Matematički fakultet

Univerzitet u Beogradu

Elektronski kurs o funkcijama u programiranju

- Master rad -

Marija Jordanović

Oktobar 2012.

Mentor:

Doc. dr Miroslav Marić

Sadržaj

1. Uvod

Funkcija u programiranju je izdvojena programska celina ili potprogram. Funkcija može, ali ne mora imati *ulazne* i *izlazne podatke*. Ulazni podaci funkcije nazivaju se još i *parametri* ili *argumenti*, a izlazni podaci predstavljaju *rezultat* funkcije koji se naziva i *povratna vrednost* funkcije.

Funkcija predstavlja neko izračunavanje, obradu koja se koristi više puta u programu. Tako se dobija kraći i jednostavniji kod. Takođe, kod je i elegantniji jer je neko izračunavanje skriveno, enkapsulirano u funkciji i ona može da se koristi čak i ako se ne zna kako je implementirana; dovoljno je znati šta funkcija radi, odnosno kakav je rezultat njenog rada za zadate argumente.

Upotreba funkcija u programiranju izuzetno je korisna iz više razloga. Najpre, funkcije se koriste da razdvoje programski kod u različite celine kako bi bio pregledniji. Takođe, na ovaj način se postiže izolacija problema. Od ostalih korisnih svojstava funkcija mogu se izdvojiti ponovno korišćenje istog programskog koda u drugim programima, kao i izbegavanje dupliranja koda u programu. Veoma je čest slučaj da se pojavi potreba za učestalom upotrebom jednog istog bloka koda unutar jednog programa. Ukoliko funkcije ne bi postojale, ovaj kod bismo, u najboljem slučaju morali kopirati na svako mesto gde nam je potrebno da pozovemo funkciju.

Osnovni delovi funkcije u većini programskih jezika su:

- Ime funkcije.
- Tip i imena (fiktivnih, odnosno formalnih) parametara.
- Tip povratne vrednosti.
- Kod (telo) funkcije.

U različitim programskim jezicima ovaj sastav se razlikuje manje ili više, u skladu sa sintaksom datog programskog jezika.

Svaka funkcija treba biti osmišljena tako da obavlja jedan dobro definisan zadatak i da korisnik funkcije ne mora poznavati detalje njene implementacije da bi je upotrebio. U tom slučaju, funkciju je moguće koristiti u različitim programima. Na primer, takve su funkcije iz standardne biblioteke.

Programi napisani u C-u sastoje se od niza međusobno povezanih funkcija čiji broj nije ograničen. U svakom programu je obavezna samo jedna funkcija, *main()* funkcija koja označava mesto na kojem počinje izvršavanje programa, dok se ostale funkcije koriste kako bi kod bio kraći i čitljiviji.

Bitna razlika između C-a i ostalih programskih jezika je u tome što u C-u ne postoje ugrađene funkcije, već se one nalaze u bibliotekama funkcija, koje se isporučuju zajedno sa prevodiocem. Biblioteke funkcija nastale su standardizacijom C-a, pa je dovoljno na početku programa najaviti da će se koristiti određena biblioteka i u celom programu dostupne su sve funkcije koje se u njoj nalaze. Primera radi, biblioteka u kojoj se nalaze funkcije za ulaz i izlaz

podataka naziva se *stdio.h*, matematičke funkcije nalaze se u biblioteci *math.h*, a funkcije za rad sa znakovnim promenljivima u biblioteci *string.h*.

Na početku svakog programa u C - u uobičajeno je pisati pretprocesorske naredbe kojima se pozivaju biblioteke funkcija. Najčešće se koristi naredba *#include<ime_datoteke>* i obavezno se navodi pre funkcije *main()*. Prilikom prevođenja programa, na mesto naredbe *#include* kopira se sadržaj navedene datoteke.

Ovde će biti prikazani primeri funkcija napisanih u programskom jeziku C. Postupak rešavanja problema pomoću funkcije isti je u svim programskim jezicima. Razlika je samo u sintaksi.

2. Definicija i deklaracija funkcije

Definicija funkcije sastoji se od zaglavlja i tela funkcije. Zaglavlje funkcije sadrži tip povratne vrednosti, identifikator (ime funkcije), par zagrada '(' i ')' sa opcionalnom listom parametara funkcije i definicije parametara funkcije. Telo funkcije je skup izvršnih iskaza i deklaracija promenljivih korišćenih u funkciji. Deklaracije promenljivih moraju se pisati pre izvršenih iskaza.

C

View Raw Code ?

```c
tip_podatka ime_funkcije(tip_1 arg_1, ... , tip_n arg_n)
{
    telo funkcije;
}
```

Slika 1. Definicija funkcije u programskom jeziku C.

Tip_podatka je tip podatka koji će funkcija vratiti kao rezultat izvršavanja. Unutar zagrada nalazi se deklaracija formalnih parametara funkcije. Prvi argument *arg_1* je tipa *tip_1* itd. Deklaracije parametara se međusobno odvajaju zarezom. Unutar vitičastih zagrada nalazi se telo funkcije koje se sastoji od deklaracije promenljivih i izvršnih naredbi. Podaci definisani na početku tela funkcije su lokalni za funkciju, tj. njihov doseg vrednosti je do kraja funkcije. To znači da se promenljive definisane u telu funkcije ne vide van te funkcije i nemaju uticaj na promenljive glavnog programa, čak ni u slučaju kada se koristi isti identifikator.

U slučaju da funkcija nema povratnu vrednost, deklariše se da ima povratni tip *void*, odnosno ispred imena funkcije navede se ključna reč *void*.

Pseudo kod | C

View Raw Code ?

```c
void ime_funkcije(tip_1 arg_1, ... , tip_n arg_n)
{
    telo funkcije;
}
```

Slika 2. Definicija funkcije bez povratne vrednosti u programskom jeziku C.

Ukoliko funkcija nema ulaznih parametara, potrebno je u deklaraciji funkcije umesto liste parametara upisati ključni reč *void* ili navesti samo prazne zagrade.

Pseudo kod | C

View Raw Code ?

```c
tip_funkcije ime_funkcije()
{
    telo funkcije;
}
```

Slika 3. Definicija funkcije bez ulaznih parametara u programskom jeziku C.

Funkcija koja ne uzima argumente poziva se najčešće u izrazu pridruživanja:

```
promenljiva = ime_funkcije();
```

Zagrade su obavezne jer one informišu prevodioca da je simbol *ime_funkcije* koji stoji ispred zagrada, u stvari, ime neke funkcije. Telo funkcije sastoji se od deklaracija promenljivih i izvršnih naredbi, isto kao i kod funkcija koje uzimaju argumente.

Deklaracija funkcije je definicija funkcije bez tela funkcije i završava se znakom tačka-zarez. Deklaracija se još naziva i *prototipom* funkcije. Deklaracija se mora pojaviti pre poziva funkcije, ako funkcija nije definisana pre toga. Definicija mora biti u skladu sa deklaracijom. Deklaracija funkcije ima oblik:

Pseudo kod | C

View Raw Code ?

```c
tip_funkcije ime_funkcije(tip_1 arg_1, ... , tip_n arg_n);
```

Slika 4. Deklaracija funkcije u programskom jeziku C.

Funkcija koja ima povratnu vrednost vraća rezultat svog izvršavanja pomoću naredbe *return*. Opšti oblik te naredbe je:

```
return izraz;
```

Vrednost izraza se vraća delu programa koji poziva funkciju. Važi sledeće pravilo:

- Funkcija može vratiti aritmetički tip, strukturu, uniju ili pokazivač, ali ne može vratiti drugu funkciju.

Ako je tip izraza u naredbi *return* različit od tipa podatka koji funkcija vraća, izraz će biti konvertovan u tip podatka. Međutim, takvu situaciju treba izbegavati.

Funkcija se poziva navođenjem imena funkcije i njenih stvarnih parametara. Naime, parametri funkcije nisu stvarni podaci koji se obrađuju tokom izvršenja funkcije. Zbog toga se oni nazivaju *formalni* ili *fiktivni* parametri. Prilikom poziva funkcije, fiktivni parametri se zamenjuju *stvarnim* parametrima čije se vrednosti koriste za izračunavanje vrednosti funkcije. Pri tom, lista stvarnih parametara mora po broju, tipu i redosledu parametara da se slaže sa listom fiktivnih parametara u definiciji funkcije. Ako se stvarni i fiktivni parametri ne slažu po tipu vrši se automatska konverzija (npr. int u float), a ako ne postoji automatska konverzija prevodilac javlja grešku. Imena stvarnih parametara mogu, ali ne moraju biti ista kao i imena fiktivnih parametara.

Jedan od najjednostavnijih C programa je čuveni program "Zdravo svete", koji ispisuje poruku "Zdravo svete" na standardni izlaz.

Pseudo kod | C

View Raw Code ?

```c
#include  <stdio.h >
main ()
{
    printf("Zdravo, svete!\n");
}

```

Slika 5. Kod programa "Zdravo, svete".

Pseudo kod | C

View Raw Code ?

```
Odštampati tekst "Zdravo svete";
Odštampati znak za novi red.

```

Slika 6. Pseudo kod programa "Zdravo, svete".

Postoje tri načina za pisanje korisničkih funkcija:

- Korisničke funkcije pišemo u istoj datoteci u kojoj se i koriste ispred funkcije *main()*.
- Navedemo prototip funkcije u datoteci čime najavljujemo njeno korišćenje, a zatim na kraju datoteke, ispod *main()* funkcije definišemo funkciju.

- Funkcije se pišu u zasebnim datotekama, pri čemu se u datoteci u kojoj se poziva funkcija navede prototip korišćene funkcije. Naredbom *#include<ime_datoteke>* datoteka koja sadrži definicije funkcija postaje vidljiva u celom programu.

Ilustracije radi, navešćemo primere funkcija u programskom jeziku C. U jednostavnije funkcije spada funkcija koja sabira dva broja. Takva funkcija ima povratnu vrednost jer kao rezultat vraća zbir dva broja. Prema tome, tip povratne vrednosti funkcije biće upravo tip rezultata sabiranja.

Primer 1. Napisati funkciju koja sabira dva cela broja, kao i program koji implementira zadatu funkciju.

Pseudo kod | C

View Raw Code ?

```c
#include <stdio.h>
int saberi(int x, int y)
{
    //uvodimo pomoćnu promenljivu
    int zbir;
    //računamo zbir
    zbir = x + y;
    //naredbom return vraćamo izraz pozivnoj funkciji
    return zbir;
}
main()
{
    int a, b, c;
    printf("Unesite prvi broj: ");
    scanf("%d", &a);
    printf("Unesite drugi broj: ");
    scanf("%d", &b);
    //poziv funkcije saberi
    c = saberi(a, b);
    printf("Zbir brojeva %d i %d je %d\n", a, b, c);
    /* Umesto prethodne dve naredbe mogli smo da napiemo i
    printf("Zbir brojeva %d i %d je %d\n", a, b, saberi(a, b)); */
}
```

Slika 7. Implementacija funkcije koja sabira dva cela broja.

Kada se u *main()* funkciji pojavi poziv funkcije npr.

```
c = saberi (a,b);
```

realizuju se sledeće akcije:

- Rezerviše se memorijski prostor za promenljive deklarisane u funkciji *saberi.*
- Izvršava se telo funkcije.

- Rezultat izračunavanja u funkciji se postavlja na mesto obraćanja toj funkciji, odnosno dodeljuje se promenljivoj c, i prelazi se na izvršenje sledeće naredbe u funkciji *main()*.

Primer 2. Napisati funkciju koja računa kvadrat nekog broja, funkciju koja računa cifru najmanje težine nekog broja i funkciju koja za ceo broj n vraća vrednost 0 ako je broj paran, odnosno 1 ako je broj neparan kao i program koji ih poziva.

Pseudo kod | C

View Raw Code ?

```c
#include <stdio.h>
int kvadrat(int x);
//funkcija koja računa kvadrat nekog broja
{
    int kv;
    kv = x*x;
    return kv;
}
//funkcija koja računa cifru najmanje težine broja
int cifra(int x)
{
    return x%10;
}
//funkcija koja za ceo broj n vrednost 0 ako je broj paran, odnosno 1 ako je broj neparan.
int neparan(int n)
{
    return n%2;
}
main()
{
    int a;
    printf("Unesite vrednost broja: ");
    scanf("%d", &a);
    //poziv funkcije koja računa cifru najmanje težine broja
    printf("Cifra najmanje težine broja %d je %d\n", a, cifra(a));
    //poziv funkcije koja računa kvadrat broja
    printf("Kvadrat broja %d je %d\n", a, kvadrat(a));
    //poziv funkcije neparan
    if(neparan(a)==0)
        printf("Broj %d je paran\n", a);
    else
        printf("Broj %d je neparan\n", a);
}
```

Slika 8. Rešenje Primera 2.

3. Funkcije ulaza i izlaza

Ulaz i izlaz u programskom jeziku C ostvaruju se posredstvom funkcija koje su definisane u standardnoj biblioteci *stdio.h*. Ove funkcije su obične C funkcije, koje se služe direktno servisima operativnog sistema (sistemskim pozivima) prilikom svog rada. Za korišćenje ovih funkcija neophodno je uključiti zaglavlje *stdio.h* navođenjem directive *#include<stdio.h>* pre definicije funkcije *main()*. Zaglavlje *stdio.h* je obilan tekstualni fajl u kome su navedene deklaracije funkcija ulaza i izlaza. Direktiva *#include* na mestu poziva uključuje kompletan sadržaj fajla koji je naveden, čime funkcije i podaci deklarisani u njemu postaju dostupni funkciji *main()*.

3.1. Funkcija printf

Funkcijom *printf()* se ispisuje poruka na standardni izlaz.

Pseudo kod | C

View Raw Code ?

```
1. printf(format, arg_1, arg_2, ... , arg_n);
```

Slika 9. Opšta forma funkcije printf.

Format je niz znakova koji sadrži informaciju o formatiranju ispisa argumenata *arg_1, arg_2, ..., arg_n*.

Eventualni konverzioni specifikatori zamenjuju se vrednostima izraza koji u tom slučaju slede nakon format stringa, kao parametri funkcije *printf()*, razdvojeni zarezima i u onom poretku u kome su odgovarajući konverzioni specifikatori navedeni. Tipovi izraza moraju biti u skladu sa tipovima koje određuju konverzioni specifikatori.

Funkcija *printf()* je bibliotečka funkcija koja prikazuje izlazne podatke u određenom formatu. Primer korišćenja funkcije *printf()* je:

```
printf("%d\t%d\n", broj1, broj2);
```

Prvi argument ove funkcije je uvek između navodnika i određuje format u kome će podaci biti ispisani. Funkcija vraća broj upisanih znakova na izlazu.

Sekvenca `\n` u okviru prvog argumenta funkcije *printf()* je C oznaka za prelazak u novi red, `\t` je oznaka za tabulator, dok `%d` označava da će na tom mestu biti ispisana celobrojna vrednost argumenta koji je sa njim u paru. Svaka `%` konstrukcija je u paru sa odgovarajućim argumentom koji sledi.

- %o – koristi se za ispis oktalnog broja.
- %x – koristi se za ispis heksadekadnog broja.
- %u – koristi se za ispis neoznačenog dekadnog broja.

Moguće je da se precizira i širina polja u kome će se ispisati odgovarajuće vrednosti. Na primer, koristimo %3c za štampanje karaktera na tri pozicije poravnato udesno, ili %3d za štampanje celog broja na tri pozicije, ili %6d za štampanje celog broja na 6 pozicija. Da bi se izvršilo levo poravnanje, između % i odgovarajućeg karaktera dodaje se znak -. Važe sledeća pravila:

- %f – štampaj kao realan broj.
- %6f – štampaj kao realan broj širok najviše šest znakova.
- %.2f – štampaj kao realan broj sa dve decimale.
- %6.2f – štampaj kao realan broj širok najviše šest znakova pri čemu su dva iza decimalne tačke.

Sledeći primer ilustruje poziv funkcije *printf()*.

Primer 3. Ispisati na standardni izlaz karakter *z* širok najviše tri znaka i karakter *Z*, širok najviše pet znakova.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
main()
{
    printf("Slova: \n%3c\n%5c\n", 'z', 'Z');
    return 0;
}

```

Slika 9. Rešenje Primera 2.

Izlaz iz programa | C

View Raw Code ?

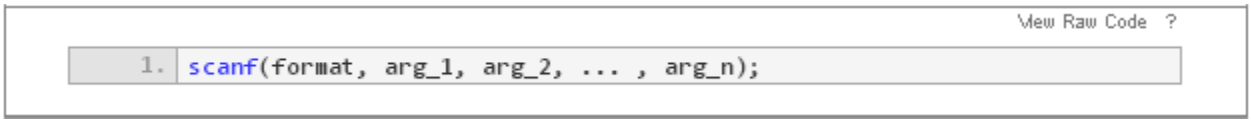
```
Slova:
  z
    Z

```

Slika 10. Prikaz ekrana nakon izvršenja programa sa Slike 8.

3.2. **Funkcija scanf**

Funkcijom *scanf()* učitavaju se podaci sa standardnog ulaza (tastature).



Slika 11. Opšta forma funkcije scanf.

Prvi parametar je format string u kome se navode konverzioni specifikatori kojima se definiše tip očekivanog podatka. Nakon format stringa slede adrese promenljivih, razdvojene zarezima, u koje treba upisati vrednosti učitane sa ulaza. Adresa promenljive *a* navodi se sa *&a*. Adrese se navode u onom poretku u kom su odgovarajući konverzioni specifikatori navedeni u format stringu. Tipovi promenljivih moraju biti u skladu sa tipovima koje određuju konverzioni specifikatori.

Parametri funkcije *scanf()* mogu biti samo pokazivači na promenljive. Ako je potrebno učitati podatak u neku promenljivu, onda *scanf()* uzima kao parametar adresu te promenljive, a ne samu promenljivu. To znači da pri pozivu funkcije *scanf()* ispred imena promenljive u koju *scanf()* učitava vrednost moramo staviti adresni operator *&*.

Funkcija *scanf()* vraća broj uspešno učitanih podataka ili *EOF*. Tu činjenicu možemo iskoristiti za proveru da li su svi traženi podaci učitani.

Primer korišćenja funkcije *scanf()* je:

```
scanf("%d %d", &a, &b);
```

Ova funkcija čita sa ulaza dva cela broja i smešta ih na adresu promenljivih `a` i `b`, redom. Kao rezultat, funkcija vraća broj uspešno dodeljenih ulaznih vrednosti. Naredni poziv funkcije *scanf()* nastavlja čitanje neposredno iza poslednjeg znaka koji je već pročitan.

*Tabela 1. Konverzioni specifikatori za funkcije **printf** i **scanf***

Specifikator	Tip	Napomena
%d	int	opciono označeni dekadni broj
%f	float	realni broj sa opcionim eksponentom
%lf	double	realni broj sa opcionim eksponentom
%Lf	long double	realni broj sa opcionim eksponentom
%hd	short	opciono označeni heksadekadni broj
%ld	long	opciono označeni dekadni broj
%c	char	ASCII karakter

Primer 4. Sledeći program prikazuje unos celog broja pomoću funkcije *scanf()*:

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
main()
{
    int x;
    printf("Unesi ceo broj: ");
    /* obratiti pažnju na znak & pre imena promenljive
    u funkciji scanf */
    scanf("%d", &x);
    // u funkciji printf nije potrebno stavljati &
    printf("Uneli ste broj:%d\n", &x);
    return 0;
}
```

Slika 12. Kod programa za unos celog broja pomoću funkcije scanf().

Primer 5. Napisati funkciju koja podiže bazu na stepen n.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
float stepen(int x, int n)
{
    int i, p, n_pos;
    p = 1;
    if (n >= 0)
        n_pos = n;
    else
        n_pos = n * -1;
    for(i=1; i <= n_pos; ++i)
        p=p*x;
    if(n >= 0)
        return p;
    else
        return 1/p;
}
int main()
{
    printf("Rezultat funkcije za x=3, n=4 je %d\n", stepen(3, 4));
    printf("Rezultat funkcije za x=3, n=-4 je %d\n", stepen(3, -4));
}
```

Slika 13. Rešenje Primera 5.

4. Prenos parametara po vrednosti

Kao što je ranije rečeno, parametri deklarisani u definiciji funkcije nazivaju se *formalni parametri*. Izrazi koji se pri pozivu funkcije nalaze na mestima formalnih parametara nazivaju se *stvarni parametri*.

Parametri se funkciji prenose po vrednosti, što znači da funkcija zapravo radi sa njihovim lokalnim kopijama. Prilikom poziva funkcije stvarni parametri se izračunavaju (ako su izrazi) i kopiraju u formalne parametre. Funkcija prima kopije stvarnih parametara i ne može ih menjati.

Ako je argument funkcije niz, funkcija dobija pokazivač na prvi element niza. Preko adrese prvog elementa niza može se doći do bilo kog drugog elementa pomoću indeksa.

Primer 6. Ilustracija načina prenosa parametara.

Izlaz iz programa | C

View Raw Code ?

```c
#include<stdio.h>
void f(int x)
{
    x+=1;
    printf("\nUnutar funkcije x=%d",x);
    return;
}
int main(void)
{
    int x=5;
    printf("\nIzvan funkcije x=%d",x);
    f(x);
    printf("\nNakon poziva funkcije x=%d",x);
    return 0;
}
```

Slika 14. Ilustracija načina prenosa parametara.

Izlaz iz programa | C

View Raw Code ?

```
Izvan funkcije x=5
Unutar funkcije x=6
Nakon poziva funkcije x=5
```

Slika 15. Prikaz ekrana nakon izvršavanja prethodnog programa.

Istaknimo najvažnija pravila o prenosu parametara:

- Broj stvarnih parametara pri svakom pozivu funkcije mora biti jednak broju formalnih parametara.
- Ukoliko se tip stvarnih parametara razlikuje od tipa odgovarajućih formalnih parametara, vrši se konverzija u tip formalnih parametara, isto kao pri pridruživanju. Takva konverzija pri tom mora biti moguća.
- Redosled izračunavanja stvarnih parametara nije definisan i može zavisiti od implementacije.

5. Lokalne i globalne promenljive

C program sastoji se od skupa spoljašnjih objekata, koji mogu biti *promenljive* i *funkcije*. U programskom jeziku C sve promenljive moramo deklarisati pre njihove upotrebe. Deklaracija obuhvata tri svojstva promenljive: tip, doseg i vek trajanja. Tip promenljive uvek se uvodi eksplicitno ključnim rečima, npr. *int, float, double*. Doseg i trajanje promenljive određeni su položajem deklaracije u programu, a mogu se modifikovati pomoću ključnih reči *static* i *extern*.

Doseg promenljive je deo programa u kome je njena deklaracija vidljiva i u kome se, stoga, promenljivoj može pristupiti preko njenog imena. Dva su osnovna tipa dosega: *blok* i *datoteka*. Promenljive s dosegom bloka nazivamo *lokalnim* (*unutrašnjim*) promenljivim, dok su promenljive s dosegom datoteke *globalne* (*spoljašnje*) promenljive. Ako je izvorni kod razbijen u više datoteka, onda su globalne promenljive deklarisane sa *extern* vidljive i izvan datoteke u kojoj su definisane.

Blok naredbi čini svaki niz naredbi ograničen vitičastim zagradama. Deklaracija lokalne promenljive unutar nekog bloka mora prethoditi prvoj izvršnoj naredbi u tom bloku. Lokalna promenljiva je vidljiva samo unutar bloka u kome je definisana, izvan bloka njeno ime nije definisano i može postojati deklaracija druge promenljive sa istim imenom. Slično, formalni parametar funkcije vidljiv je samo unutar funkcije. Dakle, doseg promenljive definisane unutar nekog bloka je taj blok, a doseg formalnog parametra je telo funkcije.

Promenljiva deklarisana u spoljašnjem bloku vidljiva je u svakom untrašnjem bloku ako u njemu nije definisana promenljiva istog imena. Ako je promenljiva definisana izvan svih blokova, onda je njen doseg čitava datoteka u kojoj je definisana. Takvu promenljivu nazivamo *globalnom*.

Spoljašnje promenljive su definisane izvan svake funkcije i zato su potencijalno na raspolaganju raznim funkcijama. Funkcije su uvek spoljašnje, jer C ne dozvoljava definisanje funkcija unutar drugih funkcija. Ukoliko funkcije razmenjuju veliki broj parametara, nekada je pogodnije da to čine preko eksternih promenljivih. Ipak, to nije preporučljivo jer narušava strukturu programa i može dovesti do neželjenih efekata.

Parametri navedeni u prototipu funkcije imaju doseg koji ne seže dalje od prototipa. To znači da imena navedena u prototipu nisu važna i mogu se podudarati s imenima drugih promenljivih u programu. Štaviše, prevodilac nam dopušta da izostavimo imena promenljivih u prototipu.

Prenošenje podataka između funkcija odvija se na tri načina:

- Preko parametara.
- Preko povratnih vrednosti.
- Preko spoljašnjih promenljivih.

Ukoliko se veliki broj promenljivih mora deliti između funkcija, bolje je koristiti spoljašnje promenljive nego dugačke liste parametara. Međutim, ovo treba pažljivo primenjivati, jer u suprotnom se dobija program sa previše veza između funkcija.

6. Vek trajanja promenljive

Svakoj promenljivoj prevodilac pridružuje određen memorijski prostor. Vek trajanja promenljive je vreme za koje joj je pridružena njena memorijska lokacija, odnosno to je vreme njene egzistencije. Prema veku trajanja, promenljive delimo na *automatske* i *statičke*.

Promenljiva kreirana unutar nekog bloka, koja nije deklarisana ključnom reči *static* je *automatska* promenljiva. Automatske promenljive se kreiraju na ulasku u blok u kome su deklarisane i uništavaju se na izlasku iz njega, osobađajući memoriju koju su zauzimale.

Uočimo razlike između spoljašnjih i automatskih promenljivih:

- Automatske promenljive su unutrašnje za funkciju. One nastaju kada otpočinje izvršavanje funkcije i nestaju kada se ono završava.
- Spoljašnje promenljive su permanentne i stoga zadržavaju svoje vrednosti između poziva funkcija. Dakle, životni ciklus ovih promenljivih je duži.

Statičke promenljive alociraju se i inicijalizuju na početku programa, a uništavaju se tek na završetku programa. Prostor za statičke promenljive alocira se u delu memorije koji nije isti kao i prostor za alokaciju automatskih promenljivih. Svaka promenljiva definisana izvan svih funkcija je statička. Promenljiva deklarisana u nekom bloku s identifikatorom memorijske klase *static* je takođe statička promenljiva.

Deklaracija *static*, koja se primenjuje na spoljašnju promenljivu ili funkciju, ograničava domet tog objekta na preostali deo izvorne datoteke. Može se primeniti i na unutrašnje promenljive. Unutrašnje statičke promenljive su lokalne za određenu funkciju baš kao i automatske promenljive, ali za razliku od automatskih promenljivih, one nastavljaju da postoje i nakon završetka funkcije.

Primer 7. Primer ukazuje na razlike između globalnih i lokalnih, i na razlike između statičkih i nestatičkih promenljivih.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
// globalna promenljiva a
int a = 0;
// uvećava se globalna promenljiva
void uvecaj()
{
    a++;
    printf("uvecaj: a = %d\n", a);
}
/* umanjuje se lokalna promenljiva a, dok globalna promenljiva
sa istim imenom zadržava vrednost */
void umanji()
{
    int a = 0;
    a--;
    printf("umanji: a = %d\n", a);
}
void nestaticka_prom()
{
    // nestatičke promenljive ne čuvaju vrednosti kroz pozive funkcije
    int s = 0;
    s++;
    printf("nestatička promenljiva: s = %d\n", s);
}
void staticka_prom()
{
    /* statičke promenljive čuvaju vrednosti kroz pozive funkcije,
    inicijalizacija se odvija samo u okviru prvog poziva */
    static int s = 0;
    s++;
    printf("statička promenljiva: s = %d\n", s);
}

```

Slika 16. Ilustracija razlika između lokalnih i globalnih, odnosno statičkih i nestatičkih promenljivih.

U nastavku ćemo prikazati primer pozivanja funkcija *nestaticka_prom()*, *staticka_prom()*, *uvecaj()*, *umanji()*, unutar funkcije *main()*, kako bi se jasnije uočile razlike između lokalnih i globalnih, kao i između statičkih i nestatičkih promenljivih.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
void uvecaj();
void umanji();
void nestaticka_prom();
void staticka_prom();
int main()
{
    // ovo su promenljive lokalne za funkciju main()
    int i;
    int x = 3;
    printf("main: x = %d\n", x);
    for(i = 0; i < 3; i++)
    {
        /* promenljiva u okviru bloka je nezavisna od spoljne
        promenljive; ovde se koristi promenljiva x, lokalna za
        blok for petlje koja ima vrednost 5, dok originalno x
        i dalje ima vrednost 3 */
        int x = 5;
        printf("for: x = %d\n", x);
    }
    // u ovom bloku x ima vrednost 3
    printf("main: x = %d\n", x);
    uvecaj();
    umanji();
    // globalna promenljiva a
    printf("main: a = %d\n", a);
    // demonstracija nestatičkih promenljivih
    for(i = 0; i < 3; i++)
        nestaticka_prom();
    // demonstracija statičkih promenljivih
    for(i = 0; i < 3; i++)
        staticka_prom();
    return 0;
}

```

Slika 17. Implementacija funkcija uvecaj(), umanji(), nestaticka_prom(), staticka_prom()

Na *Slici 18.* prikazan je program koji poziva funkcije sa *Slike 17.* Naravno, neophodno je pre funkcije *main()* definisati sve funkcije, ali pošto je njihova definicija navedena na *Slici 17*, ovde je izostavljamo. Navedeni su samo prototipovi funkcija.

Spoljašnje i statičke promenljive se uvek inicijalizuju na nulu ukoliko nisu eksplicitno inicijalizovane. Inicijalizator mora biti konstantan izraz. Inicijalizacija se obavlja samo jednom, pre početka izvršavanja programa. Automatske promenljive imaju nedefinisane početne

vrednosti ukoliko nisu eksplicitno inicijalizovane. Inicijalizator može biti svaki izraz u kojem učestvuju prethodno definisane vrednosti, pa čak i pozivi funkcija. Inicijalizacija se obavlja svaki put prilikom izvršavanja funkcije ili bloka.

7. Primeri jednostavnijih funkcija

Primer 8. Napisati funkciju koja proverava da li je zadati broj prost; ako jeste vraća 1, ako nije vraća 0.

Pseudo kod | C

View Raw Code ?

```c
#include <stdio.h>
// funkcija koja proverava da li je broj prost
int prost(int x)
{
    int i;
    if (x <= 0)
        return 0;
    if (x == 1 || x == 2)
        return 1;
    if (x%2 == 0)
        return 0;
    int brDelilaca = 2;
    for (i = 3; i <= x/2; i++)
        if (x%i == 0)
            brDelilaca++;
    if (brDelilaca > 2)
        return 0;
    else
        return 1;
}
main()
{
    int broj;
    printf("Unesite broj: ");
    scanf("%d", &broj);
    if (prost(broj) == 1)
        printf("Broj je prost.");
    else
        printf("Broj nije prost.");
}
```

Slika 18. Rešenje primera 8.

Primer 9. Napisati funkciju koja računa minimum dva broja, funkciju koja računa minimum tri broja. Koristeći funkciju za minimum dva broja napisati funkciju koja računa minimum četiri broja. Pomoću nje napisati funkciju koja određuje najmanju među poslednjim ciframa četiri broja.

Pseudo kod | C

View Raw Code ?

```c
// funkcija koja računa minimum dva broja
int min(int a, int b)
{
    int min;
    if (a < b)
        min = a;
    else
        min = b;
    /* Umesto if-else naredbe, možemo koristiti:
    min = (a < b) ? a : b; */
    return min;
}
// funkcija koja računa minimum tri broja
int min3(int a, int b, int c)
{
    int min;
    if (a < b && a < c)
        min = a;
    else if (b < c)
        min = b;
    else
        min=c;
    /* Umesto if-else naredbe, možemo koristiti:
    min = (a < b) ? (a < c) ? a : c : (b < c) ? b : c; */
    return min;
}
/* funkcija koja računa minimum četiri broja korišćenjem funkcije
koja računa minimum dva broja */
int min4(int a, int b, int c, int d)
{
    return min(min(a, b), min(c, d));
}
/* funkcija određuje najmanju među poslednjim ciframa 4 broja */
minPoslCifra(int a, int b, int c, int d)
{
    return min4(a%10, b%10, c%10, d%10);
}
```

Slika 19. Rešenje Primera 9.

Primer 10. Napisati funkciju koja za zadati ceo broj vraća broj sa istim ciframa, ali u obrnutom poretku.

Pseudo kod | C

View Raw Code ?

```c
#include <stdio.h>
// funkcija koja obrće cifre unetog broja
int obrnutiBroj(int n)
{
    int rez = 0;
    while(n!=0)
    {
        rez = rez*10 + n%10;
        n/=10;
    }
    return rez;
}
int main()
{
    int n;
    int obrnuti;
    printf("Unesite broj koji obrćemo:\n");
    scanf("%d", &n);
    obrnuti = obrnutiBroj(n);
    printf("%d nakon obrtanja postaje %d\n", n, obrnuti);
    return 0;
}

```

Slika 20. Rešenje Primera 10.

Pseudo kod | C

View Raw Code ?

```
Definišemo funkciju;
postavljamo početnu vrednost rezultata, rez=0;
pomoću while petlje rezultat množimo sa 10 i dodajemo
poslednju cifru unetog broja, sve dok ne unesemo 0;
u programu unosimo željeni broj, a zatim pozivamo funkciju
obrnutiBroj;
na kraju ispišemo dobijeni broj.

```

Slika 21. Pseudo kod za primer 10.

8. Funkcije za rad sa nizovima

Često je u programima potrebno korišćenje velikog broja srodnih promenljivih. Umesto velikog broja pojedinačno deklarisanih promenljivih moguće je koristiti nizove. Obrada elemenata nizova se onda vrši na uniforman način, korišćenjem petlji.

Razmotrimo, kao ilustraciju, program kojim se izračunava koliko se puta javlja svaka od cifara u tekstu učitanom sa standardnog ulaza, do prve pojave tačke. Bez korišćenja nizova, program zahteva 10 različitih brojačkih promenljivih.

Primer 11. Izračunati koliko se puta javlja svaka od cifara u tekstu učitanom sa standardnog ulaza do prve pojave tačke.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
int main()
{
    int b0=0, b1=0, b2=0, b3=0, b4=0,
        b5=0, b6=0, b7=0, b8=0, b9=0, c;
    while(c = getchar() != '.')
    {
        switch(c)
            {
                case '0': b0++; break;
                case '1': b1++; break;
                case '2': b2++; break;
                case '3': b3++; break;
                case '4': b4++; break;
                case '5': b5++; break;
                case '6': b6++; break;
                case '7': b7++; break;
                case '8': b8++; break;
                case '9': b9++; break;
            }
    }
    printf("%d %d %d %d %d %d %d %d %d %d\n",
           b0, b1, b2, b3, b4, b5, b6, b7, b8, b9);
    return 0;
}
```

Slika 22. Rešenje Primera 11 bez korišćenja nizova.

Umesto 10 različitih srodnih promenljivih, moguće je upotrebiti niz i time značajno uprostiti prethodni program.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
#include<ctype.h>
int main()
{
    int b[10], c, i;
    for(i = 0; i < 10; i++)
        b[i] = 0;
    while(c = getchar() != '.')
        if(isdigit(c))
            b[c - '0']++;
    for(i = 0; i < 10; i++)
        printf("%d", b[i]);
    return 0;
}
```

Slika 23. Rešenje Primera 11 pomoću niza.

Na *i*-tom mestu u nizu b se smešta broj pojavljivanja cifre *i*. Funkcija standardne biblioteke *isdigit* deklarisana u zaglavlju *ctype.h*, je upotrebljena da bi se proverilo da li je uneti karakter cifra. Ako jeste, od njegovog koda se oduzima ASCII kod karaktera '0' kako bi se dobila odgovarajuća pozicija u nizu brojača. Ova funkcija vraća ne-nula vrednost ako je c cifra, nulu inače.

Pseudo kod | C

View Raw Code ?

```c
int isdigit(char c)
{
    if (c >= '0' && c <= '9')
        return 1;
    return 0;
}
```

Slika 24. Definicija funkcije isdigit.

Za čitanje pojedinačnih karaktera sa standardnog ulaza upotrebljena je funkcija standardne C biblioteke *getchar()* deklarisana u zaglavlju *stdio.h*. Najjednostavniji mehanizam čitanja sa standardnog ulaza je da se čita jedan po jedan karakter korišćenjem funkcije *getchar()*.

```
int getchar(void);
```

Funkcija *getchar()* vraća sledeći karakter sa ulaza, svaki put kada se pozove, ili *EOF* kada dođe do kraja toka. Simbolička konstanta *EOF* je definisana u zaglavlju *stdio.h*. Njena

vrednost je najčešće -1, ali testovi bi trebalo da se vrše u odnosu na simbol *EOF*, nezavisno od specifične vrednosti na koju je ova konstanta definisana.

Funkcija *getchar()* najčešće se realizuje tako što karakter uzima iz privremenog bafera koji se puni čitanjem jedne po jedne linije ulaza. Dakle, pri interaktivnom radu sa programom, *getchar()* neće imati efekta dok se ne unese prelazak u novi red ili oznaka za kraj datoteke.

Najjednostavniji mehanizam pisanja na standardni izlaz je da se piše jedan po jedan po jedan karakter korišćenjem funkcije *putchar*:

```
int putchar(int);
```

Funkcija *putchar(c)* štampa karakter *c* na standardni izlaz, najčešće ekran, a vraća karakter koji je ispisala, odnosno *EOF* ukoliko dođe do greške.

Kao primer rada sa pojedinačnim karakterima, razmotrimo sledeći primer.

Primer 12. Napisati program koji prepisuje standardni ulaz na standardni izlaz, pretvarajući pri tom velika slova u mala.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
#include<ctype.h>
int main()
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(tolower(c));
    return 0;
}
```

Slika 25. Rešenje Primera 12.

Funkcija *tolower* deklarisana je u zaglavlju *ctype.h* i prevodi velika slova u mala, ne menjajući pri tom ostale karaktere.

Pseudo kod | C

View Raw Code ?

```c
char tolower(char c)
{
    if (c >= 'A' && c <= 'Z')
        c = c - 'A' + 'a';
    return c;
}
```

Slika 26. Definicija funkcije tolower.

Zaglavlje *ctype.h* sadrži deklaracije nekoliko funkcija za ispitivanje i konvertovanje karaktera, koje se često koriste prilikom rada sa stringovima.

- **isalpha(c)** vraća ne-nula vrednost ako je c slovo, nulu inače;
- **isupper(c)** vraća ne-nula vrednost ako je slovo c veliko, nulu inače;
- **islower(c)** vraća ne-nula vrednost ako je slovo c malo, nulu inače;
- **isdigit(c)** vraća ne-nula vrednost ako je c cifra, nulu inače;
- **isalnum(c)** vraća ne-nula vrednost ako je c slovo ili cifra, nulu inače;
- **isspace(c)** vraća ne-nula vrednost ako je c belina (razmak, tab, novi red, itd), nulu inače;
- **toupper(c)** vraća karakter c konvertovan u veliko slovo ili sam karakter c;
- **tolower(c)** vraća karakter c konvertovan u malo slovo ili sam karakter c;

8.1. **Deklaracija nizova**

Nizovi u programskom jeziku C se mogu deklarisati na sledeći način:

Pseudo kod | C

View Raw Code ?

```
1. tip ime_niza[broj_elemenata];
```

Slika 27. Deklaracija niza.

Niz se ne može preneti kao argument funkcije. Umesto toga, moguće je kao argument proslediti ime niza. Imenu niza pridružena je informacija o adresi početka niza, o tipu elemenata niza i o broju elemenata niza. Ako je dužina niza *n*, prvi član niza ima indeks 0, drugi član ima indeks 1, ..., *n*-ti član ima indeks *n-1*. Kada se ime niza prosledi kao argument funkcije, onda do funkcije stiže informacija o adresi početka niza i o tipu elemenata. Prenos takvog argumenta vrši se, kao i uvek, po vrednosti. S obzirom na to da funkcija koja je pozvana dobija informaciju o adresi početka niza, ona može neposredno da menja njegove elemente.

U programskom jeziku C niz se može formirati na *statički* i *dinamički* način. Statički način podrazumeva deklaraciju niza sa fiksnim dimenzijama koje ne možemo menjati u daljem kodu programa. Program će sam brinuti o alociranju odgovarajućeg memorijskog prostora kao i o oslobađanju istog nakon postojanja promenljive. Međutim, glavni nedostatak ovog pristupa je to što programer mora da predvidi koliko će mu prostora biti potrebno za podatke. Još jedan nedostatak je to što je životni vek svakog statički alociranog podatka ograničen.

Prethodni problemi rešavaju se dinamičkom alokacijom memorije. U memoriji postoji i određeni prostor koji se naziva hip[1]. U ovom prostoru se po potrebi mogu kreirati objekti proizvoljne veličine, odnosno nizovi proizvoljne dužine. Taj prostor se kasnije može (i treba)

[1] Termin *hip* potiče od engleske reči *heap* što znači *gomila, hrpa*.

osloboditi. U programskom jeziku C, dinamička alokacija memorije ostvaruje se preko funkcija *malloc*, *calloc* i *realloc* deklarisanih u zaglavlju *stdlib.h*. Dealokacija zauzete memorije vrši se pomoću funkcije *free* iz istog zaglavlja.

- `void *malloc(int n)` - alocira na hipu kontinualni prostor od n bajtova, i vraća adresu pocetka tog prostora; jedini način da se ovom prostoru pristupi je preko pokazivača.
- `void *calloc(int n, int s)` - alocira prostor za n susednih elemenata veličine s bajtova (zapravo alocira $n * s$ bajtova) i, za razliku od funkcije *malloc*, inicijalizuje rezervisani prostor na vrednost 0.
- `void *realloc(void *p, int n)`- vrši realokaciju prostora na koji pokazuje pokazivač p koji je prethodno bio dinamički alociran; alocira se n bajtova, kopira se stari sadržaj u novi prostor, a zatim se stari prostor briše. Ova funkcija se obično koristi da bi se proširio postojeći dinamički alocirani prostor.
- `void free(void *p)`- dealocira prostor alociran nekom od prethodnih funkcija; kada se jednom dealocira, prostor više ne sme da se koristi.

Primer 13. Sastaviti funkciju koja ispituje da li dati niz sadrži zadati element.

Pseudo kod | C

View Raw Code ?

```c
// funkcija ispituje da li dati niz sadrži zadati element
int sadrzi(int a[], int n, int x)
{
    int i;
    for(i = 0; i < n; i++)
        if(a[i] == x)
            return 1;
    return 0;
}
```

Slika 28. Rešenje Primera 13.

Pseudo kod | C

View Raw Code ?

```
a[] je zadati niz, n je dužina niza, x je zadati broj;
postavimo brojač i na vrednost 1, proveravamo da li se na poziciji
i u nizu nalazi broj x; ako se ne nalazi uvecavamo i za 1.
Ovu petlju ponavljamo dok i ne postane jednako broju elemenata niza.

```

Slika 29. Pseudo kod za primer 13.

Primer 14. Napisati funkciju koja vraća indeks prve pozicije datog broja, -1 ako ga nema, kao i funkciju koja vraća indeks poslednje pozicije datog broja, odnosno -1 ako niz ne sadrži taj broj.

Pseudo kod | C

View Raw Code ?

```c
// funkcija koja vraća indeks prve pozicije datog broja
// a[] je dati niz, n dužina niza, x dati broj
int prvaPozicija(int a[], int n, int x)
{
    int i;
    // polazimo od prvog člana u nizu, koji ima nultu poziciju
    for(i = 0; i < n; i++)
        if(a[i] == x)
            /* i je pozicija člana a[i], odnosno
            broja x ukoliko je a[i]=x */
            return i;
    return -1;
}
// funkcija koja vraća indeks prve pozicije datog broja
// a[] je dati niz, n dužina niza, x dati broj
int poslednjaPozicija(int a[], int n, int x)
{
    int i;
    //polazimo od poslednjeg člana u nizu, koji ima (n-1)-vu poziciju u nizu
    for(i = n-1; i >= 0; i--)
        // i je pozicija člana a[i], odnosno broja x ukoliko je a[i]=x
        if(a[i] == x)
            return i;
    return -1;
}
```

Slika 30. Rešenje primera 14.

Funkcija `prva_pozicija` poznata je i kao linearna pretraga niza. Linearna pretraga se vrši prostom iteracijom kroz niz. Dakle, neophodno je proći kroz sve članove niza, što algoritam linearne pretrage čini nedovoljno efikasnim. Mnogo efikasnija od linearne pretrage jeste binarna pretraga niza. Kod binarne pretrage ne moramo proći kroz sve elemente niza da bismo zaključili da li se dati element nalazi u nizu ili ne. Međutim, da bismo mogli da primenimo binarnu pretragu, niz koji pretražujemo mora da bude sortiran. Funkcija koja radi binarnu pretragu vraća indeks pozicije na kojoj je pronađen dati element, ili vrednost -1 ukoliko se element ne nalazi u nizu.

Primer 15. Binarna pretraga niza

Pseudo kod | C

View Raw Code ?

```c
int binarna_pretraga (int a[], int n, int x)
{
    int l = 0;
    int d = n  1;
    // dokle god je indeks l levo od indeksa d
    while (l <= d)
    {
        // računamo središnji indeks
        int s = (l + d) / 2;
        /* Ako je središnji element veći od x,
        tada se x mora nalaziti u levoj polovini niza */
        if (x < a[s])
            d = s - 1;
        /* Ako je središnji element manji od  x,
        tada se x mora nalaziti u desnoj polovini niza */
        else if (x > a[s])
            l = s + 1;
        else
        /* Ako je središnji element jednak x,
        tada smo pronašli x na poziciji s */
            return s;
    }
```

Slika 31. Binarna pretraga niza.

Binarna pretraga je tzv. "podeli, pa vladaj" pretraživački algoritam. Da bismo primenili binarnu pretragu moramo prethodno sortirati niz. Nakon toga, proveravamo da li je dati element manji ili veći od središnjeg člana niza. Kako je niz sortiran, lako se uočava u kojoj polovini niza je moguće pronaći zadati element, stoga se odbacuje druga polovina niza. To skraćuje vreme pretraživanja, zbog čega je ovaj algoritam mnogo efikasniji od linearne pretrage.

Ukratko, binarnu pretragu možemo opisati na sledeći način:

- Pronalazimo centralni element.
- Odbacimo polovinu u kojoj dati element ne može da se nađe (jer je zbog sortiranja niza veći, ili manji od centralnog elementa).
- Pretražimo drugu polovinu niza.
- Pretragu nastavljamo na isti način, upoređivanjem sa centralnim elementom preostalih članova niza, sve dok ne pronađemo traženi element.

Primer 16. Napisati funkciju koja vraća najmanji element niza, kao i funkciju koja vraća najmanju poziciju najvećeg elementa u nizu.

Pseudo kod | C

View Raw Code ?

```c
// funkcija koja vraća najmanji element niza
int minimum(int a[], int n)
{
    int i, min;
    // pretpostavimo da je prvi član najmanji i označimo ga sa min
    min=a[0];
    for(i = 1; i < n; i++)
        /* pomoću for petlje ispitujemo da li je neki od ostalih
        članova niza manji */
        if(a[i] < min)
            /* ukoliko je neki sledeći član manji od min, min dobija
            njegovu vrednost */
            min=a[i];
    return min;
}
// funkcija koja vraća najmanju poziciju najvećeg elementa
int pozicijaNajveceg(int a[], int n)
{
    int i, max, pozMax;
    // pretpostavimo da je prvi član i najveći i označimo ga sa max
    max=a[0];
    // pozMax označava poziciju najvećeg člana
    pozMax=0;
    for(i = 1; i < n; i++)
        /* pomoću for petlje ispitujemo da li je neki od ostalih
        članova niza veći */
        if(a[i] > max)
        {
            /* ukoliko je neki naredni član veći od max, max dobija
            njegovu vrednost */
            max=a[i];
            // pozMax dobija vrednost pozicije većeg člana
            pozMax=i;
        }
    return pozMax;
}
```

Slika 32. Rešenje Primera 16.

Primer 17. Napisati funkciju koja proverava da li je dati niz brojeva uređen neopadajuće, kao i funkciju koja štampa niz.

Pseudo kod | C

View Raw Code ?

```c
// funkcija proverava da li je niz uređen neopadajuće
int neopadajuce(int a[], int n)
{
    int i;
    for(i = 0; i < n-1; i++)
        /* proveravamo da li je svaki član niza
        manji od prethodnog */
        if(a[i+1] < a[i])
            return 0;
    return 1;
}
// funkcija štampa sve članove niza, redom
void stampajNiz(int a[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

Slika 33. Rešenje Primera 17.

Pseudo kod | C

View Raw Code ?

```
a[] je zadati niz, n je dužina niza;
funkcija neopadajuce() proverava da li je svaki član
niza manji od prethodnog; preko indeksa niza pristupiti
svakom elementu i proveriti da li je svaki element
manji od prethodnog; ukoliko je uslov ispunjen,
niz je uređen opadajuće, funkcija vraća 0;
ukoliko uslov nije ispunjen, niz je uređen neopadajuće, vraća 1;
funkcija stampajNiz(), ispisuje redom sve članove niza.

```

Slika 34. Pseudo kod za Primer 17.

Primer 18. Napisati funkciju koja ispituje da li je dati niz palindrom, a zatim i funkciju koja obrće dati niz brojeva.

Pseudo kod | C

View Raw Code ?

```c
// funkcija ispituje da li je dati niz palindrom
int palindrom(int a[], int n)
{
    int i, j;
    /* imamo dva brojača jer upoređujemo prvi i poslednji član niza,
    zatim drugi i pretposlednji itd. */
    for(i = 0, j = n-1; i < j; i++, j--)
        /* ukoliko je prvi član različit od poslednjeg,
        niz nije palindrom */
        if(a[i] != a[j])
            return 0;
    return 1;
}
// funkcija obrce dati niz brojeva
void obrni(int a[], int n)
{
    int i, j, temp;
    /* imamo dva brojača jer menjamo mesta prvom i poslednjem
    članu niza, zatim drugom i pretposlednjem itd. */
    for(i = 0, j = n-1; i < j; i++, j--)
    {
        /* temp je privremena promenljiva u koju smeštamo
        vrednost člana niza dok ne izvršimo zamenu mesta */
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

Slika 35. Rešenje Primera 18.

Palindrom je niz znakova koji se čita isto i sa desna u levo i sa leva u desno. Zbog toga prilikom proveravanja da li je niz palindrom moramo posmatrati i upoređivati članove niza s početka i s kraja, odnosno, najpre upoređujemo prvi i poslednji član, zatim drugi i pretposlednji i tako redom dok ne dođemo do središnjeg člana niza. Ukoliko je niz palindrom, svi upoređeni članovi će biti međusobno jednaki. Međutim, ako naiđemo na slučaj da neka dva upoređena člana nisu jednaka, niz nije palindrom.

Primer 19. Napisati funkciju *ucesljaj* kojom se od neopadajućih nizova a i b učešljavanjem gradi neopadajući niz c. Niz c se sastoji od svih elemenata nizova a i b.

Pseudo kod | C

View Raw Code ?

```c
/* funkcija od neopadajućih nizova a i b
učešljavanjem gradi neopadajući niz c */
void ucesljaj(int a[], int na, int b[], int nb, int c[])
{
    int i=0, j=0, k=0;
    while(i < na && j < nb)
        if(a[i] <= b[j])
            c[k++] = a[i++];
        else
            c[k++] = b[j++];
    while(i < na)
        c[k++] = a[i++];
    while(j < nb)
        c[k++] = b[j++];
    return k;
}
```

Slika 36. Rešenje Primera 19.

9. Pokazivači

U jeziku C, pokazivači imaju veoma značajnu ulogu i praktično je nemoguće napisati iole kompleksniji program bez upotrebe pokazivača.

Memorija računara organizovana je u niz uzastopnih bajtova. Uzastopni bajtovi mogu se tretirati kao jedinstven podatak. Na primer 4 ili 8 bajta (zavisno od sistema) mogu se tretirati kao jedinstven podatak tipa *int*.

Pokazivači predstavljaju tip podataka u C-u, pri čemu su vrednosti ovog tipa memorijske adrese. U zavisnosti od sistema, adrese obično zauzimaju 4 ili 8 bajta. Pokazivačke promenljive su promenljive koje sadrže memorijsku adresu objekta, jer je jedino svojom adresom svaki objekat jednoznačno određen. Kada imamo pokazivač na neki objekat, objektu možemo pristupiti preko pokazivača ili direktno. Jedan od glavnih razloga upotrebe pokazivača jeste prenos velikih struktura podataka funkcijama, a između ostalog i prenos nizova. Iako su vrednosti pokazivačkih promenljivih celi brojevi, pokazivački tipovi se striktno razlikuju od celobrojnih. Takođe, programski jezik C razlikuje više pokazivačkih tipova i tip pokazivača se određuje na osnovu tipa podataka na koji pokazuje. Ovo znači da pokazivači implicitno čuvaju

informacije o tipu onoga na šta pokazuju, sa izuzetkom pokazivača tipa *void* koji nema informaciju o tipu podatka na koji ukazuje.

Tip pokazivača koji pokazuje na promenljivu tipa *int* zapisuje se kao `int*`. Slično važi i za druge tipove. Prilikom deklaracije nije bitno da li razmak stoji između tipa i zvezdice ili identifikatora i zvezdice. Sledeći zapisi imaju isto značenje:

```
int* p1;
int *p1;
```

Kako bi pokazivačka promenljiva sadržala adresu nekog smislenog podatka, potrebno je da postoji mogućnost određivanja adrese objekata. To nam omogućava unarni operator & („operator referenciranja"), koji vraća adresu svog operanda. On može biti primenjen samo na promenljive i elemente, a ne i na izraze i konstante. Postoji i „operator dereferenciranja", koji izvršava suprotnu operaciju od operatora referenciranja. Tu ulogu vrši unarni operator *. Dakle, operator * se primenjuje na pokazivačku promenljivu i vraća sadržaj lokacije na koju ta promenljiva pokazuje, vodeći računa o tipu.

U nekim slučajevima, poželjno je imati mogućnost „opšteg" pokazivača, tj. pokazivača koji može da ukazuje na promenljive različitih tipova. Za to se koristi tip `void*`. Izraze ovog tipa je moguće eksplicitno konvertovati u bilo koji konkretni pokazivački tip (čak se u C-u, za razliku od C++-a, vrši i implicitna konverzija prilikom dodele). Medutim, naravno, nije moguće vršiti dereferenciranje pokazivača tipa `void*` jer nije moguće odrediti tip takvog izraza kao ni broj bajtova u memoriji koji predstavljaju njegovu vrednost.

Postoji čvrsta veza između pokazivača i nizova. Naime, niz se ne može preneti kao argument funkcije, ali umesto toga kao argument funkcije se može navesti ime niza i time se prenosi samo pokazivač koji ukazuje na početak niza. Funkcija koja prihvata takav argument za njegov tip ima pokazivač zapisan u jednom od sledećih oblika:

```
char* a;
```

ili

```
char a[];
```

Na taj način se kao argument (po vrednosti) prenosi samo pokazivač na početak niza, ali ne i informacija o dužini niza. Stoga, prilikom prenosa niza u funkciju, uz njegovo ime, najčešće je neophodno proslediti i broj elemenata niza, jer je ovaj broj nemoguće odrediti u okviru funkcije samo na osnovu prenete adrese početka.

Pojasnimo na primeru. Deklaracijom `int a[6]` deklarisali smo niz `a` koji ima 6 elemenata. Prvi element (početni) je `a[0]`, a poslednji `a[6]` i oni su poređani uzastopno u memoriji. Imenu niza `a` pridružena je informacija o adresi početnog elementa niza, o tipu elemenata, kao i o dužini niza. Nakon prethodne deklaracije niza `a`, vrednosti `a` odgovara pokazivač na prvi element niza, vrednosti `a+1` odgovara pokazivač na drugi element niza itd. Dakle, umesto `a[i]` možemo pisati `*(a + i)`, a umesto `&a[i]` možemo pisati `a+i`.

Primer 20. Primer ilustruje promenu vrednosti promenljive u funkciji uz pomoć pokazivača.

Pseudo kod | C

View Raw Code ?

```c
#include<stdio.h>
// funkcija uvecava vrednost promenljive za 1
void uvecaj(int* p)
{
   *p = *p + 1;
}
int main()
{
   int a = 10;
   printf("Pre poziva funkcije vrednost promenljive je %d\n", a);
   uvecaj(a);
   printf("Nakon poziva funkcije vrednost promenljive je %d\n", a);
}

```

Slika 37. Rešenje Primera 19 pomoću pokazivača.

9.1. Pokazivačka aritmetika

Ranije navedeni izraz $a + 1$ uključuje izračunavanje koje koristi pokazivačku aritmetiku i koje se razlikuje od običnog izračunavanja. Izraz $a + 1$ ne označava dodavanje vrednosti 1 na a, već označava dodavanje dužine jednog objekta tipa na koji pokazuje a.

Na primer, ako je a pokazivač na *int* koji sadrži adresu 1001 na računaru na kome *int* zauzima četiri bajta, vrednost $a + 3$ će biti adresa $1001 + 3 * 4 = 1013$.

Takođe, od pokazivača je moguće oduzimati cele brojeve (analogno kao kod sabiranja), a moguće je primenjivati i prefiksne i postfiksne operatore ++ i --. Pokazivači se mogu porediti relacijskim operacijama ukoliko pokazuju na elemente istog niza. Dva pokazivača je moguće oduzimati. Pri tom se ne vrši prosto oduzimanje dve adrese, već se razmatra veličina tipa pokazivača sa kojom se razlika deli.

Međutim, dva pokazivača nije moguće sabirati. Nema smisla sabirati pokazivače jer onda, u stvari, sabiramo dve adrese.

Prilikom izvođenja operacija nad pokazivačima, unarni operatori * i & imaju veći prioritet od binarnih aritmetičkih operatora.

10. Funkcije za rad sa niskama

Posebno mesto u programskom jeziku C zauzimaju nizovi koji sadrže karaktere, tj. niske karaktera (stringovi[2], ili jednostavno niske). Niske u programskom jeziku C se navode izmedu dvostrukih navodnika "" (na primer "ja sam niska"). U okviru niski, specijalni karakteri se navode korišćenjem specijalnih sekvenci (na primer, prvi red\ndrugi red). Tehnički, niske su predstavljene kao nizovi karaktera na čiji desni kraj se dopisuje karakter '\0' (tzv. terminalna nula). Iz ovog razloga, niske u C-u se nazivaju *niske terminisane nulom* (eng. null terminated strings). Karakter '\0' ima ASCII vrednost 0 pa se može tumačiti kao logička vrednost „netačno”. Posledica ovoga je da ne postoji ograničenje za dužinu niske u C-u, ali da je neophodno proći kroz celu nisku kako bi se odredila njena dužina. Niske mogu da se koriste i prilikom inicijalizacije nizova karaktera.

- `char s1[] = {'Z', 'd', 'r', 'a', 'v', 'o'};`
- `char s2[] = {'Z', 'd', 'r', 'a', 'v', 'o', '\0'};`
- `char s3[] = "Zdravo";`

Nakon prethodnih deklaracija, sadržaj niza s1 je:

Tabela 2. Sadržaj niza s1

0	1	2	3	4	5
‘Z’	‘d’	‘r’	‘a’	‘v’	‘o’

Sadržaj nizova s2 i s3 je:

Tabela 3. Sadržaj nizova s2 i s3

0	1	2	3	4	5	6
‘Z’	‘d’	‘r’	‘a’	‘v’	‘o’	‘\0’

Niz s1 sadrži 6 karaktera (i zauzima 6 bajtova). Deklaracije za s2 i s3 su ekvivalentne i ovi nizovi sadrže po 7 karaktera (i zauzimaju po 7 bajtova). Dakle, potrebno je jasno razlikovati karakterske konstante (npr. 'x') koje predstavljaju pojedinačne karaktere i niske (npr. "x") koje sadrže dva karaktera ('x' i '\0').

Ukoliko se u programu dve ili više niski nađu neposredno jedna uz drugu, one se automatski spajaju.

[2] Naziv *stringovi* potiče od engleske reči *strings*, što u prevodu znači *niske, nizovi*.

Sledeća dva zapisa su ekvivalentna:

- `printf("Zdravo, " "svima");`
- `printf("Zdravo, svima");`

Primer 21. Funkcija za ispis niske karaktera - demonstrira prenos niske karaktera u funkciju; funkcija za učitavanje reči sa ulaza u nisku.

Pseudo kod | C

View Raw Code ?

```c
#include <stdio.h>
#include <ctype.h>
void stampaj_nisku(char s[])
{
    int i;
    for (i = 0; s[i]; i++)
        putchar(s[i]);
}
void ucitaj_nisku(char s[])
{
    int c, i = 0;
    while (!isspace(c=getchar()))
        s[i++] = c;
    s[i] = '\0';
}
main()
{
    char s[100];
    stampaj_nisku("Informatika\n");
    ucitaj_nisku(s);
    printf("%s\n", s);
}
```

Slika 38. Rešenje Primera 20.

Pseudo kod | C

View Raw Code ?

```
Funkcija stampaj_nisku ispisuje nisku karakter po karakter;
funkcija ucitaj_nisku učitava nisku karakter po karakter
dok ne dođe do praznine.

```

Slika 39. Pseudo kod za Primer 20.

Dakle, u C-u, string je niz karaktera koji se završava terminalnom nulom. Međutim, u programskom jeziku Java, na primer, ne postoji terminalna nula. String predstavlja objekat klase *String* koji, pored niza karaktera ima i svoju dužinu. String literal je niz karaktera pod dvostrukim navodnicima. String promenljiva je promenljiva koja čuva referencu na objekat klase String. Deklariše se na isti način kao i promenljive primitivnih tipova. Takođe, može se inicijalizovati pri deklarisanju, što je generalno dobra ideja.

```
String mojString = ”Moj string”;
```

String objekti su nepromenljivi. To znači da se ne može menjati string koga taj objekat predstavlja.

Dakle, u programiranju, string, tj. niska se obično prepoznaje kao sekvenca karaktera koji mogu biti slovne konstante ili neka vrsta promenljivih. Skladištenje u memoriji vrši se tako što se svaki karakter predstavi kao neka numerička vrednost, koja zapravo predstavlja adresu tog karaktera. String se često impementira kao niz bajtova. Još jedna od razlika u odnosu na nizove je to što je string pogodan za karakter – po – karakter obrađivanje. U ovom kontekstu, string ne mora reprezentovati tekst.

Kao što je pomenuto, kod nizova u C-u je pored imena niza neophodno preneti i broj elemenata u funkciju, jer funkcija ne može samo na osnovu početne adrese izračunati broj elemenata niza. Kod funkcija koje obrađuju prosleđene niske karaktera je malo drukčija situacija jer je moguće odrediti broj elemenata niske na osnovu njenog sadržaja.

Konstantne niske tretiraju se kao nizovi karaktera koji su poređani redom na uzastopnim memorijskim lokacijama i iza kojih se nalazi terminalna nula. Deklaracija niske *"primer"* pomoću pokazivača izgleda ovako:

```
char *a = "primer";
```

Pri tom, `a` pokazuje na početak niske i važi: `a[0]=’p’`, `a[1]=’r’` i tako redom do `a[5]=’r’`, `a[6]=’/0’`. Pokušaj da se promeni sadržaj lokacije na koji ukazuje `a` nije dozvoljen, tj. dovodi do greške u fazi izvršavanja. Dakle, nije moguće napisati:

```
char a[0]=’t’;
```

Međutim, promena vrednosti samog pokazivača je dozvoljena, npr. `a++`.

Ako bismo inicijalizaciju prethodne niske karaktera izvršili na sledeći način:

```
char a[] = "primer";
```

to bi značilo da smo kreirali niz karaktera dužine 6 koji se prilikom inicijalizacije popunjava karakterima niske *"primer"*. U tom slučaju je dozvoljeno menjanje vrednosti članova niza, npr. `a[0]=’t’;` ali nije dozvoljeno menjanje vrednosti `a`.

10.1. **Standardne funkcije za rad sa niskama**

C standardna biblioteka definiše veliki broj funkcija za rad sa niskama. Prototipovi ovih funkcija se nalaze u zaglavlju *string.h*. Ilustracije radi, prikazaćemo kako se mogu implementirati neke od njih. Naravno, u svakodnevnom programiranju, ne preporučuje se definisanje ovih funkcija. Dovoljno je samo uključiti zaglavlje *string.h* na početku programa i sve funkcije koje se nalaze u tom zaglavlju biće „vidljive" unutar celog programa.

Primer 22. Funkcija *strchr* proverava da li niska sadrži dati karakter (vraća prvu poziciju karaktera *c* u stringu *s*, odnosno -1 ukoliko *s* ne sadrzi *c*); funkcija *strstr* proverava da li se jedna niska sadrži u drugoj (vraća indeks elementa niske str od kojeg počinje niska sub ili -1 ako niska str ne sadrži nisku sub).

Pseudo kod | C

View Raw Code ?

```c
/* funkcija vraća prvu poziciju karaktera c u stringu s,
odnosno -1 ukoliko s ne sadrzi c */
int strchr(char s[], char c)
{
    int i;
    for (i = 0; s[i] != '\0'; i++)
        if (s[i] == c)
            return i;
    return -1;
}
/* funkcija proverava da li niska str sadrži nisku sub,
vraća indeks elementa niske str od kojeg počinje niska
sub ili -1 ako niska str ne sadri nisku sub */
int strstr(char str[], char sub[])
{
    int i, j;
    // proveravamo da li sub počinje na svakoj poziciji i
    for (i = 0; str[i] != '\0'; i++)
        /* poredimo sub sa str počevši od pozicije i
        sve dok ne naiđemo na razliku */
        for (j = 0; str[i+j] == sub[j]; j++)
            /* Nismo naišli na razliku a ispitali smo
            sve karaktere niske sub */
            if (sub[j+1]=='\0')
                return i;
    /* Nije nadjeno */
    return -1;
}
```

Slika 40. Kod funkcija strchr i strstr.

Primer 23. Funkcija *strlen* – računa dužinu niske; funkcija *strcpy* - kopira jednu nisku u drugu.

Pseudo kod | C

View Raw Code ?

```c
// funkcija izračunava dužinu niske (verzija bez pokazivača)
int strlen(char s[])
{
    int i = 0;
    while (s[i] != '\0')
        i++;
    return i;
}
// funkcija izračunava dužinu niske (verzija sa pokazivačima)
int strlen(char *s)
{
    int n;
    // krećemo se kroz nisku dok ne naiđemo na '/0'
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}
// funkcija vrši kopiranje niske src u nisku dest
void strcpy(char dest[], char src[])
{
    int i = 0;
    /* u nisku dest se prebacuje karakter po karakter niske src
    sve dok dodeljeni karakter ne bude terminalna nula */
    while ((dest[i] = src[i]) != '\0')
        i++;
}
// funkcija vrši kopiranje niske src u nisku dest (pomoću pokazivača)
void strcpy(char *dest, char *src)
{
    while (*dest != '/0' && *src != '\0')
    {
        *dest = *src
        dest++;
        src++;
    }
}

```

Slika 41. Kod funkcija strlen i strcpy (verzija sa pokazivačima i bez istih).

Primer 24. Funkcija *strrev* obrće datu nisku; funkcija *strcmp* poredi dve niske leksikografski (kao u rečniku sortiranom po ASCII redosledu).

Pseudo kod | C

View Raw Code ?

```c
// funkcija obrće datu nisku
void strrev(char s[])
{
    int i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
    {
        int tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
    }
}
// funkcija poredi dve niske leksikografski
int strcmp(char s1[], char s2[])
{
    int i;
    // petlja teče sve dok ne naiđemo na prvi različiti karakter
    for (i = 0; s[i] == t[i]; i++)
        /* ako naiđemo na kraj obe niske, a nismo
        našli razliku, funkcija vraća 0 */
        if (s[i] == '\0') /*
            return 0;
    /* s[i] i t[i] su prvi karakteri u kojima se niske razlikuju,
    na osnovu njihovog odnosa, određuje se odnos stringova */
    return s[i]-t[i];
}
```

Slika 42. Kod funkcija strrev i strcmp.

Mnoge funkcije koje obrađuju niske, obrađuju ih karakter po karakter i sadrže petlju oblika:

```
for (i = 0; s[i] != '\0'; i++)
```

Imajući u vidu da terminalna nula ima i numeričku vrednost 0 (što predstavlja istinitosnu vrednost netačno), poređenje u prethodnoj petlji može da se izostavi:

```
for (i = 0; s[i]; i++)
```

Česta greška je pozivanje funkcije *strlen* za izračunavanje dužine niske u svakom koraku iteracije. U tom slučaju program je neefikasan.

```
for (i = 0; i < strlen(s); i++)
```

Trebalo bi izračunati dužinu niske pre ulaska u petlju.

Primer 25. Napisati funkciju koja pretvara u veliko prvo slovo svake reči. Pretpostaviti da je prvo slovo slovo na početku teksta ili ono ispred koga je blanko. Sva ostala slova pretvoriti u mala. Dopušteno je koristiti bibliotečke funkcije *toupper* i *tolower*.

Pseudo kod | C

View Raw Code ?

```c
/* funkcija pretvara u veliko prvo slovo svake reči */
void velikaSlova(char a[])
{
    int i;
    /* ukoliko niska a[] ne sadrži nijedan karakter,
    izlazimo iz funkcije */
    if(a[0]=='\0')
        return;
    /* početno slovo date reči konvertujemo u veliko slovo */
    a[0] = toupper(a[0]);
    /* za svako sledeće slovo proveravamo da li je ispred njega
    praznina; ako jeste, postaje veliko slovo,a ako nije - malo */
    for(i=1; a[i]!='\0'; i++)
        if(a[i-1]==' ')
            a[i]=toupper(a[i]);
        else
            a[i]=tolower(a[i]);
}
```

Slika 43. Rešenje Primera 24.

Pseudo kod | C

View Raw Code ?

```
Prvo proveravamo da li je uneta niska koja ima bar 1 karakter;
zatim konvertujemo prvo slovi niske u veliko;
nakon toga prolazimo kroz nisku i ispitujemo da li je ispred
tekućeg karaktera belina;
ako jeste konvertujemo slovo u veliko, jer to znači da je tekući
karakter početno slovo neke niske;
u suprotnom tekući karakter konvertujemo u malo slovo.

```

Slika 44. Pseudo kod za Primer 24.

Primer 26. Napisati funkciju koja prebrojava koliko se (ukupno) puta karakteri iz niske b pojavljuju u niski a; napisati funkciju koja skraćuje nisku na zadatu dužinu n (ukoliko je niska kraća od n, ne menjati je).

Pseudo kod | C

View Raw Code ?

```c
/* funkcija prebrojava koliko se (ukupno) puta karakteri iz
niske b pojavljuju u niski a */
int prebroj(char a[], char b[])
{
    int i, j, rez = 0;
    for(i=0; a[i]!='\0'; i++)
        for(j=0; b[j]!='\0'; j++)
            if(a[i]==b[j])
                rez++;
    return rez;
}
/* funkcija skraćuje nisku na zadatu dužinu n */
void skrati(char a[], int n)
{
    int la = strlen(a);
    if(la<=n)
        return;
    a[n]='\0';
}
```

Slika 45. Rešenje Primera 25.

10.2. **Funkcije sprintf i sscanf**

Funkcije *sprintf()* i *scanf()* definisane su u zaglavlju *stdio.h*.

```
int sscanf(const char *s, const char *format, /* args */ ...);
int sprintf(char *s, const char *format, /* args*/ ...);
```

Za razliku od funkcija *printf()* i *scanf()*, funkcije *sprintf()* i *sscanf()* kao prvi argument uzimaju pokazivač na znakovni niz. Tako, funkcija *sscanf()* čita iz stringa *s* (umesto sa tastature) prema zadatom formatu; funkcija *sprintf()* piše u string *s* (umesto na ekran), takođe prema zadatom formatu. Ove funkcije prvenstveno služe za formatiranje stringova.

11. Višedimenzionalni nizovi

Jezik C dozvoljava defnisanje klasičnih višedimenzionalnih nizova, iako se u praksi oni koriste mnogo ređe nego nizovi pokazivača. Višedimenzionalni nizovi u programskom jeziku C se mogu deklarisati kao:

View Raw Code ?

```
1. tip ime_niza[broj_elemenata][broj_elemenata]...[broj_elemenata];
```

Slika 46. Deklaracija višedimenzionalnog niza

Dvodimenzionalni nizovi se tumače kao jednodimenzioni nizovi čiji su elementi nizovi. Zbog toga se elementima pristupa sa:

```
ime_niza[vrsta][kolona];
```

Memorijski raspored elemenata dvodimenzionalnog niza, koji opisuje neku matricu, je takav da su elementi složeni po redovima matrice, najpre prvi red, zatim drugi itd.

Na primer, niz `m` deklarisan sa

```
static float m[2][3];
```

predstavlja matricu s dva reda i tri kolone. Njene elemente možemo prostorno zamisliti na sledeći način:

```
m[0][0]   m[0][1]   m[0][2]
m[1][0]   m[1][1]   m[1][2]
```

Element na mestu (i, j) matrice `m` je `m[i][j]`. Elementi višedimenzionalnog niza pamte se u memoriji računara kao jednodimenzionalni nizovi. Pri tom su elementi poređani po vrstama što znači da se pri smeštanju elemenata u memoriju najdešnji indeks najbrže menja. Kod dvodimenzionalnog niza `m` raspored elemenata u memoriji bio bi:

```
m[0][0]   m[0][1]   m[0][2]   m[1][0]   m[1][1]   m[1][2]
```

Preciznije, element `m[i][j]` biće na k-tom mestu u memoriji, gde je

```
k = i*MAXY + j,
```

a MAXY = 3 je broj kolona matrice.

Niz se može inicijalizovati navođenjem liste inicijalizatora u vitičastim zagradama; pošto su elementi opet nizovi, svaki od njih se opet navodi u okviru vitičastih zagrada. Na primer, razmotrimo dvodimenzionalni niz koji sadrži broj dana za svaki mesec, pri čemu su u prvoj vrsti vrednosti za obične, a u drugoj vrsti vrednosti za prestupne godine.

Pseudo kod | C

View Raw Code ?

```c
char broj_dana[2][13] =
{
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};
```

Slika 47. Dvodimenzionalni niz koji sadrži broj dana za svaki mesec.

Navođenje unutrašnjih vitičastih zagrada je opciono, pa se može koristiti i sledeća deklaracija:

Pseudo kod | C

View Raw Code ?

```c
char broj_dana[2][13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31, 0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Slika 48. Drugi način za definisanje dvodimenzionalnog niza koji sadrži broj dana za svaki mesec.

Međutim, ovaj drugi način se ne preporučuje jer je teže uočiti raspored elemenata po redovima i kolonama.

Primetimo da se tip *char* koristi za skladištenje malih prirodnih brojeva. Takođe, pošto zauzeće prostora nije presudno, u nultu kolonu su upisane nule kako bi se podaci za mesec `m` nalazili upravo u koloni `m` (tj. kako bi se mesecima pristupalo sa 1-12, umesto 0-11).

Višedimenzionalni nizovi se definišu rekurzivno:

- Višedimenzionalni niz je jednodimenzionalni niz čiji su elementi nizovi dimenzije manje za jedan.

Prilikom deklaracije višedimenzionalnih nizova kao parametara funkcije, pravilo je da se ne navodi prva dimenzija, kao i kod jednodimenzionalnih nizova, ali ostale dimenzije treba deklarisati kako bi program prevodilac „znao“ kojim su redom elementi složeni u memoriji.

Podsetimo se da je vrednost logičkog izraza u C-u uvek nula (netačno) ili jedan (tačno), tako da se ova vrednost može koristiti kao indeks pri pristupanju nizu. Ukoliko se dvodimenzioni niz prenosi u funkciju, deklaracija parametra u funkciji mora da uključi broj kolona; broj vrsta je nebitan jer se i u ovom slučaju prenosi samo adresa tj. pokazivač na niz vrsta, pri čemu je svaka vrsta niz. Na primer, funkcija f koja prima niz *broj_dana* može biti definisana na jedan od tri naredna načina:

```c
void f(char broj_dana[2][13]);
void f(char broj_dana[ ][13]);
```

```
void f(char (*broj_dana)[13]);
```

U poslednjem slučaju, eksplicitno je rečeno da se prenosi pokazivač na niz od 13 karaktera. Zagrade ( ) su neophodne jer zagrade [ ] imaju viši prioritet nego operator zvezdica. Bez zagrada, deklaracija

```
void f(char *broj_dana[13]);
```

označava da se prenosi niz od 13 pokazivača na karaktere.

Kada je višedimenzionalni niz argument funkcije on se može deklarisati sa svim svojim dimenzijama ili sa svim dimenzijama osim prve. Naime, broj redova nije bitan za adresiranje elemenata matrice. Sve što funkcija mora znati je da se element `m[i][j]` nalazi na k-tom mestu, gde je k = i*MAXY + j. Dakle, samo je broj kolona neophodan pri pozivu funkcije.

U slučaju dimenzija većih od 2, samo je prva dimenzija (indeks) slobodna, dok je sve naredne dimenzije neophodno navesti. Ime jednodimenzionog niza (na primer, a za int `a[10]`) može se tretirati kao pokazivač na prvi element niza. Ime dvodimenzionog niza, na primer, tipa *int* može se tretirati kao pokazivač na pokazivač na *int*. Na primer, ako je

```
int d[10][20];
```

onda je `d[0]` (kao i `d[1]`, `d[2]`, ...) pokazivač na *int*. Pokazivač `d[0]` sadrži adresu elementa `d[0][0]`, i `d[i]` sadrži adresu elementa `d[i][0]`. Vrednost `d` je tipa *int* **, ona je pokazivač na pokazivač na *int* i sadrži adresu pokazivača `d[0]`.

Analogno dvodimenzionalnim nizovima, trodimenzionalni niz je niz dvodimenzionalnih nizova.

Primer 27. Napisati program koji iz datoteke "matrica.txt" učitava prvo dimenziju kvadratne matrice (n < 10) a zatim redom elemente matrice (u okviru jedne dvostruke for petlje) pa zatim ispisuje elemente matrice na standardni izlaz (u okviru druge dvostruke for petlje). Odrediti sumu elemenata matrice i sumu elemenata koji se nalaze na glavnoj dijagonali.

Pseudo kod | C

View Raw Code ?

```
1. Prvo deklarišemo matricu a[10][10], dimenziju matrice n i
2. datoteku iz koje ucitavamo matricu, in;
3. proveravamo da li je datoteka prazna i ako jeste izlazimo iz programa;
4. učitavamo dimenziju kvadratne matrice, kao i njene elemente.
5. Zatim,ispisujemo elemente kvadratne matrice pri čemu treba da
6. vodimo računa o tome da treba da ispišemo i karakter za novi red
7. nakon svakog ispisanog reda matrice.
8.
```

Slika 49. Pseudo kod za Primer 26.

View Raw Code ?

```c
#include<stdio.h>
main()
{
    int a[10][10];
    int n;
    FILE* in;
    int i, j;
    in=fopen("matrica.txt", "r");
    if (in==NULL)
    {
        fprintf(stderr, "\nGreska pri otkrivanju datoteke.\n");
        exit(1);
    }
    fscanf(in, "%d", &n);
    for(i=0; i < n; i++)
        for(j=0; j < n; j++)
            fscanf(in, "%d", &a[i][j]);
    for(i=0; i < n; i++)
    {
        for(j=0; j < n; j++)
            printf("%d\t", a[i][j]);
        printf("\n");
    }
    /* dvostruka for petlja koja izračunava sumu elemenata matrice */
    suma = 0;
    for(i=0; i < n; i++)
        for(j=0; j < n; j++)
            suma = suma + a[i][j];
    printf("Suma elemenata matrice je: %d\n", suma);
    /* dvostruka petlja koja izračunava sumu elemenata koji
    se nalaze na glavnoj dijagonali (to su elementi kojima
    su indeksi "i" i "j" jednaki */
    suma_dijag = 0;
    for(i=0; i < n; i++)
        for(j=0; j < n; j++)
            if (i==j)
                suma_dijag = suma_dijag + a[i][j];
    printf("Suma je: %d\n", suma_dijag);

}
```

Slika 50. Rešenje Primera 26.

Primer 28. Napisati program koji iz datoteke "matrica.txt" učitava prvo dimenziju kvadratne matrice (n < 10) a zatim redom elemente matrice. Za uneto k sa standardnog ulaza ispisati sumu elemenata u k-tom redu i sumu elemenata u k-toj koloni.

Pseudo kod | C

View Raw Code ?

```c
#include
main()
{
    /* deklaracija matrice */
    int a[10][10];
    /* dimenzija kvadratne matrice */
    int n;
    /* datoteka iz koje učitavamo matricu */
    FILE* in;
    int i, j;
    /* otvaramo datoteku za čitanje */
    in=fopen("matrica.txt", "r");
    if (in==NULL)
    {
        fprintf(stderr, "\nGreška pri otkrivanju datoteke: matrica.txt\n");
        exit(1);
    }
    /* učitavamo dimenziju kvadratne matrice */
    fscanf(in, "%d", &n);
    /* učitavamo elemente kvadratne matrice */
    for(i=0; i < n; i++)
        for(j=0; j < n; j++)
            fscanf(in, "%d", &a[i][j]);
    printf("Unesite ceo broj:\n");
    scanf("%d", &k);
    /* računanje sume elemenata u k-tom redu */
    suma = 0;
    for(j=0; j < n; j++)
        suma = suma + a[k][j];
    printf("Suma elemenata koji se nalaze u %d-tom redu je: %d\n", k, suma);
    /* računanje sume elemenata u k-toj koloni */
    suma = 0;
    for(i=0; i < n; i++)
        suma = suma + a[i][k];
}
```

Slika 51. Rešenje Primera 27.

Primer 29. Napisati program za skalarno množenje matrica.

Pseudo kod | C

View Raw Code ?

```c
#include
void main()
{
    int vrsta,kol;
    float scalar;
    static float matrica[3][5]={
        {7.0,16.83,55.131,13.1,12.91},
        {15.0,10.9,42.99,0.0,7.7},
        {-2.2,1.1,2.2,4.4,9.9}
        };
    printf("Originalna matrica:\n");
    for(vrsta=0; vrsta < 3; ++vrsta)
    {
        for(kol = 0; kol < 5; ++kol)
        printf("%10f",matrica[vrsta][kol]);
        printf("\n");
    }
    printf("\n Scalar?\n");
    scanf("%f",&scalar);
    for(vrsta = 0; vrsta < 3; ++vrsta)
        for(kol = 0; kol < 5; ++kol)
            matrica[vrsta][kol]*=scalar;
    printf("\n Matrica posle skalarnog množenja:\n");
    for(vrsta = 0; vrsta < 3; ++vrsta)
    {
    for(kol = 0; kol < 5; ++kol)
    printf("%10f\t", matrica[vrsta][kol]);
    printf("\n");
    }
}
```

Slika 52. Rešenje Primera 28.

Važno je uočiti razliku između dvodimenzionalnih nizova i nizova pokazivača. Pogledajmo sledeće deklaracije:

```
int a[10][20];
int *b[10];
```

Niz a je pravi dvodimenzionalni niz. Za niz a rezervisano je 200 uzastopnih memorijskih lokacija veličine *sizeof(int)* i koristi se uobičajena računica $20 * v + k$ da bi se pronašao element $a[v][k]$. Pri tom, v je broj vrsta (redova), a k je broj kolona. Definicijom niza b alocirali smo 10 uzastopnih memorijskih lokacija za 10 pokazivača. Inicijalizacija ostatka niza b nije izvršena i

mora se izvršiti eksplicitno. Glavna prednost niza pokazivača nad dvodimenzionalnim nizom je činjenica da vrste na koje pokazuju ovi pokazivači mogu biti različite dužine. Tako, ne mora svaki element niza *b* da pokazuje na 20-to elementni niz, već različiti elementi mogu da pokazuju na nizove različitih dužina.

Razmotrimo primer niza koji sadrži imena meseci. Ukoliko bi sva imena bila iste dužine, rešenje bi mogao biti dvodimenzioni niz, međutim, pošto svi meseci imaju imena različite dužine, bolje rešenje je napraviti niz pokazivača na karaktere i inicijalizovati ga da pokazuje na konstantne niske smeštene u segmentu podataka. Taj niz bi trebalo da izgleda ovako:

```
1. char *meseci[] = {januar, februar, mart, april, maj, jun, jul,
2.                   avgust, septembar, oktobar, novembar, decembar};
```

Slika 53. Primer niza koji sadrži imena meseci.

Nizovi predstavljaju skup homogenih podataka i treba ih razlikovati od struktura koje predstavljaju skup heterogenih podataka. Elementi niza su istog tipa, dok su elementi struktura različitih tipova.

Primer 30. Napisati funkciju za množenje dve zadate matrice, kao i funkciju za transponovanje matrice.

Pseudo kod | C

View Raw Code ?

```
1.  // funkcija za množenje dve matrice
2.  void mnozenje(float m1[][10], float m2[][10],float m3[][10],int n)
3.  {
4.      int i,j,k;
5.      for(i = 0; i < n; ++i)
6.          for(j = 0; j < n; ++j)
7.              for(m3[i][j] = 0, k = 0; k < n; ++k)
8.                  m3[i][j] += m1[i][k]*m2[k][j];
9.  }
10. // funkcija za transponovanje matrice
11. void transponovana(int a[][100],int n)
12. {
13.     int i,j,p;
14.     for(i = 0; i < n; i++)
15.         for(j = 0; j < n; j++)
16.         {
17.             p = a[i][j];
18.             a[i][j] = a[j][i];
19.             a[j][i] = p;
20.         }
21. }
```

Slika 54. Rešenje Primera 29.

Primer 31. Napisati program za sabiranje matrica a i b dimenzije n i štampanje rezultujuće matrice c.

Pseudo kod | C

View Raw Code ?

```c
#include
void citaj(int a[][10],int n)
{
    int i,j;
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            scanf("%d",&a[i][j]);
}
void zbir(int a[][10],int b[][10],int c[][10],int n)
{
    int i,j;
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            c[i][j]=a[i][j]+b[i][j];
            printf("%d\t",c[i][j]);
        }
    printf("\n");
    }
}
void main()
{
    int a[10][10],b[10][10],c[10][10];
    int n;
    printf("Uneti broj vrsta i kolona matrice a i b\n");
    scanf("%d",&n);
    printf("Uneti elemente matrice a\n");
    citaj(a,n);
    printf("Uneti elemente matrice b\n");
    citaj(b,n);
    printf("Zbir matrica a i b je:\n");
    zbir(a,b,c,n);
}
```

Slika 55. Rešenje Primera 30.

12. Zaključak

Elektronski kursevi o programskom jeziku C mogu koristiti akademskoj zajednici iz više razloga:

- Mali je broj sajtova na srpskom jeziku koji na lep i pregledan način uvode studente u programiranje u programskom jeziku C.
- Funkcije predstavljaju veoma značajan koncept, kako u programskom jeziku C, tako i u ostalim programskim jezicima. Student koji nauči funkcije u C-u, lako može to znanje da primeni i u ostalim proceduralnim i objektno orijentisanim programskim jezicima.

Osnovni cilj Elektronskog kursa o funkcijama u programiranju jeste da „približi“ programiranje, kao i koncept funkcija u programiranju studentima i učenicima koji se prvi put susreću sa rešavanjem problema pomoću programa. Kurs predstavlja uvod u programski jezik C i postepeno uvodi u programiranje, počevši od najjednostavnijih primera, a sadrži i malo složenije zadatke kako bi se koncept funkcija što bolje savladao.

Još jedna od prednosti kursa je i to što se može lako unaprediti. Jedan od mogućih pravaca za unapređenje kursa jesu da se postojeće funkcije opišu i u drugim programskim jezicima. Time se omogućava paralelno učenje nekoliko programskih jezika i upoređivanje sličnosti i razlika među njima. U tom slučaju sajt bi bio posećeniji jer bi bio zanimljiv i koristan većem broju posetilaca.

Takođe, kurs se može unaprediti dodavanjem funkcionalnosti koja omogućava korisnicima da direktno na sajtu unose kod i isprobaju kako određeni program radi. To bi bila olakšavajuća okolnost za posetioce sajta, jer bi skratila vreme potrebno za učenje.

13. Literatura

1. B. Kernighan, D. Ritchie, „The C Programming Language“, *Prentice*, 1988.
2. P. Janičić, F. Marić, „Programiranje 1 – beleške sa predavanja“, *Matematički fakultet, Beograd*, 2011.
3. M. Jurak „Programski jezik C - predavanja“, 2003/04.
4. M. Milanović [Na mreži] http://poincare.matf.bg.ac.rs/~marija//p1.html
5. M. Spasić [Na mreži] http://poincare.matf.bg.ac.rs/~mirko/
6. M. Banković [Na mreži] http://poincare.matf.bg.ac.rs/~milan/scripts/color.pl?dat=download/p2/lectures/c8/1.c
7. Funkcija (programiranje) *Wikipedia* [Na mreži] [Citirano: 27.07.2012.] http://sh.wikipedia.org/wiki/Funkcija_(programiranje)