100  //donja granica velicine polja
#define P_GG 500  //gornja granica velicine polja
#define M    250  //broj ponavljanja
#define T_DG 0    //donja granica za generiranje trazenog broja
#define T_GG 1000 //gornja granica za generiranje trazenog broja

//funkcija koja trazi vrijednost unutar polja
int trazi(const int [], int, int);
//funkcija koja generira pseudo-slucajni broj unutar zadanih granica
int rand_i(int, int);
//funkcija koja popunjava polje pseudo-slucajnim brojevima unutar zadanih granica
void popuni(int [], int, int, int);

int main(void)
{
  int n, i, tr;
  int uvel = 0, up = 0;
  int *p, *r;

  srand((unsigned)time(NULL));
  r = NULL; //kako bi se prvi puta samo zauzela memorija kao pomocu f-je malloc()
  tr = rand_i(T_DG, T_GG);  //generiranje trazenog broja
  for (i = 0; i < M; i++) {
    n = rand_i(P_DG, P_GG); //generiranje velicine polja
    uvel += n;

    //zauzimanje potrebne memorije za polje, odnosno promjena njene velicine
    p = (int *)realloc(r, n * sizeof(int));
    if (p == NULL)
      return 1;

    popuni(p, n, T_DG, T_GG); //popunjavanje polja pseudo-slucajnim brojevima u [0, 1000]
```

```
    up += trazi(p, n, tr);    //trazenje broja u polju

    r = p; //prepisivanje vrijednosti pokazivaca r
  }

  free(p); //oslobadanje zauzete memorije
  printf("Prosjecna velicina polja: %.2f\n", (double)uvel / M);
  printf("Ukupno je pronaden trazeni broj: %d puta\n", up);
  return 0;
}

//slijedno pretrazivanje
int trazi(const int a[], int n, int t)
{
  int i;
  for (i = 0; i < n; i++)
    if (a[i] == t)
      return 1;

  return 0;
}

//generiranje pseudo-slucajnog cijelog broja u [dg, gg]
int rand_i(int dg, int gg)
{
  return dg + rand() % (gg - dg + 1);
}

//popunjavanje polja pseudo-slucajnim cijelim vrijednostima u [dg, gg]
void popuni(int a[], int n, int dg, int gg)
{
  int i;
  for (i = 0; i < n; i++)
    a[i] = rand_i(dg, gg);
}
```

Kratko obrazloženje [Primjer 6]: U primjeru se koriste tri pomoćne funkcije. Prva za slijedno pretraživanje polja, druga za generiranje pseudo-slučajnog broja unutar zadanih granica te treća za popunjavanje polja. Time se olakšava čitanje izvornog teksta programa kao i njegovo održavanje. Treba primijetiti kako je pri prvom pozivu funkcije `realloc()` vrijednost pokazivača koji joj se predaje kao argument jednak `NULL` te se u tom slučaju ona ponaša kao funkcija `malloc()`. Primjer se može riješiti i pomoću funkcije `malloc()` umjesto `realloc()`, ali bi trebalo nakon svakog iskorištavanja polja obaviti oslobođenje zauzete memorije. Na taj način bi se izbjeglo prekomjerno zauzimanje memorije na hrpi, koje može, u ekstremnim slučajevima, dovesti do iscrpljenja raspoloživog prostora.

POGLAVLJE 14

Standardna C biblioteka

Standardna C biblioteka (engl. *C standard library*) sadrži veliki broj raznolikih funkcija koje je moguće koristiti pri pisanju vlastitih programa. Njihove deklaracije su grupirane u nekoliko zaglavnih datoteka. Osim deklaracija, odnosno prototipova funkcija pojedine zaglavne datoteke sadrže i definicije nekih konstanti, makroa i tipova. U narednim poglavljima su navedene samo neke od češće korištenih funkcija, odnosno prikazani su njihovi prototipovi i dani su kratki opisi.

14.1 Zaglavna datoteka stdio.h

Zaglavna datoteka stdio.h sadrži deklaracije/prototipove za nekoliko standardnih funkcija za ulaz/izlaz. U tablici 14.1 prikazani su prototipovi i sažeti opisi nekih značajnih i često korištenih funkcija za ulaz/izlaz.

Tablica 14.1: Neke značajnije funkcije za ulaz/izlaz.

Prototip	Opis
`fgets(char *, int, FILE *);`	Dohvaća sljedeću liniju/red (ili do najviše naznačenog broja znakova) iz naznačenog toka
`fputs(const char *, FILE *);`	Ispisuje string u naznačeni tok
`int getchar();`	Čita sljedeći znak sa standardnog ulaza
`char *gets(char *);`	Čita sljedeću liniju/red sa standardnog ulaza
`void perror(const char *);`	Ispisuje poruku greške na standardni izlaz za greške
`int printf(const char *, ...);`	Ispisuje formatirani izlaz na standardni izlaz
`int putchar(int);`	Ispisuje dani znak na standardni izlaz
`int puts(const char *);`	Ispisuje dani string na standardni izlaz
`int scanf(const char *, ...);`	Učitava formatirani ulaz sa standardnog ulaza
`int sprintf(char *, const char *, ...);`	Ispisuje formatirani izlaz u naznačeni string
`int sscanf(const char *, const char *, ...);`	Učitava formatirani ulaz iz naznačenog stringa

Između ostalog, zaglavna datoteka sadrži definiciju za tip `FILE` i standardne ulazno/izlazne tokove `stdin` (standardni ulaz – tipkovnica), `stdout` (standardni izlaz—ekran) i `stderr` (standardni izlaz za greške—ekran) koje je moguće predati funkcijama koje zahtijevaju kao argument pokazivač na tip `FILE` (primjerice, `fgets()` i `fputs`).

14.2 Zaglavna datoteka stdlib.h

Zaglavna datoteka stdlib.h sadrži deklaracije/prototipove različitih utilitarnih funkcija. Među njih spadaju, između ostalog, funkcije za potrebe generiranja pseudo-slučajnih brojeva, pretraživanje, sortiranje i dinamičko rukovanje memorijom. U tablici 14.2 prikazani su prototipovi i sažeti opisi nekih značajnijih i često korištenih utilitarnih funkcija.

Tablica 14.2: Neke značajnije utilitarne funkcije.

Prototip	Opis
`double atof(const char *s);`	Vraća početni dio stringa `s` pretvoren u vrijednost tipa podatka `double`; pretvaranje prestaje pri pojavi znaka koji nije dio broja; početne praznine se preskaču; vraća vrijednost nula ako ne nađe broj
`int atoi(const char *s);`	Vraća početni dio stringa `s` pretvoren u vrijednost tipa podatka `int`; pretvaranje prestaje pri pojavi znaka koji nije dio broja; početne praznine se preskaču; vraća vrijednost nula ako ne nađe broj
`int rand(void);`	Vraća pseudo-slučajni cijeli broj u intervalu 0 do `RAND_MAX`
`void srand(unsigned int v);`	Postavlja početnu vrijednosti generatora pseudo-slučajnih vrijednosti na `v`; ako se pozove funkcija `rand()` prije poziva `srand()` onda je početna vrijednost jednaka 1
`void *calloc(size_t n, size_t vel);`	Zauzima memoriju za polje od `n` elemenata, gdje je svaki veličine `vel` bajta; svaki bit zauzete memorije se postavlja na nula; vraća adresu polja ako je zauzimanje uspješno, dok u suprotnom vraća `NULL`
`void *malloc(size_t vel);`	Zauzima memoriju veličine `vel` bajta; vraća adresu polja ako je zauzimanje uspješno, dok u suprotnom vraća `NULL`
`void *realloc(void *p, size_t vel);`	Mijenja veličinu bloka memorije na čiji početak pokazuje `p` na veličinu od `vel` bajta; vraća adresu memorije, koji je eventualno premješten; ako nije moguće obaviti promjenu veličine vraća `NULL`, a originalni blok memorije se ne mijenja
`void free(void *p);`	Oslobađa memoriju na čiji početak pokazuje `p`; vrijednost od `p` treba bi dobivena prethodnim pozivom funkcije `calloc()`, `malloc()`, `realloc()` ili biti `NULL`; ponašanje nije definirano za pokazivače s drugim vrijednostima
`int abs(int x);`	Vraća apsolutnu vrijednost od `x`

14.3 Zaglavna datoteka math.h

Zaglavna datoteka math.h sadrži deklaracije/prototipove za mnoge matematičke funkcije. Treba napomenuti kako su kutovi dani u radijanima (1 rad = $\frac{180^\circ}{\pi} \approx 57.296^\circ$) što se odnosi na ulaz i izlaz odgovarajućih funkcija. U tablici 14.3 prikazani su prototipovi i sažeti opisi nekih češće korištenih matematičkih funkcija.

Tablica 14.3: Korisnije matematičke funkcije.

Prototip	Opis
`double acos(double x);`	Vraća kut (0 do π radijana) čiji je kosinus `x`
`double asin(double x);`	Vraća kut ($-\pi/2$ do $\pi/2$ radijana) čiji je sinus `x`
`double atan(double x);`	Vraća kut ($-\pi/2$ do $\pi/2$ radijana) čiji je tangens `x`
`double atan2(double y, double x);`	Vraća kut ($-\pi$ do π radijana) čiji je tangens `y/x`
`double cos(double x);`	Vraća kosinus od `x` (`x` je u radijanima)
`double sin(double x);`	Vraća sinus od `x` (`x` je u radijanima)
`double tan(double x);`	Vraća tangens od `x` (`x` je u radijanima)
`double exp(double x);`	Vraća vrijednost eksponencijalne funkcije od `x` (e^x)
`double log(double x);`	Vraća prirodni logaritam od `x`
`double log10(double x);`	Vraća logaritam po bazi 10 od `x`
`double pow(double x, double y);`	Vraća potenciju `x` na `y` (x^y)
`double sqrt(double x);`	Vraća kvadratni korijen od `x`
`double ceil(double x);`	Vraća najmanju cjelobrojnu vrijednost koja nije manja od `x`
`double floor(double x);`	Vraća najveću cjelobrojnu vrijednost koja nije već od `x`
`double fabs(double x);`	Vraća apsolutnu vrijednost od `x`

Iako navedene trigonometrijske funkcije rade isključivo s radijanima to ne predstavlja ograničenje s obzirom da je pretvaranje iz radijana u stupnjeve i obratno, trivijalno. Pretvaranje iz radijana u stupnjeve ostvaruje se množenjem kuta (u radijanima) sa $\frac{180^\circ}{\pi}$. Obratni postupak pretvaranja iz stupnjeva u radijane je ostvariv množenjem danog kuta (u stupnjevima) sa $\frac{\pi}{180^\circ}$.

Funkcija `atan2()` ima dva argumenta zbog dobivanja informacija o predznacima argumenata kako bi se vratio izračunati kut iz odgovarajućeg kvadranta. Primjerice, za izračun $\arctan(\frac{-10}{-5})$ nužno je koristiti tu funkciju umjesto `atan()` jer bi ona vratila istu vrijednost kao za argument 10/5.

14.4 Zaglavna datoteka string.h

Zaglavna datoteka string.h sadraži deklaracije/prototipove funkcija za rad s poljima znakova koji sadrže *null* znak (stringovi) i poljima znakova koja to ne čine (općenito, memorija). U tablici 14.4 prikazani su prototipovi i sažeti opisi nekih korisnijih funkcija za rad sa stringovima i memorijom općenito.

Tablica 14.4: Neke korisne funkcije za rad sa stringovima i memorijom.

Prototip	Opis
`void *memcpy(void *p1, const void *p2, size_t n);`	Kopira `n` bajta s mjesta u memoriji na koje pokazuje `p2` u mjesto u memoriji na koje pokazuje `p1`; vraća vrijednost od `p1`
`void *memset(void *p1, int v, size_t n);`	Kopira vrijednost `v` (pretvorenu u tip podatka `unsigned char` u prvih `n` bajta mjesta u memoriji na koje pokazuje `p1`; vraća vrijednost od `p1`
`char *strncat(char * s1, const char *s2, size_t n);`	Nadodaje kopiju do najviše `n` znakova ili znakove do pojave *null* znaka iz stringa na koji pozakuje `s2` na mjesto u memoriji na koje pokazuje `s1`, s time da prvi znak stringa `s2` prepisuje *null* znak stringa `s1`; na kraj dodaje *null* znak; vraća vrijednost `s1`
`char *strncpy(char *s1, const char *s2, size_t n);`	Kopira do `n` znakova ili znakove do pojave *null* znaka iz stringa na koji pokazuje `s2` na mjesto u memoriji na koje pokazuje `s1`; ako se u stringu `s2` pojavi *null* znak prije nego se kopira `n` znakova, *null* znakovi se nadodaju kako bi se dobilo ukupno `n` znakova; ako se kopira `n` znakova prije pojave *null* znaka, on se ne nadodaje; vraća vrijednost `s1`
`int strncmp(const char *s1, const char *s2, size_t n);`	Uspoređuje do maksimalno prvih `n` znakova ili do prve pojave *null* znaka stringova na koje pokazuju `s1` i `s2`; oba polja su jednaka ako se svi parovi znakova slažu; znakovi se uspoređuju prema kodnoj vrijednosti znakova; vraća nula ako su polja jednaka, vrijednost manju od nula ako je prvo manje od drugog te vrijednost veću od nula ako je prvo polje veće od drugog
`int strlen(const char *s);`	Vraća broj znakova u stringu `s` (ne uključuje *null* znak)

14.5 Zaglavna datoteka ctype.h

Zaglavna datoteka ctype.h sadrži deklaracije/prototipove nekoliko funkcija za rukovanje znakovima. Te funkcije primaju argumente tipa podatka `int` čije se vrijednosti trebaju moći predstaviti s tipom `unsigned char` ili s `EOF`[1]. U tablici 14.5 prikazani su prototipovi i sažeti opisi nekih često korištenih funkcija za rukovanje znakovima.

Tablica 14.5: Korisnije funkcije za rukovanje znakovima.

Prototip	Opis
`int isalnum(int z);`	Vraća istinu ako je z slovo ili znamenka
`int isalpha(int z);`	Vraća istinu ako je z slovo
`int isdigit(int z);`	Vraća istinu ako je z znamenka
`int tolower(int z);`	Ako je z veliko slovo vraća malo, dok u suprotnom vraća samo originalni argument
`int toupper(int z);`	Ako je z malo slovo vraća veliko, dok u suprotnom vraća samo originalni argument

14.6 Riješeni primjeri

Slijedi nekoliko primjera za čije su rješavanje iskorištene neke od prethodno navedenih funkcija standardne C biblioteke. U primjerima gdje nisu eksplicitno navedeni tipovi podataka i veličine polja oni su odabrani proizvoljno prema zahtjevima zadataka. Također, pokazni primjeri uporabe napisanih funkcija su proizvoljno odabrani.

[1] `EOF` stoji za "kraj datoteke" (engl. *end of file*) i uobičajeno predstavlja vrijednost −1 (definirana u zaglavnoj datoteci stdio.h). Većina standardnih funkcija za ulaz/izlaza može vratiti tu vrijednost. Signal za "kraj datoteke" može se poslati preko tipkovnice kombinacijom tipki CRTL+Z ili na većini UNIX sustava kombinacijom CRTL+D.

Primjer 1. *Omogućiti korisniku unos stringa (rečenice) od maksimalno 200 znakova. Odrediti i u string ispisati koliko je bilo samoglasnika.*

```
#include<stdio.h>
#include<ctype.h>

int main(void)
{
  char str[201];   //200 znakova + 1 mjesto za null znak
  char izlaz[100]; //string za ispis
  int br = 0;      //broj samoglasnika
  int i;

  fgets(str, 201, stdin);

  for (i = 0; str[i] != '\0'; i++)
    switch (tolower(str[i])) { //svaki znak se pretvara u "mali"
      case 'a' :
      case 'e' :
      case 'i' :
      case 'o' :
      case 'u' : br++;
    }

  sprintf(izlaz, "Broj samoglasnika u stringu: %d\n", br); //ispis u string

  return 0;
}
```

Kratko obrazloženje [Primjer 1]: Prije provjere radi li se o samoglasniku vrši se pretvorba u malo slovo pomoću funkcije `tolower()`. Na taj način se izbjeglo pisanje provjere za velika slova. Isto bi se moglo postići funkcijom `toupper()`, u kojem bi se slučaju izbjeglo pisanje provjere za mala slova.

Primjer 2. *Napisati funkciju koja računa i vraća vrijednost formule* $y = e^{\sin(x)+\cos(x)} + \tan(2x)$. *Na primjeru u* `main()` *funkciji pokazati uporabu napisane funkcije.*

```
#include<stdio.h>
#include<math.h>

#define PI 3.141592653

double funk(double x)
{
  double y, r;

  r = sin(x) + cos(x);
  y = exp(r) + tan(2 * x);

  return y;
}

int main(void)
{
  double kut = -134;
  double kut_rad = kut * PI / 180;

  printf("f(%f) = %f", kut_rad, funk(kut_rad));

  return 0;
}
```

Kratko obrazloženje [Primjer 2]: S obzirom da korištene trigonometrijske funkcije rade isključivo s radijanima, u pokaznom primjeru izvršena je pretvorba kuta iz stupnjeva u radijane.

Primjer 3. *Deklarirati tri stringa proizvoljne veličine. Omogućiti korisniku unos jednog stringa. Kopirati prvu polovicu sadržaja unesenog u drugi string te drugu polovicu u treći. Potom ispisati na ekran drugi i treći string.*

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char s1[101], s2[51], s3[51];
  int d0, d1, d2;

  scanf("%100s", s1);
  d0 = strlen(s1);

  d1 = d0 / 2;
  strncpy(s2, s1, d1);
  s2[d1] = '\0'; //dodavanje null znaka na kraj

  d2 = d0 - d1 + 1; //+1 kako bi se kopirao i null znak
  strncpy(s3, &s1[d1], d2);

  puts(s2);
  puts(s3);

  return 0;
}
```

Kratko obrazloženje [Primjer 3]: U prvom slučaju kod kopiranja prve polovice znakova, funkcija `strncpy()` neće automatski dodati *null* znak na kraj. Stoga je potrebno dodati ga nakon poziva te funkcije. U drugom slučaju se kopiraju svi ostali znakovi uključujući i *null* znak što je postignuto uvećavanjem preostalog broja znakova za jedan. Kako bi se kopiranje započelo od prvog od preostalih znakova, u drugom pozivu funkcije, predana je adresa znaka s indeksom `d1` (podsjećanja radi: ime polja predstavlja adresu prvog elementa).

Primjer 4. *Omogućiti korisniku unos Kartezijevih koordinata x i y. Izračunati i na ekran ispisati odgovarajuće polarne koordinate r i ϕ.*

```
#include<stdio.h>
#include<math.h>

#define K 57.29578 //180/pi

int main(void)
{
  double x, y;  //Kartezijeve koordinate
  double r, fi; //polarne koordinate

  scanf("%lf %lf", &x, &y);

  r = sqrt(x * x + y * y); //radijalna udaljenost
  if (r == 0)
    fi = 0;
  else
    fi = K * atan2(y, x); //kut

  printf("Kartezijeve koordinate (%f, %f) "
  "odgovaraju polarnim (%f, %f)\n", x, y, r, fi);

  return 0;
}
```

Kratko obrazloženje [Primjer 4]: Određivanje polarnih koordinata (radijalne udaljenosti od ishodišta i polarnog kuta) iz danih Kartezijevih svodi se na uporabu sljedećih

formula: $r = \sqrt{x^2 + y^2}$ i $\phi = \arctan(\frac{y}{x})$. Korištena je funkcija `atan2()` jer funkcija `atan()` ne može razlikovati liniju s danim kutom u odnosu na liniju koja je zakrenuta za 180°. S obzirom da funkcija `atan2()` vraća kut u radijanima on je pretvoren u stupnjeve množenjem konstantom `K` $\approx \frac{180}{\pi}$.

Primjer 5. *Napisati funkciju koja će izmijeniti u predanom joj stringu sva mala slova u velika i obratno. Na primjeru u `main()` funkciji pokazati uporabu napisane funkcije.*

```
#include<stdio.h>
#include<ctype.h>

void obrat(char str[]);

int main(void)
{
  char s[] = "!?1dfkji HIOHilneln 89dfdUI -- fnoBUIOB?!";

  obrat(s);
  puts(s);

  return 0;
}

void obrat(char str[])
{
  int i, z;
  for (i = 0; str[i] != '\0'; i++) {
    z = tolower(str[i]);
    if (str[i] == z)
      str[i] = toupper(str[i]);
    else
      str[i] = z;
  }
}
```

Kratko obrazloženje [Primjer 5]: S obzirom da funkcija mijenja predani joj string ona je tipa `void` odnosno ne vraća nikakvu vrijednost. Isti učinak bi se postigao zamjenom redoslijeda poziva funkcija `tolower()` i `toupper()`.

Primjer 6. *Omogućiti korisniku unos stringa od najviše 20 znakova. Tražiti ponovni unos dok ne unese string od barem 5 znakova. Potom mu omogućiti unos još jednog stringa i tražiti ponovni unos sve dok ne unese isti kao prethodni.*

```
#include<stdio.h>
#include<string.h>

int main(void)
{
  char str1[21], str2[21];

  puts("Unesite prvi string.");
  while (1) {
    scanf("%20s", str1);
    if (strlen(str1) < 5)
      puts("String mora imati barem 5 znakova!");
    else
      break;
  }

  puts("Unesite drugi string, jednak prvom.");
  while (1) {
    scanf("%20s", str2);
    if (strncmp(str1, str2, 20) != 0)
      puts("String mora biti jednak prvom!");
```

```
    else
      break;
  }

  return 0;
}
```

Kratko obrazloženje [Primjer 6]: Koriste se dvije beskonačne petlje koje se prekidaju tek pri uspješnom unosu zahtijevanih stringova.

Primjer 7. *Popuniti polje znakova sa 100 pseudo-slučajno generiranih slova od* `A` *do* `Z` *(engleska abeceda) i ispisati ih. Kopirati prvih 10 i zadnjih 10 slova u novo polje znakova te ga potom ispisati na ekran.*

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

#define N  100
#define T  20
#define PT 10

int main(void)
{
  char abc[N];
  char z[T];
  int i;

  srand((unsigned)time(NULL)); //inicijalizacija generatora pseudo-slucajnih brojeva
  for (i = 0; i < N; i++) {
    abc[i] = 'A' + rand() % ('Z' - 'A' + 1); //pseudo-slucajno generiranje slova od A do Z
    putchar(abc[i]);
  }

  memcpy(z, abc, PT);         //kopiranje prvih 10 slova
  memcpy(&z[PT], &abc[N - PT], PT); //kopiranje zadnjih 10 slova

  puts("\n"); //dva prazna reda
  for (i = 0; i < T; i++)
    putchar(z[i]);

  return 0;
}
```

Kratko obrazloženje [Primjer 7]: Kako bi se postiglo kopiranje zadnjih 10 slova u zadnjih 10 mjesta drugog polja, funkciji `memcpy()` predane su adrese 11. elementa odredišnog polja i adresa 91. elemenata izvorišnog polja.

Primjer 8. *Deklarirati i popuniti polje od 1000 realnih brojeva. Potom kopirati sve vrijednosti od 111. do 777. elementa u novo polje i ispisati ga na ekran. Dinamički zauzeti potrebnu memoriju za polja.*

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define N  1000
#define DG 111
#define GG 777

int main(void)
{
  float *a, *b;
```

```
  int i;

  //dinamicko zauzimanje memorije za 1000 podataka tipa float
  a = (float *)malloc(N * sizeof(float));
  for (i = 0; i < N; i++)
    a[i] = 0.1 * (i + 1); //popunjavanje polja vrijednostima od 0.1 do 100 s korakom od 0.1

  //dinamicko zauzimanje memorije za 667 podatak tipa float
  b = (float *)malloc((GG - DG + 1) * sizeof(float));
  memcpy(b, &a[DG - 1], (GG - DG + 1) * sizeof(float)); kopiranje vrijed. zadanih elem.

  for (i = 0; i < GG - DG + 1; i++)
    printf("%.1f ", b[i]);

  free(a); //oslobadanje
  free(b); //zauzete memorije

  return 0;
}
```

Kratko obrazloženje [Primjer 8]: S obzirom da funkcija `memcpy()` kopira naznačeni broj bajta nužno je uzeti u obzir koliko zauzima (bajta) tip podatka kojeg se kopira ako je različit od `char`. Stoga je potrebno koristiti `sizeof` operator čiji se rezultat množi s brojem elemenata kojeg se želi kopirati.

ZADACI ZA VJEŽBU A

Upravljanje tijekom izvođenja programa

Opis

U nastavku slijedi nekoliko zadataka za vježbu, a odnose se na upravljanje tijekom izvođenja programa. Točnije, odnose se na gradivo pokriveno poglavljem 4 i 5.

Napomena: U zadacima gdje se od korisnika traži unos n (ili je korišteno neko drugo slovo) brojeva, a gdje je vrijednost n ograničena (primjerice, $8 \leq n \leq 18$), potrebno je od korisnika tražiti unos vrijednosti n sve dok ona ne bude unutar zadanih granica. Navedeno je ilustrirano primjerom koji slijedi.

[Pr. 1.] Omogućiti korisniku unos $6 < n < 16$ cijelih brojeva.

```c
#include <stdio.h>

int main(void)
{
  int n, broj, i;

  do {
    scanf("%d", &n);
  } while (n <= 6 || n >= 16);

  for (i = 0; i < n; i++) {
    scanf("%d", &broj);
  }

  return 0;
}
```

Zadaci

1. Omogućiti korisniku unos tri realna broja te ih potom ispisati uzlaznim i silaznim redoslijedom. Također, izračunati i na ekran ispisati omjer/kvocijent udaljenosti najmanjeg do srednjeg i srednjeg do najvećeg.

2. Omogućiti korisniku unos $5 \leq n \leq 20$ realnih brojeva koji moraju biti unutar intervala $[-5, 5]$. Prilikom svakog unosa provjeriti je li uneseni broj unutar zadanog intervala te ako nije ponovo tražiti unos. Na ekran ispisati srednju vrijednost svih n brojeva.

3. Omogućiti korisniku unos jednog znaka te ako je unešeni znak samoglasnik ispisati koji je po redu (od samoglasnika) i ispisati njegovu ASCII vrijednost. Ako unešeni znak nije samoglasnik ispisati odgovarajuću poruku.

4. Omogućiti korisniku unos $2 \leq n < 11$ cijelih brojeva te odrediti i na ekran ispisati koliko je brojeva bilo dijeljivo s 3, a da pri tome nije bilo dijeljivo s 7. Pri tome ako broj zadovoljava zadani uvjet ispisati ga na ekran.

5. Omogućiti korisniku unos tri znaka te odredite koji ima najveću, a koji najmanju ASCII vrijednost. Ispisati na ekran oba znaka i pripadajuće ASCII vrijednosti.

6. Omogućiti korisniku unos $3 \leq n \leq 15$ realnih brojeva koji moraju biti negativni. Prilikom svakog unosa provjeriti je li unešeni broj negativan te ako nije ponavljati unos. Nakon svakog unosa izračunati i na ekran ispisati srednju vrijednost do tada unešenih brojeva.

7. Omogućiti korisniku unos $4 < n < 26$ cijelih brojeva koji moraju biti dvoznamenkasti. Prilikom svakog unosa provjeriti je li unešeni broj dvoznamenkast te ako nije ponavljati unos. Pronaći i na ekran ispisati zbroj znamenki najvećeg unešenog broja.

8. Omogućiti korisniku unos dva realna broja. Ako je njihov zbroj veći od prvog broja ispisati zbroj, u suprotnom izračuanti njihovu razliku i ispasati ju na ekran.

9. Omogućiti korisniku unos 16 dvoznamenkastih cijelih brojeva. Ako se ne unese dvoznamenkasti broj ponovo tražiti unos. Nakon unosa tih 16 brojeva na ekran ispisati srednju vrijednost negativnih, a ukoliko ih nije bilo ispisati odgovarajuću poruku.

10. Omogućiti korisniku unos dva realna broja. Ako je prvi veći od drugog ispisati na ekran količnik, a ako je drugi veći od prvog ispisati umnožak. Ukoliko su jednaki ispisati odgovarajuću poruku.

11. Omogućiti korisniku unos 10 troznamenkastih cijelih brojeva. Ako se ne unese troznamenkasti broj ponovo tražiti unos. Nakon unosa tih 10 brojeva na ekran ispisati srednju vrijednost neparnih, a ukoliko ih nije bilo ispisati odgovarajuću poruku.

12. Omogućiti korisniku unos tri cijela broja a, b i c. Izračunati $(a \cdot b - c)^2 + \max\{a, b, c\}$ i rezultat ispisati na ekran.

13. Omogućiti korisniku unos $8 \leq n \leq 18$ realnih brojeva koji moraju biti veći od nule (0). Prilikom svakog unosa provjeriti je li uneseni broj zadovoljava uvjet, ako ne zadovoljava tražiti ponovni unos. Izračunati omjer (količnik) zbroja kvadriranih unesenih vrijednosti i zbroja unesenih vrijednosti.

14. Omogućiti korisniku unos tri realna broja x, y i z. Izračunati $x^2 \cdot y^2 - \min\{x, y, z\}$ i rezultat ispisati na ekran.

15. Omogućiti korisniku unos $6 < m < 16$ cijelih brojeva koji ne smiju biti parni. Prilikom svakog unosa provjeriti je li uneseni broj zadovoljava uvjet, ako ne zadovoljava tražiti ponovni unos. Izračunati produkt zbroja unsenih vrijednosti i zbroja negiranih unsenih vrijednosti.

16. Omogućiti korisniku unos tri cijela broja a, b i c. Izračunati $\min\{a, b, -1\}/(|c^3| + 1)$ i rezultat ispisati na ekran. Za izračun nije dozvoljeno koristiti funkcije standardne biblioteke.

17. Omogućiti korisniku unos $4 \leq n < 19$ cijelih brojeva. Osigurati da svaki od tih unosa bude negativan neparan broj (ako nije zadovoljen uvjet, tražiti ponavljanje). Odrediti kvocijent najmanjeg i najvećeg od navedenih n unosa.

18. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.hr>

#define N = 10

int main(void)
{
  int i
  float br, zb = 0,

  for (i = 0, i < N, ++i) {
    scanf("%f", br);
    zb += br;
  }
  printf("%f, zb / N");

  return ;
}
```

19. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <studio.h>

int main(void)
{
  int c; s = 0,

  While (1) {
    scanf("%d", &c);
    if (10 < c < 20)
      ++s;
    else if (c = 0)
      break;
  }

  return 0;
}
```

d

ZADACI ZA VJEŽBU B

Jednodimenzionalna i dvodimenzionalna polja te stringovi

Opis

U nastavku slijedi nekoliko zadataka za vježbu, a odnose se na jednodimenzionalna (1-D) i dvodimenzionalna (2-D) polja te stringove. Točnije, odnose se na gradivo pokriveno poglavljem 6, 7 i 8.

Napomena: U zadacima gdje se od korisnika traži unos n (ili je korišteno neko drugo slovo) vrijednosti u polje (cijelih ili realnih brojeva), a gdje je vrijednost n ograničena (primjerice, $8 \leq n \leq 18$), potrebno je od korisnika tražiti unos vrijednosti n sve dok ona ne bude unutar zadanih granica. Potom u polje učitati tih n vrijednosti. Naravno, potrebno je deklarirati polje odgovarajuće veličine. Nadalje, ako je tražen unos stringa ili rečenice od maksimalno n znakova (vrijednost n je zadana u zadatku), nužno je onemogućiti da se pokuša spremiti veći broj znakova u polje. Veličina deklariranog polja mora biti barem $n + 1$ zbog mjesta potrebnog za *null* znak. Navedeno je ilustrirano primjerima koji slijede.

[Pr. 1.] Omogućiti korisniku unos $6 < n < 16$ vrijednosti u polje cijelih brojeva.

```
#include <stdio.h>

int main(void)
{
  int n, i;
  int brojevi[15];

  do {
    scanf("%d", &n);
  } while (n <= 6 || n >= 16);

  for (i = 0; i < n; i++) {
    scanf("%d", &brojevi[i]);
  }

  return 0;
}
```

[Pr. 2.] Omogućiti korisniku unos jedne rečenice od maksimalno 50 znakova.

```
#include <stdio.h>

int main(void)
{
  char str[51];

  fgets(str, 51, stdin);

  return 0;
}
```

Zadaci

1. Omogućiti korisniku unos $7 < m < 21$ vrijednosti u polje realnih brojeva. Pronaći element s parnim indeksom koji ima najveću vrijednost te element s neparnim indeksom koji ima najmanju vrijednost. Na ekran ispisati indekse i vrijednosti navedenih elemenata.

2. Omogućiti korisniku unos $5 \leq p \leq 25$ vrijednosti u polje realnih brojeva. Odrediti i ispisati na ekran aritmetičku sredinu elemenata koji su manji od aritmetičke sredine svih elemenata polja i aritmetičku sredinu elemenata koji su veći od aritmetičke sredine svih elemenata polja.

3. Omogućiti korisniku unos $5 \leq n < 21$ vrijednosti u polje cijelih brojeva. Pronaći element s najmanjom vrijednosti i izračunati srednju vrijednost svih. Potom promijeniti sve elemente čija je vrijednost veća od srednje na vrijednost elementa s najmanjom.

4. Omogućiti korisniku unos $7 < m \leq 17$ vrijednosti u polje cijelih brojeva. Pronaći element s najvećom vrijednosti manjom od 0 i najmanji broj veći od 0 te na ekran ispisati njihovu razliku. Ako nema neke od traženih vrijednosti ispisati odgovarajuću poruku.

5. Omogućiti korisniku unos $3 < n < 27$ vrijednosti u polje realnih brojeva. Izračunati srednju vrijednost elemenata polja. Potom, izračunati produkt svih vrijednosti elemenata koju su strogo veći od srednje vrijednosti i strogo manji od najveće vrijednosti u polju.

6. Omogućiti korisniku unos stringa od maksimalno 85 znakova. Promijeniti sve znakove čija je ASCII vrijednost manja od srednje vrijednosti (znakova stringa po ASCII) u znak +.

7. Omogućiti korisniku unos $4 \leq m \leq 30$ vrijednosti u polje realnih brojeva. Izračunati srednju vrijednost elemenata polja. Potom, izračunati zbroj svih vrijednosti elemenata koju su veći ili jednaki od najmanje vrijednosti u polju i strogo manji od srednje vrijednosi.

8. Omogućiti korisniku unos rečenice od maksimalno 70 znakova. Izračunati srednju vrijednost znakova po ASCII. Zatim zamijeniti sve znakove čija je ASCII vrijednost veća od srednje sa znakom &.

9. Omogućiti korisniku unos $10 \leq n \leq 25$ vrijednosti u polje realnih brojeva. Pronaći među unesenim brojevima onaj koji je najbliži najvećem elementu polja.

10. Omogućiti korisniku unos stringa od maksimalno 90 znakova. Izračunati aritmetičku sredinu znakova stringa po ASCII. Sve znakove stringa čija je ASCII vrijednost manja od izračunate aritmetičke sredine, promijeniti u prvi znak stringa. Potom, ispisati string na ekran.

11. Omogućiti korisniku unos $6 \leq n \leq 24$ vrijednosti u polje realnih brojeva. Pronaći među unesenim brojevima onaj koji je najbliži najmanjem elementu polja.

12. Omogućiti korisniku unos $20 \leq m \leq 50$ vrijednosti u polje cijelih brojeva. Dozvoliti samo unos 0 ili 1 kao vrijednost elementa. Odrediti koliko je elemenata s vrijednosti 1. Ako ih je manje nego elemenata s vrijednosti 0, promijeniti sve vrijednosti 0 u vrijednost 1.

13. Omogućiti korisniku unos $14 < n \leq 45$ vrijednosti u polje realnih brojeva. Izračunati i na ekran ispisati srednju vrijednosti samo onih elemenata polja čija je vrijednosti u intervalu $[-10, 10]$. Ako nema elemenata s takvim vrijednostima u polju ispisati odgovarajuću poruku na ekran.

14. Omogućiti korisniku unos $25 \leq n < 51$ vrijednosti u polje realnih brojeva. Povećati vrijednost svakog elemenata većeg od srednje vrijenosti za 25%, a ostalim smanjiti vrijednost za 15%.

15. Omogućiti korisniku unos $7 < x < 19$ vrijednosti u polje realnih brojeva. Ispisati na ekran sve elemente polja čija je vrijednost barem za 10% veća od srednje vrijednosti svih elemenata tog polja.

16. Omogućiti korisniku unos $19 < m < 46$ vrijednosti u polje realnih brojeva. Pronaći i na ekran ispisati element s najmanjom vrijednosti koja je veća od srednje vrijednosti svih elemenata polja.

17. Omogućiti korisniku unos jedne riječi od maksimalno 15 znakova. Zatim, omogućiti korisniku unos još jedne riječi od maksimalno 15 znakova. Ispisati onu riječ koja sadrži više samoglasnika ili odgovarajuću poruku ukoliko sadrže jednak broj istih ili ih ne sadrže.

18. Omogućiti korisniku unos jedne rečenice od maksimalno 70 znakova. Odrediti i na ekran ispisati koliko znakova ima najdulja riječ u rečenici. Pretpostaviti kako je svaka riječ odvojena od druge isključivo razmakom.

19. Omogućiti korisniku unos $7 \leq n \leq 17$ vrijednosti u polje cijelih brojeva. Zamijeniti najveći element u polju sa srednjom vrijednosti svih elemenata.

20. Omogućiti korisniku unos jedne rečenice od maksimalno 50 znakova. Zatim, izbrojati i na ekran ispisati koliko ona sadrži znakova koji predstavljaju razmak, točku ili zarez.

21. Tražiti od korisnika unos jedne rečenice od maksimalno 200 znakova. Odrediti i na ekran ispisati broj riječi u rečenici.

22. Omogućiti korisniku unos jedne rečenice od maksimalno 50 znakova. Zatim, odrediti i na ekran ispisati koliko ona sadrži riječi koje završavaju na znak `a`.

23. Omogućiti korisniku unos rečenice od najviše 250 znakova. Odrediti i na ekran ispisati koji samoglasnik ili samoglasnici se najčešće pojavljuju. Uzeti u obzir mogućnost pojave velikih i malih slova (primjerice, `A` i `a` su isti samoglasnik).

24. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) $m \times m$ $(2 \leq m < 9)$ i njeno popunjavanje realnim brojevima. Izračunati i na ekran ispisati element s najmanjom vrijednosti u drugom retku.

25. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) $n \times n$ ($1 < n \leq 6$) i njeno popunjavanje realnim brojevima. Izračunati i na ekran ispisati srednju vrijednost elemenata prvog stupca.

26. Omogućiti korisniku unos dimenzija matrice (2-D polja) cijelih brojeva $m \times n$ ($3 < m < 17$ i $4 < n \leq 16$). Potom, omogućiti korisniku njeno popunjavanje. Odrediti i na ekran ispisati u kojem se stupcu nalazi najmanja vrijednost.

27. Omogućiti korisniku unos dimenzija matrice (2-D polja) realnih brojeva $m \times n$ ($2 \leq m < 11$ i $5 < n < 21$). Potom, omogućiti korisniku njeno popunjavanje. Odrediti i na ekran ispisati u kojem se retku nalazi najveća vrijednost.

28. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($8 \leq n \leq 64$). Potom, omogućiti korisniku njeno popunjavanje. Ispisati odgovarajuću poruku nalazi li se najveća vrijednost u matrici na glavnoj dijagonali.

29. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) realnih brojeva $m \times m$ ($13 < m < 71$). Potom, omogućiti korisniku njeno popunjavanje. Ispisati odgovarajuću poruku nalazi li se najmanja vrijednost u matrici na sporednoj dijagonali.

30. Omogućiti korisniku popunjavanje matrice (2-D polja) realnih brojeva dimenzija 8×8. Provjeriti i na ekran ispisati odgovarajuću poruku je li najveći broj na glavnoj dijagonali ujedno i najveći broj u matrici.

31. Omogućiti korisniku popunjavanje matrice (2-D polja) realnih brojeva dimenzija 5×4. Pronaći i na ekran ispisati najveću vrijednost u drugom stupcu te najmanju vrijednost u trećem retku.

32. Omogućiti korisniku popunjavanje matrice (2-D polja) cijelih brojeva dimenzija 6×6. Pronaći i na ekran ispisati najveću i najmanju vrijednost na glavnoj dijagonali.

33. Omogućiti korisniku popunjavanje matrice (2-D polja) cijelih brojeva dimenzija 11×7. Odrediti koliko je parnih brojeva u šestom retku, a koliko je neparnih brojeva u sedmom stupcu.

34. Omogućiti korisniku popunjavanje matrice (2-D polja) realnih brojeva dimenzija 25×8. Pronaći najveću vrijednost elementa u matrici. Potom, podijeliti vrijednost svakog elementa matrice s najvećom te ispisati matricu na ekran.

35. Omogućiti korisniku popunjavanje matrice (2-D polja) cijelih brojeva dimenzija 17×18. Potom omogućiti korisniku unos jednog cijelog broja x. Provjeriti i na ekran ispisati koliko se puta vrijednost x pojavljuje u matrici i koliko je vrijednosti elemenata djeljivo s x.

36. Omogućiti korisniku unos dimenzija (mora biti parna vrijednost) kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($10 \leq n \leq 20$). Potom, omogućiti korisniku popunjavanje matrice. Pronaći i na ekran ispisati najmanju vrijednost ispod te najveću vrijednost iznad sporedne dijagonale.

37. Omogućiti korisniku unos dimenzija (mora biti parna vrijednost) kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($5 \leq n \leq 20$). Potom, omogućiti korisniku njeno popunjavanje. Izračunati aritmetičku sredinu gornje i donje polovice matrice.

38. Omogućiti korisniku unos dimenzija (mora biti neparna vrijednost) kvadratne matrice (2-D polja) cijelih brojeva $n \times n$ ($9 < n \leq 24$). Potom, omogućiti korisniku popunjavanje matrice. Izračunati zbroj dane matrice i transponirane joj matrice. Pohraniti zbroj u novu matricu te ju potom ispisati na ekran.

39. Omogućiti korisniku unos dimenzija (mora biti neparna vrijednost) kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($7 < n < 17$). Potom, omogućiti korisniku njeno popunjavanje. Izračunati aritmetičku sredinu vrijednosti elemenata iznad glavne dijagonale i aritmetičku sredinu vrijednosti elemenata ispod glavne dijagonale te ispisati na ekran onu vrijednost koja je veća.

40. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($10 \leq n \leq 50$). Potom, omogućiti korisniku popunjavanje matrice. Izračunati srednju vrijednost elemenata iznad i ispod presjeka glavne i sporedne dijagonale (elementi koji su iznad odnosno ispod obje dijagonale).

41. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($5 < n \leq 25$). Potom, omogućiti korisniku njeno popunjavanje. Izbrojati negativne elemente svakog stupca i redom ih spremati u zasebno 1-D polje.

42. Omogućiti korisniku unos dimenzija kvadratne matrice (2-D polja) realnih brojeva $n \times n$ ($10 \leq n \leq 30$). Potom, omogućiti korisniku popunjavanje matrice. Ispisati na ekran stupac (elemente matrice koji ga čine) s najvećom srednjom vrijednosti elemenata.

43. Omogućiti korisniku unos dimenzija matrice (2-D polja) cijelih brojeva $m \times n$ ($5 \leq m \leq 30$ i $11 < n < 21$). Potom, omogućiti korisniku popunjavanje matrice. Odrediti i na ekran ispisati stupac (vrijednosti elemenata koji ga čine) koji sadrži najmanje parnih vrijednosti.

44. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

int main(void)
{
  float brojevi[] = {1; 2.2; 4.1; 5.25; 6.4}

  for (i = 0; i < 5; i++)
    printf("%f ", &brojevi[i]);

  return ;
}
```

45. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

int main(void)
{
  int[10] polje;
  int i;
```

```
  for (i = 0; i < 10; i++)
    scanf("%d", polje[i]);

  return 0;
}
```

46. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

int main(void)
{
  char str[] = 'Programiranje jedan';

  for (i = 0; i != '\0'; i++)
    putchar(str[i] + 1);

  return 0;
}
```

47. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

#define M = 3

int main(void)
{
  unsigned int matrix[M][] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} }

  int i;
  for (i = 0, i <= M, i++)
    for (j = 0, j <= M, j++)
      printf("%d ", matrix[i, j])

  return ;
}
```

48. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

#Define ROWS 5
#Define COLS 4

int main(void)
{
  float array2D[ROWS][COLS];
  float avg;

  int i, j;
  for (i = 0; i < COLS; i++)
    for (j = 0; j < ROWS; j++)
      scanf("%f", array2D[j][i]);

  for (i = 0; i < ROWS; i++) {
    sum = 0
    for (j = 0; j < COLS; j++)
      sum += array2D[i][j];
    avg = sum / COLS;
    printf("%f"\n , avg);
  }

  return 0
}
```

ZADACI ZA VJEŽBU C

Jednostavne funkcije

Opis

U nastavku slijedi nekoliko zadataka za vježbu, a odnose se na funkcije. Točnije, odnose se na gradivo pokriveno poglavljem 9.

Zadaci

1. Napisati funkciju koja će izračunati i vratiti vrijednost formule $y = -a^2 + b^2$. U svrhu testiranja u funkciji `main()` pomoću napisane funkcije izračunati vrijednost formule za $(a, b) = (2, 3)$ i $(a, b) = (-2, 4)$ te ispisati parametre a i b za koje formula ima manju vrijednost.

2. Napisati funkciju koja će izračunati i vratiti vrijednost formule $y = 2a^2 - 3b$. U svrhu testiranja u funkciji `main()` pomoću napisane funkcije izračunati vrijednost formule za $(a, b) = (-2, 1)$ i $(a, b) = (1, 2)$ te ispisati parametre a i b za koje formula ima veću vrijednost.

3. Napisati funkciju koja računa i vraća vrijednost formule $y = |x_1 \cdot x_2 + x_1| + x_1^2 - x_2^2$. U svrhu testiranja u funkciji `main()` provjeriti i na ekran ispisati za koje vrijednosti x_1 i x_2 funkcija vraća manju vrijednost. Provjeru obaviti za sljedeće parove x_1 i x_2: $(2.125, -5.5)$ i $(1.45, 8.752)$.

4. Napisati funkciju koja računa i vraća vrijednost formule $y = |x_1 - x_2 \cdot x_1| - x_1^2 \cdot x_2$. U svrhu testiranja u funkciji `main()` provjeriti i na ekran ispisati za koje vrijednosti x_1 i x_2 funkcija vraća veću vrijednost. Provjeru obaviti za sljedeće parove x_1 i x_2: $(0.15, 9.575)$ i $(-4.725, -5.87)$.

5. Napisati funkciju koja određuje i vraća koliko je znamenki predanog joj cijelog broja parno (funkcija mora moći raditi s brojevima proizvoljnog broja znamenki). U svrhu testiranja u funkciji `main()` pozvati funkciju s vrijednosti 12345 kao argumentom i ispisati na ekran povratnu vrijednost.

6. Napisati funkciju koja određuje i vraća koliko je znamenki predanog joj cijelog broja djeljivo s tri (funkcija mora moći raditi s brojevima proizvoljnog broja znamenki). U

svrhu testiranja u funkciji `main()` pozvati funkciju s brojem 1346 kao argumentom i ispisati na ekran povratnu vrijednost.

7. Napisati funkciju koja određuje i vraća koliko neparnih znamenki sadrži predani joj cijeli broj. U svrhu testiranja u funkciji `main()` pozvati funkciju s brojem 2456 kao argumentom i na ekran ispisati povratnu vrijednost.

8. Napisati funkciju koja će izračunati i vratiti vrijednost formule $y = \frac{\alpha^3+\beta^{-2}}{\gamma^2}$. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos tri realna broja te osigurati da budu međusobno različiti. Pozvati napisanu funkciju s unesenim vrijednostima kao argumentima i na ekran ispisati povratnu vrijednost. Za izračun potencija nije dozvoljeno koristiti gotovu funkciju.

9. Napisati funkciju koja će odrediti i vratiti koliko je cijelih brojeva između x i y koji su djeljivi sa z. U svrhu testiranja u funkciji `main()` pozvati napisanu funkciju s argumentima 8, 243 i 7. Ispisati na ekran povratnu vrijednost funkcije.

10. Napisati funkciju koja će izračunati i vratiti zbroj svih djelitelja danog broja. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos broja $10 < x < 100$ te pozvati napisanu funkciju s x kao argumentom. Ispisati povratnu vrijednost funkcije.

11. Napisati funkciju koja će izračunati i vratiti omjer (kvocijent) produkta prvih n prirodnih brojeva i zbroja istih. U funkciji `main()` omogućiti korisniku unos jednog prirodnog broja. Pozvati napisanu funkciju s unesenom vrijednosti kao argumentom i na ekran ispisati povratnu vrijednost.

12. Napisati funkciju koja će izračunati i vratiti vrijednost formule $y = \frac{1}{2}|\alpha^2 - \beta^2|$. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos dvije realne vrijednosti koje predstavljaju parametre α i β. Pozvati napisanu funkciju s unesenim vrijednostima kao argumentima i na ekran ispisati povratnu vrijednost. Za izračun apsolutne vrijednosti ($|\cdot|$) ne koristiti gotovu funkciju.

13. Napisati funkciju koja će izračunati zbroj znamenki za oba predana joj cijela broja. Funkcija vraća vrijednost jednaku broju čiji je zbroj znameki veći. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos dva cijela broja. Pozvati napisanu funkciju s unesenim vrijednostima kao argumentima i na ekran ispisati povratnu vrijednost.

14. Napisati funkciju koja provjerava i vraća 1 ako je zbroj znamenki predanog joj cijelog broja jednoznamenkast, dok u suprotnom vraća vrijednost 0. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos jednog cijelog broja. Pozvati funkciju s unesenom vrijednosti kao argumentom i ispisati na ekran povratnu vrijednost.

15. Napisati funkciju koja vraća razliku broja znamenki predanog joj prirodnog broja po bazi 2 i po bazi 10. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos jednog broja, osigurati da bude prirodan. Pozvati funkciju s unesenim vrijednostima kao argumentima i ispisati na ekran povratnu vrijednost.

16. Napisati funkciju koja vraća od dva predana joj cijela broja onaj koji ima veći produkt znamenki. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos dva cijela

broja. Pozvati funkciju s unesenim vrijednostima kao argumentima i ispisati na ekran povratnu vrijednost.

17. Napisati funkciju koja vraća vrijednost 1 u slučaju da predana joj dva cijela broja imaju jednak broj znamenki, dok u suprotnom vraća vrijednost 0. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos dva cijela broja. Pozvati funkciju s unesenim vrijednostima kao argumentima i ispisati na ekran povratnu vrijednost.

18. Napisati funkciju koja vraća vrijednost 1 u slučaju da je prva znamenka predanog joj cijelog broja manja od 5, a u suprotnom vraća vrijednost 0. (funkcija mora moći raditi s brojevima proizvoljnog broja znamenki). U svrhu testiranja u funkciji `main()` pozvati funkciji s vrijednosti 31982 kao argumentom i ispisati na ekran povratnu vrijednost.

19. Napisati funkciju koja računa i vraća kvocijent prve znamenke i broja znamenki prednog joj cijelog broja (funkcija mora moći raditi s brojevima proizvoljnog broja znamenki). U svrhu testiranja u funkciji `main()` pozvati funkciji s vrijednosti 23415 kao argumentom i ispisati na ekran povratnu vrijednost.

20. Napisati funkciju koja određuje i vraća broj prostih znamenaka predanog joj prirodnog broja. U svrhu testiranja u funkciji `main()` pozvati funkciju s vrijednosti 1579 kao argumentom i ispisati na ekran povratnu vrijednost.

21. Napisati funkciju koja određuje i vraća s koliko je svojih znamenki predani joj cijeli broj djeljiv. U svrhu testiranja u funkciji `main()` pozvati napisanu funkciju s brojem 1806 kao argumentom i na ekran ispisati povratnu vrijednost.

22. Napisati funkciju koja računa i vraća razliku prvih znamenki (prve znamenke s lijeva na desno) predana joj dva cijela broja. U svrhu testiranja u funkciji `main()` pozvati napisanu funkciju s brojevima -51826 i 321 kao argumentima i na ekran ispisati povratnu vrijednost.

23. Napisati funkciju koja od predana joj dva cijela broja određuje i vraća onaj koji ima manje parnih djelitelja. Ukoliko imaju jednak broj traženih djelitelja funkcija vraća prvi broj. U svrhu testiranja u funkciji `main()` pozvati napisanu funkciju s brojevima -21116 i 32421 kao argumentima i na ekran ispisati povratnu vrijednost.

24. Napisati funkciju koja predani joj prirodni broj vraća zapisan naopako. Primjerice za broj 1539, treba vratiti broj 9351. U svrhu testiranja u funkciji `main()` pozvati funkciju s vrijednosti 11579 kao argumentom i ispisati na ekran povratnu vrijednost. Funkcija standardne biblioteke za izračun potencije a^b ima prototip: `double pow(double a, double b);` (opisana u `math.h`)

25. Napisati funkciju koja prima dva cijela broja i vraća onaj čija je aritmetička sredina znamenki veća (ukoliko su jednake funkcija treba vratiti prvi broj). Predani brojevi ne moraju imati jednak broj znamenki. U svrhu testiranja u funkciji `main()` pozvati funkciju s tim brojevima 1293 i -3281 kao argumentima te ispisati na ekran povratnu vrijednost.

26. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

void produkt(double, double)

int main(void)
{
  double a = produkt(12; 115.825);
  printf('%f', &a);

  return ;
}

void produkt(double a, double b)
{
  return a * b
}
```

27. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

int faktorijel(int n);

int main(void)
{
  int r = faktorijel 10
  printf("%d", r);

  return 0;
}

int faktorijel(int m)
{
  for (i = 2; i <= n; i++)
    p *= i;

  return p;
}
```

ZADACI ZA VJEŽBU D

Funkcije koje rade s poljima i stringovima

Opis

U nastavku slijedi nekoliko zadataka za vježbu, a odnose se na funkcije koje rade s poljima i stringovima. Točnije, odnose se na gradivo pokriveno poglavljem 10 i 11.

Napomena: Podrazumijeva se kako funkcije moraju moći raditi s poljima proizvoljnih veličina, odnosno stringovima proizvoljnih duljina. To znači kako je u zadacima opisan samo jedan mogući slučaj za potrebe testiranja [dio zadatka koji se odnosi na funkciju `main()`]. Nadalje, u zadacima gdje je potrebno omogućiti korisniku unos rečenice, pretpostavlja se kako string kojeg unosi korisnik može sadržavati praznine (razmake i tabulatore). Prema tome, valja koristiti odgovarajuću funkciju za učitanje ulaza s tipkovnice. Uz to je važno osigurati da se ne premaši veličina polja u koje se unos sprema. Na kraju, osim ako je drukčije navedeno, nije obavezno koristiti pokazivaču notaciju za pristup elementima polja/stringa.

Zadaci

1. Napisati funkciju koja će pronaći i vratiti element s najvećom vrijednosti u polju realnih brojeva. Funkciji se također predaju početni i završni indeks polja unutar čijih granica se obavlja pretraga. U svrhu testiranja u funkciji `main()` deklarirati i inicijalizirati polje od 10 realnih brojeva, za početni indeks postaviti 3, za završni 7 te pozvati napisanu funkciju s navedenim argumentima. Na ekran ispisati povratnu vrijednost funkcije.

2. Napisati funkciju koja će izračunati i vratiti srednju vrijednost elemenata polja realnih brojeva. Funkciji se također predaju početni i završni indeks polja unutar čijih granica se obavlja izračun. U svrhu testiranja u funkciji `main()` deklarirati i inicijalizirati polje od 12 realnih brojeva, za početni indeks postaviti 1, za završni 9 te pozvati napisanu funkciju s navedenim argumentima. Na ekran ispisati povratnu vrijednost funkcije.

3. Napisati funkciju koja će izračunati i vratiti apsolutnu vrijednost razlike najmanjeg i najveceg elementa prednog joj polja realnih brojeva. U svrhu testiranja u funkciji

`main()` omogućiti korisniku unos 20 elemenata u polje realnih brojeva. Potom, pozvati funkciju i ispisati na ekran povratnu vrijednost.

4. Napisati funkciju koja će izračunati i vratiti kvadrat razlike srednje vrijednosti elemenata i najveceg elementa prednog joj polja realnih brojeva. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos 17 vrijednosti u polje realnih brojeva. Potom, pozvati funkciju i ispisati na ekran povratnu vrijednost.

5. Napisati funkciju koja određuje i vraća znak s najmanjom ASCII vrijednosti u predanom joj stringu. Koristiti pokazivačku notaciju prilikom pristupa elementima polja. U svrhu testiranja u funkciji `main()` deklarirati i inicijalizirati polje znakova koje predstavlja string, proizvoljnog sadržaja. Pozvati funkciju i ispisati na ekran povratnu vrijednost.

6. Napisati funkciju koja negativne vrijednosti elemenata predanoga joj polja realnih brojeva mijenja u apsolutne vrijednosti. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos 10 vrijednosti u polje realnih brojeva. Pozvati funkciju i ispisati vrijednosti elemenata polja na ekran.

7. Napisati funkciju koja određuje i vraća znak s najvećom ASCII vrijednosti u predanom joj stringu. Koristiti pokazivačku notaciju prilikom pristupa elementima polja. U svrhu testiranja u funkciji `main()` deklarirati i inicijalizirati polje znakova koje predstavlja string, proizvoljnog sadržaja. Pozvati funkciju i ispisati na ekran povratnu vrijednost.

8. Napisati funkciju koja pozitivne vrijednosti elemenata predanoga joj polja realnih brojeva mijenja u kubne vrijednosti. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos 16 vrijednosti u polje realnih brojeva. Pozvati funkciju i ispisati vrijednosti elemenata polja na ekran.

9. Napisati funkciju koja određuje i vraća koliko elemenata predanog joj polja realnih brojeva ima vrijednost do 10 % veću od aritmetičke srednje vrijednosti svih elemenata tog polja. U svrhu testiranja u funkciji `main()` deklarirati polje od 20 elemenata i omogućiti korisniku njegovo popunjavanje. Pozvati funkciju s navedenim poljem kao argumentom i ispisati na ekran povratnu vrijednost.

10. Napisati funkcija koja određuje i vraća koliko različitih znakova sadrži predani joj string. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od maksimalno 200 znakova te pozvati funkciju s navedenim stringom kao argumentom i na ekran ispisati povratnu vrijednost.

11. Napisati funkciju koja će pronaći i vratiti najveću ASCII vrijednost znaka u predanom joj stringu. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od najviše 100 znakova. Pozvati napisanu funkciju sa navedenim stringom kao argumentom i na ekran ispisati povratnu vrijednost.

12. Napisati funkciju koja će odrediti i vratiti broj elemenata u polju realnih brojeva čija je vrijednost u intervalu $\langle a, b \rangle$. Funkciji se također predaju navedene vrijednosti a i b. U svrhu testiranja u funkciji `main()` deklarirati i inicijalizirati polje od 8 realnih brojeva, te korisniku omogućiti unos vrijednosti a i b takvih da su $0 \leq a, b \leq 20$ i $a < b$. Na ekran ispisati povratnu vrijednost funkcije.

13. Napisati funkciju koja u danom stringu svaku pojavu traženog znaka zamjenjuje s novim zadanim znakom. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa, znaka kojeg se želi zamijeniti i novog znaka te pozvati funkciju s navedenim kao argumentima. Nakon toga ispisati na ekran string.

14. Napisati funkciju koja će u predanom joj polju cijelih brojeva sve vrijednosti dvoznamenkastih elemenata pomnožiti s njihovom drugom znamenkom. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos 15 elemenata u polje cijelih brojeva. Pozvati napisanu funkciju s navedenim poljem kao argumentom.

15. Napisati funkciju koja će vratiti 1 ako u danom stringu ima više samoglasnika u odnosu na druge znakove, a u suprotnom će vratiti 0. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa. Nakon toga, pozvati napisanu funkciju s unesenim stringom kao argumentom i na ekran ispisati povratnu vrijednost.

16. Napisati funkciju koja će obaviti *normalizaciju* vrijednosti elemenata polja realnih brojeva. Normalizacija predstavlja dijeljenje svih vrijednosti u skupu s najvećom. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos 15 elemenata u polje realnih brojeva. Ispisati unesene elemente, a potom pozvati napisanu funkciju te ponovo ispisati elemente polja.

17. Napisati funkciju kojoj je moguće predati polje realnih brojeva i jedan dodatni realni broj g. Funkcija pronalazi i vraća koliko elemenata polja ima vrijednost veću od g. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos 18 elemenata u polje realnih brojeva. Potom, pozvati funkciju i ispisati na ekran povratnu vrijednost.

18. Napisati funkciju kojoj je moguće predati dva polja cijelih brojeva jednake duljine. Funkcija izračunava koliko ima jednakih vrijednosti elemenata s jednakim indeksom iz oba polja te vraća postotak takvih u odnosu na ukupnu veličinu polja. U svrhu testiranja u funkciji `main()` deklarirati dva polja od 10 elemenata. Omogućiti korisniku popunjavanje oba polja. Pozvati funkciju i ispisati na ekran povratnu vrijednost.

19. Napisati funkciju kojoj je moguće predati dva polja realnih brojeva jednake duljine. Funkcija izračunava razliku elemenata s jednakim indeksom iz oba polja te vraća zbroj tih razlika. U svrhu testiranja u funkciji `main()` deklarirati dva polja od 12 elemenata. Omogućiti korisniku popunjavanje oba polja. Potom, pozvati funkciju i ispisati na ekran povratnu vrijednost.

20. Napisati funkciju kojoj je moguće predati string. Funkcija vraća koliko se puta u danom stringu pojavljajuje veznik i. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos rečenice od maksimalno 140 znakova. Potom, pozvati funkciju i ispisati na ekran povratnu vrijednost.

21. Napisati funkciju kojoj je moguće predati string. Funkcija vraća vrijednost koja predstavlja omjer/kvocijent broja samoglasnika i ostalih znakova. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od maksimalno 100 znakova. Potom, pozvati funkciju i ispisati na ekran povratnu vrijednost.

22. Napisati funkciju kojoj je moguće predati string. Funkcija mijenja string tako da znakove postavlja obrnutim redoslijedom. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos rečenice od maksimalno 100 znakova. Pozvati funkciju te potom, ispisati string/rečenicu na ekran.

23. Napisati funkciju koja određuje i vraća koliko se puta pojavljuje znak s najvećom ASCII vrijednosti u stringu. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos rečenice od maksimalno 220 znakova. Pozvati funkciju i na ekran ispisati povratnu vrijednost.

24. Napisati funkciju koja određuje i vraća najmanji element predanog joj polja realinh brojeva koji je ujedno veći od srednje vrijednosti svih elemenata. U svrhu testiranja u funkciji `main()` deklarirati polje realnih brojeva od 22 elemenata i omogućiti korisniku njegovo popunjavanje. Potom, pozvati funkciju i na ekran ispisati povratnu vrijednost.

25. Napisati funkciju koja računa i vraća razliku srednjih vrijednosti elemenata predana joj dva polja realnih brojeva (predana dva polja ne moraju biti jednake veličine). U svrhu testiranja u funkciji `main()` deklarirati jedno polje realnih brojeva od 25 elemenata, i drugo od 15 elemenata. Omogućiti korisniku popunjavanje oba polja. Potom, pozvati funkciju i na ekran ispisati povratnu vrijednost.

26. Napisati funkciju koja znakove stringa sortira po ASCII vrijednostima od najveće prema najmanjoj. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od maksimalno 125 znakova. Pozvati funkciju i potom na ekran ispisati izmijenjeni string.

27. Napisati funkciju koja preslaguje vrijednosti elemenata prednog joj polja realnih brojeva tako da budu u obrnutom u odnosu na početno zadani redoslijed. Primjerice, obrnuti redoslijed elemenata s vrijednostima $[1, 2, 3, 4]$ je $[4, 3, 2, 1]$. U svrhu testiranja u funkciji `main()` deklarirati polje realnih brojeva od 30 elemenata i omogućiti korisniku njegovo popunjavanje. Pozvati funkciju, a potom na ekran ispisati vrijednosti elemenata polja.

28. Napisati funkciju koja predani joj string pretvara u palindrom tako što prepisuje drugu polovicu znakova prvom, ali u obratnom redoslijedu (funkcija mora moći raditi sa stringovima proizvoljnog broja znakova; pretpostaviti kako će uvijek biti paran broj znakova). Primjerice, funkcija treba string "abc2341a" promijeniti u "abc22cba". U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od maksimalno 255 znakova. Pozvati funkciju i potom ekran ispisati izmijenjeni string.

29. Napisati funkciju koja određuje i vraća prvi sljedeći broj po veličini u odnosu na najveći u predanom joj polju realnih brojeva. U svrhu testiranja u funkciji `main()` deklarirati polje realnih brojeva od 25 elemenata te omogućiti korisniku njegovo popunjavanje. Potom, pozvati funkciju i na ekran ispisati povratnu vrijednost.

30. Napisati funkciju koja u predanom joj stringu zamjenjuje samoglsnike prvim suglasnikom u stringu. Primjerice, funkcija treba string alm3ek promijeniti u llm3lk. U

svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od maksimalno 215 znakova. Pozvati funkciju i potom na ekran ispisati izmijenjeni string.

31. Napisati funkciju koja računa i vraća kvocijent broja elemenata manjih i broja elemenata većih od srednje vrijednosti svih u predanom joj polju cijelih brojeva. U svrhu testiranja u funkciji `main()` deklarirati polje cijelih brojeva od 125 elemenata te omogućiti korisniku njegovo popunjavanje. Potom, pozvati funkciju i na ekran ispisati povratnu vrijednost.

32. Napisati funkciju koja u predanom joj stringu zamjenjuje samoglsnike znakom znamenke koja odgovara ostatku pri cjelobrojnom dijeljenju indeksa s dva (indeks samoglasnika u polju modulo 2). Primjerice, funkcija treba string AhigeU promijeniti u `0h0g01`. U svrhu testiranja u funkciji `main()` omogućiti korisniku unos stringa od maksimalno 125 znakova. Pozvati funkciju i potom na ekran ispisati izmijenjeni string.

33. Napisati funkciju koja računa i vraća Lehmerovu srednju vrijednost elemenata predanog joj polja realnih brojeva. Funkcija računa Lehmerovu srednju vrijednosti prema:

$$s_q(x_1, \ldots, x_n) = \frac{\sum_{i=1}^{n} x_i^q}{\sum_{i=1}^{n} x_i^{q-1}},$$

gdje je parametara/eksponent $q \in \mathbb{R}$. U svrhu testiranja funkciji `main()` deklarirati polje realnih brojeva od 25 elemenata te omogućiti korisniku njegovo popunjavanje. Potom, pozvati funkciju i na ekran ispisati povratnu vrijednost. Za izračun potencija koristiti funkciju `pow()` iz standardne biblioteke (opisana u math.h), čiji je prototip: `double pow(double baza, double eksponent);`

34. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

void mean(double a[])
{
  double sum = 0
  int i,
  for (i = 0; i < n; i++)
    sum += *(a + i);

  return sum / n;
}

int main(void)
{
  double []seq = {1.23, 3.234, 4.3, 6.6, 0.394};
  int s = sizeof seq / sizeof(double);

  printf("%f\n", mean(seq, s));
  return 0,
}
```

35. Pronaći i ispraviti pogreške u sljedećem tekstu programa

```
#include <stdio.h>

int aboveAvg(float a, int n)
```

```
int main(void)
{
  float p[7] = {2.23, 3.24, 4.3, 1.6, 0.3, 0.02, 0.23};

  int r = aboveAvg(p, 7);
  printf("%d\n", r);

  return 0;
}

int aboveAvg(float a, int n)
{
  int sum = 0; c;
  int i, avg;

  for (i = 0; i < n; i++)
    sum + a[i];
  avg = sum / n;

  for (i = 0; i < n; i++)
    if (a[i] > avg) ++c;
}
```