Програмирање
Материјал за смене IV и V - Први блок

Београд, 2019.

Садржај

1

Увод

Шта је C? Најједноставнији одговор би био веома користан програмски језик развијен у раним 70-тим годинама у *Bell* Лабораторијама (као саставни део UNIX оперативног система, аутора Ken Thompson, Dennis Ritchie и других). Данас популарни C - засновани језици су:

- C, C++, Java, C#, Perl

Познавање C-а даје бољи увид у наведене нове програмске језике и добру основу за учење истих. Ипак, када је меморија ограничена за развој софтвера C има кључну улогу ради своје једноставности. Одлике овог језика су:

- C је програмски језик ниског новоа, погодан за системско програмирање јер омогућује приступ концептима машинског нивоа (бајтовима и адресама), које остали програмски језици прикривају. Омогућава операције које одговарају уграђеним инструкцијама рачунара па програми могу бити брзи. Како апликациони програми зависе од управљања улазно-/излазним подацима и другим сервисима, оперативни систем не сме бити спор.

- C је "мали" језик, у смислу да обезбеђује ограничен скуп компоненти за разлику од других програмских језика. Да би остао "мали" језик, C се ослања много на библиотеку стандардних функција. C библиотеке су велике колекције функција које долазе у саставу сваког компајлера.

- C је "попустљив" језик, у смислу да претпоставља да знате шта радите, дозвољава већи степен ширине него други програмски језици. Такође не врши детаљну проверу грешке као други програмски језици.

Предности и снага овог програмског језика су следећи:

- Ефиксаност. Подразумевано је да програми писани у овом програмском језику треба да буду брзи и не захтевају велику количину меморије.
- Портабилност.
- Моћан је програмски језик јер подржава велику колекција типова података и оператора. У само неколико линија кода је могуће постићи много.
- Флексибилност. Иако је првобитно настао sa системско програмирање, није ограничен само за те сврхе. Користи се за све врсте апликација, од системских уграђених до комерцијалних обрада података. Неке операције које нису дозвољене у другим програмским језицима су у C-у дозвољене. Нпр. C дозвољава да карактер буде додат целобројној вредности. Ово може олакшати програмирање али може и изазвати грешке које се лако провлаче (тешко прате).
- Стандардна библиотека. Једна од највећих снага овог програмског језика је његова стандардна библиотека, која садржи хиљаде функција за улаз/излаз, манипулисање стринговима, алокацију меморије, итд.
- Интеграција са UNIX-ом. C је практично моћан у комбинацији UNIX-ом, уствари већина UNIX-алата подразумава познавање C-а.

Ипак треба нагласити да постоје и ограничења и слабости програмског језика C, које проистичу из истог домена из кога су и предности овог језика, а то је његова "близина" машинском језику:

- Склон грешкама. Цена флексибилности је склоност ка грешкама, које у већини других програмских језика не би биле дозвољене. По том питању је C доста сличан асемблеру где се већина грешака ухвати у фази извршавања програма, а не у фази компајлирања.
- Програми написани у C-у могу бити тешко разумљиви. Иако мали језик има компоненте које нису подржане у другим програмским језицима те их је тешко протумачити без искуства.
- Програми написани у C-у су тешки за модификовање. Модерни програмски језици омогућавају класе и пакете који подржавају логичку поделу великих програма, то овде није случај.

Ефикасно коришћење C-а подразумева избегавање ограничења овог језика и коришћење снаге истог. Усвајање неке конвенције писања кода је најважније. Препоручује се избегавање коришћења нестандардних функција и библиотека осим ако баш није неопходно (стандарди C89 и C99).

1.1 Писање једноставног програма

Уобичајен први пример је: написати програм који на екрану штампа једноставну поруку "Здраво свима!".

```
#include <stdio.h>

int main()
{
   printf("Zdravo svima!\n");
   return 0;
}
```

Прво је неопходно да направимо датотеку са одговарајућим именом (једна реч) и екстензијом *c*. Екстензија говори компајлеру да је у питању C програм, име није битно (образложење на вежбама). Затим је неопходно да се програм преведе у облик који је машини препознатљив. Ова процедура обично укључује 3 корака:

- Препроцесирање. Програм прво иде препроцесору, који ослушкује препроцерске директиве (почињу тарабом). Препроцесор је сличан едитору, он може додати и мењати програм.
- Компајлирање. Модификовани програм сада иде у компајлер, који га трансформише у машинске инструкције (објектни код). Програм још није спреман за извршавање.
- Повезивање. У последњем кораку линкер комбинује објектни програм са додатним кодом неопходним да се програм изврши. Додатни код у овом примеру укључује функцију из библиотеке printf.

Овај процес је обично аутоматизован, препроцесор је обично интегрисан у компајлер па се ови кораци изводе као један. Команде за компајлирање и линковање зависе од компајлера и оперативног система. После ових корака је направљен извршни програм са екстензијом *.out* на UNIX оперативним системима, односно *.exe* на Windows оперативним системима.

1.1.1 Интегрисано развојно окружење

На вежбама користимо интегрисано развојно окружење $Dev-C++$ доступно на локацији http://www.bloodshed.net/dev/. $Dev-C++$ је софтверски пакет

који омогућава мењање, компајлирање, повезивање, извршавање и дебаговање програма у самом окружењу. Компоненте овог окружења су дизајниране тако да раде заједно. На пример, ако компајлер детектује грешку у програму у едитору се маркира одговарајућа линија која садржи грешку. Дозвољено је коришћење других развојиних окружења као и оперативних система. У лабораторијским учионицама је студентима на располагању ОС Windows.

1.1.2 Генерална форма једноставног програма

Једноставан програм има форму

direktive

```
int main(void)
{
```

naredbe

```
}
```

У овом шаблону, као и у осталим примерима италик словима је наглашен текст који треба да се допуни у шаблону. Може се приметити да заграде у примеру означавају почетак и крај главне функције, што је врло слично кључним речима begin и end у другим програмским језицима. Пример илуструје и главне особине овог програмског језика, који се умногоме ослоња на скраћенице и специјалне симболе. Овај једноставан програм се ослања на 3 особине:

- директиве (команде које мењају програм пре компајлирања),
- фунцкије (именовани блокови извршног кода, `main` је само један пример),
- и наредби (команде које се извршавају у тренутку извршавања програма).

1.1.3 Директиве

Пре него што се програм компајлира, прво се мења у фази предпроцесирања. Команде упућене препроцесору се називају директиве. Биће детаљно објашњене касније, за сада треба објасници `#include` директиву.

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

Ова директива наглашава да библиотека `<stdio.h>` треба да буде укључена пре компајлирања. `<stdio.h>` садржи информације о стандардној C I/O библиотеци. C има велики број заглавља као што је `<stdio.h>`, сваки садржи информације о неком делу стандардне библиотеке. Разлог укључивања заглавља `<stdio.h>`

је што C, за разлику од других програмских језика, нема уграђене команде за читање и писање. Уместо уграђених команди, читање и писање је омогућено коришћењем функција из стандардне библиотеке. Директиве увек почињу симболом #, што их разликује од других команди програмског језика. Уобичајено је да се директиве налазе у једном реду, не ставља се ; или било који други симбол на крају директиве.

1.1.4 Функције

Функције су као процедуре или подпрограми у другим програмским језицима, оне су извршни блокови од којих је програм састављен. Уствари у C-у су програми нешто више од колекције функција. Функције спадају у две категорије: оне које сами корисници пишу и оне које су обезбеђене као део C-имплементације (обезбеђене као библиотеке функција и долазе са компајлером). Термин функција долази из математике, где је функција правило за израчунавање вредности када је дат један или више аргумената:

$$f(x) = x + 1 g(y, z) = y^2 - z^2$$

C користи термин функција мало ослабљеније. У C-у се под функцијом подразумева серија наредби које се групишу заједно и именују. Неке функције израчунавају вредност али неке не. Функције које израчунавају (враћају вредност) користе `return` наредбу да би вратиле одговарајућу вредност. На пример, функција која увећава свој аргумент за 1 може имати следећу наредбу:

```
return  x + 1;
```

Иако се програм може састојати од много функција само једна је обавезна за сваки C програм - `main`. Ова функција је специјална, она се позива аутоматски када се извршава програм. Кључан је назив `main` који не сме да се мења. Такође функција `main` има повратну вредност, враћа статус код оперативном систему када програм заврши са радом. Реч `int` пре речи `main` говори да гункција враћа целобројну вредност. Реч `void` унутар заграда говори да `main` функција нема аргумената. Наредба

```
return 0;
```

има два ефекта: она говори `main` функцији да заврши са радом (а тиме и програм завршава са радом), и наглашава да `main` функција враћа 0 по завршетку рада, што се подразумева као исправно извршен програм. Чак иако се изо-

стави ова нараедба, програм ће исправно завршити са радом али ће већина компајлера пријавити упозорење.

1.1.5 Наредбе

Наредбе су команде које се извршавају када се покрене програм на извршавање. Приказани пример користи само две наредбе једна је `return` а друга је позив функције. Позивање функције да одради посао који јој је придружен. Програми писани у C-у захтевају да се свака команда завршава са симболом ; изузетак су здружене наредбе. Симбол ; говори компајлеру где се наредба завршава јер наредба може да се простире у више редова. Директиве су по правилу у једном реду и зато не захтевају симбол за крај наредбе.

1.1.6 Штампање стрингова

`printf` је веома моћна функција. За сада је коришћена само да прикаже стринг (ниску карактера), који се налазе између " ". У приказу стринга се не наводе наводници. Ова функција не прелази аутоматски у следећи ред по испису из тог разлога за наглашавање када треба да се настави у новом реду се користи симбол $\backslash n$.

1.1.7 Коментари

Сваки написани програми би требало документовати нрп. да садржи основне информације шта програм ради, ко је аутор програма, када је програм написан и сл. Ове информације се могу написати у виду коментара.

```
/*************************************************************
* Naziv : primer1.c                                          *
* Svrha: demonstrira stampanje stringa                       *
* Autor: D. Jandrlic                                         *
*************************************************************/

/* Neophodna informacija o standardnoj C I/O biblioteci */
#include <stdio.h>


/* Izvrsni kod C-a ide u okviru main funkcije, koja predstavlja glavni
   program */
int main(void)
```

```
{
   /* Jedina linija koda koristi printf funkciju (komandu) za stampanje,
      ciji je rezultat izvrsavanja formatirana poruka */
   /* '\n' kod naynacava printf komandi da predje u novi red posle
       ispisivanja poruke */
   // Jednoredni komentari

   printf("Zdravo ");
   printf("svima!\n");

   return 0; // Ova komanda inicira da program vrati 0 operativnom sistemu
       kada zavrsi sa radom
}
```

Коментари могу да се налазе било где у програму, било у новој линији или истој у којој се налази команда или да се простиру у више линија. Треба бити пажљив са вишередним коментарима, ако се изостави симбол за крај коментара може се десити да компајлер третира део наредби као коментар и не изврши их.

1.1.8 Промењиве и додељивање вредности

За разлику од приказаног једноставног програма, већина програма изводи серију рачунања пре него што заврши са радом, и зато је неопходно да се подаци привремено чувају током извршавања програма. Ове меморијске локације се називају промењиве.

1.1.9 Типови података

Свака промењива мора да има тип (eng. *type*), који специфира коју врсту података ће промењива да чува. C програмски језик подржава велику колекцију типова података, за сада ћемо се ограничити само на два типа `int` и `float`. Избор типа података је веома важан, јер он одређује како ће се промењива чувати и које операције могу да се спроводе над промењивом. Нумерички типови података условљавају највећу и најмању вредност коју промењива може да прихвати, као и да ли су или не цифре после децималне тачке дозвољене. Промењиве типа `int` могу да чувају целе бројеве: 0, 1 495, -2553. Ипак распон вредности које промењива може да прихвати је ограничен. Промењиве типа `float` могу да чувају много већи распон вредности. Још важније, могу да

чувају вредности из скупа реалних бројева. Ипак постоји и друго ограничење, операције над овим типом података могу да буду спорије. И оно што је још значајније вредност `float` промењиве је најчешће апроксимација вредности која је првобитно сачувана у променљивој (грешка заокруживања).

Основни типови података

`int` — целобројна вредност, најчешће је величине 4 бајта (може и 2 бајта — зависи од рачунара),
`char` — један знак (целобројна вредност), један бајт,
`float` — реалан број једноструке тачности, најчешће 4 бајта,
`double` — реалан број двоструке тачности, најчешће 8 бајтова.
Постоје модификатори које можемо придружити основним типовима а то су `short` и `long`. Типу `int` се могу придружити оба при чему навођење кључне речи `int` није обавезно:

```
long int dugacak_broj; /*zauzima 4 bajta*/
short kratak;
long dugacak;
```

На сваком рачунару важи:

```
broj bajtova(short) <= broj bajtova(int) <= broj bajtova(long)
```

Типу `double` се може придружити `long` док се типу `float` не може придружити ни један модификатор. Постоје још и модификатори `signed` и `unsigned` који се могу придружити целобројним типовима (`int`, `char`, `short` и `long`). Они се односе на означене и неозначене целе бројеве (ненегативне целе бројеве). Ако је променљива целобројног типа који заузима n бајтова, односно $8*n$ битова, то значи да вредности које могу бити смештене у тој променљивој припадају опсегу од -2^{8n-1} до $2^{8n-1}-1$. Ако су типа `unsigned` онда припадају опсегу од 0 до 2^{8n-1}. Нпр. променљива типа `signed char` узима вредности од -128 до 127 (односно од -2^7 до 2^7-1) док променљива типа `unsigned char` узима вредности од 0 до 255 (односно од 0 до 2^8-1). Напомена:Важнији степени броја 2: $2^7 = 128$, $2^8 = 256$, $2^15 = 32768$, $2^16 = 65536$, $2^31 = 2147483648$ i $2^32 = 4294967296$. На пример, оптимални тип за скуп вредности $S = 3, 5, 189, ..., 15700$ је `short` или `unsigned short` зато што све вредности из скупа припадају опсегу од -2^15 до 2^15-1 као и опсегу од 0 до 2^16-1.

1.1.10 Декларација променљивих

Променљиве морају бити декларисане (описане компајлеру) пре него што се користе. Да би се декларисала променљива прво се наводи тип податка а затим назив променљиве (биће објашњено у секцији Именовање променљивих).

```
int main(void)
{
   int visina;        // Deklaracija celobrojne promenljive
   float prosek;    // Deklaracija realne promenljive

   /* Ako vise promenljivih ima isti tip. */
   int visina, tezina, sirina, zapremina;

   /* Obratiti paznju samo na kraju se stavlja ; dok se promenljive
      razdvajaju zarezom */
   float prosek, profit;
   . . .
}
```

Напомена: добра пракса је оставити један слободан ред између декларације променљивих и наредби. У C99 стандарду није неопходно декларисати све променљиве на почетку програма (али наравно обавезно пре употребе), тако да је могуће да у програму постоји одељак за ддекларацију па наредбе па декларација... Иначе пракса у објектно оријентисаним језицима је да се променљива не декларише пре него што је потребно да се први пут користи.

1.1.11 Додељивање вредности променљивим

```
   int visina;        // Deklaracija celobrojne promenljive
   visina = 172;    // Ddodeljivanje vrednosti

   float profit;
   profit = 2150.48;  // ili bolje
   profit = 2150.48f;
/* izostavljanje f oznake za tip moze izazvati upozorenje od kompajlera */
```

Могуће је додељивање целобројних вредности `float` типу податка, и обрнуто

али није увек сигурно!

```
int i, j = 5;
float x = 3.82;
i = x;
x = j;
```

Резултат извршења наведене секвенце наредби је да променљива i добије вредност 3, а не 3.82, док променљива x има вредност 5.0. Једном када је променљивој додељена вредност она се може користити у изразима.

1.1.12 Штампање вредности променљивих

`printf` функција се може користити и за штампање вредности променљивих.

```
printf("Visina: %d\n", visina); /* %d cuva mesto za vrednost
    promenljive "visina" i koristi se za celobrojne tipove */
printf("Profit: %f\n", profit); /* za stampanje float vrednosti koristi
    se %f, koja podrazuvena prikazuje 6 decimala posle decimalne tacke */

/* Da bi sami odredili koliko cifara treba da bude prikayano npr. 2
    %.2f */
printf("Profit: %.2f\n", profit);
```

Све променљиве могу бити приказане и само једним позивом `printf` функције. Не постоји ограничење у броју променљивих које могу да се наводе у овој функцији. (Дељење и целобројно дељење).

1.1.13 Иницијализација

Неке променљиве подразумевано добијају вредност 0 када програм започне извршавање, али већина не. Променљиве које не добијају подрзумевану вредност и није им додељена вредност, се зову неиницијализоване променљиве. Покушај приступа и употребе неиницијализоване променљиве може довести до неочекиваног резултата или програм престаје са радом. Из тог разлога је неопходно или доделити вредност променљивој пре употребе или је иницијализовати (доделити јој неку почетну вредност при декларацији).

```
int visina = 150;          //Deklaracija sa inicijaliyacijom
int visina = 150, tezina = 50, sirina = 48; //inicijalizacija vise
    promenljivih
```

1.1.14 Штампање вредности израза

Функција за штампање није ограничена само на штампање вредности променљивих, може приказати вреднот било ког нумеричког израза. Ова могућност се може искористити да се поједностави код, смањи број линија кода и увођење нових променљивих.

```
int visina = 172, duzina = 100, sirina = 50, zapremina; // zapremina je
    neinicijalizovana promenljiva

   zapremina = visina*duzina*sirina;
   printf("Zapremina: %d\n", zapremina);

   // ili bez uvodjenja promenljive zapremina
   printf("Zapremina: %d\n", visina*duzina*sirina);
```

1.1.15 Читање са улаза

За прихватање улазних информација користимо `scanf` функцију. Суфикс f у обе функције (за читање и писање) означава да је у питању форматирано читање/писање. То значи да обе функције захтевају формат у коме се очекује улаз односно треба приказати излаз. Да би се учитао само један цео број довољно је написати следећу наредбу:

```
scanf("%d", &i);
```

Овај стринг $\%d$ говори функцији за читање да чита са улаза целобројну вредност. i је int променљива у коју желимо да функција смести улаз. Симбол & у овом моменту не објашњавамо (биће објашњен касније) за сада треба прихватити да је тај симбол неопходан за учитавање.

1.1.16 Дефинисање константи

Када је потребно у програму користити константе уобичајено је да им се задају имена. Константе се могу дефинисати коришћењем макро дефиниција

```
#define MAX 165

/* #define je preprocesorska direktiva, kao i #include na kraju se ne
    stavlja ;
```

```
 * Kada se program kompajlira preprocesor zamenjuje svaki definisani makro
    sa predefinisanom vrednoscu
 * Vrednosti makroa mogu biti dodeljenje i iyrazima */

#define RECIPROC_OF_PI (1.0f / 3.14159f)
/* U tom slucaju izraz mora biti u zagradama. Konvencija je pisati imena
   makroa velikim slovima */
```

Свака директива препроцесору захтева засебан ред!

1.1.17 Идентификатори

Писање програма захтева именовање промењљивих, функција, макроа... Ова имена се sову идентификатори. У C-у идентификатори могу да се састоје од слова, цифара, подвлаке, АЛИ МОРАЈУ ДА ПОЧНУ СЛОВОМ ИЛИ ПОДВЛАКОМ. Примери неких валидних идентификатора:

```
skola, x, y, moja_promenljiva, ____
```

Примери невалидних идентификатора:

```
int //to je ključna reč
5nedelja //počinje cifrom
x-y //sadrži znak -
for //ključna reč
```

C је осетљив на мала и велика слова, разликује мала и велика слова стога промењљиве: ана, Ана, аНа, АНа ... нису једнаке и могу да се користе симултано у програму за различите сврхе. Уобичајена конвенција у C-у је да се иденфикатори именују малим словима где се подвлака користи за спајање речи, или се спајају речи а свака нова реч у имену идентификатора почиње великим словом. Не постоји ограничење у броју слова у имену (напомена: 31 према старом стандарду и 63 према новом, објашњење на вежбама).

1.1.18 Кључне речи

Кључне речи имају специјално значење за компајлер и не смеју се користити за именовање идентификатора. Кључне речи се пишу малим словима.

1.1.19 Изглед програма

Свака команда може да се наставља у новом реду. Број бланкова, табова - празнина није битан. Празни редови и увлачење је дозвољено и препоручује се да програм буде што читљивији.

1.2 Форматирани улаз и излаз

`scanf` и `printf` функције, које подржавају форматирано читање и писање су две најчешће коришћене функције у C-у. Обе функције су веома моћне али морају се користити пажљиво.

1.2.1 `printf` функција

Ова функција је дизајнирана да прикаже садржај стринга, са вредностима уметнутим на посебним местима у стрингу. У тренутку позива ове функције мора бити обезбеђен стринг за форматирање, који прате одговарајуће вредности које треба уметнути у стринг приликом штампања.

```
printf(string, expr1, expr2,...);
```

Вредности које се приказују могу бити константе, променљиве, или сложенији изрази. Не постоји ограничење у броју ових вредности које могу бити приказане позивом једне `printf` функције. Стринг за форматирање може садржати уобичајене карактере и спецификацију за конверзију која мора почињати са % карактером. Спецификација за конверзију је место које се резервише за смештање вредности у току штампања. Информација која следи карактер % специфицира како се вредност конвертује из своје примарне (бинарне) форме у коначном запису. Уобичајени карактери се штампају дословно онако како су написани, конверзионе спецификације се мењају одговарајућим вредностима. Напомена: C компајлер не проверава исправност конверзионе ниске. Неисправно уношење већег броја вредности него што је наведено у ниски ће резултовати неприказивањем свих вредности, навођење мањег броја вредности ће резултовати несмисленим приказом, исто се дешава ако конверзиона ниска није исправно наведена за одговарајуће типове података.

1.2.2 Конверзиона ниска

Конверзиона ниска има главну улогу у контроли приказивања резултата, али може бити компликована и тешка за читање. Конверзиона ниска може да

буде облика: $\%m.pX$ или $\%$-$m.pX$ где су m и p целобројне константе а X је слово. Обе целобројне константе су опционе, ако је p специфицирано тачка која раздваја p и m се изоставља. Нпр. $\%10.2f$, $m = 10$ $p = 2$ $X = f$. m специфицира минимални број карактера за приказивање. Ако је вредност која се приказује мање ширине него што је прецизирано вредности су десно поравнате (празна поља се штампају са леве стране). Ако је вредност која се штампа шира од специфицираног броја карактера, ширина поља за приказ се аутоматски проширује до неопходне величине. Уметање знака - означава лево поравнање (празнине ако их има се налазе после записа - десно). Значење прецизности p није тако једноставно за објаснити јер зависи од избора X спецификатора конверзије. X говори која конверзија треба да буде примењена на вредност пре штампања.

- d - се користи за приказ децималних (у основи 10) бројева. p говори који је најмањи број цифара за приказ (додатне нуле се исписују на почетку ако је неопходно) ако се изостави подразумена вредност је 1.

- e - приказује бројеве у покретном зарезу у експоненацијалном запису. p говори колико се цифара приказује после децималне тачке (подразумевано је 6). Ако је p 0 децимална тачка се не приказује.

- f - приказује броје у покретном зарезу са фиксним децималама без експонента. (p исто значање као и пре).

- g - приказује реалне бројеве или експоненцијалном запису или са фиксним децималним записом, у зависности од величине броја. p говори које је ограничење за значајне цифре (не цифре иза децималне тачке). За разлику од претходне конверзије ова конверзија неће приказати водеће нуле.

Последња спецификација је посебно значајна када је потребно приказати број за који се унапред не може предвидети величина и може узимати значајно другачије вредности.

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

   i = 40;
   x = 839.21f;

   // Konverziona niska za celobrojne promenljive
```

```
    printf(" |%d|%5d|%-5d|%5.3d|\n", i, i, i, i);

    // Konverziona niska za realne promenljive
    printf("|%10.3f|%10.3e|%-10g|\n sa minusom\n |%9.3f|%8.3e|%-10g|\n", x,
        x, x, -x, -x, -x);

    return 0;
}
```

Објашњење:

- %5.3d приказује целобројну вредност користећи најмање 5 карактера и најмање 3 цифре, како цео број има само две цифре додаје се водећа нула на почетку и 2 празна места испред.
- %10.3f Приказује реална број у фиксном децималном запису користећи 10 карактера укупно и 3 цифре после децималне тачке. Како x захтева само 7 карактера (3 пре децималне тачке, 3 после децималне тачке и децималну тачку) 3 празна места се умећу на почетку.
- %10.3e приказује x у експоненцијалном запису користећи свеукопно 10 карактера, са три цифре иза децималне тачке. x захтева девет места (ту је укључен и симбол за експонент) значи једно место се умеће лево.
- %-10g -приказује x или y фиксном запису или експоненту користећи свеукупно 10 карактера (лево поравнање).
-

Специјални знаци

Овде се подразумевају знаци који имају специјално значење за компајлер:

- Alert (bell) \a
- Backspace \b
- New line \n
- Horizontal tab \t
- \\\" ...

Scanf функција

Баш као што `printf` функција штампа према задатом формату тако и `scanf` функција чита према одређеном формату. И ова функција као аргумент има конверзиону ниску која у суштини има исто значење као до сада, може садржати регуларне карактере и конверзионе спецификације. Ипак, ова функција најчешће садржи само конверзионе спецификације. Уствари ова функција може много више, она покушава да групише улазне карактере са задатом спецификацијом тако што започиње читање са леве стране покушава да лоцира садржај који јој одговара прескачући празне симболе ако је неопходно, заустављa читање када наиђе на карактер који не може припадати одређеном формату. Ако је садржај алоциран и прочитан функција наставља са читањем преосталог садржаја конверзионе ниске. Ако неки садржај није прочитан према коректно функција завршава са радом без читања остатка конверзионе ниске.

```
1-20.3-4.0e3
scanf("%d%d%f%f", &i, &j, &x, &y)
```

%i се у `printf` функцији не разликују, док у `scanf` функцији %d се користи само за читање децималних бројева (основа 10), %i се може користити за учитавање октални, децималних и хексадецималних бројева, нпр. ако улаз започиње 0 (нпр. 056) са %i се учитава октални запис, са 0x и 0X се учитава хексадецимални број.

2

Изрази

Најједноставнији изрази су променљиве и константе. Сложенији изрази се добијају комбиновањем оператора и операнада (који су сами по себи изрази). Оператори су основни елементи за формирање израза и C има богату колекцију оператора. Елементарни оператори који су присутни и у другим програмским језицима укључују:

- Аритметичке операторе: + , - , * , \, %
- Релациони оператори за поређење.
- Логички оператори за конструкцију услова.

C подржава јако велики скуп оператора који ће се уводити постепено (због обима) кроз сва поглавља. Овде ће бити описани основни оператори, аритметички, оператори доделем и оператори инкрементирања и декрементирања.

2.1 Аритметички оператори

Унарни и бинарни +, - . Опрез са операторима \и % када је други аргумент 0 могу изазвати недефинисано понашање. Такође када је један од операнада негативан у различитм имплементацијама се добијају различити резултати. Ипак у C99 стандарду заокруживање при дељењу увек иде ка 0, а резултата примене оператора % је знака као први аргумент.

2.1.1 Приоритет оператора и асоцијативност

Приоритет операција увек може да се регулише навођењем заграда. Уколико заграде нису наведене приоритет аритметичких оператора је следећи:

Највећи:	+	-	(унарни)
	*	/	%
Најнижи:	+	-	(бинарни)

Приоритет оператора није довољан као правило када се у истом реду нађу оператори истог приоритета. У таквим случајевима се гледа асоцијативност оператора. Лево асоцијативни оператори се групишу са лева на десно, такви су сви аритметички оператори (*,/,%, + и -). Десно асоцијативни оператори се групишу са десна на лево. Такви су унарни аритметички оператори (+ и -).

2.1.2 Оператори инкрементирања и декрементирања

Две од најчешћи операција над променљивим су увећење променљиве за 1 (инкрементирање) и умањење променљиве за 1 (декрементирање).

```
y++;
--x;
```

Наизгле једноставни оператори могу бити прави изазов за коришћење. Разлог томе је што ови оператори могу бити префиксни у постфиксни. Други разлог је бочни ефекат ових оператора, као и оператори доделе и ови оператори мењају свој операнд. Префиксни оператор подразумева да се операнд ажурира одмах, док постфиксни захтева да се операнд ажурира касније. У случају постфиксног оператора операнд се ажурира пре следеће наредбе. Постификсни оператори инкрементирања и декрементирања имају јачи приоритет у односу на унарне аритметичке операторе и лево су асоцијативни. Префиксни оператори инкрементирања и декрементирања имају исти приоритет као унарни + и - и десно су асоцијативни.

2.2 Оператори доделе

Јендом када је израз израчунат његова вредност треба негде да се смести. Најједноставнији оператор доделе се користи у том случају (=). За ажурирање већ сачуване вредности у променљивој C подржава здружене операторе доделе.

```
i = 5;  /* i dobija vrednost 5 */
j = i;  /* j dobija vrednost 5 */
l = 6 * i - j; /* l dobija vrednost 25 */
...
int i;
```

```
float f;
i = 65.32; /* i dobija vrednost 65 */
f = 345; /* f je sada 345.0 */
...
```

Већина оператора не мења своје операнде, што није случај са оператором доделе где се израчунава вредност и придружује једном операнду. Овај ефекат је познат као бочни ефекат оператора доделе.

```
i = 5;  /* i dobija vrednost 5 */
f = 8.2 /* f dobija vrednost 8.2 */


j = i = f; /* prvo i dobija vrednost 8 yatim i j dobija vrednost 8 */
l = 6 * i - j; /* l dobija vrednost 25 */
...
/* zdruzeni operatori dodele */
i += 2; // isto sto i: i = i + 2;

i += j += k; // isto sto i: i += (j += k);

...
```

Лева страна израза

Већина оператора дозвољава да операнди буду промењљиве, константе, или изрази који и сами садрже операторе. Ипак, оператор доделе захтева да лева страна израза буде објекат у меморији, не константа нити резултат израчунавања. Промењљиве могу да буду лева страна израза.

Здружени оператори доделе

Додељивања у којима се користи стара вредност промењљиве како би се израчунала нова вредност су уобичајена у програмском језику C. Здружени оператори додељивања су +=, -=, *=, /=, %= . Има их укупно 9. Ови оператори су десно асоцијативни.

primer

2.2.1 Оценивање израза

Следећа табела приказује приоритет до сада наведених оператора:

Приоритет	Назиив	Симбол	Асоцијативност
1	инкрементирање (постфиксни)	++	лево
	декрементирање (постфиксни)	–	лево
2	инкрементирање (префиксни)	++	десно
	декрементирање (префиксни)	–	
	унарни +	+	
	унарни -	-	
3	множење/дељење..	*,/,%,	лево
4	адитивни	+, -	лево
5	додељивање	=, *=, /=, %=, +=, -=,	десно

Редослед извршавања подизраза

```
a = 5;
c = (b = a + 2) - (a = 1);
```

Ефекат извршавања друге наредбе у изразу је недефинисан јер није познато који подизраз ће се први извршити $(b = a + 2)$ или $(a = 1)$. Уколико се први израз израчунава прво b добија вредност 7, међутим ако се прво израчуна други израз b добија вредност 3. Неки компајлери чак могу да пријаве упозорење. Да би се овакви проблеми избегли треба избегавати коришћење оператора доделе у подизразима, увек је боље направити серију одвојених додела.

2.3 Примери - Задаци са вежби

Задатак 2.1 *Написати програм који на стандардном излазу исписује поруку: 'Zdravo svete!'.*

```
#include <stdio.h>

int main()
{
    printf("Zdravo svete!\n");
    return 0;
```

```
}
```

Задатак 2.2 *Шта се исписује извршењем следећег програма?*

```
#include <stdio.h>

int main() {
    printf("\"Zdravo, svima\"\n");
    printf("\\n\tprelazak u novi red\n");
    printf("\\t\ttabulator\n");
    printf("\\\\\tkosa crta\n");
    printf("%%%%\tprocenat\n");

    return 0;
}
```

Задатак 2.3 *Написати програм који илуструје декларацију и иницијализацију целобројне променљиве и испис њене вредности.*

```
#include <stdio.h>

int main()
{
    int i = 2;
    printf("Vrednost promenjive i je %d\n", i);

    return 0;
}
```

Задатак 2.4 *Написати програм у којем се са стандардног улаза уноси вредност целобројне променљиве и исписује њена вредност на стандардни излаз.*

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

```c
int main()
{
    int x;
    printf("Unesite ceo broj: ");
    /* Obratiti paznju na znak & pre imena promenljive
       u funkciji scanf. */
    scanf("%d",&x);

    /* U funkciji printf ne treba stavljati &. */
    printf("Uneli ste broj %d\n", x);

    return 0;
}
```

Задатак 2.5 *Написати програм у којем се са стандардног улаза уноси вредност реалне промењиве и исписује њена вредност на стандардни излаз заокружена на две децимале.*

```c
#include <stdio.h>

int main()
{
    float x;
    printf("Unesite realan broj: ");
    scanf("%f", &x);
    printf("Broj zaokruzen na dve decimale je: %.2f\n", x);

    return 0;
}
```

Задатак 2.6 *Написати програм који сабира два цела броја задата са стандардног улаза.*

```c
#include <stdio.h>

int main()
```

```
{
    int a, b, c;
    printf("Unesi prvi broj : ");
    scanf("%d", &a);
    printf("Unesi drugi broj : ");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);

   return 0;
}
```

Задатак 2.7 *Написати програм којим се размењују вредности два цела броја унета са стандардног улаза.*

```
#include <stdio.h>

int main()
{
    int a, b;
    int pom;
    printf("Unesi prvi broj : ");
    scanf("%d", &a);
    printf("Unesi drugi broj : ");
    scanf("%d", &b);

    printf("Pre razmene: a = %d, b = %d\n", a, b);

    pom = a;
    a = b;
    b = pom;

    printf("Posle razmene: a = %d, b = %d\n", a, b);

    return 0;
}
```

Задатак 2.8 *Програм демонстрира аритметичке операције над два унета цела броја.*

```c
#include <stdio.h>

int main()
{
    int a, b;
    printf("Unesi prvi broj: ");
    scanf("%d",&a);

    printf("Unesi drugi broj: ");
    scanf("%d",&b);

    printf("%d + %d = %d\n", a, b, a + b);
    printf("%d - %d = %d\n", a, b, a - b);
    printf("%d * %d = %d\n", a, b, a * b);
    printf("Celobrojni kolicnik a / b je: %d\n", a/b);
    printf("Pogresan pokusaj racunanja realnog kolicnika a/b \
           je: %f\n", a/b);
    printf("Realni kolicnik a/b je: %f\n", (float)a/(float)b);
    printf("Ostatak pri deljenju a/b je: %d\n", a%b);

    return 0;
}
```

Задатак 2.9 *Napisati program koji izračunava obim i površinu kruga poluprečnika* r*, koji se unosi sa standardnog ulaza.*

```c
#include <stdio.h>

#define PI 3.14159

int main() {
    int r;

    printf("Unesi poluprecnik kruga: ");
```

```
    scanf("%d",&r);

    printf("Obim: %g\n", 2 * r * PI);
    printf("Povrsina: %g\n", r * r * PI);

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

Задатак 2.10 *Šta se ispisuje sledećim programom?*

```
/*
   Operator ++ uvecava vrednost promenjive za 1.
   Operator -- je potpuno analogan operatoru ++ osim sto
               umanjuje vrednost za 1
*/
#include <stdio.h>
int main() {
    int x, y;
    int a = 0, b = 0;

    printf("Na pocetku : \na = %d\nb = %d\n", a, b);

    /* Ukoliko se vrednost izraza ne koristi, prefiksni i
      postfiksni operator se ne razlikuju */
    a++;
    ++b;
    printf("Posle : a++; ++b; \na = %d\nb = %d\n", a, b);

    /* Prefiksni operator uvecava promenjivu, i rezultat
       je uvecana vrednost */
    x = ++a;

    /* Postfiksni operator uvecava promenjivu, i rezultat je
       stara (neuvecana) vrednost */
    y = b++;

    printf("Posle : x = ++a; \na = %d\nx = %d\n", a, x);
```

```
    printf("Posle : y = b++; \nb = %d\ny = %d\n", b, y);

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

3

Управљање током програма

Иако C има много оператора насупрот томе има ограничен број команди. До сада су анализиране само две команде: `return` и команде за формирање израза. Преостале команде спадају у једну од три категорије, у зависности од тога како утичу на редослед извршавања програма:

- **Наредбе избора.** Наредбе `if` и `switch` омогућавају програму да изабере један од путева извршавања од неколико могућих.
- **Итерационе наредбе.** Наредбе `while`, `do` и `for` подржавају итерације (петље-извршавање у циклусима одређене групе наредби).
- **Наредбе скока.** Наредбе `break`, `continue`, `goto` омогућавају безусловно скакање на неко друго место у програму. У ову категорију спада и `return` наредба.

Поред набројаних група наредби преостале су само здружене наредбе, које групишу неколико наредби у једну као и тзв. *null* наредба која нема никакав ефекат извршавања. Пре увођења наредби избора, неопходно је објаснити логичке изразе који се формирају у оквиру наредби избора где се тестира њихова вредност. Логички изрази се формирају од релационих оператора ($<, <=, >, >=$), оператора једнакости ($==, !=$) и логичких оператора ($\&\&, ||, !$).

3.1 Логички изрази

Неке наредбе у C–у укључујући `if` наредбу тестирају услов у оквиру наредбе и проверавају да је вредност израза `true` или `false`. Пример поређења је `i < j`, треба напоменути да у другим програмским језицима резултат оваквог поређења може бити типа "boolean" или "logical" док је у C-у оваквом изразу придружена целобројна вредност 0(false) или 1(true).

3.2 Релациони оператори

Релациони оператори су исти као у математици $<, <=, >, >=$ само што је резултат њиховог извршавања 1 или 0. Нпр. израз `10 < 11` има вредност 1. Релациони оператори могу да се примењују над целобројним и реалним операндима. Такође је дозвољено да операнди буду различитог типа: `1 < 2.5` има вредност 1, док `4 > 5.6` има вредност 0. Израз `i < j < k` је дозвољен у C-у и еквивалента је изразу `(i < j)< k` јер су релациони оператори лево асоцијативни. Дакле у овом примеру се не проверава да ли се вредност променљиве i налази између вредности променљивих j и k.

3.3 Оператори једнакости

Оператори једнакости $(==, !=)$ су као и релациони оператори лево асоцијативни и резултат примене ових оператора даје вредности 0 или 1. Оператори једнакости имају нижи приоритет у односу на релационе операторе:

```
i < j == j < k
```

Овај израз је еквивалентан изразу:

```
(i < j) == (j < k)
```

3.4 Логички оператори

Сложенији логички изрази могу да се направе укључивањем и логичких оператора: and (&&), or (||) и not(!) . Прва два оператора су бинарна док је оператор негације унарни. Примена логичких оператора такође резултује вредностима 0 или 1. Операнди најчешће узимају вредности 0 или 1, али ово није неопходно напротив свака вредност различита од 0 се третира као тачно, док се нула третира као нетачно. Логички оператори се понашају на следећи начин:

- `!izraz` има вредност 1 ако *izraz* има вредност 0.
- `izraz1 && izraz2` има вредност 1 ако оба операнда-израза имају вредност 1.
- `izraz1 || izraz2` има вредност 1 ако било који од операнда-израза има вредност 1.

У свим другим случајевима је вредност наведених израза 0. Јако битна особина оператора коњукције и дисјункције је да они иводе лењо-израчунавање, у смислу да прво израчунавају вредности левог операнда а затим десног. Уколико се коначна вредност израза може закључити на овнову вредности левог операнда десни операнд-израз се не израчунава. Нпр.: `(i != 0)&& (j / i > 0)`. Ако је у овом изразу вредност променљиве i различита од 0, десна страна израза мора да се израчуна како би се утврдила коначна вредност израза. Уколико је вредност променљиве i једнака 0, израз `(j / i > 0)` се не извршава чиме се избегава дељење са 0. Треба бити јако пажљив са бочним ефектима у логичким изразима! Захваљујући лењом извршавању логичких оператора коњукције и дисјункције некада се бочни ефекти неће догодити. Нпр. у следећем изразу: `i > 0 && ++j > 0` Увећање променљиве j је бочни ефекат извршавања другог израза (операнда у коњукцији). Међутим ако је резултат извршавања првог израза `i > 0` false, израз `++j > 0` се не извршава а тиме се и не инкрементира променљива j. Проблем се може решити заменом места првог и другог израза `++j > 0 && i > 0` или још боље увећањем вредности променљивој j ван израза. Оператор негације има исти приоритет као унарни оператори + и -. Приоритет осталих логичких оператора је нижи од релационих и оператора једнакости. Нпр. `i < j && k == m` је исто што и `(i < j)&& (k == m)`. Оператор негације је десно асоцијативан док су остали логички оператори лево асоцијативни.

3.4.1 Наредба - `if`

Наредба - if омогућава управљане током програма (избор једног од више могућих путева), тестирањем израза у оквиру if наредбе. Најједноставнији облик ове наредбе је `if (izraz)naredba;`. Заграде око услова су обавезне оне су део if наредбе а не део израза. Када се if наредба извршава, израчунава се вредност израза у заградама: уколико је вредност израза различита од нуле извршава се *naredba* Напомена: не мешати операторе = и == они имају различито значење! Најчешће се ова наредба тестира да ли се променљива налази између неких вредности (у интервалу). `if (0 <= i && i < n)` За тестирање супротног услова користимо оператор дисјункције `if (i < 0 || i >= n)` У случају да је услов испуњен може постојати потреба да се изврши више од једне наредбе (здружене наредбе), у том случају је неопходно користити заграде: {*bloknaredbi*}. Груписањен наредби унутар заграда компајлеру се наглашава да се наредбе извршавају као једна.

3.4.2 Клаузула - `else`

```
if (izraz) naredba else naredba
```

Не постоји ограничење за израз који може да се појави у оквиру if наредбе, уствари могуће је и угњеждавати if наредбе. (Пример макс 3 броја). Угњеждавање је могуће до произвољне дубине. Равнање одговарајуће `if` и `else` гране омогућава лакше читање и разумевање програма. Стављање заграда чак и када нису потребне омогућава лакше разумевање кода и смањује могућност прављења грешке.

3.4.3 Каскадна `if` наредба

Често ћемо бити у прилици да тестирамо више услова и да се тестирање заврши чим је један од услова испуњен. За то је каскадна if наредба најбоље решење. (пример проверити да ли је вредност променљиве једнака, већа или мања нули)

3.4.4 Услововни изрази

`if` наредба омогућава програму да изврши једну од две акције у зависности од вредности израза који се проверава. C такође подржава оператор који омогућава изразу да изврши једну од две акције у зависности од вредности израза. Овај оператор се зове условни оператор и састоји се од симбола : и ? који се користе заједно у изразу: `izraz1 ? izraz2 : izraz3` Изрази 1, 2, 3 могу бити било ког типа. Условни оператор захтева три операнда па се на њега реферише и као на тернарни оператор.

3.4.5 `Boolean` тип података

Према стандарду C89 не постоји одговарајући логички тип података. Овај недостатак се једноставно превазилази увођењем целобројне променљиве којој се додељују вредности 1 или 0 и третирају као тачно односно нетачно. Такође је могуће увести макрое TRUE и FALSE. У новом стандардну C99 је уведен тип података `_Bool` (`<stdbool.h>`).

3.4.6 Наредба - switch

У случајевима када је потребно поредити вредност израза са више могућих вредности алтернатива if наредби је switch наредба.

```
switch (ocena) {
case 4: printf("Vrlo dobar");
      break;
case 3: printf("Dobar");
      break;
case 5: printf("Odlican")
      break;
case 1: printf("nedovoljan");
      break;
default: printf("nije korektna ocena");
      break;
}
```

Када се ова наредба изврши вредност променљиве оцена се пореди са вредностима 4,3,5,1. Ако се поклапа са 4 порука *Врло добар* се исписује и контрола тока се пребацује на прву наредбу после switch наредбе. Ако се вредност не поклопи ни са једном од наведених вредности default наредба се извршава и исписује се порука није коректна оцена. Ова наредба се једноставније чита од каскадне if наредбе, и бржа је што је посебно значајно када је више case израза. У оквиру switch наредбе треба да стоји целобројна вредност, карактери се третирају као целобројне вредности, а наредба није подржана за реалне вредности и стрингове. Свака ставка case почиње баш том кључном речју иза које стоји константан израз и не сме садржати променљиву (осим ако не представља макро) нити позиве функцијама. После ове наредбе може да се нађе произвољан број израза, уколико их је више није неопходно оградити их заградама. Дуплиране case наредбе нису дозвољене, редослед навођења у принципу није битан па тако ни default наредба не мора да буде на последњем месту. Само једна константа може да прати једну case наредбу, ипак неколико case израза може да претходи једној групи наредби. Нажалост не постоји начин да се напишу case наредбе које специфицирају интервал вредности. default наредба није неопходна.

3.4.7 Улога `break` наредбе

Улога `break` наредбе је да прекине извршавање `switch` блока и да програм крене са извршавањем прве наредбе након `switch`. Када се евалуира израз и крене са извршавањем одговарајућег case блока уколико `break` не постоји све наредбе се извршавају редом до краја `switch`. У следећем примеру ако је вредност израза *ocena* 3, биће одштампано *DobarOdlicannedovoljan* (извршавају се наредбе редом док се не наиђе на `break`).

```
switch (ocena) {
case 4: printf("Vrlo dobar");
      break;
case 3: printf("Dobar");

case 5: printf("Odlican");

case 1: printf("nedovoljan");
      break;
default: printf("nije korektna ocena");
      break;
}
```

Постоје ситуације када је згодно изоставити `break` наредбу:

```
int pozitivnih = 0; // promenljiva cuva broj pozitivnih ocena
    int ukupno = 0;   // racunaju se sve ocene
    switch (ocena) {
          case 5: case 4: case 3: case 2:
               pozitivnih++;
               // nema break - nastavlja se izvrsavanje (i uvecava broj
                   ukupnih)

               case 1:
               ukupno++;
}
printf("Prolaznost na ispitu je:%5.2f\n", 1.0*pozitivnih/ukupno*100);
```

3.5 Задаци са вежби

Задатак 3.1 *Написати програм који проналази максимум два цела броја унета са улаза.*

```c
#include <stdio.h>

int main() {
    int a, b;
    int max; // Vrednost maksimuma
    printf("Unesi prvi broj: ");
    scanf("%d",&a);

    printf("Unesi drugi broj: ");
    scanf("%d",&b);

    // Pretpostavimo da je prvi broj veci.
    max = a;

    // Ukoliko to nije slucaj, vrednost se azurira.
    if(max < b) {
      max = b;
    }

    printf("Maksimum je: %d\n", max);

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

Задатак 3.2 *Написати програм који за унето* `x` *израчунава* `y` *по формули:*

$$y = \begin{cases} \frac{-x^3}{3} & za\ x > 0 \\ \frac{x+1}{2} & inače \end{cases}$$

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

```c
int main() {
    float x;
    float y;

    printf("x = ?\n");
    scanf("%f", &x);

    if(x > 0) {
        y = - pow(x, 3) / 3.0;
    }
    else {
        y = (x + 1) / 2;
    }

    printf("y = %f\n", y);

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

Задатак 3.3 *Написати програм који проверава да ли се од задатих одсечака* a, b *и* c *може конструисати троугао. Ако може израчунати површину троугла, ако не, исписати одговарајућу поруку.*

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

int main() {
    double a, b, c;

    printf("Unesite stranice trougla a = ?, b = ?, c = ?\n");
    scanf("%lf%lf%lf", &a, &b, &c);

    if(a + b > c && a + c > b && b + c > a) {
        double s = (a + b + c) / 2;
        double P = sqrt(s * (s - a) * (s - b) * (s - c));

        printf("Povrsina trougla je: %.2g\n", P);
```

```
    }
    else {
        printf("Od unetih stranica ne moze se formirati trougao!\n");
    }

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

Задатак 3.4 *Написати програм који проналази реална решења квадратне једначине задате у облику*

$$ax^2 + bx + c = 0$$

.

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

int main() {
    float a, b, c;
    float D; // diskriminanta

    printf("Unesite koeficijente kvadratne jednacine \
           a = ?, b = ?, c = ?\n");
    scanf("%f%f%f", &a, &b, &c);

    if(a == 0.0) {
        printf("Greska a = 0, jednacina nije kvadratna!\n");

        system("pause");
        return 1;
    }

    // Izracunava se diskriminanta
    D = b * b - 4 * a * c;
    if(D == 0.0) {
        printf("Jednacina ima jedinstveno resenje \
               x1 = x2 = %.2f\n", - b / (2 * a));
    }
```

```
    else if(D > 0.0) {
         printf("Jednacina ima dva realna resenja:\n");
         printf("x1 = %.2f\n", (-b - sqrt(D)) / (2 * a));
         printf("x2 = %.2f\n", (-b + sqrt(D)) / (2 * a));
    }
    else {
         printf("Jednacina nema realnih resenja.\n");
    }

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

Задатак 3.5 *Написати програм који за задати редни број дана у недељи исписује његово име. У случају некоректног уноса исписати одговарајућу поруку.*

```
#include <stdio.h>

int main() {
    // Redni broj dana u nedelji.
    int dan;

    printf("Redni broj dana u nedelji:\n");
    scanf("%d", &dan);

    switch(dan) {
        case 1:
             printf("Ponedeljak\n");
             break;
        case 2:
             printf("Utorak\n");
             break;
        case 3:
             printf("Sreda\n");
             break;
        case 4:
             printf("Cetvrtak\n");
             break;
```

```
        case 5:
             printf("Petak\n");
             break;
        case 6:
             printf("Subota\n");
             break;
        case 7:
             printf("Nedelja\n");
             break;
        default:
             printf("Nekorektan unos, redni broj dana treba da \
                    je u intervalu [1, 7]\n");
    }

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

Задатак 3.6 *Написати програм који симулира рад упрошћеног калкулатора. Са улаза се уноси цео број затим једна од аритметичких операција +, -, *, /, %, и на крају цео број, а затим се на стандарни излаз исписује вредност тог израза.*

```
#include <stdio.h>

int main() {
    int a, b; // operandi
    char op;  // operacija

    printf("Unesti izraz u obliku operand operacija operand:\n");
    scanf("%d%c%d", &a, &op, &b);

    switch(op) {
        case '+':
             printf("%d %c %d = %d\n", a, op, b, a + b);
             break;
        case '-':
             printf("%d %c %d = %d\n", a, op, b, a - b);
```

```
            break;
        case '*':
            printf("%d %c %d = %d\n", a, op, b, a * b);
            break;
        case '/':
            // Obratiti paznju na deljenje!
            printf("%d %c %d = %f\n", a, op, b, a / (b + 0.0));
            break;
        case '%':
            printf("%d %c %d = %d\n", a, op, b, a % b);
            break;
        default:
          // Obratiti paznju na ispis \ i %
          printf("Nekorektan unos, operacija mora biti jedna \
                 od: +, -, *, \\, %%\n");
    }

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

3.5.1 Самосталан рад

Задатак 3.7 *Napisati program koji izračunava maksimum i minimum tri cela broja sa ulaza.*

Задатак 3.8 *Napisati program koji rešava linearnu jednačinu*

$$ax + b = 0$$

.

Задатак 3.9 *Ispitati da li se prave*

$$y = k_1 x + n_1$$

i

$$y = k_2 x + n_2$$

seku i ako se seku naći koordinate presečne tačke.

```c
#include <stdio.h>

int main() {
    float k1, n1; // koeficijenti prve prave
    float k2, n2; // koeficijenti druge prave

    printf("Koeficijenti prve prave: k1 = ?, n1 = ?\n");
    scanf("%f%f", &k1, &n1);

    printf("Koeficijenti druge prave: k2 = ?, n2 = ?\n");
    scanf("%f%f", &k2, &n2);

    if(k1 == k2) {
         printf("Prave su paralelne!\n");
    }
    else {
        float x, y; // Koordinate presecne tacke

        x = (n1 - n2) / (k2 - k1);
        y = k1 * x + n1;

        printf("Prave se seku u tacki (%.2f, %.2f)\n", x, y);
    }

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

Задатак 3.10 *Napisati program koji za zadati redni broj meseca ispisuje broj dana u tom mesecu. Za mesec februar pretpostaviti da ima 28 dana.*

4

Циклуси

Итеративне наредбе имају за задатак да понављају извршавање наредбе или блока наредби (тело циклуса). Сваки циклус има ткз. контролни израз. Кад год се изврши тело циклуса проверава се и контролни израз, уколико је његова вредност тачно (вредност различита од нуле) наставља се са извршавањем тела циклуса. Циклус прекида са извршавањем када вредност контролног израза постане нетачна. У C-у постоје три обика итеративних наредби (наредби за формирање циклуса): `while`, `do` и `for`. `while` наредба се користи за формирање циклуса код којих се услов испуњености контролног израза проверава на почетку пре него тело циклуса започне са извршавањем. Јасно је да у случају да контролни израз има вредност нетачно тело циклуса се неће извршити ни једном. `do` наредба се користи за формирање циклуса где се услов испуњености контролног израза проверава на крају, тј. загарантовано је извршавање тела циклуса макар једном. `for` наредба је погодна за формирање циклуса код којих се унапред зна колико пута треба да се изврши тело циклуса.

4.1 Наредба `while`

Наредба `while`, сматра се основном наредбом за формирање циклуса, а има следећи облик:

```
while (izraz) naredba;
```

Израз унутар заграда је контролни израз, иза заграда се налази наредба која овде представља тело цикуса. Проверава се тачност контролног израза, уколико је његова вредност тачно извршава се наредба затим се понавља провера контролног израза. Поступак се понавља док год је вредност контролног израза тачно у супротном се излази из циклуса. Пример када се из циклуса не

излази (формира се бесконачан циклус):

```
while (1) naredba;
```

Осим ако је наредба `break` која за задатак има кад год се нађе у циклусу да прекине исти и програм наставља са извршавањем од прве наредбе на коју се наилази након циклуса. Наредни пример демонстрира штампање целебројне вредности променљиве i која је на почетку иницијализована на 1. Контролни израз проверава да ли је вредност ове променљиве мања од 10 и уколико јесте извршава се тело циклуса. У конкретном примеру у телу циклуса је само једна наредба (у супротном би блок наредби морао бити унутар витичастих заграда) која штампа вреднот променљиве i, али истовремено и извршава увећање дате променљиве `i++`. Оператор ++ је постфиксни што значи да ће променљива бити увећана након штампања.

```
i = 1;
while (i < 10)
printf("%d\n", i++);
```

Важно је напоменути да је вредност променљиве i након изласка из циклуса 10, тј. из циклуса се излази оног момента када више није испуњен контрлни услов. Вредност промењљиве i након изласка из циклуса нема никакве везе са употребом постфиксног оператора, исто би било и да је код написан на следећи начин:

```
i = 1;
while (i < 10)
{
   printf("%d\n", i);
   i++;
}
```

Сада су у телу циклуса две наредбе па је потребно назначити блок који представља тело циклуса употребом заграда.

4.2 Наредба `do`

Формирање циклуса наредбом `do` има за ефекат потпуно исто што наредба `while` са кључном разликом да се услов не проверава на почетку циклуса већ на крају, што обезбеђује бар један пролазак кроз циклус. Синтакса ове наредбе

има следећу форму:

```
do naredba while (izraz);
```

4.3 Наредба `for`

Ова наредба је погодна за бројачке циклусе, али може послужити и за формирање других типова циклуса. Синтакса за формирање цикуса наредбом `for` је следећа:

```
for(izraz1; izraz2; izraz3) nareba
```

Први израз се израчунава само једном на почетку, и обично служи за иницијализацију бројачке променљиве (променљивих). Други израз је контролни израз и проверава се сваким проласком кроз циклус, уколико вредност овог израза није тачно тело циклуса се неће извршити, у супротном се извршава све док је вредност овог израза тачно. Трећи израз се извршава након сваког извршавања тела цикуса. Обично представља израз за ажурирање бројачке променљиве. Овакав циклус готово увек може да се преведе у циклус формиран `while` наредбом.

```
izraz1;
while(izraz2){
    nareba;
    izraz3;}
```

Пример када циклус формиран `for` наредбом није еквивалентан циклусу написаном `while` наредбом, као у претходном случају, је када се у телу циклуса налази `continue` наредба:

```
for(izraz1; izraz2; izraz3)
{
   ...
   continue;
   ...
}
```

Израз3 се увек извршава у оквиру `for` циклуса без обзира да ли је текућа итерација прекинута `continue` наредбом.

```
izraz1;
```

```
while(izraz2)
{
   ...
   continue;
   ...
   izraz3;
}
```

Овде се израз3 неће извршити када се `continue` наредба изврши.

4.4 Наредбе `break`, `continue` и `goto`

- `break` наредба преноси извршавање прорама у тачку после самог краја тела `switch`, `while`, `do` односно `for` наредбе.
- `continue` наредба преноси извршавање прорама у тачку pre самог краја тела `while`, `do` односно `for` циклуса (заправо прекида тренутну итерација и форсира почетак следеће итерације).
- `goto` наредба преноси извршавање прорама на произвољну наредбу (у оквиру исте функције) која започиње одговарајућом лабелом (ознаком).

```
for(d = 2; d < n; d++)
   if(n % d == 0) goto done;
done:
if(d < n)
   printf("%d je deljiv sa %d\n", n, d);
else
   printf("%d je prost broj\n", n);
```

У сучају угњеждених циклуса или `switch` наредби, `break` наредба прекида тело циклуса у коме се сама наредба налази. Док `goto` наредба форсирала скок на наредбу обележену одговарајуом лабелом која може да буде ван или унутар неког од угњеждених циклуса. Дакле, могућ је скок и прекидање свих угњеждених циклуса.

До сада смо са улаза читали податке користећи `scanf` наредбу и штампали на стандардни излаз `printf` наредбом. У оквиру ове главе се користе још две функције `getchar` и `putchar` чији рад је демонстриран следећим примером:

```c
/* Cita se jedan karakter i ispisuje se - demonstracija putchar i getchar
   */
#include <stdio.h>

main()
{
   int c;        /* Karakter - obratiti paznju na int */
   c = getchar(); /* cita karakter sa standardnog ulaza */
   putchar(c);   /* pise karakter c na standardni izlaz */

   putchar('\n'); /* prelazak u novi red */
   putchar('a'); /* ispisuje malo a */
   putchar(97);  /* ekvivalentno prethodnom */
}
```

4.5 Задаци

Задатак 4.1 *Написати програм који штампа првих 10 природних бројева.*

```c
#include <stdio.h>

int main() {
    int i; // Brojac u petlji

    for(i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }

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

Задатак 4.2 *Написати програм који израчунава* `n!`.

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

```c
int main() {
    int n;     // Broj ciji se faktorijel racuna
    long fakt; // Vrednost faktorijela
    int i;     // Brojac u petljama

    printf("n = ?\n");
    scanf("%d", &n);

    // Inicijalizacija (obratiti paznju!)
    fakt = 1;
    for(i = 1; i <= n; i++) {
        fakt *= i;
    }

    printf("%d! = %ld\n", n, fakt);

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

Задатак 4.3 *Шта се добија извршавањем следећег дела програма?*

```c
int i, j;
for(i = 1; i <= 3; i++) {
    for(j = 1; <= 3; j++) {
        printf("%d * %d = %d\t", i, j, i * j);
    }
    printf("\n");
}
```

Задатак 4.4 *Написати програм који преписује карактере унете са стандардног улаза на стандардни излаз.*

```c
#include <stdio.h>

int main() {
```

```
    char c;

    while((c = getchar()) != EOF) {
        putchar(c);
    }

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

Задатак 4.5 *Написати програм који врши пребројавање цифара у тексту који се уноси са стандардног улаза.*

```
#include <stdio.h>

int main() {
    char c;
    int br_cifara = 0;

    while((c = getchar()) != EOF) {
        if(c >= '0' && c <= '9') {
            br_cifara++;
        }
    }

    printf("Broj unetih cifara je: %d\n", br_cifara);

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

Напомена: Задатак је могао да се реши и коришћењем функције `isdigit(c)` дефинисане у заглављу `ctype.h`.

Задатак 4.6 *Написати програм који врши пребројавање великих и малих слова у тексту који се уноси са стандардног улаза.*

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

```c
#include <ctype.h>

int main() {
    char c;

    int br_malih, br_velikih;

    printf("Tekst?\n");
    br_malih = br_velikih = 0;
    while((c = getchar()) != EOF) {
        if(islower(c)) {
            // moglo je i if(c >= 'a' && c <= 'z')
            br_malih++;
        }
        else if(isupper(c)) {
            // mogle je i if(c >= 'A' && c <= 'Z')
            br_velikih++;
        }
    }

    printf("Ukupan broj malih slova je: %d\nUkupan broj velikih slova je:
        %d\n", br_malih, br_velikih);

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

Задатак 4.7 *Написати програм који сва мала слова у тексту који се уноси са стандардног улаза претвара у велика и обрнуто.*

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

int main() {
    char c;

    while((c = getchar()) != EOF) {
        char out = c;
        if(isupper(c)) {
```

```
            out = tolower(c);
        }
        else if(islower(c)) {
            out = toupper(c);
        }

        putchar(out);
    }

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

Задатак 4.8 *Написати програм који израчунава суму позитивних целих бројева који се задају са стандардног улаза. Унос прекинути када се унесе 0.*

```
#include <stdio.h>

int main() {
    int suma;

    suma = 0;
    // Beskonacna petlja
    while(1) {
        int x;

        printf("Broj (0 za kraj): ");
        scanf("%d", &x);

        if(x == 0) {
            break;
        }

        if(x > 0) {
            suma += x;
        }
    }
```

```
    printf("Suma pozitivnih je %d\n", suma);

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

Задатак 4.9 *Написати програм који израчунава суму парних целих бројева који се задају са стандардног улаза. Унос прекинути када се унесе 0.*

```
#include <stdio.h>

int main() {
    int x;
    int suma_parnih;

    suma_parnih = 0;
    while(1) {
        printf("x = ?, 0 za kraj\n");
        scanf("%d", &x);

        if(x == 0) {
            break;
        }

        if(x % 2 == 0) {
            suma_parnih += x;
        }
    }

    printf("Suma parnih je: %d\n", suma_parnih);

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

Задатак 4.10 *Почевши тренирање такмичар је прве недеље претрчао* `A` *километара, а касније сваке недеље је раздаљину повећавао за 10%. Одредити недељу у којој је претрчао растојање једнако дужини маратона (растојање 42.195km).*

```c
#include <stdio.h>

#define DUZINA_MARATONA 42.195

int main() {
    float A;
    int nedelja;

    printf("Unesite duzinu kojom je takmicar poceo treninge, u km: ");
    scanf("%f", &A);

    nedelja = 1;
    while(A < DUZINA_MARATONA) {
        A *= 1.1;
        nedelja++;
    }

    printf("U %d nedelji takmicar je pretrcao %.3f km\n", nedelja, A);

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

Задатак 4.11 *Написати програм који рачуна n - ти степен реалног броја задатког у двострукој прецизности.*

```c
#include <stdio.h>

int main() {
    double x;
    int n;

    double step; // tekuca vrednost stepena x - a

    int i;

    printf("n = ?\n");
    scanf("%d", &n);
```

```
    printf("x = ?\n");
    scanf("%lf", &x);

    step = 1.0;
    for(i = 1; i <= n; i++) {
        step *= x;
    }

    printf("Trazeni stepen je %.2lf\n", step);

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

Задатак 4.12 *Написати програм који штампа све делиоце целог броја који се уноси са улаза.*

```
#include <stdio.h>

int main() {
    int n;
    int i;

    printf("n = ?\n");
    scanf("%d", &n);

    printf("Delioci broja %d su:\n", n);
    for(i = 1; i <= n; i++) {
        if(n % i == 0) {
            printf("%d\n", i);
        }
    }

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

4.6 Задаци за вежбу

Задатак 4.13 *Написати програм којим се израчунава сума цифара целог броја унетог са стандардног улаза.*

Задатак 4.14 *Написати програм којим се обрћу цифре целог броја унетог са стандардног улаза.*

Задатак 4.15 *Написати програм којим се израунава следећа сума :*

$$S = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$$

.

Задатак 4.16 *Написати програм који за унети број n рачуна* 2^n.

Задатак 4.17 *Написати програм који налази прво појављивање цифре c у унесеном целом броју.*

Задатак 4.18 *Написати програм који имплементира функцију* $|x|$.

Задатак 4.19 *Написати програм који налази НЗД два цела броја a и b (Еуклидов алгоритам).*

Задатак 4.20 *Написати програм који рачуна збир највеће и најмање цифре датог целог броја.*

Задатак 4.21 *Написати програм који рачуна n-ти члан Фибоначијевог низа*

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2), \quad n > 1$$

.

Задатак 4.22 *Написати програм који израчунава збир првих n чланова реда:*

$$1 + \frac{1}{3} + \frac{1}{5} + \cdots + \frac{1}{2n-1}$$

Задатак 4.23 *Написати програм који израчунава збир првих n чланова реда:*

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$$

5

Низови

5.1 Једнодимензиони низови

До сада смо се сусретали само са променљивим које су скаларне вредности, значи могу чувати само једну вредност у складу са дефинисаним типом. C омогућава и чување скупа вредности кроз структуре и низове. Низови су посебне структуре података у C-у које омогућавају чување више вредности истог типа. Ове вредности се зову елементи низа, а може им се приступа преко позиције (индекса) у низу. Елементи једнодимензионог низа су смештени један за другим (у врсти или колони).

5.1.1 Декларација низа

Низ се декларише тако што се прво специфицира тип података у низу (тип елемената низа), име низа, а затим број елемената.

```
int a[10];
```

Елементи низа могу да буду било ког типа, а сама дужина низа може да се специфицира било којим целобројним изразом. Како може постојати потреба за каснију промену димензије низа није лоша пракса да се дефинише макро за димензију низа.

```
#define N 10
....
int a[N];
```

5.1.2 Индексирање елемената низа

За приступање појединачном елементу низа наводимо име низа за којим следи у заградама `[]` целобројна вредност која представља позицију одговарајућег елемента. Индексирање елемента увек полази од 0, па ако низ има n елемената индекси иду од 0 до `n-1`. Појединачни елементи низа могу да се нађу и учествују у изразу као и скаларне промењиве.

```
a[0] = 3;          // dodeljuje se vrednost 3 prvom elementu niza
printf("%d\n",a[4]); // stampa se 5.element niza
++a[i];            // uvecava se za jedan (i+1) element niza
```

Рад са низовима не може да се замисли без циклуса, тако се `for` циклус практично подразумева у употреби са низовима. То се односи готово на све примере где се нека акција изводи над свим елементима низа. Неке од типичних акција су приказане у примерима који следе (учитавање елмената низа, штампање елемената низа итд.):

```
#include <stdio.h>

int main() {
    int a[10];
    int i;
   /* Ucitavaju se elementi niza a */
    for (i = 0; i<10; i++){
        printf("a[%d]=", i);
        scanf("%d", &a[i]);
    }

    printf("Unazad: \n");
   /* Stampaju se elementi niza od pocev od poslednjeg do prvog */
    for (i = 9; i >= 0; i--) {
        printf("a[%d]=%d\n", i, a[i]);
    }

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

Напомена: C не захтева проверу индекса, уколико се деси да је индекс ван граница димензије низа понашање програма је недефинисано. Нпр. следећи део кода на неким компајлерима може да узрокује бесконачан циклус (објашњење на вежбама).

```
int a[10], i;

for(i = 1; 1 <= 10; i++)
   a[i] = 0;
```

У индексирању елемената низа може да учествује било који целобројан израз нпр.

```
a[i + j * 15] = 0;
```

Приликом индексирања изрази могу да имају и бочне ефекте нпр.:

```
i = 0;
while(i < N)
a[i++] = 0; // dodeljuje se nula tekucem elementu niza, a indeks niza se
    uvecava za 1
```

Употребу израза са бочним ефектима треба избегавати када може да изазове недефинисано понашање као нпр.:

```
i = 0;
while(i < N)
a[i] = b[i++]; // detaljno obrazlozenje na vezbama
```

Недефинисано понашање се може избећи на седећи начин:

```
for(i = 0; i < N; i++)
a[i] = b[i];
```

5.1.3 Иницијализација низа

Вредности елементима низа могу да се доделе у тренутку декларације унутар витичасти зарада и раздвојене зарезима. Уколико је иницијализатор краћи од димензије низа, преосталим елементима низа се додељује вредност 0. Ипак, није дозвољено да иницијализатор буде празан или прекорачује димензију низа, док може бити изостављена димензија низа.

```c
int a[6] = {1, 2, 3, 4, 5, 6};

int a[10] = {1, 2, 3, 4, 5, 6};
/* {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */

int a[10] = {0};
/* {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} */


int a[10] = {0, 0, 1, 0, 0, 6, 0, 0, 0, 0};
/* Ili jednostavnije koriscenjem dezignatora, koji mora biti celebrojna
   vrednost */
int a[10] = {[2]=1,[5]=6};

/* Redosled nije bitan */
int a[10] = {[5]=6, [2]=1};

/* Izostavljena je dimenzija niza, niz ce imati 21 element */
int a[] = {[5]=6, [2]=1, a[20]=3};

/* Mogu se i kombinovati oba nacina inicijalizacije */
int a[10] = {1,2,[4]=6, 7, 8,[2]=1, 23};
/* {1 2 1 23 6 7 8 0 0 0} */
```

Задатак 5.1 *Декларација, иницијализација низа и употреба* ***sizeof*** *оператора:*

```c
#include <stdio.h>

int main() {
    /* Niz se inicijalizuje tako sto mu se navode vrednosti
       u viticastim zagradama. Dimenzija niza se odredjuje
       na osnovu broja inicijalizatora */
    int a[] = {1, 2, 3, 4, 5, 6};

    /* Isto vazi i za niske karaktera */
    char s[] = {'a', 'b', 'c'};

    /* Ekvivalentno prethodnom bi bilo
```

```
    char s[] = {97, 98, 99};
    */

    /* Broj elemenata niza */
    int a_br_elem = sizeof(a) / sizeof(int);
    int s_br_elem = sizeof(s) / sizeof(char);

    /* Ispisuju se nizovi */
    int i;
    for (i = 0; i < a_br_elem; i++) {
        printf("a[%d]=%d\n", i, a[i]);
    }

    for (i = 0; i < s_br_elem; i++) {
        printf("s[%d]=%c\n",i, s[i]);
    }

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

Задатак 5.2 *Написати програм који израчунава суму елемената и средњу вредност низа целих бројева дужине не веће од 50.*

```
#include <stdio.h>

#define MAX_DUZINA_NIZA 50

int main() {
    int a[MAX_DUZINA_NIZA];
    int n; // Stvarna duzina niza
    int suma;

    int i;

    printf("Duzina niza? (<= %d): ", MAX_DUZINA_NIZA);
    scanf("%d", &n);
```

```
    suma = 0;

    printf("Unesite elemente niza:\n");
    for(i = 0; i < n; i++) {
       printf("a[%d] = ?\n", i);
       scanf("%d", &a[i]);
       // Suma se izracunava odmah nakon unosa tekuceg elementa
       suma += a[i];
    }

    printf("Suma = %d, srednja vrednost = %f\n", \
                  suma, suma / (n + 0.0));

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

Задатак 5.3 *Написати програм који проналази минимални елемент низа реалних бројева дужине не веће од 50, и његов индекс.*

```
#include <stdio.h>

#define MAX_DUZINA_NIZA 50

int main() {
    float a[MAX_DUZINA_NIZA];
    int n;     // Stvarna duzina niza
    int min_ix; // Indeks minimalnog elementa niza

    int i;

    printf("Duzina niza? (<= %d): ", MAX_DUZINA_NIZA);
    scanf("%d", &n);

    // Pretpostavimo da je prvi element niza najmanji
    min_ix = 0;

    printf("Unesite elemente niza:\n");
```

```
    for(i = 0; i < n; i++) {
       printf("a[%d] = ?\n", i);
       scanf("%f", &a[i]);

       /*
         Uporedjuje se tekuci element sa elementom na poziciji
         min_ix, ukoliko je tekuci element manji azurira se
         vrednost indeksa najmanjeg elementa.
       */
       if(a[i] < a[min_ix]) {
           min_ix = i;
       }
    }

    printf("Min = %f na poziciji = %d\n", \
                 a[min_ix], min_ix);

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

Задатак 5.4 *Дат је низ целих бројева дужине не веће од 50, и цео број* `x`*. Написати програм који проверава колико се пута задати број појављује у низу.*

```
#include <stdio.h>

#define MAX_DUZINA_NIZA 50

int main() {
    int a[MAX_DUZINA_NIZA];
    int n; // Stvarna duzina niza

    int x; // Broj koji se trazi
    int br_pojavljivanja;

    int i;

    printf("Duzina niza? (<= %d): ", MAX_DUZINA_NIZA);
```

```
    scanf("%d", &n);

    printf("Unesite elemente niza:\n");
    for(i = 0; i < n; i++) {
       printf("a[%d] = ?\n", i);
       scanf("%d", &a[i]);
    }

    printf("Unesite broj koji se trazi:\n");
    scanf("%d", &x);

    br_pojavljivanja = 0;
    for(i = 0; i < n; i++) {
        if(a[i] == x) {
            br_pojavljivanja++;
        }
    }

    printf("Broj %d, pojavljuje se %d puta\n", \
                 x, br_pojavljivanja);

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

Задатак 5.5 *Написати програм који рачуна скаларни производ два вектора у* `n` *- димензионом векторском простору (*`n <= 50`*).*

```
#include <stdio.h>

#define MAX_DUZINA_NIZA 50

int main() {
    float a[MAX_DUZINA_NIZA];
    float b[MAX_DUZINA_NIZA];
    int n; // Stvarna duzina niza

    float skalarni_proizvod;
```

```
    int i;

    printf("Duzina niza? (<= %d): ", MAX_DUZINA_NIZA);
    scanf("%d", &n);

    printf("Unesite prvi vektor:\n");
    for(i = 0; i < n; i++) {
       printf("a[%d] = ?\n", i);
       scanf("%f", &a[i]);
    }

    printf("Unesite drugi vektor:\n");
    for(i = 0; i < n; i++) {
       printf("b[%d] = ?\n", i);
       scanf("%f", &b[i]);
    }

    skalarni_proizvod = 0.0;
    for(i = 0; i < n; i++) {
       skalarni_proizvod += a[i] * b[i];
    }

    printf("Skalarni proizvod je %f\n", \
                  skalarni_proizvod);

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

Задатак 5.6 *Написати програм који сортира растуће низ целих бројева дужине* `n`, `n <= 50`. *(selection sort)*

```
#include <stdio.h>

#define MAX_DUZINA_NIZA 50

int main() {
```

```
    int a[MAX_DUZINA_NIZA];
    int n; // Stvarna duzina niza

    int i, j;

    printf("Duzina niza? (<= %d): ", MAX_DUZINA_NIZA);
    scanf("%d", &n);

    printf("Unesite elemente niza:\n");
    for(i = 0; i < n; i++) {
       printf("a[%d] = ?\n", i);
       scanf("%d", &a[i]);
    }

    printf("Pre sortiranja:\n");
    for(i = 0; i < n; i++) {
       printf("%d ", a[i]);
    }
    printf("\n");

    for(i = 0; i < n - 1; i++) {
       for(j = i + 1; j < n; j++) {
            if(a[i] > a[j]) {
                int pom = a[i];
                a[i] = a[j];
                a[j] = pom;
            }
       }
    }

    printf("Posle sortiranja:\n");
    for(i = 0; i < n; i++) {
       printf("%d ", a[i]);
    }
    printf("\n");

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

Задатак 5.7 *Решити претходни задатак применом bubble sort алгоритма.*

```
for(i = n - 1; i > 0; i--) {
    for(j = 0; j < i; j++) {
        if(a[j] > a[j + 1]) {
            int pom = a[j];
            a[j] = a[j + 1];
            a[j + 1] = pom;
        }
    }
}
```

Задатак 5.8 *Написати програм који за низ целих бројева дужине* `n <= 20` *проверава да ли је палиндром. Низ је палиндром ако је редослед елемената исти када се низ посматра спреда и отпозади.*

Задатак 5.9 *На Светском првенству у трчању за мушкарце на 100м, низом дужине* `n <= 10` *су представљени резултати. Приказати резултате сортиране растуће по времену трчања. Ако је познато да светски рекорд износи 9.58сец, испитати да ли је на овом такмичењу постављен нови.*

Задатак 5.10 *Написати програм који израчунава вредност полинома степена не већег од 20 чији су коефицијенти реални бројеви задати низом, у некој тачки* `x`.

Задатак 5.11 *Написати програм који израчунава и представља полином који се добија као производ два задата полинома степена не већег од 20 чији су коефицијенти реални бројеви (задати низовима - са улаза).*

5.2 Низови, ниске и функције

Задатак 5.12 *Функције и низови:*

```
#include <stdio.h>

/* Funkcija koja stampa niz celih brojeva */
```

```c
void print_array(int x[], int n) {
     int i;
     for(i = 0; i < n; i++) {
          printf("%d ", x[i]);
     }

     printf("\n");
}

/* Funkcija koja stampa niz realnih brojeva brojeva */
void print_arrayd(double x[], int n) {
     int i;
     for(i = 0; i < n; i++) {
          printf("%f ", x[i]);
     }

     printf("\n");
}

/* Funkcija koja siftuje elemente niza ulevo */
void cycle_left(int x[], int n) {
     int i;

     int first = x[0];
     for(i = 1; i < n; i++) {
        x[i - 1] = x[i];
     }
     x[n - 1] = first;
}

/* Funkcija koja siftuje elemente niza udesno */
void cycle_right(int x[], int n) {
     int i;

     int last = x[n - 1];
     for(i = n - 2; i >= 0; i--) {
          x[i + 1] = x[i];
     }
     x[0] = last;
}
```

```
/* Funkcija koja obrce elemente niza */
void reverse_array(int x[], int n) {
     int i;

     for(i = 0; i < n / 2; i++) {
        int pom = x[i];
        x[i] = x[n - i - 1];
        x[n - i - 1] = pom;
     }
}

/* Funkcija koja sortira elemente niza rastuce */
void bubble_sort(int x[], int n) {
    int i, j;

    for(j = n - 1; j > 1; j--) {
         for(i = 0; i < j; i++) {
              if(x[i] < x[i + 1]) {
                 int pom = x[i];
                 x[i] = x[i + 1];
                 x[i + 1] = pom;
              }
         }
    }
}

/* Funkcija koja transformise elemente niza, demonstracija poziva funkcije
    unutar funkcije */
void transform(int x[], int n) {
     int i;
     double y[n];

     for(i = 0; i < n / 2; i++) {
          y[i] = (x[2 * i] + x[2 * i + 1]) / 2.0;
     }

     print_arrayd(y, n / 2);
}
```

```c
/* Glavni program i demonstracija poziva funkcija. Napomena:
 * Sve navedene funkcije su tipa void - nemaju povratnu vrednost */
int main() {
    int x[] = {-2, 4, 0, -1, -6, 8, 12, 6, 7};

    int n = sizeof(x)/sizeof(int);

    printf("Niz inicijalno:\n");
    print_array(x, n);

    printf("Bubble:\n");
    bubble_sort(x, n);
    print_array(x, n);

    printf("Obrtanje:\n");
    reverse_array(x, n);
    print_array(x, n);

    printf("Transformacija:\n");
    transform(x, n);

    printf("Ulevo:\n");
    cycle_left(x, n);
    print_array(x, n);

    printf("Udesno:\n");
    cycle_right(x, n);
    print_array(x, n);

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

Задатак 5.13 *Функције и ниске:*

```c
#include <stdio.h>
/* Izracunava duzinu datog stringa. */
int duzina(char s[]) {
```

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

    return i;
}

/* Obrce dati string. */
void obrni(char s[]) {
    int i, j;

    /*
       Jedan brojac krece od pocetka stringa, drugi od kraja i u svakom
       koraku razmenjuju se elementi sa njihovim indeksima. Pritom se vodi
           racuna
       da se obrtanje vrsi samo dokle se brojaci ne sretnu, u suprotnom
           string
       bi se vratio na pocetni.
    */
    for(i = 0, j = duzina(s) - 1; i < j; i++, j--) {
        char c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

/* Prevodi neoznaceni ceo broj u binarni zapis. */
void prevedi_u_binarni(unsigned n, char binarni_zapis[]) {
     int i = 0;

     do {
         // U rezultujuci string se upisuje ostatak pri deljenju sa 2.
         binarni_zapis[i++] = (n % 2) + '0';
         // Dati broj se deli sa dva, dokle god se ne dodje do 0.
         n >>= 1; //Operatori nad bitovima ce biti detaljno objasnjeni u
             nastavku
     } while( n != 0);

     // Na kreju se rezultujuci string terminira 0 i obrce.
     binarni_zapis[i] = '\0';
     obrni(binarni_zapis);
```

```
}

/*
   Rekurzivna varijanta prevodjenja u binarni zapis.
*/
void prevedi_u_binarni_rek(unsigned n) {
     if(n) {
         prevedi_u_binarni_rek(n / 2);
         printf("%u", n % 2);
     }
}

int main() {
    unsigned n;

    while(1) {
       printf("Unesite prirodan broj, 0 za kraj unosa: ");
       scanf("%u", &n);

       if( n == 0) {
          break;
       }

       printf("rekurzivno: (");
       prevedi_u_binarni_rek(n);
       printf(")\n");

       char binarni_zapis[32];
       prevedi_u_binarni(n, binarni_zapis);
       printf("nerekurzivno: (%s)\n", binarni_zapis);
    }

    getch();
    return 0;
}
```

5.3 Операције над битовима

Нагласак у досадашњем раду су биле инструкције и особине језика C независне од машине. Манипулација над битовима и остале операције ниског нивоа су корисне за системско програмирање (писање компајлера и ОС), енкрипцију, графику и писање програма где је искористивост меморије веома важна. Овде су описани само основни оператори над битовима.

Оператори шифтовања

Симбол	Значење
<<	шифтовање у лево
>>	шифтовање у десно

Операнди могу бити било ког целебројног типа укључујући и `char`. Резултат израза `i << j` је померање битова у `i` за `j` места улево. За сваки бит померен улево, десно се додаје бит 0. Резултат израза `i >> j` је померање битова у `i` за `j` места удесно.Ако је `i` неозначеног типа или је ненегативан , нуле се додају са леве стране.

```
unsigned short i, j;
i = 13;    /* i je 13 binarno zapisano: 0000000000001101 */
j = i << 2; /* j dobija vrednost 52: 0000000000110100 */
j = i >> 2; /* j dobija vrednost 3:  0000000000000011 */
```

Ови оператори не модификују своје операнде. Употребом здружених оператора `<<=` и `>>=` је могуће модификовати операнде:

```
i = 13;    /* i je 13 binarno zapisano: 0000000000001101 */
i <<= 2;   /* i je sada 52:             0000000000110100 */
i >>= 2;   /* i je sada 13:             0000000000001101 */
```

Оператори шифтовања су нижег приоритета у односу на аритметичке операторе: `i <<= 2 + 1` је исто што и `i <<= (2 + 1)`

Остали оператори над битовима

```
unsigned short i, j, k;
i = 21;    /* i je 21 binarno zapisano: 0000000000010101 */
```

Симбол	Значење
~	комплемент
&	коњукција над битовима
^	ексклузивна дисјункција над битовима
|	дисјункција над битовима

```
j = 56;    /* j je 56 binarno zapisano: 0000000000111000 */
k = ~i;    /* k je sada 65514 :      1111111111000111 */
k = i & j;       /* k je sada 16:    0000000000010000 */
k = i ^ j ;      /* k je sada 45:    0000000000101101 */
k = i | j ;      /* k je sada 61:    0000000000111101 */
```

6

Датотеке

C - ова улазно/излазна библиотека је највећа и најважнија стандардна библиотека (`<stdio.h>`). До сада смо користили неке од важних функција ове библиотека `scanf`, `printf` за форматирано читање/писање, `getc` и `putc` које читају/пишу карактер по карактер `gets`, `puts` читају/пишу ред по ред, све за читање и писање са стандардног улаза/излаза. Функције аналогне наведеним, за читање и писање из/у датотеку, се такође налазе у оквиру исте библиотеке : `fscanf`, `fprintf`, `fgets` и `fputs`.

За комуникацију са улазно/излазним уређајима се користи механизам тока података. Сваком току података се придружује одређена структура дефинисана у заглављу `<stdio.h>`. Иако ток може бити асоциран са било којим уређајем, предмет овог курса су токови асоцирани са датотекама. Приступ току података се врши преко одговарајућег идентификатора који је типа `FILE *`. Пример декларације идентификатора за два нова тока:

```
FILE *fp1, *fp2;
```

6.1 Стандардни токови

Постоје три основна тока података дефинисана у `<stdio.h>`. За њих није потребно дефинисање идентификатора (нити отварање/затварање) већ су спремни за коришћење. На вежбама објаснити редирекцију!

Показивач на ток	Ток	Подразумевани уређај
stdin	стандардни улаз	тастатура
stdout	стандардни излаз	монитор
stderr	стандардни излаз за грешку	монитор

6.1.1 Текстуалне и бинарне датотеке

У текстуалним датотекама сваком бајту одговара један карактер, што значи да је текстуална датотека читљива (за човека). Нпр. изворни код написан у C-у се чува у текстуалној датотеци. У бинарној датотеци један бајт не мора да представља један карактер, и она није читљива, групе бајтова могу представљати типове података и сл. Извршни програми се чувају у бинарним датотекама. Текстуалне датотеке су, за разлику од бинарних, подељене по редовима (специјалан карактер за ознаку краја реда), такође поседују ознаку (маркер) за крај датотеке (EOF). У бинарним датотекама нема специјалних маркера, сви бајтови се третирају једнако.

Пример записа броја 32767 у бинарној и текстуалној датотеци. Значајна уштеда меморије при чувању у бинарној датотеци.

Када се пише програм који чита/пише у датотеку потребно је знати да ли је датотека бинарна или текстуална. Уколико та информација није доступна боље је претпоставити да се ради о бинарној датотеци. Отварање датотеке:

```
FILE * fopen(const char[] naziv datoteke, const char[] mod);
```

Отварање датотеке захтева позив функције `fopen`, која за први аргумент има стринг (ниску карактера) који садржи име датотеке коју треба отворити. Други аргумент ове функције је мод-стринг који специфицира коју операцију са датотеком намеравамо да изведемо.

Уколико се датотека не налази у истом фолдеру као изворни програм неопходно је навести целу путању на један од два приказана начина:

```
fp = fopen("c:\\project\\test.dat", "r");

fp = fopen("c:/project/test.dat", "r");
```

Функција `fopen` враћа показивач на ток, тј. идентификатор за приступ отвореној датотеци који ће програм користити за даљу обраду датотеке. У конкретном примеру за обраду датотеке "test.dat" ће се користити идентификатор fp, под условом да је датотека успешно отворено. Уколико датотека са задатим именом не постоји или путања није исправна задата или корисник нема право приступа датотеци fp ће бити `null` показивач.

Напомена: Увек треба проверити да ли је датотека успешно отворена.

Мод за отварање датотеке може бити из следеће групе стрингова:

"r"	читање, датотека већ постоји
"w"	писање од почетка (ако датотека не постоји биће направљена)
"a"	писање у наставку постојећег садржаја(ако датотека не постоји биће направљена)
"r+"	читање и писање од почетка
"w+"	читање и писање, преко постојећег садржаја
"a+"	читање и писање, надовезивање ако садржај већ постоји

Затварање, претходно отворене, датотеке:

```
int fclose(FILE * stream);
```

Аргумент ове функције је идентификатор датотеке коју затварамо. Функција враћа 0 уколико је датотека успешно затворена.

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

#define NAZIV_DAT "primer.dat"

int main()
{
   FILE* fp;

   fp = fopen(NAZIV_DAT, "r");

   if (fp == NULL)
   {
      printf("Datoteku pod nazivom %s nije moguce otvoriti\n", NAZIV_DAT);
      exit(EXIT_FAILURE);
   }

   fclose(fp);
   return 0;
}
```

6.1.2 Задавање аргумената командне линије

Пример покретања програма који из командне линије узима два аргумента, два стринга, који представљају називе датотека:

```
primer.exe ulaz.dat izlaz.dat
...
int main(int argc, char* argv[])
```

У наведеном примеру arg има вредност 3, док су $argv[0]$ назив програма у овом случају `primer.exe`, $argv[1]$ `ulaz.dat` и $argv[2]$ `izlaz.dat`.

Задатак 6.1 *Пример илуструје отварање датотеке, чије име се задаје као аргумент командне линије, за читање.*

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

int main(int argc, char* argv[])
{
   FILE* fp;
   if(argc != 2)
   {
      printf("Nije zadat odgovarajuci broj argumenata\n");
      exit(EXIT_FAILURE);
   }


   if ((fp = fopen(argv[1], "r")) == NULL)
   {
      printf("Datoteku pod nazivom %s nije moguce otvoriti\n", argv[1]);
      exit(EXIT_FAILURE);
   }

   printf("%s moze biti otvorena\n", argv[1]);
   fclose(fp);

   return 0;
}
```

6.2 Читање и уписивање у датотеку

```
int fscanf(FILE * tok, const char[] format, ...);
```

```
int fprintf(FILE * tok, const char[] format, ...);
```

Рекапитулација основног о датотекама:

```
/* Osnovne funkcije za rad sa datotekama */
#include <stdio.h>

//Otvaranje datoteke
FILE *f = fopen(char[] nazivdat, char[] mod);

// Kada se program pokrene operativni sistem otvara 3 datoteke sa
    pokazivacima:
FILE* stdin;
FILE* stdout;
FILE* stderr;

// Funkcija:
int fclose(FILE *fp);
/* je inverzna funkciji fopen i prekida vezu izmedju programa i datoteke,
 * prazne se baferi koji privremeno cuvaju sadraj datoteke i obezbedjuju
     da sadrzaj bude trajno upisan na disk...
 * Napravljena struktura FILE se brise i vise nije dostupna */

// Funkcije koje mogu da se koriste i sa datotekama:
int getc(FILE *fp);
int putc(int c, FILE *fp);

/* Primer funkcije koja kopira sadrzaj datoteke ulaz.txt u datoteku
    izlaz.txt - neefikasan nacin. */

void kopirajDatoteku(FILE* fin, FILE* fout)
{
   int c;
   while((c=getc(fin)) != EOF)
   {
      putc(c,fout);
```

```
   }
}

int main()
{
   FILE *in, *out;
   in = fopen("ulaz.txt","r");
   if(in == NULL)
      return -1; //Neuspesno otvaranje datoteke
   out = fopen("izlaz.txt", "w");
   if(out == NULL)
   return -1;
   kopirajDatoteku(in, out);
   fclose(in);
   fclose(out);
   return 0;
}
```

```
/* Naredni primer cita cifre iz datoteke i formira dekadni broj od
    procitanih cifara
 * ako nije procitana cifra, karakter se vraca nazad */

#include <stdio.h>

int citajCifru(FILE* in)
{
   int c, d = 0;
   while(isdigit(c=getc(in)))
   {
      d = d * 10 + c - '0';
   }
   ungetc(c, in);
return d;
}
int main()
{
   FILE * in;
   in = fopen("ulazCifre.txt", "r");
```

```
   if(in == NULL)
      return -1;
   printf("%d \n", citajCifru(in));
   fclose(in);
return 0;
}
```

```
/* feof primer: brojanje bajtova
* f-ja feof vraca vrednost tacno (razlicito od nule) ukoliko se doslo do
    kraja datoteke
*/
#include <stdio.h>

int main ()
{
   FILE * in;
   long n = 0;
   in = fopen ("ulaz.txt","r");
   if (in == NULL)
      /* perror primer - Interpretira prikaz vrednosti globalne promenljive
          errno */
      perror ("Greska prilikom otvaranja datoteke");
   else
   {
      while (!feof(in)) {
         fgetc (in);
         n++;
      }
   fclose (in);
   printf ("Ukupan broj bajtova: %d\n", n-1);
   }
return 0;
}
```

```
/* ferror primer: obrada greske
 * f-ja ferror vraca tacno (vrednost razlicitu od nule) ako je doslo do
     greske
```

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

int main ()
{
   FILE * in;
   in = fopen("ulaz.txt","r");
   if (in == NULL)
      perror ("Greska prilikom otvaranja datoteke");
   else {
      /* Datoteka je otvorena za citanje! */
      fputc ('x', in);
      if (ferror (in))
         printf ("Greska prilikom pisanja u ulaz.txt\n");
      fclose (in);
      }
return 0;
}
```

```
#include <stdio.h>

/* Funkcija fgets cita sledecu liniju iz datoteke (ukljucujuci oznaku
    kraja reda) */
char * fgets(char *line, int maxline, FILE *fp);
/* rezultat smesta u nisku line, najvise maxline -1 karater ce biti
    smesten u nisku. Niska je terminisana */
int fputs(char *line, FILE *fp)
/* smesta nisku u datoteku, ne mora da sadrzi oznaku kraja reda */
/* Za vezbu fputs - prepisati sadrzaj jedne datoteke u drugu */
int getline(char *line, int max, FILE* in)
{
   if (fgets(line, max, in) == NULL)
      return 0;
   else{
      printf(line);
      return strlen(line);
      }
}
```

```c
int main()
{
   FILE* in, *out;
   char s[200];
   in = fopen("ulaz.txt", "r");
   if (in == NULL)
      return -1;
   out = fopen("izlazPrepisano.txt", "w");
   if( out == NULL)
      perror("greska prilikom otvaranja/kreiranja datoteke");
   while(!feof(in))
   {
      int d = getline(s, 100, in);
      printf("%d\n", d);
      fputs(s, out);
   }
 fclose(in);
 fclose(out);
 return 0;
 }
```

6.3 Баферовање

6.4 Бинарне датотеке

```c
#include <stdio.h>
/* Standardna biblioteka nudi funkcije za direktno citanje i pisanje
    bajtova u binarne datoteke.
 * Na raaspolaganju su i funkcije za pozicioniranje u datoteci.
 * Funkcija fread se koristi za citanje niza slogova iz binarne datoteke,
    fwrite za pisanje niza slogova. */
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
/* 1. argummenta je adresa u memoriji u kojoj se smestaju ucitani slogovi
 * 2. argument je velicina jednog sloga
 * 3. argument je broj slogova
 * 4. pokazivac na odgovarajucu datoteku
```

```c
Funkcija fseek sluzi za pozicioniranje na mesto u datoteci sa koga ce biti
    procitan ili na koje ce biti upisan sledeci podatak.
Funkcija ftell vraca trenutnu poziciju u datoteci (u obliku pomeraja od
    pocetka izrazenog u broju bajtova). */
int fseek (FILE * stream, long int offset, int origin);
/* - drugi argument je pomeraj izrazen u broju bajtova
   - SEEK_SET koja oznacava da se pomeraj racuna u odnosu
   na pocetak datoteke, SEEK_CUR koji oznacava da se pomeraj racuna u
       odnosu
   na tekucu poziciju i SEEK_END koji oznacava da se pomeraj racuna u
       odnosu na kraj datoteke. */
long int ftell (FILE * stream);

int main()
{
   FILE* f;
   int i, kvadrat, j;
   int niz[] = {1, 2, 3, 4, 5};
   /* Upisivanje niza u binarnu datoteku */
   if ((f = fopen("niz.bin", "w+b")) == NULL)
   {
      fprintf(stderr, "Greska prilikom otvaranja datoteke niz.bin");
      return 1;
   }
   fwrite(niz, sizeof(niz), 1, f);
   fclose(f);

    /* Citanje iz binarne datoteke */
    if ((f = fopen("niz.bin", "rb")) == NULL)
   {
      fprintf(stderr, "Greska prilikom otvaranja datoteke niz.bin");
      return 1;
   }
   fread(niz, sizeof(niz), 1, f);
   fclose(f);
   printf("velicina niza je u bajtovima: %d\, a broj elemenata: %d\n",
       sizeof(niz), sizeof(niz)/sizeof(int));
   if ((f = fopen("ulazb.txt", "w+b")) == NULL) {
      fprintf(stderr, "Greska prilikom otvaranja datoteke");
      return 1;
```

```
     }
for (i = 1; i <= 5; i++) {
   kvadrat = i*i;
   fwrite(&i, sizeof(i), 1, f);
   fwrite(&kvadrat, sizeof(kvadrat), 1, f);
}
printf("Upisano je %ld bajtova\n", ftell(f));
for (i = 5; i > 0; i--) {
   fseek(f, (i-1)*sizeof(i), SEEK_SET);
   fread(&j, sizeof(j), 1, f);
   printf("%3d %3d\n", i, j);
}
fclose(f);
}
```