Darko Drakulić

Osnove programskog jezika C sa zbirkom zadataka
-skripta-

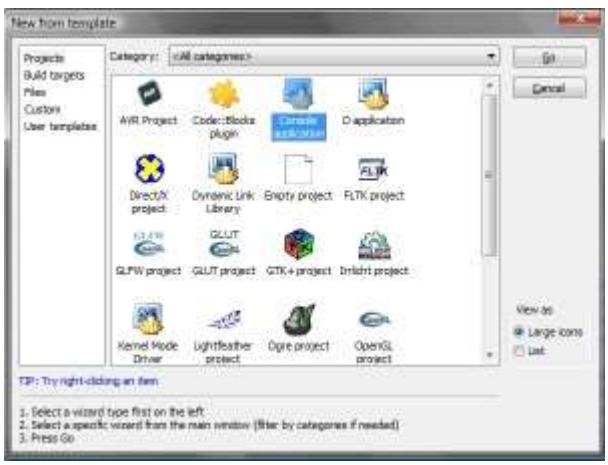
Rad u Code::Blocks okruženju

Da bi se napisao i izvršio program napisan na programskom jeziku C, potreban je tekst editor u kojem će program biti napisan, kompajler i bilder koji će napisani kod kompajlirati i prevesti u mašinski kod, razumljiv računaru. Postoji dosta aplikacija koji u sebi integrišu ove alate i one nose zajedničko ime IDE (integrated development environment). One u sebi sadrže pomenute alate – kod editor, kompajler ili interpreter, bilder i alat za testiranje programa (debugger). Jedan od takvih alata je Code::Blocks IDE, open source alat koji je besplatan i koji se može skinuti sa adrese http://www.codeblocks.org. Ovdje će ukratko biti opisano kako napisati, kompajlirati i pokrenuti program u pomenutom alatu. Da bi se program mogao kompajlirati i pokrenuti, sa pomenute adrese je potrebno skinuti instalaciju codeblocks-xxx-mingw-setup koja u sebi sadrži GCC kompajler i GDB dibager. xxx predstavlja trenutnu verziju alata.

Nakon uspešne istalacije i pokretanja programa Code::Blocks dobija se sledeći izgled ekrana

Slika 1

Da bi napisali i pokrenuli program u programskom jeziku C, potrebno je kreirati novi projekat (*File->_ew->Project*) zatim izabrati *Console application* (slika 2). U polju za izbor jezika izabrati *C* (slika 3), i konačno dati ime projektu kao i lokaciju na koju će se kod sačuvati (slika 4).



Slika 2.

Slika 3.

Slika 4.

Nakon ovih koraka, klikom na dugme *Finish*, u levom delu ekrana u delu *Projects* pojaviće se projekat koji je kreiran, kao i datoteka (*main.c*) u kojoj se nalazi glavna (main) funkcija kreiranog projekta.

Otvaranjem te datoteke, dobija se kod najprostijeg C-programa, koji na ekran ispisuje tekst „Hello world“.

Osnove programskog jezika C

Struktura programa

Program koji Code::Blocks automatski kreira ima sledeći kod

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

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Na ovom primeru će biti objašnjena struktura programa napisanog u programskom jeziku C.

C je tzv. case-sensitive programski jezik, što znači da **razlikuje** mala i velika slova. Naredbe `return` i `RETURN` su dve različite naredbe.

Nakon svake naredbe u C-u mora se nalaziti karakter `;`.

C ne poznaje razmake u kodu, tako da se prelazak u novi red nakon svake komande vrši samo iz estetskih razloga, radi lakšeg čitanja koda.

Komentari se mogu pisati na dva načina: ukoliko se komentar nalazi u jednom redu, dovoljno je na početak tog reda staviti karaktere `//`, a ukoliko se komentar nalazi u više redova, potrebno je komentar ograničiti parovima karaktera `/*` i `*/`. Primeri komentara su

```
// Komentar u jednom redu
/* Komentar
   u više redova */
```

Prva dva reda reda programa

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

u postojeći kod uključuju datoteke za prevođenje, u kojma se nalaze funkcije koje su nam potrebne za uspešno izvršavanje našeg programa. U datoteci `stdio.h` se nalaze funkcije za ulaz/izlaz, a u datoteci `stdlib.h` se nalazi većina funkcija koje su u najčešćoj upotrebi. O ovim funkcijama će kasnije biti više riječi.

Funkcija

```
int main()
```

je glavna funkcija programa i izvršavanje programa počinje od prve linije koda ove funkcije. Svaki projekat mora u nekom od fajlova imati ovu funkciju. Ona može imati drugačiju definiciju (može biti nekog drugog tipa, npr. `void` ili može imati listu argumenata).

Par vitičastih zagrada

```
{}
```

predstavlja granice bloka naredbi, kao `begin` i `end` u programskom jeziku PASCAL.

Poslednji red programa

```
return 0;
```

predstavlja izlaz iz glavnog programa, odnosno kraj glavnog programa. U ovom slučaju, glavni program vraća vrednost 0.

Red

```
printf("Hello world!\n");
```

štampa tekst „Hello world“ na ekran. Funkcija `printf` je definisana u datoteci `stdio.h` i ona na standardni izlaz (ekran) ispisuje niz karaktera koji je njen argument. Nizovi karaktera su ograničeni dvostrukim navodnim znacima (`""`). Karakter `\n` koji se nalazi na kraju ovog niza je specijalni karakter koji označava novi red. Osim karaktera za novi red, postoje sledeći specijalni karakteri:
`\t` – tabulator,
`\b` – backspace,
`\"` – dvostruki navodnici,
`\\` - obrnuta kosa crta (backslash).

Promenljive i konstante

Definisanje promenljivih

Promenljive su objekti koji imaju svoje ime i čija se vrednost može menjati tokom izvršenja programa.

Definisanje promenljivih u C-u vrši se na sledeći način:

```
tip promenljive ime promenljive;
```

Tip promenljive mora biti neki od ugrađenih tipova u C jezik ili neki od korisnički definisanih tipova.

Četiri osnovna tipa podataka su

`int` - cjelobrojni tip,

`float` - realni tip jednostruke tačnosti,

`double` - realni tip dvostruke tačnosti,

`char` - znakovni tip, karakter.

z osnovnog celobrojnog tipa se mogu dobiti izvedeni tipovi:

`short int` (`short`),

`long int` (`long`),

`unsigned int` (`unsigned`) – pozitivan celi broj.

Ime promenljive je proizvoljan niz karaktera koji ispunjava sledeće zahteve:

- ime se sastoji od karaktera, cifara i donje crte _,
- ime ne sme počinjati cifrom,
- ime ne može biti neka od rezervisanih reči jezika C (tabela 1.)

Pravila koja važe za imena promenljivih, važe i za imena konstanti, funkcija i korisničkih tipova.

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

Tabela 1. Lista rezervisanih riječi programskog jezika C

Prema opisanom, sledeće definicije promenljivih su ispravne

```
int x;
char karakter;
float realni_broj1;
double realni_broj2;
```

Ukoliko je potrebno definisati više promenljivih istog tipa, to se može učiniti u jednoj naredbi. Naredba za definisanje celobrojnih promenljivih x, y i z je

```
int x, y, z;
```

Definisanje alternativnih imena tipovima podataka

Da bi se definisalo alternativno ime postojećem tipu podatka koristi se ključna riječ typedef. Sintaksa je sledeća

```
typdef ime tipa novo ime tipa;
```

Tako, npr. možemo definisati tip "celi_broj" i dve promenljive tog "novog" tipa na sledeći način

```
typedef int cijeli_broj;
cijeli_broj a;
cijeli broj b;
```

Dodela vrednosti promenljivima

Da bi se dodelila neka vrednost nekoj promenljivoj, koristi se operator =. Naredba za dodelu ima oblik

```
ime promenljive = vrednost;
```

Ukoliko se radi o celim brojevima (tip `int`), vrednosti se mogu zadati u tri različita brojna sistema. Ukoliko vrednost počinje cifrom koja nije nula, broj se tretira kao dekadni. Ukoliko počinje nulom, broj se tretira kao oktalni a ukoliko počinje sa 0x ili 0X broj se tretira kao heksadekadni.

```
int x, y, z;
x = 123; // dekadni broj
y = 0123; // oktalni broj
z = 0x123; // heksadekadni broj
```

Ukoliko se radi o realnim brojevima, vrednosti se mogu pisati samo u dekadnom sistemu, a za decimalni zarez se koristi tačka (`.`). Vrednosti se mogu i pisati u obliku `oEk` (`oek`), gdje je `o` osnova a `k` eksponent.

```
float x, y, z;
x = 123;
y = 123.456;
z = 123E+2; // broj 12300
```

Ukoliko je promenljiva tipa `char`, vrednost mora biti jedan karakter, ograničen jednostrukim navodnim znacima (`'`). Specijalni karakteri (`\n`, `\t`..) se tretiraju kao jedan karakter, iako se, praktično sastoje od dva karaktera. Tip `char` je ustvari celobrojna vrednost ASCI koda karaktera.

```
char a, b, c, d;
a = 'a';
b = '\n';
c = '1';
d = '.';
```

Moguće je u istoj naredbi definisati promenljivu i dodeliti joj vrednost

```
int x = 5;
float y = 1.2, z = 234e-1;
char c1, c2 = 'a';
```

Štampanje vrednosti promenljivih

Na početku je bilo reči o funkciji `printf` koja štampa proizvoljan niz karaktera na standardni izlaz. Ista funkcija se koristi za štampanje vrednosti uz određene izmene. Funkcija ima sledeći oblik

```
printf("niz_znakova_za_ispis", ime_promenljive1, ...);
```

Niz znakova za ispis predstavlja niz karaktera, pri čemu se znakom `%` pokazuje mesto gde treba da se upišu vrednosti promenljivih koje se nalaze iza niza znakova za ispis. Nakon znaka `%` ide odgovarajuće slovo, koje zavisi od tipa promenljive:

`%d` – celobrojni tip, zapisan u dekadnom sistemu,

`%o` – celobrojni tip, zapisan u oktalnom sistemu,

`%x` – celobrojni tip, zapisan u heksadekadnom sistemu,

`%f` – realni tip (`float` i`double`)

`%c` – karakter.

Primeri naredbi za štampanje

```
printf("%d", x); // štampa promenljive x u dekadnom sistemu
printf("%f %f", a, b); // štampa realnih promenljivih a i b
printf("%d %o %x", x, x, x); /* štampa promenljive x u
                dekadnom, oktalnom i heksadekadnom sistemu */
printf("%c", c); // štampa vrednosti promenljive c tipa char

printf("Vrednost promenljive x je %d", x);
```

Opseg važenja promenljivih

U jednom bloku naredbi nije dozvoljeno definisanje više promenljivih istog imena. Tako, na primer, kompajliranje sledećeg koda bi javilo grešku

```
{
     int x;
     char x;
}
```

dok je sledeći kod ispravan

```
{
     int i = 1;
     {
          int i = 2;
          printf("Unutrasnje i = %d\n", i);
     }
     printf("Spoljasnje i = %d\n", i);
}
```

i njegovo izvršavanje daje rezultat

```
Unutrasnje i = 2
Spoljasnje i = 1
```

Veličina memorijskog prostora za promenljive

Nakon svake definicije promenljive, alocira se potrebna memorija za njeno čuvanje. Veličina te memorije zavisi od tipa promenljive, ali i sistema na kojem se program izvršava. U C-u je ugrađen operator `sizeof` koji određuje veličinu promenljive ili tipa u bajtovima. Ukoliko se želi odštampati veličina (u bajtovima) tipa `int`, kod bi izgledao

```
int velicina = sizeof(int);
printf("%d", velicina);
```

Vrednost se može odštampati i bez uvođenja nove promenljive, izračunavanjem veličine direktno u argumentu funkcije `printf`.

```
printf("%d", sizeof(int));
```

Celobrojne promenljive se zapisuju u formatu potpunog komplementa, realne u zapisu IEEE754 a karakteri u ASCI kodu. Za više informacija o ovim zapisima pogledati [?].

Primer 1. Napisati program koji definiše dvije celobrojne promenljive, dodeljuje im vrednost i štampa njihov zbir.

Rešenje 1

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

int main()
{
     int a, b;
     a = 5;
     b = 7;
     int c = a+b;
     printf("Zbir je %d\n", c);
     return 0;
}
```

Rešenje 2

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

int main()
{
     int a = 5, b = 7;
     printf("Zbir je %d\n", a+b);
     return 0;
}
```

Primer 2. Napisati program koji štampa veličine memorijskog prostora za osnovne tipove podataka.

Rešenje

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

int main()
{
     printf("int: %d\n", sizeof(int));
     printf("short: %d\n", sizeof(short));
     printf("long: %d\n", sizeof(long));
     printf("unsigned: %d\n", sizeof(unsigned));
     printf("float: %d\n", sizeof(float));
     printf("double: %d\n", sizeof(double));
     printf("char: %d\n", sizeof(char));
     return 0;
}
```

zlaz ovog programa nam daje odgovor na veličinu osnovnih tipova u C-u

```
int: 4
short: 2
long: 4
unsigned: 4
float: 4
double: 8
char: 1
```

Konstante

Slično promenljivima, konstante su objekti koji imaju svoje ime ali im se vrednost ne može menjati tokom izvršenja programa. Za definisanje konstanti koristi se ključna reč `const` ispred tipa i imena konstante.

```
const int x = 5;
```

Drugi način definisanja konstante je korišćenje preprocesorske direktive `#define`. Ona ima oblik

```
#define ime konstante vrednost konstante
```

OvaIpreprocesorska direktiva se piše na početku programa, ispod direktiva `#include` a ove konstante se nazivaju simboličke konstante.

Učitavanje vrednosti promenljivih sa tastature

Jasno je da loše rešenje da se vrednosti promenljivim dodeljuju u samom kodu. Takav način dodele zahteva promenu koda svaki put kada se menja vrednost promenljivih. Osnovna ideja je da se korisniku omogući dodela vrednosti promenljivima u toku izvršavanja programa. Za tu svrhu se koristi funkcija `scanf` oblika

```
scanf("format", lista promenljivih,...);
```

`format` i `lista_promenljivih` predstavljaju argumente slične kao kod funkcije `printf`. Prvi je niz karaktera koji u sebi sadrži oznaku tipa koji se unosi - `%` i odgovarajuće slovo koje je identično kao kod Ifunkcije `printf`. Što se tiče liste promenljivih, razlika u odnosu na funkcije `printf` je ta da se ispred svake promenljive mora napisati karakter `&`. Zašto baš on, biće objašnjeno u delu kada se bude govorilo o pokazivačima i referencama.

Unos vrednosti celobrojne promenljive `x` i karaktera `c` se vrši na sledeći način

```
scanf("%d", &x);
scanf("%c", &c);
```

Primer 3. Napisati program koji sa tastature učitava dva cela broja i štampa njihov zbir.

Rešenje

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

int main()
{
     int a, b;
     scanf("%d", &a);
     scanf("%d", &b);
     printf("Zbir je %d\n", a+b);
     return 0;
}
```

Operatori

Aritmetički operatori

Aritmetički operatori, kao što im samo ime govori, služe za izvršavanje aritmetičkih operacija. Do sada je već korišćen operator sabiranja, a postoji ukupno pet aritmetičkih operatora:

`+` sabiranje,

- oduzimane,

* množenje,

/ deljenje,

% ostatak pri deljenju (samo za cele brojeve).

Svi ovi operatori su binarni, što znači da imaju dva argumenta i njihova svrha je jasna, osim možda poslednjeg. Npr. izraz `8%3` daje ostatak pri deljenju broja `8` sa brojem `3`, tj. daje vrednost `2`.

Primer 4. Napisati program koji sa tastature čita dva cela broja i ispisuje njihov zbir, razliku, proizvod i količnik.

Rešenje

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

int main()
{
     int a, b;
     scanf("%d", &a);
     scanf("%d", &b);
     printf("a + b = %d\n", a+b);
     printf("a - b = %d\n", a-b);
     printf("a * b = %d\n", a*b);
     printf("a / b = %f\n", (float)a/b);
     printf("a % b = %d\n", a%b);
     return 0;
}
```

Svi redovi u kodu su poznati i jasni, osim

```
printf("a / b = %f\n", (float)a/b);
```

odnosno, izraza `(float)a/b`. U ovom izrazu se koristi tzv. cast operator.

Cast operator

U programskom jeziku C postoji ugrađena automatska konverzija tipova. To znači da je moguće nekoj promenljivoj dodeliti vrednost nekog drugog tipa. Npr. kod

```
int a = 5;
float b = a;
```

je ispravan, iako se u drugom redu promjenljivoj b tipa `float` dodeljuje vrednost promenljive a koja je tipa `int`. Tu dolazi do automatske konverzije tipa `int` u `float`. U ovom slučaju ne postoji nikakav gubitak podataka, dok bi, recimo konverzija tipa `float` u `int` dovela do gubitka podataka, i to cifara iza decimalnog zareza.

Moguća je i eksplicitna konverzija tipova, kada neki izraz želimo konvertovati u odgovarajući tip. Takva konverzije je moguća korišćenjem unarnog operatora koji se naziva cast a ima oblik

```
(tip) izraz;
```

Izraz `(float)a/b` je upravo konvarzija rezultata `a/b` u tip `float`. To je neophodno jer su `a` i `b` tipa `int` i da nema eksplicitne konverzije, i rezultat `a/b` bi bio tipa `int` i ne bi bio tačan.

Relacijski i logički operatori

Ovi operatori imaju isti smisao kao i u matematici. U ovu grupu operatora spadaju

`>` veće,

`<` manje,

`<=` veće ili jednako,

`>=` manje ili jednako,

`==` jednako (**dvostruko jednako** - jednostruko jednako je operator dodele),

`!=` različito (`!` je znak negacije).

Osim ovih, u ovu grupu spadaju i logički vezinici.

`&&` logičko I,

`||` logičko ILI.

Rezultat primene ovih operatora je logička vrednost koja može biti tačna ili netačna, npr.

```
(x>4) && (x%2 == 0)
```

Ovaj izraz ispituje da li je x paran broj veći od 4.

O prioritetima operatora ovdje neće biti govora, za detaljnije pogledati [?].

Bitovni operatori

Bitovni operatori su operatori koji manipulišu sa celobronim tipovima podataka, na nivou bitova. Za potpuno razumjevanje rada ovih operatora potrebno je poznavati način zapisivanja celih brojeva u potpunom komplementu[?]. Postoji 6 operatora za rad s bitovima

`&` binarno I,

`|` binarno ILI,

`^` binarno ekskluzivno ILI,

`<<` lijevi pomak (lijevi shift),

`>>` desni pomak (desni shift),

`~` unarna negacija.

Primer 5. Izračunati vrednosti izraza `24&12, 17|12, 8^11, 17>>2, 21<<3, ~11`.

Rešenje.

Zbog zapisa, veličina podataka će biti ograničena na `8` bitova.

```
 00011000 = 24          10001 = 17           1000 =  8
 00001100 = 12           1100 = 12           1011 = 11
-------------------------------------------------------
&00001000 =  8    |    11101 = 29     ^       11 = 12

17 >> 2 = 0001   1 >> 2 = (00)000100|01 =      100 = 4
21 << 3 = 00010101 << 3 = 000|10101(000) = 10101000 = 168
~11 = ~(00001011) = 11110100 = -12
```

Primer 6. Napisati program koji izračunava vrednosti izraza iz primera 5.

Rešenje

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

int main()
{
     printf("24&12 = %d\n", 24&12);
     printf("17|12 = %d\n", 17|12);
     printf("8^11 = %d\n", 8^11);
     printf("17>>2 = %d\n", 17>>2);
     printf("21<<3 = %d\n", 21<<3);
     printf("~11 = %d\n", ~11);
     return 0;
}
```

Operatori uvećavanja i umanjivanja

U C-u postoje dva uobičajena operatora uvećavanja i umanjivanja za jedan. To su operatori `++` i `--`. Operator inkrementiranja `++` uvećava svoj operand za jedan, dok operator dekrementiranja `--` umanjuje svoj operand za jedan.

To znači da izrazi `a=a+1` i `a++`, odnosno `a=a-1` i `a--` imaju isto značenje.

Oba operatora se mogu koristiti i kao prefiksni (`++a`) i kao sufiksni (`a++`). Razlika je što se u prefiksnom korišćenju vrednost promenljive `a` uveća pre njenog korišćenja, a u sufiksnom obrnuto.

Operatori i izrazi dodeljivanja vrednosti

Slično kao kod operatora inkrementiranja i dekrementiranja, postoje skraćeni zapisi i za operatore u kojima je promenljiva na levoj strani a neki od operatora na desnoj, kakav je `i=i+2`. Skraćeni zapis za ovaj izraz je `i+=2`.

Ovakvi skraćeni zapisi su mogući za operatore `+`, `-`, `*`, `/`, `%`, `<<`, `>>`, `&`, `^` i `|`.

Kontrola toka

`if-else` izraz

Uslovni `if` izraz ima sledeći oblik

```
if(uslov)
     naredba1;
else
     naredba2;
```

`uslov` mora biti neki logički izraz. U slučaju da je uslov tačan, izvršava se `naredba1` a u suprotnom `naredba2`. `if` izraz se može pojaviti i bez `else` izraza. Ako je potrebno da se izvrši više naredbi, tada je potrebno da se stave u blok, ograničen sa vitičastim zagradama

```
if(uslov)
{
     naredba1_1;
     naredba1_2;
     naredba1_3;
     ...
}
else
{
     naredba2_1;
     naredba2_2;
     naredba2_3;
     ...
}
```

Primer 7. Ispraviti rešenje primera 4. tako da isključuje mogućnost deljenja nulom.

Rešenje

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

int main()
{
     int a, b;
     scanf("%d", &a);
     scanf("%d", &b);
     printf("a + b = %d\n", a+b);
     printf("a - b = %d\n", a-b);
     printf("a * b = %d\n", a*b);
     if(b != 0)
          printf("a / b = %f\n", (float)a/b);
     else
          printf("Greska! Deljenje nulom\n");
```

```
    printf("a % b = %d\n", a%b);
    return 0;
}
```

if-else if-else izraz

`if-else` naredba ima samo dve grane – kada je uslov ispunjen i kada nije. Međutim nekada je potrebno ispitati više uslova i u zavisnosti od njihovih vrednosti izvršavati određene naredbe. U tim slučajevima se koristi `if-else if-else` izraz, koji će biti prikazan na sledećem primeru.

Primer 8. Napisati program koji ispituje da li je uneseni broj pozitivan, negativan ili je jednak nuli.

Rešenje

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

int main()
{
    int x;
    scanf("%d", &x);
    if(x > 0)
        printf("Broj je pozitivan\n");
    else if(x < 0)
        printf("Broj je negativan\n");
    else
        printf("Broj je jednak nuli\n");
    return 0;
}
```

Operator ?

U programskom jeziku C postoji operator ? i to je jedini ternani operator, tj. ima tri operanda. Izgled operatora je

`(logicki_izraz) ? vrijednost1 : vrijednost2`

Ovaj operatoj proverava istinitost logičkog izraza i u slučaju njegove tačne vrednosti vraća vrednost1 a u suprotnom vraća vrednost2.

switch

Ovaj izraz je sličan izrazu `if-else if-else` i proverava da li izraz odgovara nekoj od konstantnih celobrojnih vrednosti (ili karaktera). `switch` izraz ima sledeći oblik

```
switch(izraz)
{
    case vrednost1:
        naredbe1
        break;
```

```
     case vrednost2:
          naredbe2
          break;
     ...
     default:
          naredbeN;
}
```

U slučaju da `izraz` ima vrednost `vrednost1` , tada se izvršavaju `naredbe1`, u slučaju da ima `vrednost2` tada se izvršavaju `naredbe2`, a u slučaju da nema nijednu od ponuđenih vrednosti tada se izvršavaju `naredbeN`.

Primer 9. Napisati program koji proverava da li je uneseni karakter razmak, tačka ili neki neki drugi znak.

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

int main()
{
     char c;
     scanf("%c", &c);
     switch(c)
     {
          case ' ':
               printf("Uneseni karakter je razmak\n");
               break;
          case '.':
               printf("Uneseni karakter je tacka\n");
               break;
          default:
               printf("Uneseni karakter nije ni tacka ni
razmak\n");
     }

    return 0;
}
```

Petlje

`for`

Primer 8. Napisati program koji štampa prvih `5` prirodnih brojeva.

Rešenje

U ovom programu je potrebno izvršiti `5` sličnih naredbi, tj. `5` poziva funkcije `printf` sa različitim argumentima. U slučajevima kada se trebaju izvršiti slične naredbe koriste se petlje. Jedan od načina da se reši ovaj zadatak je sledeći

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

```
#include <stdlib.h>

int main()
{
     int i;
     for(i=1; i<=5; i=i+1)
          printf("%d\n", i);
     return 0;
}
```

Ovde je korišćen izraz `for` koji ima sledeći oblik

```
for(inicijalizacija; uslov; korak)
```

`inicijalizacija` – izraz u kojem se promenljivima dodeljuju početne vrednosti,

`uslov` – logički izraz koji uslovljava izvršenje `for` petlje – petlja se izvršava sve dok ovaj logički izraz tačan,

`korak` – izraz u kojem se vrši uvećavanje brojača. U prethodnom primeru, u svakom se izvršavanju `for` petlje vrednost brojača i povećava za jedan.

Sva tri izraza u `for` petlji su opcioni, što znači da ne moraju postojati. Izraz

```
for( ; ; )
```

je ispravan izraz i predstavlja beskonačnu petlju.

Treba primetiti da iza linije u kojoj se nalazi `for` petlja ne nalazi karakter `;`. To je zato što je kraj komande tek u sledećem redu. Kao i kod `if` naredbe, ukoliko se for petlja odnosi na više komandi, potrebno ih je ograničiti vitičastim zagradama `{}`.

while

`while` petlja ima iste osobine kao `for` petlja i sve što se može uraditi preko `for` petlje može i preko `while` i obrnuto. `while` petlja ima sledeći oblik

```
while(uslov)
{
     komande...
}
```

Slično kao `for` petlja, samo što nema delaIza inicijalizaciju i Ikorak, već se inicijalizacija mora mora obaviti pre a korak u samoj petlji.

Primer 9. Prethodni zadatak rešiti korišćenjem `while` petlje.

Rešenje

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

int main()
{
```

```
	int i=1;
	while(i<=5)
	{
		printf("%d\n", i);
		i++;
	}
	return 0;
}
```

do-while

Kao i `while` petlja, samo što se kod `do-while` petlje prvo izvršavaju naredbe pa se onda proverava uslov. `do-while` petlja ima sledeći oblik

```
do
{
	komande...
} while(uslov);
```

Za razliku od `while` petlje, komande u `do-while` petlji će se izvršiti bar jednom.

Primer 10. Prethodni zadatak rešiti korišćenjem `do-while` petlje.

Rešenje

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

int main()
{
	int i=1;
	do
	{
		printf("%d\n", i);
		i++;
	} while(i<=5);
	return 0;
}
```

break i continue

Nekada postoji potreba izlaska iz petlje pre potpunog ispunjenja potrebnog uslova za izlazak. Za to se koristi naredba `break`.

Primer 11. Odštampati prvi broj (ako postoji) koji je manji od 500 a deljiv sa brojevima 3, 4, 5 i 7.

Rešenje

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

```
int main()
{
     int i;
     for(i=1; i<=500; i++)
          if(i%3 == 0 && i%4 == 0 && i%5 == 0 && i%7 == 0)
          {
               printf("%d\n", i);
               break;
          }
     return 0;
}
```

Naredba `continue` je slična naredbi `break`, a njen poziv otpočinje sledeću iteraciju petlje.

Primer 12. Odštampati sve brojeve manje od 20 koji nisu deljivi sa brojem 3.

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

int main()
{
     int i;
     for(i=1; i<=20; i++)
     {
          if(i%3 == 0)
               continue;
          printf("%d\n", i);
     }
     return 0;
}
```

`goto` i labele

C ima mogućnost rada sa labelama, ali se ne preporučuje njihovo korišćenje osim u slučajevima kada je potrebno izaći iz više petlji odjednom, kao u sledećem primeru

```
for(...)
     for(...)
     {
          if(...)
               goto kraj;
     }
...

kraj:
...
```

Rad sa karakterima

Kao što je već pomenuto, u programskom jeziku C tip `char` je ustvari celobrojna vrednost ASCI koda datog karaktera. To znači da se `char` može štampati i kao karakter i kao celi broj

```
char c = 'A';
printf("%d\n", c); // štampa ASCI kod karaktera
printf("%c\n", c); // štampa izgled karaktera
```

Primer 13. Napisati program koji štampa ASCI tabelu karaktera.

Rešenje

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

int main()
{
     char i;
     for(i=1; i<127; i++)
          printf("%d %c\t", i, i);

    return 0;
}
```

Ako se pogleda ASCI tabela, vidi se da, recimo karakter `'A'` ima vrednost `65`, karakter `'B'` vrednost `66`, karakter `'a'` ima vrednost `97`, karakter `'b'` ima vrednost `98` itd. Tako da vrednost izraza `'A' + 1` daje vrednost `75` odnosno karakter `'K'`.

Znakovni ulaz i izlaz

U standardnoj biblioteci, osim funkcija `printf` i `scanf` postoje funkcije koje služe za ulaz odnosno izlaz znakovnog tipa podataka. To su funkcije `getchar` i `putchar`.

Funkcija `getchar()` čita jedan karakter sa standardnog ulaza (tastature) i kao rezultat vraća njegovu ASCI vrednost, npr. kod

```
char c = getchar();
```

čita jedan karakter sa tastature i njegovu vrednost dodeljuje promenljivoj `c`.

Funkcija `putchar(c)` štampa vrednost karaktera `c` na standardnom izlazu, npr

```
putchar(c);
```

Kod unosa teksta, kao indikator za kraj unosa se koristi znak za kraj datoteke, koji se označava `EOF` (end of file). Ako je u pitanju unos sa tastature, znak `EOF` se dobija kombinacijom tastera ctrl i z.

Primer 14. Napisati program koji sa tastature čita karaktera sve dok se ne unese karakter `EOF` i ispisuje ih na ekranu.

Rešenje

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

int main()
{
	char c;
	int i;
	while(c!=EOF)
	{
		c = getchar();
		putchar(c);
	}
	return 0;
}
```

Rešeni zadaci

Zadatak 1. Napisati program koji za uneseno n štampa sumu prvih n prirodnih brojeva i n!

Rešenje

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

int main()
{
     int n, i, suma=0, fakt=1;
     scanf("%d", &n);
     for(i=1; i<=n; i++)
          suma += i;
     printf("Suma prvih %d je %d\n", n, suma);

     for(i=1; i<=n; i++)
          fakt *= i;
     printf("%d! = %d\n", n, fakt);

     return 0;
}
```

Zadatak 2. Odštampati sve delioce unesenog broja.

Rešenje

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

int main()
{
     int n, i;
     scanf("%d", &n);


     for(i=1; i<=n; i++)
          if (n%i == 0)
               printf("%d ", i);

    return 0;
}
```

Zadatak 3. Odštampati sve savršene brojeve manje od 10000 (broj je savršen ako je jednak zbiru svojih delilaca, osim sebe samog).

Rešenje

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

int main()
{
    int n, i,suma;

    for(n=1; n<1    ; n++)
    {
     suma = 0;
          for (i=1; i<n; i++)
               if (n%i == 0)
                    suma+=i;

          if (suma==n)
               printf("broj %d je savrsen\n", n);
    }

    return 0;
}
```

Zadatak 4. Napisati program koji za uneseni prirodni broj štampa zbir njegovih cifara.

Rešenje

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

int main()
{
     int n, suma = 0;
     scanf("%d", &n);
     while(n > 0)
     {
          suma += n % 10; //  sa n%10 se dobija poslednja cifra
          n /= 10;
     }
     printf("%d", suma);
     return 0;
}
```

Zadatak 5. Napisati program koji za uneseni prirodan broj ispisuje broj čije su cifre u obrnutom redosledu.

Rešenje

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

int main()
{
     int n, obrnuti = 0;
     scanf("%d", &n);
     while(n > 0)
     {
          obrnuti = obrnuti*10 + n % 10;
          n /= 10;
     }
     printf("%d", obrnuti);
     return 0;
}
```

Zadatak 6. Izračunati kvadratni korjen datog broja Njutnovom metodom sa zadatom tačnosti.

Rešenje

Njutnova formula glasi $y_{i+1} = \frac{1}{2}\left(y_i + \frac{x}{y_i}\right)$, gdje je x broj za koji se traži koren, a početna iteracija $y_0 = x + 1$. Za rešavanje ovog zadatka potrebna je funkcija koja računa apsolutnu vrednost datog broja. To je funkcija `abs(float)` koja je definisana u datoteci `math.h`[1].

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

int main()
{
     float x, y, z, eps;
     scanf("Unesite broj i tacnost: ");
     scanf("%f", &x);
     scanf("%f", &eps);
     y = x+1;
     do
     {
          z = y;
          y = .5 * (y + x/y);
     } while(abs(z-y) > eps);
     printf("Korjen iz %.2f je %f", x, y);
     return 0;
}
```

[1] Pregled načajnijih funkcija standardne biblioteke je prikazano u dodatku A

Zadatak 7. Napisati program koji simulira rad kalkulatora sa osnovnim operacijama +, −, *, /. Izraz se unosi u obliku `operator operand1 operand2`.

Rješenje

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

int main()
{
     float op1,op2;
     char op;
     scanf ("%c", &op);
     scanf ("%f", &op1);
     scanf ("%f", &op2);
     printf("izraz: %.2f %c %.2f\n",op1,op,op2);

     switch(op)
     {
          case '+':
               printf("vrednost: %.2f\n", op1 + op2);
               break;
          case '-':
               printf("vrednost: %.2f\n", op1 - op2);
               break;
          case '*':
               printf("vrednost: %.2f\n", op1 * op2);
               break;
          case '/':
               if (op2 == 0)
                    printf("Greska! Deljenje nulom\n");
               else
                    printf("vrednost: %.2f\n", op1 / op2);
               break;
          default:
               printf("Greska! Pogresan operator\n");
     }
    return 0;
}
```

Zadatak 8. Napisati program koji pronalazi rešenja kvadratne jednačine $ax^2 + bx + c = 0$

Rešenje

Za rešavanje ovog zadatka potrebna je funkcija koja računa kvadratni koren datog broja, a to je funkcija `sqrt(float)` koja je definisana u datoteci `math.h`

Moguća su tri slučaja – jednačina nema realna rešenja (D<0), ima dvostruko rešenje (D=0) i ima dva različita rešenja (D>0).

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

int main()
{
     float a, b, c;
     scanf("%f", &a);
     scanf("%f", &b);
     scanf("%f", &c);
     printf("Jednacina: %.2f*x^2 + %.2f*x + %.2f\n", a, b, c);

     float D = b*b - 4*a*c;
     if(D < 0)
          printf("Resenja nisu realna\n");
     else if(D == 0)
     {
          float x = -b/(2*a);
          printf("Jednacina ima jedno dvostruko resenje x =
%.2f\n", x);
     }
     else
     {
          float x1 = (-b + sqrt(D))/(2*a);
          float x2 = (-b - sqrt(D))/(2*a);
          printf("Resenja su x1=%.2f, x2=%.2f\n", x1, x2);
     }

    return 0;
}
```

Zadatak 9. Napisati program koji ispituje da li se dve prave $y=a_1x+b_1$ i $y=a_2x+b_2$ seku.

Rešenje

Moguća su tri slučaja – prave su paralelne, prave se poklapaju ili se seku u jednoj tački.

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

int main()
{
     float a1,a2,b1,b2;
     scanf ("%f",&a1);
     scanf ("%f",&b1);
     scanf ("%f",&a2);
     scanf ("%f",&b2);
     printf("y =%5.2fx + %5.2f\n",a1,b1);
     printf("y =%5.2fx + %5.2f\n",a2,b2);
```

```
     if (a1 == a2 && b1== b2)
          printf ("prave se poklapaju\n");
     else if (a1 == a2 && b1 != b2 )
          printf ("prave su paralelne\n");
     else
     {
          float x=(b2-b1)/(a1-a2);
          float y=a1*x+b1;
          printf ("prave se seku u tacki
(%5.2f,%5.2f)\n",x,y);
     }

    return 0;
}
```

Zadatak 10. Podaci o krugovima se sastoje od koordinata centra i dužine poluprečnika. Napisati program koji ispituje u kom su položaju dati krugovi.

Rešenje

Moguća su tri slučaja. Da se krugovi ne seku (udaljenost među centrima je veća od zbira poluprečnika), da se dodiruju (udaljenost među centrima je jednaka zbiru poluprečnika) i da se seku (udaljenost među centrima je manja od zbira poluprečnika).

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

int main()
{
    int x1,x2,y1,y2,r1,r2;
    scanf ("%d", &x1);
    scanf ("%d", &y1);
    scanf ("%d", &r1);
    scanf ("%d", &x2);
    scanf ("%d", &y2);
    scanf ("%d", &r2);

    float d = sqrt((x2-x1)*(x2-x1)+ (y2-y1)*(y2-y1));

    if (d>r1+r2)
          printf ("kruznice se ne seku");
    else if (d<r1+r2)
          printf ("kruznice se seku");
    else
          printf ("kruznice se dodiruju");

    return 0;
}
```

Zadatak 11. Napisati program koji za uneseno n štampa romb čija je dužina stranice n, kao na slici (na slici n=2)

```
  *
* *
  *
```

Rešenje

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

int main()
{
     int n, i, j, k;
     scanf("%d", &n);
     for(i=1; i<=n; i++)
     {
          for(k=n; k>i; k--)
               printf(" ");
          for(j=0; j<i; j++)
               printf("* ");
          printf("\n");
     }
     for(i=n-1; i>0; i--)
     {
          for(k=n; k>i; k--)
               printf(" ");
          for(j=0; j<i; j++)
               printf("* ");
          printf("\n");
     }

    return 0;
}
```

Zadatak 12. Napisati program koji štampa sve trocifrene brojeve kod kojih je druga cifra za dva veća od prve a treća za jedan veća od druge.

Rešenje

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

int main()
{
     int i, j, k;
     for(i=1; i<=9; i++)
          for(j=3; j<=9; j++)
               for(k=4; k<=9; k++)
                    if(j == i+2 && k == j+1)
                         printf("%d%d%d\n", i, j, k);
    return 0;
```

```
}
```

Zadatak 13. Napisati program koji uneseno mali slovo pretvara u veliko.

Rešenje

Ideja: vrednost između malog slova i njemu odgovarajućeg velikoj je konstantna za sva slova.

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

int main()
{
     char maliC;
     scanf("%c", &maliC);

     char velikiC = maliC - ('a' - 'A');
     printf("%c", velikiC);

     return 0;
}
```

Zadatak 14. Napisati program koji za uneseni karakter proverava da li je malo slovo, veliko slovo, cifra ili neki drugi karakter.

Rešenje

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

int main()
{
     char c;
     scanf("%c", &c);

     if(c >= 'A' && c <= 'Z')
          printf("Uneseni karakter je veliko slovo\n");
     else if(c >= 'a' && c <= 'z')
          printf("Uneseni karakter je malo slovo\n");
     else if(c >= '0' && c <= '9')
          printf("Uneseni karakter je cifra\n");
     else
          printf("Uneseni karakter nije slovo ni cifra\n");

    return 0;
}
```

Zadatak 15. Napisati program koji sa tastature čita karaktera sve dok se ne unese karakter `EOF` i broji unesenie samoglasnike i suglasnike.

Rešenje

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

int main()
{
     char c;
     int i, suglasnici=0, samoglasnici=0;
     while(c!=EOF)
     {
          c = getchar();
          if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
               if(c == 'a' || c == 'e' || c == 'i' || c == 'o'
|| c == 'u' || c == 'A' || c == 'E' || c == 'I' || c == 'O' ||
c == 'U')
                    samoglasnici++;
               else
                    suglasnici++;
     }
     printf("Samoglasnika: %d, suglasnika: %d", samoglasnici,
suglasnici);

     return 0;
}
```

Zadaci za vežbu

1. Napisati program koji za uneseno n izračunava sumu $\sum_{i=1}^{n} 2^{2}$
2. Napisati program koji za zadatu tačnost eps izračunava broj $e = \sum_{i=0}^{\infty} \frac{1}{i!}$
3. Napisati program koji za unesene n i m računa broj n^m (bez korišćenja funkcije pow) .
4. Napisati program koji za uneseno n i m (n > m) računa $\binom{n}{m}$
5. Napisati program koji učitava n, zatim učitava n brojeva i izračunava njihovu aritmetičku sredinu.
6. Napisati program koji određuje za koliko procenata je porasla cena CI$_1$ ako ona sad iznosi CI$_2$.
7. Napisati program koji izračunava sumu i proizvod dva kompleksna broja x1+iy1 i x2+iy2
8. Napisati program koji računa zbir, razliku i proizvod razlomaka a/b i c/d.
9. Napisati program koji za uneseno n izračunava n-ti stepen kompleksnog broja, korišćenjem De Moavrove formule[3]
10. Napisati program koji učitava brojeve m i n i pravi Dekartov proizvod skupova {1,2,...,n} i {1,2,...,m}.
11. Napisati program koji za uneseno n ispisuje trougao kao na slici (na slici n=3)

```
    1
  2 2
3 3 3
```

12. Napisati program koji za uneseno n štampa kvadrate brojeva 1, 2, ... , n-1.
13. Napisati program koji za uneseno x izračunava vrijednost y po formuli

$$y = \begin{cases} \frac{1}{x}, & x > 0 \\ 0, & x = 0 \\ -\frac{1}{x}, & x < 0 \end{cases}$$

14. Napisati program koji izračunava aritmetičku sredinu cifara datog broja.
15. Napisati program koji u datom četverocifrenom broju izračunava razliku između prve i poslednje cifre.
16. Napisati program koji za unesene n i m (m<n) izračunava sumu posljednjih m cifara broja n.
17. Napisati program koji za unesene n i m učitava n brojeva i izračunava aritmetičku sredinu brojeva koji su veći od m.
18. Napisati program koji učitava broj m i zatim učitava brojeve sve dok im suma ne pređe m.
19. Napisati program koji za n unijeti cijelih brojeva računa zbir negativnih.
20. Napisati program koji za uneseno n računa n-ti član Fibonačijevog niza
21. Napisati program koji za unesena tri cijela broja ispisuje najveći.

---

[2] Za funkciju stepenovanja pogledati dodatak A

[3] Za funkcije sin i cos pogledati dodatak A

22. Napisati program koji za unesena tri cijela broja ispituje da li čine Pitagorinu trojku.
23. *Napisati program koji nalazi NZD za dva broja korišćenjem Euklidovog algoritma.
24. *Napisati program koji za uneseno n izračunava $\sqrt{1+\sqrt{\frac{1}{2}+\ldots+\sqrt{\frac{1}{n}}}}$
25. Napisati program koji za uneseno n učitava n brojeva i određuje najveći i najmanji među njima.
26. Napisati program koji rješava jednačinu ax+b=0.
27. Napisati program koji ispituje da li je prava y=kx+n tangenta parabole $y_I^2=2px$ (vrijednosti k, n i p se unose sa tastature).
28. Napisati program koji ispituje da li se sijeku prava y=kx+n i kružnica $x_I^2+y_I^2=r_I^2$ (vrijednosti k, n i r se unose sa tastature).
29. Napisati program koji ispituje da li je vrijednost polinoma $2x_I^3+x_I^2-2x+1$ pozitivna ili negativna u tački n koja se unosi sa tastature.
30. Napisati program koji ispituje da li je uneseni broj Armstrongov. Amrstrongov broj je jednak zbiru kubova svojih cifara.
31. Napisati program koji ispituje da li je uneseni broj Nivenov. Nivenov broj je broj koji je djeljiv sumom svojih cifara.
32. Napisati program koji učitava broj b i osnovu o < 10 i zatim ispisuje broj o u dekadnom brojnom sistemu.
33. Napisati program koji izračunava površinu pravougaonika čije su stranice paralelne koordinatnim osama, a zada je preko dijagonalnih tjemena (x_I, y_I) i (x_1, y_1)
34. Napisati program koji ispituje da li tačke $A(x_I, y_I)$, $B(x_1, y_1)$, $C(x_2, y_2)$ i $D(x_3, y_3)$ čine paralelogram ABCD.
35. Napisati program koji štampa kvadrant u kojem se nalazi tačka (x_I, y_I).
36. Napisati program koji učitava vrijednost n (n < 180) i ispituje da li je ugao od n stepeni oštar, prav ili tup.
37. Napisati program koji računa maksimim za unesena četiri broja.
38. Napisati program koji ispituje da li tri unesene duži mogu da naprave trougao (koristiti nejednakost trougla).
39. Napisati program koji računa površinu trougla datog preko svojih tjemena (x_I, y_I), (x_1, y_1), (x_2, y_2) (koristiti Heronov obrazac).
40. Napisati program koji provjerava da li je uneseni karakter slovo, tačka ili razmak.
41. Napisati program koji za uneseni broj n ispisuje n-ti dan u nedjelji (npr. za 1 se ispisuje „Ponedjeljak“.
42. Napisati program koji provjerava da li tri cijela broja koja se unose sa tastature h, m i s čine ispravno vrijeme (h:m:s).
43. Napisati program koji provjerava da li tri cijela broja koja se unose sa tastature d, m i g čine ispravan datum (d.m.g).
44. Napisati program koji za uneseni broj n ispisuje n-ti mjesec u godini.
45. Napisati program koji provjerava da li je unesena godina prestupna.
46. Napisati program koji ispisuje ocjenu koju je student dobio na ispitu na osnovu osvojenih bodova (5 – od 0 do 50, 6 – od 51 do 60 itd).

Funkcije

Do sada je bilo riječi o funkcijama. Pominjane su funkcije za ulaz/izlaz (`printf`, `scanf`, `getchar`, `putchar`...), matematičke funkcije (`fabs`, `sqrt`, `sin`, `cos`...) itd. Moguće je da korisnici sami definišu funkcije koje su im neophodne za izvršavanje programa.

Deklaracija funkcije izgleda

```
tip funkcije ime funkcije(lista argumenata...)
```

`tip_funkcije` predstavlja tip rezultata kojeg funkcija vraća. To može biti bilo koji od standarnih ili korisnički definisanih tipova programskog jezika C ili tip `void`

`ime_funkcije` je proizvoljan identifikator funkcije za koje važe ista pravila kao i za davanje imena promjenljivima

`lista_argumenata` je niz od nijednog, jednog ili više argumenata funkcije. Svaki argument mora imati svoje ime i tip. U slučaju da postoji više argumenata svaki novi je potrebno odvojiti zarezom od prethodnog.

Primjeri deklaracija funkcija

```
int funkcija();
float funkcija1(int x);
void funkcija2(intx, char y, float z);
```

Osim deklaracije funkcije potrebno je definisati šta ta funkcija radi. Definicija funkcije se piše u bloku naredbi ograničenim `{}`. Npr. za prvu funkciju bi to izgledalo

```
int funkcija()
{
     Tijelo funkcije
     ...
}
```

Deklaracija funkcije mora biti iznad `main()` funkcije, dok definicija može biti i ispod, ali u prostijim programima je lakše napisati definiciju funkcije odmah iza deklaracije.

U Idefiniciji Ifunkcije Imora Ipostojati I(osim Ieventualno Iu Ifunkcijama Itipa `void`) naredba koja prekida izvršavanje funkcije i vraća njenu vrijednost. To je naredba `return`.

Funkcija za sabiranje dva realna broja bi izgledala

```
float saberi(float x, float y)
{
     float rezultat = x+y;
     return rezultat;
}
```

Pozivu ove funkcije je potrebno kao argumente zadati dva realna broja a ona kao rezultat vraća njihov zbir

```
float rezultat = saberi(2.4, 3.5);
```

U slučaju da je funkcija tipa `void`, ona ne vraća nikakv rezultat. Ukoliko je potrebno izaći iz te funkcije, može se pozvati komanda `return` bez vrijednosti iza nje.

Bitno je Inapomenuti da Ikod ovakvog Idefinisanja funkcija Iargumenti Ine Imijenjaju svoje Ivrijednosti Inakon Ipoziva Ifunkcije, Itj. Injihova Ivrijednost Iostaje Inepromjenjena. IU sljedećem programu vrijednost promjenljive `n` ostaje nepromjenjena u nakon pozova funkcije `f`, iako se u funkciji vrijednost argumenta uvećava za jedan.

```
void f(int n)
{
     n++;
}

int main()
{
     int n = 1;
     f(n);
     printf("%d\n", n);
     return 0;
}
```

Primjer 15. Napisati funkciju koja izračunava faktorijel datog broja.

Rješenje

```
int faktorijel(unsigned int broj)
{
     int i, fakt=1;

     for(i=1; i<=broj; i++)
          fakt *= i;
     return fakt;
}
```

Primjer 16. Napisati funkciju koja ispituje da li je dati broj prost.

Rješenje

```
int prost(unsigned int broj)
{
     int i;
     for(i=2; i<broj; i++)
          if(broj % i == 0)
               return 0;

     return 1;
}
```

Obzirom da u C-u ne postoji logički tip, ukoliko se radi o nekoj funkciji koja treba da vrati vrijednost tačno/netačno, kao što je slučaj sa ovom, tada je ona tipa `int` i vraća `1` ako treba da vrati tačno a I ako treba da vrati netačno.

z tog razloga ova funkcija je tipa `int` i u zavisnosti od (ne)ispunjenja uslova vraća se vrijednost I ili vrijednost `1`.

Primjer 17. Napisati program koji štampa sve proste brojeve manje od 1000.

Rješenje

Ovdje Ićemo Iiskoristiti Ifunkciju Iiz Iprethodnog Iprimjera. IZa Isvaki Ibroj Iod I1000 pozivanjem funkcije iz prethodnog primjera ćemo ispitati da li je prost.

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

int prost(unsigned int broj)
{
     int i;
     for(i=2; i<broj; i++)
          if(broj % i == 0)
               return 0;
     return 1;
}

int main()
{
     int i;
     for(i=1; i<=1   ; i++)
          if(prost(i) == 1)
               printf("%d\t", i);

     return 0;
}
```

Primjer 18. Napisati program koji ispisuje sve brojeve manje od 500 koji su Nivenovi i Armstrongovi.

Rješenje

Kod zadataka koji su kompleksniji, najlakše je problem razbiti na podprobleme i za svaki od njih uvesti funkciju. Ovdje će biti uvedene četiri funkcije – za računanje sume cifara datog broja, za računanje sume kubova cifara datog broja i funkcija koje ispituju da li je broj Nivenov odnosno Armstrongov.

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

int sumacifara(int n)
{ //
     int suma = 0 ;
     for ( ; n>0; n/=10)
          suma += n%10;
     return suma;
}

int sumakubova(int n)
{
     int suma = 0 ;
```

```
     for ( ; n>0; n/=10)
          suma += pow(n%10, 3);
     return suma;
}

int nivenov(int n)
{
     if(n % sumacifara(n) == 0)
          return 1;
     return 0;
}

int armstrongov(int n)
{
     if(n == sumakubova(n))
          return 1;
     return 0;
}

int main()
{
     int i;
     for(i=1; i<=500; i++)
          if(armstrongov(i) && nivenov(i))
               printf("%d\t", i);

     return 0;
}
```

Rekurzija

U programskom jeziku C moguće je pisati rekurzivne funkcije. To znači da funkcija može pozivati samu sebe, bilo direktno ili indirektno. Kod Irekurzivnih funkcija bitno je obezbijediti izlaz iz rekurzije, tj. uslov u kojem se rekurzija prekida.

Primjer 19. Napisati rekurzivnu funkciju koja izračunava faktorijel datog broja.

Rješenje

Funkcija faktorijela se Irekurzivno može Ipredstaviti Ikao `f(n) = n*f(n-1)` Isa početnim uslovom `f(1) = 1`.

```
int faktorijel(unsigned int n)
{
     if(n == 1)
          return 1;
     return n*faktorijel(n-1);
}
```

Primjer 20. Napisati rekurzivnu funkciju koja štampa prvih n prirodnih brojeva u obrnutom redoslijedu.

Rješenje

```
void stampa(unsigned int n)
{
     if(n == 0)
          return;
     printf("%d\n", n);
     stampa(n-1);
}
```

Nizovi

Deklaracija niza u programskom jeziku C ima sljedeći oblik

```
tip niza ime niza[maksimalna veličina]
```

Sljedeće deklaracije nizova su ispravne

```
int niz_cijelih_brojeva[10];
int niz_realnih_brojeva[5];
char niz_karaktera_brojeva[15];
```

Elementima Iniza Ise Ipristupa Ipreko Injihovih Iindeksa Ikoji Ise Inavode Iu Iuglastim zagradama iza imena niza, s tim što prvi element niza ima indeks nula. Tako, prvom elementu niza iz prošlog primjera ćemo dodijeliti neki vrijednost na sljedeći način

```
niz cijelih brojeva[0] = vrijednost;
```

nicijalizacija elemenata niza može se izvršiti pri samoj deklaraciji niza tako što se nakon deklaracije niza u uglu vitičastih zagrada navedu vrijednosti elementa niz. Izraz

```
int niz[5] = {1, 2, 3, 4, 5};
```

deklariše niz od pet elemenata čije su početne vrijednosti `1`, `2`, `3`, `4` i `5`, odnosno dobija se niz promjenljivih

```
int niz[0] = 1;
int niz[1] = 2;
int niz[2] = 3;
int niz[3] = 4;
int niz[4] = 5;
```

U radu s nizovima, ondnosno njihovim indeksima prirodno se nameće korišćenje `for` petlji, pri čemu se brojači petlje koriste za indekse u nizu. U takvim petljama brojači idu od I do `duzina_niza-1`.

Sljedeći Iprimjer Ipokazuje Ikako Ideklarisati Iniz Iod I10 Icijelih Ibrojeva, Iučitati Imu vrijednosti elemenata sa tastature i odštampati ih

```
int niz[10], i;
for(i=0; i<10; i++)
     scanf("%d", &niz[i]);
for(i=0; i<10; i++)
     printf("%d ", niz[i]);
```

U C se mogu na sličan način definisati i višedimenzioni nizovi, samo je potrebno pri deklaraciji Iniza Inavesti Iveličine Isvake Iod Idimenzija. IDovodimenzioni Iniz Icijelih Ibrojeva dimenzija `2x2` bi definisali na sljedeći način

```
int dvo niz[2][2];
```

ndeksi višedimenzionalnih nizova imaju iste osobine kao i kod jednodimenzionih.

Primjer 21. Učitati niz sa tastature i izračunati aritmetičku sredinu elemenata, i minimalni i maksimalni element.

Rješenje

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

int main()
{
    int a[10], i, suma=0, min, maks;
    for(i=0; i<10; i++)
          scanf("%d", &a[i]);
     min = maks = a[0];

     for(i=0; i<10; i++)
     {
          suma += a[i];
          min = (a[i] < min) ? a[i] : min;
          maks = (a[i] > maks) ? a[i] : maks;
     }

     printf("Artimeticka sredina je %.2f\n", (float)suma/10);
     printf("Minimalni element %d\n", min);
     printf("Maksimalni element %d\n", maks);

    return 0;
}
```

Primjer 22. Učitati niz sa tastature dvije matrice 3x3 pronaći njihov zbir i odštampati ga na ekranu.

Rješenje

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

int main()
{
     int mat1[3][3], mat2[3][3], zbir[3][3];

     int i, j;
     printf("Unesite elemente matrice 1\n");
     for(i=0; i<3; i++)
          for(j=0; j<3; j++)
               scanf("%d", &mat1[i][j]);

     printf("Unesite elemente matrice 2\n");
     for(i=0; i<3; i++)
          for(j=0; j<3; j++)
               scanf("%d", &mat2[i][j]);
```

```
     printf("Zbir matrica je\n");
     for(i=0; i<3; i++)
     {
          for(j=0; j<3; j++)
               printf("%d ", mat1[i][j] + mat2[i][j]);
          printf("\n");
     }

     return 0;
}
```

Primjer 23. Napisati rekurzivnu i iterativnu funkciju koja izračunava n-ti član (n < 20) Fibonačijevog niza.

Rješenje:

n-ti član Fibonačijevog niza zadat je rekurentnom relacijom $f_n = f_{n-1} + f_{n-2}$.

```
int fib_rek(int n)
{
     if(n > 20)
     {
          printf("Greska! n mora biti manje od 20\n");
          return -1;
     }
     if(n == 1 || n == 2)
          return 1;
     return fib_rek(n-1) + fib_rek(n-2);
}

int fib_iter(int n)
{
     if(n > 20)
     {
          printf("Greska! n mora biti manje od 20\n");
          return -1;
     }

     int niz[20], i;
     niz[0] = niz[1] = 1;
     for(i=2; i<20; i++)
          niz[i] = niz[i-1] + niz[i-2];
     return niz[n];
}
```

Nizovi karaktera

U programskom jeziku C, nizovi karaktera se ponašaju drugačije od nizova brojeva. Na kraju svakog niza karaktera se automatski dodaje karakter `'\0'` koji predstavlja kraj niza. Tako niz karaktera "`abcd`" u C se predstavlja kao `[a,b,c,d,\0]`.

Ova činjenica nam omogućava da koristimo nizove karaktera, čak i kada ne znamo njihovu dužinu, kretanjem kroz niz sve dok se ne dođe do karaktera `'\0'`.

Osim toga, niz karaktera se može unijeti i štampati u cjelini, a ne element po element kao nizovi brojeva. Konverzija za niz karaktera je `%s`.

Sljedeći primjer pokazuje kako sa tastaure možemo učitati niz karaktera i odrediti mu dužinu

```
char niz[10];
scanf("%s", &niz);
int i, duzina = 0;
for(i=0; niz[i] != '\0'; i++)
     duzina++;
printf("%d", duzina);
```

Nizovi kao argumenti funkcija

Ukoliko želimo da definišemo funkciju koja za argument ima niz, niz kao argument funkcije možemo prenijeti na jedan od sljedećih načina

```
tip_funkcije f1(tip_niza ime_niza[]);
tip funkcije f2(tip niza* ime niza);
```

Funkcija koja računa dužinu niza karaktera izgledala bi

```
int duzina(char niz[])
{
     int i, duzina = 0;
     for(i=0; niz[i] != '\0'; i++)
          duzina++;
     return duzina;
}
```

Kada je niz argument funkcije, vrijednost elemenata se može promjeniti u funkciji, za razliku od argumenata osnovnih tipova. Nakon izvršavanja sljedećeg programa vrijednost elemenata niza postaju 2 i 3.

```
void f(int a[])
{
     a[0]++;
     a[1]++;
}

int main()
{
     int niz[2] = {1, 2};
     f(niz);
     printf("%d\n%d", niz[0], niz[1]);
     return 0;
}
```

Razlozi za ovo biće objašnjeni kada se bude govorilo o pokazivačima.

Primjer 24. Napisati funkciju koja dopisuje jedan niz karaktera na drugi.

Rješenje

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

void prepis(char prvi[], char drugi[], char rezultat[])
{
     int i, j=0;
     for(i=0; prvi[i] != '\0'; i++)
          rezultat[j++] = prvi[i];
     for(i=0; drugi[i] != '\0'; i++)
          rezultat[j++] = drugi[i];
     rezultat[j] = '\0';
}

int main()
{
     char prvi[10], drugi[10], rezultat[20];
     scanf("%s", &prvi);
     scanf("%s", &drugi);
     prepis(prvi, drugi, rezultat);
     printf("%s", rezultat);
     return 0;
}
```

Riješeni zadaci

Zadatak 16. Napisati funkciju koja provjerava da li dat niz karaktera predstavlja ispravan hekdasecimalni broj u obliku #broj i funkciju koja pretvara dati niz karaktera koji predstavlja heksadekadni broj u dekadni.

Rješenje:

Radi lakšeg rješavanja ovog zadatka, uvešćemo pomoćne funkcije koje ispituju da li dati karakter predstavlja cifru, slovo između a i f i slovo između A i F. U rješenju se koristi funkcija `strlen(char s[])` biblioteke `string.h`[4] koja vraća duzinu niza karaktera s i koja je analogna funkciji `int duzina(char s[])`.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

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

int slovoa_f(char c)
{
     if(c >= 'a' && c <= 'f')
          return 1;
     return 0;
}

int slovoA_F(char c)
{
     if(c >= 'A' && c <= 'F')
          return 1;
     return 0;
}

int ispravan(char hex[])
{
     if(hex[0] != '#')
          return 0;
     int i;
     for(i=1; hex[i] != '\0'; i++)
          if(cifra(hex[i]) == 0 && slovoa_f(hex[i]) == 0 &&
             slovoA_F(hex[i]) == 0)
               return 0;
     return 1;
}
```

[4] Opis datoteke `string.h` dat je u dodatku A

```
int hexudek(char hex[])
{
     int i, rez = 0;
     int d = strlen(hex);
     for(i=1; hex[i] != '\0'; i++)
     {
          if(cifra(hex[i]) == 1)
               rez += (hex[i] - '0') * (int) pow(16, d-i-1);
          else if(slovoa_f(hex[i]) == 1)
               rez += (hex[i] - 'a' + 10) *
                         (int) pow(16, d-i-1);
          else
               rez += (hex[i] - 'A' + 10) *
                         (int) pow(16, d-i-1);
     }
     return rez;
}

int main()
{
     char niz[10];
     scanf("%s", &niz);
     if(ispravan(niz) == 0)
          printf("Broj nije ispravan\n");
     else
          printf("%s = %d", niz, hexudek(niz));
     return 0;
}
```

Zadatak 17. Sa tasture se unosi niz karaktera oblika `<broj1><operator><broj2>`, pri čemu su broj1 i broj2 dvocifreni brojevi a operator +, −, * ili /. Napisati program koji učitava niz karaktera, ispituje da li je ispravan izraz i izračunava vrijednost unesenog izraza.

Rješenje

U programu ćemo koristiti ugrađenu funkciju `isdigit(char c)` koja je definisana u Idatoteci `ctype.h`[5] koja Iispituje Ida Ili Ije Idati Ikarakter Icifra Ii Ianalogna Ije Ifunkciji `cifra(char c)` iz prethodnoh zadatka.

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

int ispravan(char izraz[])
{
     if(!isdigit(izraz[0]) || !isdigit(izraz[1])
          || !isdigit(izraz[3]) || !isdigit(izraz[4]))
               return 0;
     if(izraz[2] != '+' && izraz[2] != '-'
          && izraz[2] != '*' && izraz[2] != '/')
               return 0;
```

[5] Opis datoteke ctype.h dat je u dodatku A

```
     return 1;
}

int main()
{
     char izraz[5];
     scanf("%s", &izraz);
     if(ispravan(izraz) == 0)
     {
          printf("Izraz nije ispravan!\n");
          return -1;
     }

     int broj1 = (izraz[0] - '0') * 10 + (izraz[1] - '0');
     int broj2 = (izraz[3] - '0') * 10 + (izraz[4] - '0');

     printf("%s = ", izraz);
     switch(izraz[2])
     {
          case '+':
               printf("%d", broj1 + broj2);
               break;
          case '-':
               printf("%d", broj1 - broj2);
               break;
          case '*':
               printf("%d", broj1 * broj2);
               break;
          case '/':
               printf("%.2f", (double)broj1 / broj2);
               break;
     }
     return 0;
}
```

Zadatak 18. Napisati program koji sa tastature učitava polinom (stepen < 10 i koeficijente), štampa ga na ekran, učitava vrijednost v i štampati vrijednost polinoma u tački v.

Rješenje

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

int main()
{
     float koef[11];
     int stepen, i;
     printf("Unesite stepen polinoma:");
     scanf("%d", &stepen);

     for(i=0; i<=stepen; i++)
     {
          printf("Unesite koeficijent a[%d]: ", i);
          scanf("%f", &koef[i]);
```

```
    }

    printf("P(x) = ");
    for(i=0; i<=stepen; i++)
        printf("%.2f*x^%d + ", koef[i], i);
    printf("\n");

    float v, vr = 0;
    printf("Unesite vrijednost v: ");
    scanf("%f", &v);

    for(i=0; i<=stepen; i++)
        vr += koef[i] * pow(v,i);

    printf("P(%.2f) = %.2f\n", v, vr);

    return 0;
}
```

Zadaci za vježbu

1. Sa ulaza se unosi osnova <= 10 i broj. Provjeriti da li je taj broj ispravan broj za datu osnovu i ako jeste izračunati njegovu vrijednost u osnovi 10.
2. Sa ulaza se unosi broj u osnovi deset i osnova <= 10. Odštampati vrijednost datog broja u datoj osnovi
3. Napisati program koji sa tastature čita broj n (najviše 100), niz od n cijelih brojeva i vrijednost n a zatim provjerava da li je v u nizu, na kom je mjestu i koliko u nizu ima brojeva manjih od n
4. Za dati niz realnih brojeva ispitati da li je „testerast“ (niz je testarast ako važi $x_1<x_2>x_3<x_4..$ ili $x_1>x_2<x_3>x_4..$)
5. Napisati program koji ispisuje sve proste brojeve manje od 10000 koristeći Eratostenovo sito.
6. Zadatu matricu ispisati spiralno, u smjeru kazaljke na satu, počevši od $A[1,1]$*
7. Napisati program koji učitava dvije realne kvadratne matrice (prvo dimenzije pa elemente, najviše 10x10) i ispisuje njihov proizvod.
8. Napisati program koji učitava dva polinoma i ispiruje nihov proizvod.
9. Napirati program koji izračunava n-ti stepen datog polinoma.
10. Napirati program koji izračunava n-ti izvod datog polinoma.
11. Napisati rekurzivnu funkciju koja ispisuje romb čija je dužina stranice n, kao na slici (na slici n=2)

```
    *
   * *
    *
```

12. Napisati program koji sa tastature učitava broj n i štampa sve permutacije tih n elemenata.
13. Napisati funkciju koja dati realni broj prevodi u odgovarajući niz karaktera, npr. 123.123 u „123,123“
14. Napisati funkcije koje ispituju da li su sva niza karaktera jednaka i da li su jednaki na prvih n mjesta (ne koristiti datoteku string.h)

15. Napisati funkciju koja koja dati niz karaktera razbija na dva niza karaktera date dužine.
16. Data su dva niza karaktera. Ispitati da li je drugi niz karaktera podniz prvog.
17. Data su dva niza karaktera. Napisati program koji pravi novi niz karaktera koji se sastoji od karaktera koji se pojavljuju u oba niza.
18. Data su dva niza karaktera. Napisati program koji pravi novi niz karaktera koji se sastoji od karaktera koji se pojavljuju u prvom, a ne pojavljuju se u drugom nizu.
19. Napisati program koji vrši šifrovanje ulaznog teksta: svako slovo sa ulaza zamjenjuje se sa slovom koje je na tri mjesta udaljeno od njega u abecedi.
20. Napisati funkciju koja na osnovu date reči formira šifru koja se dobija tako što se svako slovo u riječi zameni sa naredna tri slova koja su mu susedna u abecedi. Na primer, reč "tamo" treba da bude zamenjena sa "uvwbcdnoppqr".
21. Napisati program koji u datom tekstu uklanja sve višestruke razmake.
22. Napisati program koji iz datog teksta štampa riječi (neprekidne nizove karaktera). Svaka riječ treba da se nalazi u novom redu.
23. Napisati funkcije koje dati niz slova pretvara u mala, odnosno u velika slova.
24. Napisati program koji radi s dvodimenzionim matricama koristeći nizove (učitavanje, štampa i sabiranje)
25. Napisati funkciju koja provjerava da li je data matrica simetrična
26. Napisati funkciju koja provjerava da li je data matrica magični kvadrat (magični kvadrat je kadratna matrica kod koje je suma brojeva u svakom redu i koloni jednaka)
27.
    a. Napisati funkciju koja za datu matricu računa transponovanu matricu
    b. Napisati funkciju koja provjerava da li je data matrica jedinična

8. Za datu matricu ispitati da li je ortogonalna (matrica je ortogonalna ako važi $A \cdot A^T = E$)

Pokazivači

Osnovni tipovi koji su do sada korišćeni sadrže vrijednosti promjenljivih. Pokazivači su promjenljive koja sadrže memorijsku adresu promjenljive. Za razumjevanje principa rada pokazivača Ipotrebno Ije Ipoznavati Iosnovne Iprincipe Iorganizacije Imemorije Iračunara. Memorija je, ustvari, niz lokacija u koje možemo upisivati vrijednosti ili iz kojih možemo čitati vrijednosti. Svaka lokacija predstavlja prostor u koji se može zapisati jedan bajt (osam binarnih cifara) ima svoju adresu. Tako se tip `char` može zapisati na jednu memorijsku lokaciju, `int` na četiri i `double` na osam lokacija (o veličini osnovnih tipova je bilo govora u dijelu o tipovima podataka).

Da bi se dobila adresa nekog objekta koristi se unarni operator `&`.

Sljedeći kod prikazuje kako odštampati adresu neke promjenljive x.

```
int x;
printf("Adresa promjenljive: %x", &x);
```

U Iprincipu, Ivrijednost Isame Iadrese Inas Ii Ine Izanima, Iali Ipomoću Iovoga Imožemo provjeriti jednu važnu činjenicu vezanu za alokaciju memorije za nizove.

Primjer 24. Napisati program koji štampa memorijske lokacije članova niza od 10 cijelih brojeva.

Rješenje

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

int main()
{
    int niz[10], i;
    for(i=0; i<5; i++)
          printf("Adresa clana %d: %d\n", i, &niz[i]);

    return 0;
}
```

Rezultat izvršavanja ovog programa izgleda ovako[6]:

```
Adresa clana 0: 2293520
Adresa clana 1: 2293524
Adresa clana 2: 2293528
Adresa clana 3: 2293532
Adresa clana 4: 2293536
Adresa clana 5: 2293540
Adresa clana 6: 2293544
Adresa clana 7: 2293548
Adresa clana 8: 2293552
Adresa clana 9: 2293556
```

Ovo nam pokazuje da se elementi niza u memoriji nalaze na susjednim lokacijama, tj. drugi element se nalazi iza lokacije na kojoj se nalazi prvi itd.

[6] Rezultat ovog programa neće biti uvijek isti, moguće je da se razlikuju vrijednosti adresa.

Ako u kodu pokušamo odštampati vrijednost promjenljive „niz“, tj.

```
printf("Vrijednost promjenljive \"niz\": %d\n", niz);
```

dobićemo

```
Vrijednost promjenljive "niz": 2293520
```

Ovim Ije Ipokazano, Ida Ije Ipromjenljiva Ikoja Ioznačava Iniz, Iustvari Ipokazivač Ikoji pokazuje na prvi element niza.

Da bi se definisala promjenljiva koja sadrži pokazivač na neki tip, potrebno je iza tipa promjenljive dodati znak `*`. Tako dobijamo četiri tipa pokazivača, na svaki od osnovnih tipova:

```
char*
int*
float*
double*
```

Potrebno je napomenuti da ako u jednom redu definišemo više pokazivača, potrebno je ispred imena svakog od njih dodati *. Tako, npr. kod

```
int *a, *b;
```

definiše dva pokazivača tipa `int`, a kod

```
int *a, b;
```

definiše promjenljivu `a` koja je pokazivač tipa `int` i promjenljivu `b` koja je tipa `int`.

Kada definišemo pokazivač (promjenljivu pokazivačkog tipa) u nju možemo smjestiti adresu neke promjenljive. Sljedeći kod pokazuje kako se pokazivaču `p` može dodjeliti adresa objekta x.

```
int x = 5;
int* p = &x;
```

zrazom `p=&x` Ipromjenljivoj `p` Ise Idodjeljuje Iadresa Ipromjenljive `x` Ii Iukoliko pristupamo promjenljivoj `p` mi ustvari pristupamo adresi na kojoj se nalazi promjenljiva `x`, odnosno adresi gdje se nalazi njena vrijednost. Ukoliko želimo da pristupimo vrijednosti na koju pokazuje pokazivač p, moramo koristiti unarni operator dereferenciranja `*`. U kodu

```
printf("%d\n", p);
printf("%d\n", *p);
```

prvi red štampa adresu na koju `p` pokazuje, a drugi red štampa vrijednost koja se nalazi u memoriji na koju pokazuje `p` (tj. vrijednost 5).

Nakon ovoga, postaje jasno šta je karakter `&` ispred imena promjenljive u funkciji `scanf` - funkcija za argument ima adresu promjenljive a ne njenu vrijednost. Razlozi za to biće detaljno objašnjeni u dijelu u kojem će se govoriti o pokazivačima kao argumentima funkcija.

Dinamička alokacija memorije

Kada se pokazivač definiše, on ne sadrži adresu lokacije kojoj možemo pristupiti i da bi Iradili Isa Injim Ineophodno Ije Idodjeliti Imu Iadresu Ineke Ilokacije. IPokazivaču Imože Ibiti dodjeljena Iadresa Ineke Iveć Ipostojeće Ipromjenljive Iili Imu Ise Imože Idodjeliti Iadresa Ineke lokacije u memoriji. Da bi mogli pristupati toj adresi neophodno je alocirati memoriju na toj lokaciji, tj. "reći" operativnom sistemu tu memorijsku lokacija dodjeljuje programu koji se

izvršava. Ova operacija se naziva dinamička alokacija memorije, i ona se vrši u vrijeme izvršavanja Iprograma. IKod Ikorišćenja I"nepokazivačkih" Itipova Ise Ine Ivrši Idinamička alokacija, tj. automatski se alocira potreban prostor za memorisanje promjenljivih i nakon završetka izvršavanja program se taj prostor oslobađa.

Da bi se memorija dinamički alocirala, koristi se funkcija

```
void *malloc(size t size);
```

Ova Ifunkcija Iima Ijedan Iargument, Ia Ito Ije Iveličina Imemorijskog Iprostora Ikoji Ise alocira, u bajtima. Rezultat funkcije je pokazivač na alociranu memoriju ili NULL pokazivač ukoliko je alokacija nemoguća. Obzirom da se ovom funkcijom alokacija vrši za sve tipove pokazivača, njen rezultat je tipa `void*`, pa ja potrebno da se uradi cast-ovanje rezultata u potreban tip.

Definisanje Ipokazivača Ina Icijeli Ibroj Ii Ialokacija Imemorije Ina Ikoju Ion Ipokazuje izgledaće

```
int *p = (int*) malloc(sizeof(int));
```

Obzirom da veličina tipova nije ista na svim sistemima, dobra praksa je da se pri alokaciji koristi operator sizeof o kojem je ranije bilo govora.

U Iopštem Islučaju, Idefinisanje Ipokazivača Ii Ialokacija Imemorije Iza Iproizvoljan Itip podataka izgleda

```
TIP *p = (TIP*) malloc(sizeof(TIP));
```

pri čemu TIP može biti bilo koji od osnovnih ili korišnički-definisanih tipova.

Nakon izvršenja programa koji koristi dinamičku alokaciju memorije ta memorija se ne oslobađa automatski. Da bi program bio potpuno ispravan, potrebno je tu memoriju ručno osloboditi. Ukoliko se to ne učini, program će raditi bez grešaka, ali će nakon izvršavanja programa dinamički alocirana memorija ostati zauzeta. Ovaj problem je poznat kao "curenje memorije" ili ne engleskom "memory leak". Da bi se dealocirala memorija na koju pokazuje neki pokazivač koristi se funkcija

```
void free(void *pointer);
```

pri Ičemi Ije Ipointer Ipokazivač Ina Imemorijsku Ialokaciju Ikoju Iželimo Idealocirati. Funkcije `malloc` i `free` se uvijek trebaju pojavljivati u paru, tj. koliko imamo fukncija `malloc`, moramo imati toliko i funkcija `free`.

Primjer 25. Napisati program koji sa tastature čita dva broja i ispisuje njihov zbir na ekranu. Memoriju za brojeve alocirati dinamički.

Rješenje

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

int main()
{
    int *p1, *p2;
    p1 = (int*) malloc(sizeof(int));
    p2 = (int*) malloc(sizeof(int));
    scanf("%d", p1);
    scanf("%d", p2);
    printf("%d + %d = %d", *p1, *p2, *p1 + *p2);
    free(p1);
    free(p2);
```

```
    return 0;
}
```

U ovom primjeru u funkciji `scanf` ne postoji karakter `&` ispred imena promjenljivih, jer su promjenljive `p1` i `p2` pokazivači, pa oni u sebi sadrže adrese - a to je upravo ono šta funkcija scanf očekuje za argument.

Nizovi i pokazivači

Dosadašnje korišćenje pokazivača i nije bilo tako efektno, tj. za svaku promjenljivu smo morali definisati pokazivač i alocirati potrebnu memoriju a to se automatski dešava kada ne koristimo pokazivače. Pokazivači pokazuju svoju snagu i svrsishodnost korišćenja kada je potrebno alocirati memoriju za nizove. Do sada je svaki niz imao ograničenu veličinu - pri definisanju niza smo navodili maksimalnu veličinu koliku niz može da ima. Ovo je loše iz dva razloga. Prvi od njih je postojanje mogućnosti da je korisniku potreban veći niz, a drugi razlog je da postoje memorijske lokacije koje alociramo bez potrebe. Nrp. ako definišemo niz od maksimalno 100 elemenata, našem korisniku može zatrebati niz od više elemenata, ili pak, korisniku je možda potreban niz od samo nekoliko elemenata pri čemu svi ostali elementi zazumaju memoriju bez potrebe.

Pokazivači omogućavaju i dinamičku alokaciju nizova, tačnije omogućavaju alokaciju memorije za proizvoljan niz u toku izvršavanja programa. Osobina niza da mu se svi elementi nalaze jedan za drugim i da je ime promjenljive koja označava niz ustvari pokazivač na prvi element omogućava vrlo laku alokaciju memorije za nizove.

Dovoljno je definisati pokazivač određenog tipa i alocirati onoliko memorije koliko je potrebno Iza Ismještanje Icijelog Iniza. IU Iopštem Islučaju, Ialokacija Imemorije Iza Iniz Iod n elemenata niza proizvoljnog tipa izgleda

```
TIP* niz = (TIP*) malloc(sizeof(TIP)*n);
```

pri čemu TIP može biti bilo koji od osnovnih ili definisanih tipova.

Sa Iovako Idefinisanim Inizom Ise Iradi Ikao Ii Isa I"običnim" Inizovima I- Imoguće Ije indeksiranje članova niza pri čemu prvi element niza ima indeks nula.

Primjer 26. Napisati program koji sa tastature čita proizvoljan broj `n`, zatim definiše niz od `n` cijelih, učitava elemente niza i pronalazi im aritmetičku sredinu.

Rješenje

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

int main()
{
    int n, i;
    scanf("%d", &n);
    int *niz = (int*) malloc(sizeof(int)*n);
    for(i=0; i<n; i++)
        scanf("%d", &niz[i]);

    double zbir=0.0;
    for(i=0; i<n; i++)
        zbir += niz[i];

    printf("%3.2f", zbir/n);
```

```
    free(niz);
    return 0;
}
```

Članovima niza je moguće pristupati i drugačiji način - preko operatora +. Ukoliko imamo neku promjenljivu `p` koja je pokazivačkog tipa i na nju primjenimo operator + sa argumentom `n`, Ikao Irezultat Idobićemo Imemorijsku Ilokaciju Ikoja Ise Inalazi Ina (`n*veličina_tipa`) mjesta od mjesta na koje pokazuje pokazivač `p`. Prostije, ako imamo pokazivač koji pokazuje na neki niz, elementi niza se nalaze na memorijskim lokacijama
`niz+1, niz+2, ...`
a vrijednosti elemenata niza na lokacijama
`*(niz+1), *(niz+2), ...`

Primjer 27. Napisati program koji sa tastature čita proizvoljan broj `n`, zatim definiše niz od `n` realnih brojeva, učitava i štampa elemente niza pristupajući im preko njihovih adresa.

Rješenje

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

int main()
{
    int n, i;
    scanf("%d", &n);
    float *niz = (float*) malloc(sizeof(float)*n);
    for(i=0; i<n; i++)
        scanf("%f", niz+i);

    for(i=0; i<n; i++)
        printf("%.2f ", *(niz+i));

    free(niz);
    return 0;
}
```

Pokazivači kao argumenti funkcija

Do sada je bilo govora da funkcija `scanf` za argument zahtjeva adresu promjenljive, a ne njeno ime (odnosno vrijednost), dok recimo funkcija `printf` zahtjeva ime (vrijednost) a ne adresu. Koja je razlika?

Funkciji se argumenti mogu proslijediti na dva načina - po vrijednosti i po adresi.

U prvom slučaju se kao argument Ifunkcije zadaje ime promjenljive a funkciji se prosljeđuje vrijednost te promjenljive. Funkcija radi sa vrijednosću promjenljive i ne postoji način da izmjeni stvarnu vrijednost promjenljive. Sljedeći kod

```
int x = 5;
f(x);
```

se tumači kao:

- promjenljiva `x` ima vrijednost `5`

- funkcija f se poziva sa argumentom 5, tj. f(5) i daje rezultat.

Pri prenosu adrumenata po adresi funkciji se prosljeđuje adresa promjenljive i funkcija barata Isa Iadresom Ina Ikojoj Ise Inalazi Ipromjenljiva. IU Itom Islučaju Istvarna Ivrijednost promjenljive se može promjeniti nakon izvršavanja funkcije.

Sljedeći primjer to najbolje ilustruje:

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

void fv(int x)
{
    x++;
}

void fa(int* x)
{
    (*x)++;
}

int main()
{
    int x = 5;

    fv(x);
    printf("%d\n", x);

    fa(&x);
    printf("%d\n", x);

    return 0;
}
```

Funkcija fv za argument dobija vrijednost promjenljive x, tj. broj 5 i ona ustvari samo tu vrijednost povećava za jedan dok vrijednost promjenljive x ostaje 5 i nakon izvršavanja funkcije fv. Funkcija fa za argument dobija adresu promjenljive x i komandom (*x)++ se povećava za jedan vrijednost koja se nalazi na lokaciji x, tj. mijenja se stvarna vrijednost promjenljive x.

Poziv prve funkcije ne mjenja vrijednost promjenljive x, a poziv druge je mijenja. Rezultat izvršavanja ovog programa je

```
5
6
```

Ova osobina se naziva bočni efekat ili side-effect jer omogućava mjenjanje vrijednosti promjenljivih kroz pozivanje funkcija. Ova osobina omogućava da funkcija vrati više od jedne Ivrijednosti. IPotrebno Ije Iu Iglavnom Iprogramu Idefinisati Ipromjenljive, Iproslijediti funkciji Injihove Iadrese Ii Inakon Iizvršavanja Iprograma Iu Itim Ipromjenljivima Idobijamo određene rezultate.

Primjer 27. Napisati funkciju koja za dati niz nalazi minimum, maksimum i aritmetičku sredinu.

Rješenje

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

void mmas(int* niz, int d, int* min, int* max, float* as)
{
    int i, suma = 0;
    int mi=niz[0], ma=niz[0];
    for(i=0; i<d; i++)
    {
        if(mi > niz[i])
            mi = niz[i];
        if(ma < niz[i])
            ma = niz[i];
        suma += niz[i];
    }
    *as = (float) suma / d;
    *min = mi;
    *max = ma;
}

int main()
{
    int niz[5] = {4, 2, 8, 6, 1};
    int min, max;
    float as;

    mmas(niz, 5, &min, &max, &as);

    printf("Minimalni elemenat: %d\n", min);
    printf("Maksimalni elemenat: %d\n", max);
    printf("Aritmeticka sredina: %f\n", as);

    return 0;
}
```

Kada su opisivani nizovi kao argumenti funkcija rečeno je da kada je niz argument funkcije, vrijednost elemenata niza se može promjeniti u funkciji. Sada je jasno i zašto - ime niza Ije, Iustvari, Ipokazivač Ina Iprvi Ielemenat Iniza, Ipa Ije Iargument Iustvari Iadresa Iprvog elementa a to omogućava i mijenjanje vrijednosti niza.

Strukture

Struktura je skup jedne ili više promjenljivih koje su grupisane radi lakše manipulacije sa njima. Sintaksa definisanja strukture je

```
struct ime_strukture
{
     tip1 ime1;
     tip2 ime2;
     ...
};
```

Definicijom strukture se, ustvari, definiše tip imena `struct ime_strukture` i u programu se mogu praviti promjenljive ovoga tipa. Svaka promjenljiva ovog tipa se sastoji od skupa Ipromjenljivih Ikoje Isu definisane Iu strukturi. Tim Ipromjenljivim Ise pristupa Ipreko operatora tačka (`.`) i sa njima se radi kao sa običnim promjenljivim.

Primjer 28. Napisati strukturu za čuvanje podataka o tačkama u ravni, funkciju za štampanje tačke i funkciju za izračunavanje udaljenosti između dvije tačke.

Rješenje

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

struct tacka
{
     float x;
     float y;
};

double udaljenost(struct tacka t1, struct tacka t2)
{
     return sqrt((t1.x-t2.x)*(t1.x-t2.x) + (t1.y-t2.y)*(t1.y-
t2.y));
}

void odstampaj(struct tacka t)
{
     printf("(%.2f, %.2f)\n", t.x, t.y);
}

int main(int argc, char *argv[])
{
     struct tacka t1, t2;

     scanf("%f", &t1.x);
     scanf("%f", &t1.y);
     scanf("%f", &t2.x);
     scanf("%f", &t2.y);

     odstampaj(t1);
     odstampaj(t2);

     printf("d = %.2f\n", udaljenost(t1, t2));

     system("PAUSE");
     return 0;
}
```

U praksi se najčešće definiše tip koji predstavlja tip strukture, da se ne bi svaki put morala navoditi ključna riječ `struct`. Tako bi npr. u prošlom primjeru mogli definisati tip `Tacka` pomoću ključne riječi `typedef`:

```
typedef struct tacka Tacka;
```

i onda bi koristili tip `Tacka` umjesto `struct tacka`. Korišćenjem ovog novog naziva tipa možemo definisati dvije promjenljive tipa tačka na sljedeći način:

```
Tacka t1, t2;
```

Još kraći zapis je da se pri definiciji strukture definiše i tip, u prošlom primjeru bi to izgledalo

```
typedef struct tacka
{
     float x;
     float y;
} Tacka;
```

Pokazivači na strukture

Kao i na svaki drugi tip, možemo definisati pokazivač i na strukturu. Sintaksa je identična kao sa ostalim tipovima, jedina razlika je u načinu pristupanja članovima strukture. Kod Iobičnih Ipromjenljivih Ioperator Ipristupa Ičlanovima Ije Itačka I(`.`) Ia Ikod Ipokazivača članovima Ise Ipristupa Ipreko Ioperatora Iminus Iveće I(`->`). IKorišćenje Istrukture Iiz Iprošlog primjera preko pokazivača izgledalo bi

```
Tacka *p = (Tacka*) malloc(sizeof(Tacka));
p->x = 10.0;
p->y = 5.0;
odstampaj(*p);
free(p);
```

Zadaci za vježbu

Algoritmi sortiranja i pretrage

Binarna pretraga

Primjer 29. Napisati funkciju koja pretražuje dati neopadajući niz pomoću algoritma binarne pretrage

Rješenje

```
int BinPretraga(int* niz, int d, int x)
{  // Iterativna binarna pretraga
    0int lijevi = 0;
     int desni = d-1;
     while(lijevi <= desni)
     {
          int m = (lijevi+desni) / 2;
          if(x == niz[m])
               return m;
          else if(x < niz[m])
               desni = m-1;
          else
               lijevi = m+1;
     }
     return -1;
}

int BinPretragaR(int* niz, int lijevi, int desni, int x)
{ // Rekurzivna binarna pretraga
     if(lijevi >= desni)
         return -1;
     int m = (lijevi + desni) / 2;
     if(x < niz[m])
         return BinPretragaR(niz, lijevi, m, x);
     else if(x > niz[m])
         return BinPretragaR(niz, m, desni, x);
     else
         return m;
}
```

Bubble sort

Primjer 30. Napisati funkciju koja dati niz sortira pomoću algoritma Bubble sort

Rješenje

```
void SortBubble(int* niz, int duzina)
{
     int i, j;
     for(i=0; i<duzina; i++)
          for(j=i; j<duzina; j++)
               if(niz[i] > niz[j])
               {
                    int x = niz[i];
                    niz[i] = niz[j];
                    niz[j] = x;
               }
```

```
}
```

Sortiranje umetanjem

Primjer 31. Napisati funkciju koja dati niz sortira pomoću algoritma sortiranja umetanjem.

Rješenje

```
void SortUmetanje(int* a, int d)
{ // Osnovna varijanta sortiranja umetanjem
      int i, j;
      for(i=1; i<d; i++)
      {
          int x = a[i];
          j = i-1;
          while(j>0 && x > a[j])
          {
              0a[j+1] = a[j];
               j = j-1;
          }
          a[j+1] = x;
     }
}

void SortBinUmetanje(int* a, int d)
{ // Poboljsana varijanta, pri cemu se binarnom pretraga
pronalazi mjesto na koje se umece elemenat
     int i, j;
     for(i=1; i<d; i++)
     {
          int x = a[i];
          int lijevi = 0;
          int desni = i-1;
          while(lijevi <= desni)
          {
               int m = (lijevi+desni) / 2;
               if(x < a[m])
                  desni = m-1;
               else
                  lijevi = m+1;  0
          }
          for(j=i-1; j>=lijevi; j--)
             a[j+1] = a[j];
           a[lijevi] = x;
     }
}
```

Sortiranje izborom

Primjer 32. Napisati funkciju koja dati niz sortira pomoću algoritma sortiranja izborom.

Rješenje

```
void SortIzborom(int* a, int d)
{ // Varijanta 1
     int i, j, indexNajmanjeg;
     for(i=0; i<d-1; i++)
```

```
     {
          indexNajmanjeg = i;
          int najmanji = a[i];
          for(j=i+1; j<d; j++)
          {

              0if(a[j] < najmanji)
              0{
                    indexNajmanjeg = j;
                   0najmanji = a[j];
               }
          }
          a[indexNajmanjeg] = a[i];
          a[i] = najmanji;
     }
}

void SortIzborom1(int* a, int d)
{ // Varijanta 2
     int i, j, indexNajmanjeg;
     for(i=0; i<d-1; i++)
     {
          indexNajmanjeg = i;
          for(j=i+1; j<d; j++)
              0if(a[j] < a[indexNajmanjeg])
                    indexNajmanjeg = j;
          int najmanji = a[indexNajmanjeg];
          a[indexNajmanjeg] = a[i];
          a[i] = najmanji;
     }
}
```

QuickSort

Primjer 33. Napisati funkciju koja dati niz sortira pomoću algoritma QuickSort.

Rješenje

```
void swap(int v[], int i, int j)
{
     int temp;
     temp = v[i];
     v[i] = v[j];
     v[j] = temp;
}

void QuickSort(int v[], int lijevi, int desni)
{
   int i, zadnji;

   if (lijevi >= desni)
       return;
   swap(v, lijevi, (lijevi + desni)/2);
   zadnji = lijevi;
```

```
    for (i = lijevi + 1; i <= desni; i++)
        if (v[i] < v[lijevi])
            swap(v, ++zadnji, i);
    swap(v, lijevi, zadnji);
    QuickSort(v, lijevi, zadnji-1);
    QuickSort(v, zadnji+1, desni);
}
```

MergeSort

Primjer 34. Napisati funkciju koja dati niz sortira pomoću algoritma MergeSort.

Rješenje

```
void Merge(int* a, int d1, int d2)
{ // Funkcija koja spaja dva podniza tako da se dobije
sortirani niz
     int* novi = (int*) malloc(sizeof(int) * (d1+d2));
     int i=0, j1=0, j2=0;

     while(j1 < d1 && j2 < d2)
          novi[i++] = (a[j1] <= a[d1+j2]) ? a[j1++] :
a[d1+j2++];
     while(j1 < d1)
          novi[i++] = a[j1++];
     while(j2 < d2)
          novi[i++] = a[d1+j2++];

     for(i=0; i<d1+d2; i++)
          a[i] = novi[i];

     free(novi);
}

void SortMerge(int* a, int d)
{ // Funkcija koja sortira niz
     if(d>1)
     {
          int d1 = d/2;
          int d2 = d - d1;
          SortMerge(a, d1);
          SortMerge(a+d1, d2);
          Merge(a, d1, d2);
     }
}
```

U svakom algoritmu sortiranja niz koji se sortira može biti proizvoljan: niz cijelih brojeva, niz realnih brojeva, niz karaktera ili pak niz promjenljivih nekog korisnički definisanog tipa. Jedini uslov je da postoji kriterijum po kojem se elementi niza mogu upoređivati.

Primjer 35. Podaci o vektoru se sastoje od koordinata x, y i z. Napisati strukturu za čuvanje podataka o vektoru. Sa tastature učitati broj n a zatim učitati n vektora i sortirati ih po intenzitetu korišćenjem sortiranja umetanjem.

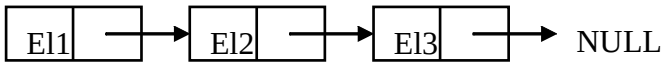
```
#include <stdio.h>
#include <stdlib.h>

typedef struct v
{ float x;
  float y;
  float z;
}vektor;

void ucitaj(vektor *niz,int d)
{
    int i;


     for(i=0;i<d; i++)
     {
         scanf("%f", &niz[i].x);
         scanf("%f", &niz[i].y);
         scanf("%f", &niz[i].z);
     }
}

void ispisi(vektor *niz,int d)
{
     int i;
     for(i=0;i<d; i++)
         printf("(%.2f, %.2f, %.2f) ", niz[i].x, niz[i].y,
niz[i].z);
}

float intenzitet(vektor v)
{
     return (float) sqrt(v.x*v.x+v.y*v.y+v.z*v.z);
}

void sort(vektor* a, int d)
{
      int i, j;
      for(i=1; i<d; i++)
      {
          vektor x = a[i];
          j = i-1;
          while(j>0 && intenzitet(x) < intenzitet(a[j]))
          {
               a[j+1] = a[j];
               j = j-1;
          }
          a[j+1] = x;
```

```
     }
}

int main()
{
     int n;
    0scanf("%d", &n);
    0vektor *niz=(vektor*)malloc(sizeof (vektor)*n);
    0ucitaj(niz,n);

    0sort(niz,n);

    0ispisi(niz,n);

    0free(niz);
     return 0;
}
```

Stek i red

Stek je struktura podataka koja predstavlja kontejner promjenljivih istog tipa. Nad stekom su definisane dvije operacije: push (dodaj) i pop (ukloni). Osobina steka je da se posljednje dodat elemenat prvi uklanja sa steka. Zato se stek naziva LIFO (last-in-first-out) struktura podataka.

Ovdje Iće Ibiti Iprikazana Idva Inačina Iimplementacije Isteka I- Istatički, Ipreko Iniza unaprijed definisane maksimalne dužine i dinamički - preko pokazivača na strukturu koja predstavlja jedan elemenat steka. Stek se sastoji od elemenata koji međusobno čine povezanu listu, kao na slici.



Svaki član liste se sastoji od nekog podatka i pokazivača na sljedeći član liste. Posljednji član pokazuje na NULL

Primjer 36. Napisati program koji omogućava rad sa stekom cijelih brojeva.

Rješenje - statičko, preko niza

```
#include <stdio.h>
#include <stdlib.h>
#define MAXVALUE 100

int stek[MAXVALUE];
int brElem = 0;

void Push(int v)
{
    if(brElem == MAXVALUE)
    {
        printf("Stek je pun\n");
        return;
    }
    stek[brElem++] = v;
    printf("Dodaje se element %d\n", v);
}

void Pop()
{
    if(brElem == 0)
    {
        printf("Stek je prazan\n");
        return;
    }
    printf("Uklanja se element %d\n", stek[brElem--]);
}

void Print()
{
    int i;
```

```
    printf("Stek: ");
    for(i=0; i<brElem; i++)
        printf("%d ", stek[i]);
    printf("\n");
}

int main()
{
    Push(5);
    Push(10);
    Push(0);
    Print();
    Pop();
    Print();
    return 0;
}
```

Rješenje - dinamičko, preko pokazivača na strukturu

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

typedef struct el
{
     int v;
     struct el* sljedeci;
} Elem;

Elem* stek;

void Push(int v)
{
     Elem* novi = (Elem*) malloc(sizeof(Elem));
     novi->v = v;
     novi->sljedeci = stek;
     stek = novi;
     printf("Dodaje se element %d\n", v);
}

void Pop()
{
     if(stek == NULL)
     {
          printf("Stek je prazan\n");
          return;
     }
     int v = stek->v;
     Elem* prvi = stek;
     stek = stek->sljedeci;
     free(prvi);
     printf("Uklanja se element %d\n", v);
}
```

```
void Print()
{
     Elem* t = stek;
     printf("Stek: ");
     while(t != NULL)
     {
         0printf("%d ", t->v);
          t = t->sljedeci;
     }
     putchar('\n');
}

int main()
{
     Push(5);
     Push(7);
     Push(6);
     Print();
     Pop();
     Print();
     Push(1);
     Print();
     return 0;
}
```

Primjer 37. Napisati program koji omogućava rad sa redom cijelih brojeva.

Red je slična struktura steku, s tim da se iz reda uklanja onaj element koji je prvi došao u red, zato se red naziva FIFO (first-in-first-out) struktura. Red će se implementirati na dva načina kao i stek.

Rješenje - statičko, preko niza

```
#include <stdio.h>
#include <stdlib.h>
#define MAXVALUE 100

int red[MAXVALUE];
int brElem = 0;

int Push(int v)
{
    if(brElem == MAXVALUE)
    {
        printf("Red je pun\n");
        return;
    }
    red[brElem] = v;
    brElem++;
    printf("Dodaje se element %d\n", v);
}
```

```
void Pop()
{
    if(brElem == 0)
    {
        printf("Red je prazan\n");
        return -1;
    }
    int v = red[0];
    int i;
    for(i=0; i<brElem; i++)
        red[i] =red[i+1];
    brElem--;
    printf("Uklanja se element %d\n", v);
}

void Print()
{
    int i;
    printf("Red: ");
    for(i=0; i<brElem; i++)
        printf("%d ", red[i]);
    printf("\n");
}

int main(int argc, char *argv[])
{
    Push(3);
    Push(6);
    Push(4);
    Print();
    Pop();
    Print();
    return 0;
}
```

Rješenje - dinamičko, preko pokazivača na strukturu

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

typedef struct el
{
    int v;
    struct el* sljedeci;
} Elem;

Elem* red;

void Push(int v)
{
    Elem* novi = (Elem*) malloc(sizeof(Elem));
    novi->v = v;
    novi->sljedeci = red;
```

```
    red = novi;
    printf("Dodaje se element %d\n", v);
}

void Pop()
{
    if(red == NULL)
        printf("Red je prazan\n");
    else if(red->sljedeci == NULL)
    {
        int v = red->v;
        free(red->sljedeci);
        red = NULL;
    }
    else
    {
        Elem *e = red;
        while(e->sljedeci->sljedeci != NULL)
            e = e->sljedeci;
        int v = e->sljedeci->v;
        free(e->sljedeci);
        e->sljedeci = NULL;
        printf("Uklanja se element %d\n", v);
    }
}

void Print()
{
    Elem* t = red;
    printf("Red: ");
    while(t != NULL)
    {
        printf("%d ", t->v);
        t = t->sljedeci;
    }
    putchar('\n');
}

int main()
{
    Push(5);
    Push(7);
    Push(6);
    Print();
    Pop();
    Print();

    return 0;
}
```

Jednostruko povezane liste

mplementacija Isteka Ii Ireda Ipreko Ipokazivača Ina Istrukture Ipredstavljaju Ijedno korišćenje jednostruko povezane liste. Jednostruko povezana lista može imati mnogo više operacija osim dodavanja elementa i uklanjanja elementa s vrha (kod steka), odnosno sa dna (kod reda). U jednostruko povezanoj listi se može uklanjati elemenat s proizvoljne pozicije u listi, može se vršiti pretraga liste, vršiti ubacivanje elemenata tako da lista bude sortirana po nekom kriterijumu itd. Ovdje će biti prikazane implementacije nekih od ovih operacija.

Primjer 38. Napisati program koji omogućava rad sa jednostruko povezanim listama čiji su elementi parovi (kljuc, vrijednost). Implementirati funkcije za dodavanje elementa u listu (na proizvoljno mjesto), pretragu liste po ključu i štampu elemenata liste.

Rješenje

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

typedef struct el
{
     int kljuc;
     char vrijednost[10];
     struct el* sljedeci;
} Elem;

Elem* lista = NULL;

void dodaj(Elem* e, int k, char* v)
{
    Elem *novi = (Elem*) malloc(sizeof(Elem));
    novi->kljuc = k;
    strcpy(novi->vrijednost, v);
    novi->sljedeci = lista;
    lista = novi;
}

void pronadji(Elem* e, int k)
{
    if(e == NULL)
    {
        printf("Ne postoji element sa kljucem %d\n", k);
        return;
    }
    else if(k == e->kljuc)
    {
        printf("Vrijednost s kljucem %d: %s\n", k,
e->vrijednost);
        return;
    }
    else
```

```
        pronadji(e->sljedeci, k);
}

void odstampaj(Elem* e)
{
    if(e == NULL)
    {
        printf("---Kraj liste---\n");
        return;
    }
    printf("%d %s\n", e->kljuc, e->vrijednost);
    odstampaj(e->sljedeci);
}

void oslobodi_memoriju(Elem* e)
{
    if(e == NULL)
        return;
    Elem *n = e->sljedeci;
    free(e);
    oslobodi_memoriju(n);
}

int main()
{
    dodaj(lista, 3, "tri");
    dodaj(lista, 2, "dva");
    dodaj(lista, 6, "sest");
    dodaj(lista, 4, "cetiri");
    odstampaj(lista);

    pronadji(lista, 6);
    pronadji(lista, 12);

    oslobodi_memoriju(lista);

    return 0;
}
```

Primjer 39. Napisati program koji omogućava rad sa jednostruko povezanim listama čiji su elementi parovi (kljuc, string). Implementirati funkcije za dodavanje elementa u listu tako da lista bude sortirana po ključu i funkciju za izbacivanje elementa iz liste sa zadatim ključem kao i štampu elemenata liste.

Rješenje

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

typedef struct el
{
```

```
     int kljuc;
     char string[10];
     struct el* sljedeci;
} Elem;

Elem* lista = NULL;

void dodaj(Elem* e, int k, char* s)
{
     if(lista == NULL)
     { // lista je prazna
          Elem* novi = (Elem*) malloc(sizeof(Elem));
          novi->kljuc = k;
          strcpy(novi->string, s);
          novi->sljedeci = NULL;
          lista = novi;
          return;
     }
     else if(k < lista->kljuc)
     { // element koji se dodaje ide na prvo mjesto
          Elem* novi = (Elem*) malloc(sizeof(Elem));
          novi->kljuc = k;
          strcpy(novi->string, s);
          lista = novi;
          novi->sljedeci = e;
          return;
     }
     else if(e->sljedeci == NULL)
     { // dodaje se na posljednje mjesto u listi
          Elem* novi = (Elem*) malloc(sizeof(Elem));
          novi->kljuc = k;
          strcpy(novi->string, s);
          novi->sljedeci = NULL;
          e->sljedeci = novi;
          return;
     }
     else if(k < e->sljedeci->kljuc)
     {
          Elem* novi = (Elem*) malloc(sizeof(Elem));
          novi->kljuc = k;
          strcpy(novi->string, s);
          novi->sljedeci = e->sljedeci;
          e->sljedeci = novi;
          return;
     }
    else
        dodaj(e->sljedeci, k, s);
}

void izbaci(Elem* e, Elem* p, int k)
{
    if(e == NULL)
```

```
    {
        printf("Lista je prazna!\n");
        return;
    }
    else if(k == lista->kljuc)
    {
        Elem* e = lista;
        lista = lista->sljedeci;
        free(e);
        return;
    }
    else if(k == e->kljuc)
    {
        p->sljedeci = e->sljedeci;
        free(e);
        return;
    }
    else if(e == NULL)
    {
        printf("Ne postoji element sa zadatim kljucem!\n");
        return;
    }
    else
        izbaci(e->sljedeci, e, k);
}

void odstampaj(Elem* e)
{
    if(e == NULL)
    {
        printf("---Kraj liste---\n");
        return;
    }
    printf("%d %s\n", e->kljuc, e->string);
    odstampaj(e->sljedeci);
}

void oslobodi_memoriju(Elem* e)
{
    if(e == NULL)
        return;
    Elem *n = e->sljedeci;
    free(e);
    oslobodi_memoriju(n);
}

int main()
{
    dodaj(lista, 3, "tri");
    dodaj(lista, 2, "dva");
    dodaj(lista, 6, "sest");
    dodaj(lista, 4, "cetiri");
```

```
    dodaj(lista, 1, "jedan");
    dodaj(lista, 8, "osam");
    dodaj(lista, 9, "devet");
    dodaj(lista, 7, "sedam");
    dodaj(lista, 5, "pet");
    odstampaj(lista);
    izbaci(lista, NULL, 6);
    odstampaj(lista);
    izbaci(lista, NULL, 1);
    odstampaj(lista);
    izbaci(lista, NULL, 9);
    odstampaj(lista);
    oslobodi_memoriju(lista);

    return 0;
}
```

Grafovi

Graf se definiše kao uređeni par G=(V,E), gde je V konačan, neprazan skup čvorova, a E je skup grana (veza između čvorova).

Osnovni pomjovi vezani za graf su:

Stepen čvora grafa je broj grana koje uparuju taj čvor.

Dve grane su susjedne ako imaju isti čvor.

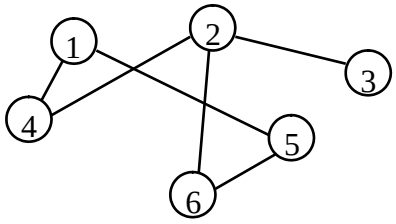
Grana koja spaja čvor sa samim sobom naziva se petljom.

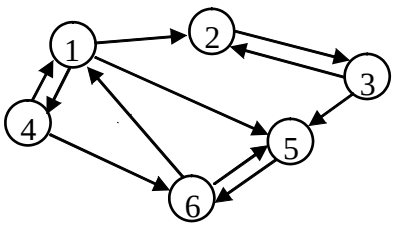
Graf koji nema nijednu petlju nazivaju se prostim grafom.

Graf je regularan ako su svi čvorovi istog stepena.

Graf je usmjeren ukoliko su mu grane usmjerene, odnosno ukoliko se zna koji čvor je početak a koji čvor je kraj grane.

Svaki graf se može grafički prikazati kao crtež koji se sastoji od tačaka i linija koje predstavljaju čvorove i grane grafa.
Na slici ... je prikazan jedan neusmjeren graf a na slici ... usmjeren graf.





Postoje dva načina da se predstavi graf - preko matrica povezanosti i preko lista povezanosti.

Matrica povezanosti za graf G koji ima *n* čvorova je matrica dimenzija *n*x*n* pri čemu je element $a_{kj} = 1$ akko postoji grana između čvorova c_k i c_j, a $a_{kj}=0$ ako ne postoji grana između tih čvorova. Jasno je da su matrice povezanosti neusmjerenih grafova simetrične.

Matrice povezanosti grafova sa slika ... i ... su

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

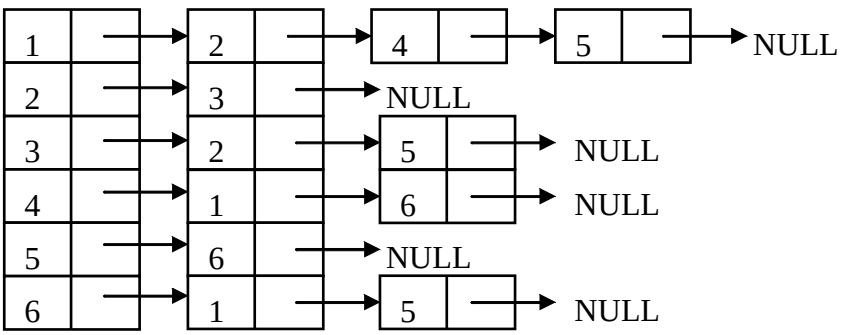
Primjer 40. Implementirati graf preko matrice povezanosti.

Rješenje

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

int **matp;

void OdstampajMatricu(int **m, int stepen)
{
     int i, j;
     for(i=0; i<stepen; i++)
     {
          for(j=0; j<stepen; j++)
               printf("%d\t", m[i][j]);
          printf("\n");
     }
}

void UcitajMatricu(int stepen)
{
     int i, j, pov;
     matp = (int **) malloc(stepen*sizeof(int*));
     for(i=0; i<stepen; i++)
         matp[i] = (int*) malloc(stepen*sizeof(int));


     for(i=0; i<stepen; i++)
          for(j=0; j<stepen; j++)
               matp[i][j] = 0;

     for(i=0; i<stepen; i++)
     {
          printf("Unesite 0cvorove 0koji 0su 0povezani 0sa 0%d.
cvorom:\n", i);
         0for(; ;)
         0{
         0     scanf("%d", &pov);
         0     if(pov < 0 || pov >= stepen)
                    break;
               matp[i][pov] = 1;
               matp[pov][i] = 1;
         0}
```

```
     }
}

int main(int argc, char *argv[])
{
     int stepen, i;
     printf("Unesite broj cvorova grafa: ");
     scanf("%d", &stepen);

     UcitajMatricu(stepen);
     OdstampajMatricu(matp, stepen);

     for(i=0; i<stepen; i++)
          free(matp[i]);
     free(matp);

     return 0;
}
```

Drugi način predstavljanja grafa je preko liste povezanosti. U takvoj implementaciji se pravi lista čvorova pri čemo svaki čvor pokazuje na čvorove sa kojima je povezan. Tako npr, lista povezanosti za graf na slici ... izgleda



Primjer 41. Implementirati graf preko liste povezanosti.

Rješenje

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

typedef struct c
{
    int v;
     struct c* s;
} Cvor;

Cvor* UcitajGraf(int brCvorova)
{
     Cvor *graf;
     int i, j, pov;
    0for(i=0; i<brCvorova; i++)
          graf = (Cvor*) malloc(brCvorova*sizeof(Cvor));
```

```
     for(i=0; i<brCvorova; i++)
     {
          graf[i].v = i;
          graf[i].s = NULL;
          Cvor* p = &graf[i];
         printf("Unesite 0cvorove 0koji 0su 0povezani 0sa 0%d.
cvorom:\n", i);
          for( ; ;)
          {
         0     scanf("%d", &pov);
         0     if(pov < 0 || pov >= brCvorova)
                    break;
               Cvor* c = (Cvor*) malloc(sizeof(Cvor));
               c->v = pov;
               c->s = NULL;
               p->s = c;
               p = c;
          }
     }
     return graf;
}

void OdstampajGraf(Cvor* graf, int brCvorova)
{
     int i, j;
     for(i=0; i<brCvorova; i++)
     {
          Cvor* c = &graf[i];
          printf("%d->", graf[i].v);
          while(c->s != NULL)
          {
               c = c->s;
               printf("%d, ", c->v);
          }
          printf("\n");
      }
}

int main(int argc, char *argv[])
{
     Cvor *graf;
     int brCvorova, i;
     printf("Unesite broj cvorova grafa: ");
     scanf("%d", &brCvorova);
     graf = UcitajGraf(brCvorova);
     OdstampajGraf(graf, brCvorova);
     return 0;
}
```

Stabla

Zadaci za vježbu

1. Podaci o procesu se sastoje od naziva procesa i vremena izvršavanja. Implementirati stek koji čuva podatke o procesima.
2. Napisati program koji učitava jednostruko povezanu listu i iz nje uklanja najveći element.
3. Napisati program koji učitava jednostruko povezanu listu i pronalazi aritmetičku sredinu pozitivnih elemenata te liste.
4. Napisati program koji učitava jednostruko povezanu listu i od pozitivnih elemenata te liste pravi novu listu.
5. Napisati program koji učitava jednostruko povezanu listu i ispisuje njene elemente u obrnutom redoslijedu.
6. Napisati program koji učitava dvije jednostruko povezane liste i provjerava da li su te dvije date liste identične.
7. Napisati program koji učitava i štampa jednostruko povezanu listu. Lista ne smije sadržavati jednake elemente.
8. Napisati program koji učitava jednostruko povezanu listu i štampa elemente koji se nalaze na neparnim mjestima.
9. Napisati program koji učitava jednostruko povezanu listu i od nje pravi dvije jednostruko povezane liste tako da se u prvoj nalazi polovina elemenata a druga polovina u drugoj listi.
10. Napisati program koji učitava i štampa cikličnu jednostruko povezanu listu.
11. Napisati program koji učitava dva grafa i ispituje da li su identični.
12. Napisati program koji učitava graf i ispituje da li je usmjeren.
13. Napisati program koji učitava graf i ispituje da
14. Napisati program koji učitava uređeno binarno stabla i izračunava koliko njegovih čvorova ima tačno dva podčvora.

15. Dodatak A

Pregled važnijih funkcija biblioteke math.h

Ime funkcije	Opis
`acos`	nverzni kosinus
`acosh`	nverzni hiperbolički kosinus
`asin`	nverzni sinus
`asinh`	nverzni hiperbolički sinus
`atan`	nverzni tangens
`atanh`	nverzni hiperbolički tangens
`cbrt`	Kubni korjen
`cos`	Kosinus
`cosh`	Hiperbolički kosinus
`exp`	Eksponencijalna funkcija
`exp2(x)`	2^x
`fabs`	Apsolutna vrijednost realnog broja
`fmax(x,y)`	Vraća veću vrijednost od x i y
`fmin(x,y)`	Vraća manju vrijednost od x i y
`hypot(x,y)`	Hipotenuza - sqrt(x2 + y2)
`log`	Prirodni logaritam
`log2`	Logaritam po bazi 2
`log10`	Lograitam po bazi 10
`pow(x,y)`	x^y
`round`	Zaokruživanje realnog broja
`sin`	Sinus
`sinh`	Hiperbolički sinus
`sqrt`	Kvadratni korjen
`tan`	Tangens
`tanh`	Hiperbolički tangens

Pregled važnijih funkcija biblioteke string.h

Pregled važnijih funkcija biblioteke ctype.h