ПРОГРАМИРАЊЕ

ЗА ТРЕЋИ РАЗРЕД

Садржај

Низови

Низови представљају сложене типове података, и базирају се на осталим типовима променљивих. Низови представљају скупове коначног броја елемената истог типа обележене једним именом. Сваки податак у низу се назива његовим елементом, а сваки елемент има своју позицију. Сваком елементу низа се може приступити преко његове позиције у низу. Позиција елемента у низу одређена је његовим индексом. Индекс мора бити целобројна вредност и увек почиње нулом. За низ је битно познавати његове димензије да бисмо га могли исправно индексирати. Елементи низа се увек смештају у повезаним меморијским локацијама.

Низ може бити једнодимензионалан (када га зовемо једноставно низ), дводимензионалан (када га називамо матрицом због аналогије са истоименим математичким појмом), и вишедимензионалан (коцка, четвородимензионална коцка итд.).

Код једнодимензионалног низа, димензија се поистовећује са дужином низа. Нпр. кажемо да је низ димензије n.

Код више димензионалних низова, међутим, не постоји појам дужине, него се увек каже да је низ димензија m x n x ... x z или (m,n,...,z).

Једнодимензионални низови

То су низови који имају само један индекс, који мора бити целобројан. Индекс првог елемента низа увек почиње редним бројем 0. У суштини индекс елемента представља позицију елемента низа у односу на први елемент низа у меморији.

У програмском језику C++, низови се индексирају користећи оператор средње заграде ([]).

Напомена: у оквиру оператора средње заграде ([]) мора стајати цео број.

Декларација низа

Начин декларације низа је следећи:

```
тип име [димензија][димензија]...
```

тип – тип података сваког од елемента у низу

име – име низа и за њега важе правила као и за све остале промењиве

[димензија] – цели број између средњих заграда показује број елемената

Напомена: у оквиру оператора средње заграде ([]) не смете користити промењиве приликом декларисања низова. Нпр: int n=3; int a[n];

Следи пример декларације низа целих бројева:

```
int niz[40]; /* декларишемо статички низ величине 40 */
```

Пример декларације дводимензионалног низа целих бројева:

```
int niz[10][10]; /* декларишемо статички низ величине 10x10 */
```

Пример: Декларисати низ који садржи:

10 целих бројева	-->	`int nizbrojeva [10];`
пет реалних бројева	-->	`float nizrealnih [5];`
8 слова	-->	`char nizslova [8];`

Пример: Декларисати низ са 10 елемената типа целих бројева и сваком елементу низа доделити вредност његовог индекса

```
int x[10],i;
for (i=0; i<10; i++)
     x[i]=i;
```

када би приказали елементе оваквог низа добили би следеће:

```
x[0] = 0
x[1] = 1
x[2] = 2
x[3] = 3
x[4] = 4
x[5] = 5
x[6] = 6
x[7] = 7
x[8] = 8
x[9] = 9
```

Иницијализација низова и приступ елементима низа

Елементима низа може се доделити одређена вредност приликом декларисања низа. Додела вредности елеметнима низа зове се иницијализација низа.

Иницијализованим елементима се вредности додељују тако што се при декларацији низа наводе вредности елемената низа у оквиру витичастих заграда.

Ако је број елемената низа већи од броја иницијализатора тада сви елементи низа који немају одговарајући иницијализатор постају једнаки нули.

Пример: Иницијализација целобројног низа врши се на следећи начин:

```
int X[10] = {11, 12, 55, 66, 2, 4, 5, 67, 34, 12};
```

Име низа је X. Приликом иницијализације овог низа елементу X[0] се додељује вредност првог броја након велике заграде, односно 11, другом елементу X[1] наредна вредност, тј. 12 и тако редом. У табели испод је приказано ком индексу је додељена која вредност.

Индекс елемента низа	0	1	2	3	4	5	6	7	8	9
Вредност елемента низа	11	12	55	66	2	4	5	67	34	12

Пример: Иницијализовати низ од 5 реалних чланова низа.

```
float Realni_niz[5] = {1.1, 1.2, 5.5, 6.6, 3.3};
```

Пример: Иницијализовати низ који у себи садржи слова која чине реч ЗДРАВО.

```
char s[6]={'z', 'd', 'r', 'a', 'v', 'o'};
```

вредности елемената оваквог низа су следећи:

```
s[0] = z
s[1] = d
s[2] = r
s[3] = a
s[4] = v
s[5] = o
```

Пример: Иницијализованом низу од 3 реална броја сваки елемент увећати два пута.

```
float a[3];
a[0] = 3.4;
a[1] = 4.5;
a[2] = 5.6;
int i;
for (i=0; i<3; i++)
    a[i] = a[i]*2;
```

приказани елементи оваквог низа су следећи:

```
a[0] = 6.8
a[1] = 9.0
a[2] = 11.2
```

Пример: Иницијализовати низ 1, 3, 9, 6, 5, 4, 2, 8, 7 и приказати све елементе низа.

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int x[9]={1, 3, 9, 6, 5, 4, 2, 8, 7}, i;
    for(i = 0; i<9; i++)
          cout<<x[i]<<" ";
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
1 3 9 6 5 4 2 8 7
```

Приступ елементима низа

Приступање елементима низа врши се преко његовог индекса. С обзиром да индекси чланова низа почињу од нуле, низ има индексе од 0 до n-1. Следећи ту логику четврти члан низа имаће индекс 3.

Пример: Иницијализовати низ 1, 2, 3, 4, 5, 6, 7, 8 и приказати суму другог, четвртог и шестог елемента низа.

```
#include <iostream>
using namespace std;
int main()
{
    int x[8]={1, 2, 3, 4, 5, 6, 7, 8}, suma;
    suma = x[1] + x[3] + x[5];
    cout<<"suma 2-og, 4-tog i 6-tog elementa niza je "<<suma<<endl;
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
suma 2-og, 4-tog i 6-tog elementa niza je 12
```

Пример: Унети низ од 5 целих бројева и приказати само оне који су већи од 0.

```
#include <iostream>
using namespace std;
int main()
{
    int x[5];
    cout<<"Unesi 5 celih brojeva"<<endl;
    for(int i = 0; i<5; i++)
          cin>>x[i];
    for(int i = 0; i<5; i++)
          if(x[i]>0)
                cout<<x[i]<<" ";
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Unesi 5 celih brojeva
2
3
0
7
-5
2 3 7
```

Пример: За унети низ од n елемента целобројног типа одредити суму елемената у низу.

```
#include <iostream>
using namespace std;
int main()
{
    int n, i, x[1000], s=0;
    cout<<"Koliko elemenata ima niz?"<<endl;
    cin>>n;
    cout<<"unesi elemente niza:"<<endl;
```

```
    for(i = 0; i<n; i++)
    {
          cin>>x[i];
          s = s + x[i];
    }
    cout<<"suma svih elemenata niza je "<<s<<endl;
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Koliko elemenata ima niz?
4
unesi elemente niza:
2
4
6
8
suma svih elemenata niza je 20
```

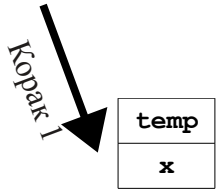
Пример: Унети низ од 5 целих бројева и заменити вредности другом и четвртом елементу низа, а затим приказати низ.

Приликом замене места елементима низа потребно је да користимо додатну промењиву истог типа као и низ како би привремено сачували вредност једног од елемената низа.

Поступак би био следећи:

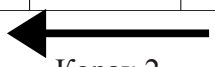
Корак 1: У привремену промењиву сместити вредност једног елемента низа: **`temp = X[m]`**

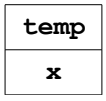
Индекс елемента низа	0	...	m	...	n	...
Вредност елемента низа	a	...	x	...	y	...



Корак 2: Вредност другог елемента низа преместити у први елемент низа: **`X[m] = X[n]`**

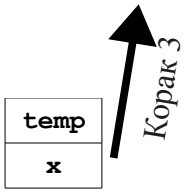
Индекс елемента низа	0	...	m	...	n	...
Вредност елемента низа	a	...	y	...	y	...





Корак 3: Вреднст привремене промењиве преместити у други елемент низа: **`X[n] =  temp`**

Индекс елемента низа	0	...	m	...	n	...
Вредност елемента низа	a	...	y	...	x	...




```
#include <iostream>

using namespace std;

int main()
{
  int X[5], temp, i;
```

```
  cout << "Unesi elemente niza: "<< endl;
  for (i = 0; i < 5; i++)
    cin >> X[i];

  // prikazujemo niz kako je unet od strane korisnika
  cout << "uneti niz: " ;
  for (i = 0; i < 5; i++)
    cout << X[i] << " ";
  cout << endl;

  //menjamo vrednosti elementima niza
  temp=X[1];
  X[1]=X[3];
  X[3]=temp;

  // prikazujemo niz nakon zamene vrednosti elemenata
  cout << "novi niz: " ;
  for (i = 0; i < 5; i++)
    cout << X[i] << " ";
  cout << endl;

  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Unesi elemente niza:
1 2 3 4 5
uneti niz: 1 2 3 4 5
novi niz: 1 4 3 2 5
```

Питања:

1. Како се декларише низ?
2. Која правила важе за име низа?
3. Колико елемената имају низови a[10] и s[3][5]?
4. Да ли димензија низа може бити реалан број?
5. Који је први индекс елемената низа?
6. Ако низ има к елемената који је претпоследњи елемент низа?
7. Које вредности имају елементи низа ако само декларишемо низ?
8. Како се иницијализује низ?
9. Шта се дешава ако не иницијализујемо све елементе низа?
10. Који је индекс првог елемента низа?

Задаци за вежбање

1. Иницијализовати низ од 8 елемената и приказати трећи.

```
#include <iostream>
using namespace std;
int main()
{
    int a[8]={3, 23, 13, 43, 53, 63, 22, 33};
    cout<<"treci element je: "<<a[2]<< endl;
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
treci element je: 13
```

2. Написати програм којим ће се унети n целобројних чланова низа и приказати их.

```
#include <iostream>
using namespace std;
```

```
int main()
{
     int i,n, a[100];
     cout<<"Uneti n:"<<endl;
     cin>>n;

     for(i=0;i<n;i++)
     {
           cout<<"a["<<i<<"]=";
           cin>>a[i];
     }
     for(i=0;i<n;i++)
           cout<<a[i]<<" ";
     return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Uneti n:
3
a[0]=1
a[1]=2
a[2]=3
1 2 3
```

3. Израчунати збир првог и десетог члана реалног низа унетог са тастатуре и штампати га.

```
#include <iostream>
using namespace std;
int main()
{
     float s, a[10];
     int i;
     for (i=0; i<=9; i++)
     {
           cout<<"a["<<i<<"]=";
           cin>>a[i];
     }
     s=a[0]+a[9];
     cout<<"zbir prvog i desetog clana je: "<<s<<endl;
     return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5
a[5]=6
a[6]=7
a[7]=8
a[8]=9
a[9]=10
zbir prvog i desetog clana je: 11
```

4. Написати таблицу множења бројева до 10 помоћу низова A[10], B[10] и C[100].

```
#include <iostream>
using namespace std;
int main()
{
     int i,j,k,a[10],b[10],c[100];
     for (i=0; i<10; i++)
     {
           a[i] = i+1;
           b[i] = i+1;
     }
```

```
    for (i=0; i<10; i++)
    {
          cout<<endl;
          for (j=0; j<10; j++)
          {
                k=10*i+j;
                c[k]= a[i] * b[j];
                cout<<a[j]<<"*"<<b[i]<<"="<<c[k]<<"\t";
          }
    }
    cout<<endl;
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
1*1=1   2*1=2   3*1=3   4*1=4   5*1=5   6*1=6   7*1=7   8*1=8   9*1=9   10*1=10
1*2=2   2*2=4   3*2=6   4*2=8   5*2=10  6*2=12  7*2=14  8*2=16  9*2=18  10*2=20
1*3=3   2*3=6   3*3=9   4*3=12  5*3=15  6*3=18  7*3=21  8*3=24  9*3=27  10*3=30
1*4=4   2*4=8   3*4=12  4*4=16  5*4=20  6*4=24  7*4=28  8*4=32  9*4=36  10*4=40
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25  6*5=30  7*5=35  8*5=40  9*5=45  10*5=50
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36  7*6=42  8*6=48  9*6=54  10*6=60
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49  8*7=56  9*7=63  10*7=70
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64  9*8=72  10*8=80
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81  10*9=90
1*10=10 2*10=20 3*10=30 4*10=40 5*10=50 6*10=60 7*10=70 8*10=80 9*10=90 10*10=100
```

5. Написати програм којим ће се исписати све могуће комбинације бројева 1, 2 и 3.

```
#include <iostream>
using namespace std;
int main()
{
  int i, j, k, A[3]={1,2,3};
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
      for (k=0; k<3; k++)
        cout<<A[i]<<A[j]<<A[k]<<" ";
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
111 112 113 121 122 123 131 132 133 211 212 213 221 222 223 231 232 233 311 312 313 321
322 323 331 332 333
```

6. Саставити програм којим ће се сабрати елементи низа од N члана.
7. Саставити програм којим ће се израчунати аритметичка средина за 10 иницијализованих елемената.
8. Иницијализовати низ -4, 3, -2, 0, -8, 9, -6, 7 и приказати суму другог, четвртог и шестог елемента низа.
9. Унети низ од 10 целих бројева и приказати само оне који су већи од 5.
10. Иницијализовати низ 2.1, 2.2, 3.7, 9.8, 1.4, 4.6, 5.5 и одредити индекс елемента са најмањом вредности.
11. За унети низ одредити средњу вредност елемената у низу.
12. Иницијализовати низ 1, 3, 9, 6, 5, 4, 2, 8, 7 и приказати све елементе низа.
13. Иницијализовати низ 3.1, -4.2, 2.7, -7.8, 6.4, -5.6, 1.5 и одредити индекс елемента са највећом вредности.
14. Унети низ од 11 реалних бројева и приказати само оне који су мањи од 2.7
15. За унети низ целих бројева одредити суму свих елемената који имају парне вредности.

Показивачи

Оперативна меморија је низ меморијских локација које су нумерисане целим бројевима од 1 до m, где је m капацитет меморије. Бројеви који означавају меморијску локацију зову се адресе.

Најмања меморијска јединица која може самостално да се адресира је бајт (8 битова).

Подаци се смештају у те меморијске локације и заузимају различити број бајтова.

```
char    ->   заузима 1 бајт,
int     ->   заузима 2 или 4 бајта,
float   ->   заузима 4 бајта,
double  ->   заузима 8 бајтова.
```

Показивач (pointer) је прост податак у који може да се смести адреса неке меморијске локације.

Уколико желимо да покажемо меморијску локацију у коју је смештен неки податак користићемо показиваче. Показивачи заузимају 2 или 4 бајта. Тај број зависи од опсега адреса на неком конкретном рачунару, а не од тога колики простор заузима нека промењива на коју показивач указује.

Декларисање показивача

Показиваче обележавамо знаком * и увек стоји са леве стране имена показивача.

Као и сви други подаци показивач мора да има и свој тип и дефинише се на исти начин као и било који други податак.

```
Тип * име;
```

Тип показивача мора бити истог типа као и промањива на чију адресу показује!!!

Пример: `int *a;`

Дакле, реч је о показивачу а који показује на целобројни податак. Уколико имамо више показивача истог типа испред сваког мора да се стави *.

Пример: `int *a, *b, c;` где су a и b показивачи, а c је нека целобројна променљива.

Напомена: речено је да приликом декларисања показивача испред имена мора да стоји „*“ и зато треба напоменути да не постоји разлика између следећих декларација:

```
int *pokazivac;
int* pokazivac;
int * pokazivac;
int*pokazivac;
```

Усвојићемо правило да у наставку користимо да непосредно испред показивача ставимо „*“ и да испред имена показивача додамо слово „p“ како би нагласили да се ради о показивачу.

Нпр: `int *pPokazivac;`

Најчешће се за рад са показивачима користе два оператора: „*“ и „&“.

Оператор „*“ се назива оператор дереференцирања, док се оператор „&“ назива оператором адресе.

Оператор „*“ се користи када желимо да приступимо податку уписаном на адреси на коју указује показивач, односно вредности промењиве на тој адреси.

Оператор „&“ се користи када желимо да сазнамо на којој меморискоj локацији је уписан податак.

Ови оператори нам омогућују да показивачу доделимо адресу неког податка. То се постиже на следећи начин:

```
pa=&a;
```

Тако да сад можемо и да прикажемо вредност коју смо доделили неком показивачу. Поступак је следећи:

Пример: декларисати промењиву и показивач на њу, а затим преко показивача променити вредност промењиве.

```
#include <iostream>
using namespace std;
int main ()
{
  int a,*pa;  // definišemo promenjivu
  pa = &a;    // pokazivacu dodeljujemo vrednost adrese promenjive a
  *pa = 10;   // u memorijskoj lokaciji na koju pokazuje pokazivac upisujemo vrednost 10
  cout << "promenjiva a ima vrednost " << a << endl;
  cout << "adresa promenjive a je " << pa << endl;
  cout << "na adresi promenjive a je vrednost " << *pa << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
promenjiva a ima vrednost 10
adresa promenjive a je 0x7fff9253b994
na adresi promenjive a je vrednost 10
```

Иницијализација показивача

Показивачи као и остале промењиве могу да се иницијализују, односно да им се додели почетна вредност. Како је сваки показивач адреса неког податка тако и почетна вредност мора да буде адреса неког податка. То се може урадити као у следећем примеру:

```
int a, *pa = &a;
```

Дакле, имамо променљиву a и показивач на ту променљиву. Показивачу додељујемо почетну вредност тј. адресу променљиве a, помоћу оператора „&“.

Пример: Декларисати променљиве: један цео број и два децимална, па одредити њихове адресе.

```
#include <iostream>
using namespace std;
int main ()
{
  int x=5,*px=&x;
  float y=2.1, *py=&y;
  double z=1.22, *pz=&z;
  cout<<"promenjiva x ima vrednost "<<x<<" smestenu je na adresi "<<px<<endl;
  cout<<"promenjiva y ima vrednost "<<y<<" smestenu je na adresi "<<py<<endl;
  cout<<"promenjiva z ima vrednost "<<z<<" smestenu je na adresi "<<pz<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
promenjiva x ima vrednost 5 smestenu je na adresi 0x7fff7892bed4
promenjiva y ima vrednost 2.1 smestenu je na adresi 0x7fff7892bed0
promenjiva z ima vrednost 1.22 smestenu je na adresi 0x7fff7892bec8
```

Напомена: Ако показивачу доделимо вредност 0, тада он постаје нул-показивач. Такав показивач не указује на никакву локацију у меморији и може направити много проблема у програму. Програмер је обавезан да води рачуна о таквим проблемима. Понекада се уместо изједначавања са 0, показивачу додељује вредност NULL, како би се лакше уочило да се ради о нул-показивачу.

Сигурно се питате зашто би неко користио ово?

Понекада се пре доделе вредности показивачу прво испита да ли је његова вредност NULL или у функцијама које враћају показивач, у случају грешке добијамо вредност NULL.

Аритметика показивача

Показивачи су прости целобројни подаци и над њима се могу вршити следеће аритметичке и релационе операције:

- додељивање вредности једног показивача другом
- сабирање два показивача,
- одузимање два показивача,

- сабирање показивача и целог броја,
- одузимање показивача и целог броја,
- поређење два показивача,
- поређење показивача са нулом.

Показивачи и низови

Показивачи се највише примењују код низова. Они показују адресу неког елемента низа. Први елемент низа је уствари показивач на низ, јер је само име низа управо адреса првог елемента низа. Ако те вредности нису иницијализоване приликом дефинисања дужине низа сви елементи у тој дужини узимају неку случајну вредност.

Пример: Шта би се десило ако би смо показивач увећали за 1? У том случају позиционирали би се на следећи елемент низа.

```
#include <iostream>
using namespace std;
int main ()
{
  int x[5]={4, 6, 14, 3, 6}; //inicijalizujemo niz
// Napomena: ime niza „x" – ukazuje na adresu prvog elementa niza
  int *px;                   //deklarisemo pokazivac
  px=x+1;                    //pokazivacu dodeljujemo adresu drugog elementa niza
  cout<<"vrednost drugog elementa niza je "<<*px<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
vrednost drugog elementa niza je 6
```

Ако показивачу доделимо вредност адресе низа тада увећањем тог показивача за неку вредност к приступамо елементу низа к-1.

Ово ћемо најбоље да видимо кроз пример:

```
#include <iostream>
using namespace std;
int main ()
{
  int a[10]={1,2,3,4,5,6,7,8,9,10}; // inicijalizujemo niz
  int *pa;                          // deklarisemo pokazivac
  pa=a;                             // pokazivacu dodeljujemo adresu niza
  for(int i=0;i<10;i++)
    /* u telu for petlje pomeramo pokazivac za po jednu lokaciju udesno
       i prikazujemo podatak na toj lokaciji */
    cout<<"vrednost "<<i+1<<"-tog elementa niza je "<<*(pa+i)<<endl;

  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
vrednost 1-tog elementa niza je 1
vrednost 2-tog elementa niza je 2
vrednost 3-tog elementa niza je 3
vrednost 4-tog elementa niza je 4
vrednost 5-tog elementa niza je 5
vrednost 6-tog elementa niza je 6
vrednost 7-tog elementa niza je 7
vrednost 8-tog elementa niza je 8
vrednost 9-tog elementa niza je 9
vrednost 10-tog elementa niza je 10
```

Могуће је доделити показивачу адресу било ког елемента у низу, тако што испред жељеног елемента низа наведемо оператор „&“.

Пример: Показивачу px доделити адресу 5-ог елемента, показивачу py адресу 1-ог елемента низа 1,2,3,4,5,6,7,8,9,10.

```
#include <iostream>
using namespace std;
int main ()
{
  int a[10]={1,2,3,4,5,6,7,8,9,10};
  int *px,*py;
  py = &a[4]; // pokazuje vrednost petog elementa niza
  px = &a[0]; // pokazuje vrednost prvog elementa niza
  cout<<"vrednost 5-tog elementa niza je "<<*py<<endl;
  cout<<"vrednost 1-tog elementa niza je "<<*px<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
vrednost 5-tog elementa niza je 5
vrednost 1-tog elementa niza je 1
```

Пример: Заменити места првом и последњем елементу низа a[10]={1,2,3,4,5,6,7,8,9,10}.

Напомена : Погледај пример на страни 7.

```
#include <iostream>
using namespace std;
int main ()
{
  int a[10]={1,2,3,4,5,6,7,8,9,10};
  int z,*px,*py;
  py = &a[9]; // pokazuje vrednost poslednjeg elementa niza
  px = &a[0]; // pokazuje vrednost prvog elementa niza
  cout<<"pre zamene mesta:"<<endl;
  cout<<"vrednost prvog elementa niza je "<<a[0]<<endl;
  cout<<"vrednost poslednjeg elementa niza je "<<a[9]<<endl;
  z = *px;
  *px = *py;
  *py = z;
  cout<<"posle zamene mesta:"<<endl;
  cout<<"vrednost prvog elementa niza je "<<a[0]<<endl;
  cout<<"vrednost poslednjeg elementa niza je "<<a[9]<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
pre zamene mesta:
vrednost prvog elementa niza je 1
vrednost poslednjeg elementa niza je 10
posle zamene mesta:
vrednost prvog elementa niza je 10
vrednost poslednjeg elementa niza je 1
```

Пример: Иницијализовати низ 1,2,3,4,5,6,7,8,9,10 и приказати адресу првог и последњег елемента низа, као и разлику тих адреса.

```
#include <iostream>
using namespace std;
int main ()
{
  int a[10]={1,2,3,4,5,6,7,8,9,10};
  int *p1,*p10;
  p1 = a;
  p10 = &a[9];
  cout<<"adresa prvog elementa niza je "<<p1<<endl;
```

```
    cout<<"adresa poslednjeg elementa niza je "<<p10<<endl;
    cout<<"razlika adresa poslednjeg i prvog elementa niza je "<<p10-p1<<endl;
    return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
adresa prvog elementa niza je 0x7fff40ceb900
adresa poslednjeg elementa niza je 0x7fff40ceb924
razlika adresa poslednjeg i prvog elementa niza je 9
```

Пример: Помоћу показивача приказати низ 1,2,3,4,5,6,7,8,9,10.

```
#include <iostream>
using namespace std;

int main()
{
  int a[10]={1,2,3,4,5,6,7,8,9,10};
  int *pa;
   for(pa=a;pa<a+10;pa++) /*pokazivacu pa formiramo petlju. Rekli smo vec da je ime niza
ustvari adresa prvog clana tj. clana sa indeksom 0, tako da mozemo da napisemo pa=a i
time se pokazivacu dodeljuje pocetna adresa niza. Imamo 10 elemenata pa se prema tome i
formira vrednost za kraj niza tj. a+10. U svakom prolazu vrednost pokazivaca se povecava
za 1 tj. prelazi na sledecu adresu, odnosno na sledeci element niza) */
    cout<<*pa<<" ";
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
1 2 3 4 5 6 7 8 9 10
```

Пример: Помоћу показивача унети и приказати елементе целобројног низа од 10 елемената.

```
#include <iostream>
using namespace std;
int main ()
{
  int a[10], *pa;
  cout<<"uneti 10 celih brojeva"<<endl;
  for(pa=a;pa<a+10;pa++)
    cin>>*pa;
  cout<<"Uneti niz je: ";
  for(pa=a;pa<a+10;pa++)
    cout<<*pa<<" ";
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
uneti 10 celih brojeva
2 32 4 56 47 123 47 98 9 12
Uneti niz je: 2 32 4 56 47 123 47 98 9 12
```

Питања:

1. Шта је показивач?
2. Како се декларише показивач?
3. На какву вредност указује показивач?
4. Како приказати вредност на адреси показивача?
5. Да ли показивач типа int може указивати на адресу реаланог броја?
6. Како може да се добије адреса неког податка?
7. Које операције могу да се изводе над показивачима?
8. Шта се дешава када показивачу додамо неки цео број?
9. Шта се дешава када од показивача одузмемо неки цео број?
10. Шта се дешава ако показивач има вредност 0?
11. Шта је име низа без индекса?

12. Ако низ није иницијализован на какве вредности показује показивач?
13. Како доделити адресу неког елемента низа показивачу?

Задаци за вежбање

1. Целобројној промењивој доделити показивач, а затим помоћу показивача променити вредност промењивој. Приказати вредност промењиве, показивача, као и колико простора заузимају у меморији.

```
#include <iostream>
using namespace std;
int main ()
{
  int a,*pa;
  pa = &a;
  *pa = 10;
  cout << "promenjiva a ima vrednost " << a << endl;
  cout << "adresa podatka je vrednost " << pa << endl;
  cout << "velicina podatka u memoriji je " << sizeof(a) << " bajta" <<endl;
  cout << "velicina pokazivaca u memoriji je " << sizeof(pa) << " bajta" <<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
promenjiva a ima vrednost 10
adresa podatka je vrednost 0x7fff99c6f814
velicina podatka u memoriji je 4 bajta
velicina pokazivaca u memoriji je 4 bajta
```

2. Декларисати две целобројне променљиве, па одредити њихове адресе и одредити која меморијска локација је већа.

```
#include <iostream>
using namespace std;
int main ()
{
  int x=5,*px=&x;
  int y=2, *py=&y;
  cout<<"Broj "<<x<<" smesten je na adresi "<<px<<endl;
  cout<<"Broj "<<y<<" smesten je na adresi "<<py<<endl;
  if(px>py)   // pitanje: da li adrese ovih podataka mogu biti jednake?
    cout<<"Adresa podatka x je veca od adrese podatka y."<<endl;
  else
    cout<<"Adresa podatka y je veca od adrese podatka x."<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Broj 5 smesten je na adresi 0x7fff3233b6ac
Broj 2 smesten je na adresi 0x7fff3233b6a8
Adresa podatka x je veca od adrese podatka y.
```

3. Помоћу показивача унети елементе целобројног низа од 10 елемената, а затим приказати све елементе низа са парним индексима.

```
#include <iostream>
using namespace std;
int main ()
{
  int a[10]={1,2,3,4,5,6,7,8,9,10};
  int *pa;
  for(pa=a+1;pa<a+10;pa+=2)
    cout<<*pa<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
2 4 6 8 10
```

4. Помоћу показивача унети елементе целобројног низа од 5 елемената, а затим приказати суму свих елемената низа.

```
#include <iostream>
using namespace std;
int main ()
{
  int a[5], *pa, S=0;
  cout<<"uneti 5 celih brojeva"<<endl;
  for(pa=a;pa<a+5;pa++)
  {
    cin>>*pa;
    S=S+*pa;
  }
  cout<<"suma elemenata niza je "<<S<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
uneti 5 celih brojeva
2 6 95 144 5664
suma elemenata niza je 5911
```

5. За унети низ целих бројева помоћу показивача приказати производ свих елементе низа.

6. Унети низ 10 реалних бројева и помоћу показивача сумирати елементе са индексима већим од 3 и мањим од 8.

7. За унети низ целих бројева помоћу показивача одредити суму свих елементе низа са парним индексима.

8. Унети низ од 20 реалних бројева приказати адресу елемента са најмањом вредности.

9. За унети низ целих бројева одредити адресу највећег елемента низа.

Функције

Сложени проблеми се лакше решавају ако се поделе на мање целине, па се онда свака целина посебно реши и онда се све споји у финални производ.

Делови програма којима се решавају потпроблеми зову се потпрограми. Постоје две врсте потпрограма. То су функције и процедуре, али у програмском језику C++ користе се само функције.

Функције су потпрограми који на основу одређеног броја параметара дају један резултат који се зове вредност функције.

Процедуре су потпрограми који на основу одређеног броја параметара дају више резултата.

Програмски језик C++ користи само функције као потпрограме, али оне омогућавају да постоје и додатни резултати функције. Ти додатни резулатати се зову бочни ефекти функције. Тако да, функције у C++-у у себи обједињују и функције и процедуре.

Дефинисање функција

Уколико желимо да дефиннишемо функцију општи облик је:

```
tip_funkcije ime_funkcije (tip_parametra1 ime_parametra1, tip_par2 ime_par2, … tip_parn ime_parn)
{
        telo funkcije
        return izraz ili vrednost funkcije;
}
```

tip_funkcije је било који тип који се користи и за променљиве. Уколико функција не враћа ништа користи се тип void, а ако се не наведе тип подразумева се да је int (integer).

ime_funkcije је било које име (као и име променљиве) мора да се пише спојено и не садржи специјалне знакове нити да почиње бројем или неким знаком, такође мора да се води рачуна о малим и великим словима.

tip_parametra је било који тип који је унапред дефинисан (као и за променљиве).

ime_parametra је било које име (као и за било коју другу променљиву) мора да се пише спојено и не садржи специјалне знакове нити да почиње бројем или неким знаком, такође мора да се води рачуна о малим и великим словима.
За параметре функција важи следеће:

- параметри функције нису обавезни,
- њихов број није ограничен,
- испред сваког параметра морамо написати ког је типа,
- ово нису стварне промењиве, већ копије аргумената приликом позива функције

Напомена: Ако се у функцији наведу параметри није потребно да их накнадно декларишемо. Такође није дозвољено декларисати нове промењиве које имају иста имена као и имена параметара.

Telo funkcije је низ наредби које се извршавају у функцији (своди се на било који програм који је и до сада решаван).

return izraz наредбом се постиже следећи ефекат: то је вредност која се враћа у главни програм и која, уствари, представља вредност функције. То може бити цео израз или само једна променљива која представља вредност функције. Тип ове вредности одговара типу функције. Ова наредба стоји увек на крају функције, а изузетак су функције типа void где наредба return није потребна.

Потребно је нагласити да параметри трају док траје и функција, тј. они имају локални карактер. Дакле, параметри се стварају са почетком коришћења функције, а уништавају се при напуштању функције.

Функција се пише пре почетка главног програма, мада постоје и ситуације када се пише на крају програма, али се то зове прототип функције и о томе ће бити речено нешто касније.

Пример: написати функцију збир којом ће се израчунати збир два цела броја.

ф-ја ће се звати збир и целобројног је типа и има два параметра а и б, који су такође цели бројеви

први начин:

```
int zbir (int a, int b)
{
  //u glavni program se vraca vrednost f-je koja je ustvari matematicki izraz
  return a+b;
}
```

други начин:

```
int zbir (int a, int b)
{
  int z;     // posebno se definise promenljiva z koja predstavlja zbir dva broja
  z=a+b;     // izracunavamo zbir parametara f-je
  return z;  // u glavni program vracamo vrednost promenljive z
}
```

Када завршимо са писањем функције враћамо се у главни програм који почиње функцијом **main()** и решавамо даље задатак како смо и досад радили. С тим што сад негде у главном програму треба да позовемо функцију да се она изврши.

Позивање функција

Функција у C++-у се позива тако што се наводи име функције а онда и аргументи на које функција утиче.

```
ime_funkcije (argument, argument....);
```

ime_funkcije је исто оно име које стоји и дефинисању функције.

argument је вредност којом се иницијализује параметар функције.
По броју и типу морају да одговарају типовима параметара при дефинисању функције, тј. број аргмента мора бити једнак броју параметара.
Тип првог аргумента мора бити једнак типу првог параметра, тип другог мора бити исти као и тип другог параметра, итд. Ако то није тако последице су непредвидљиве.

Функција може да се позива као операнд у изразима.

Пример: написати функцију збир којом ће се израчунати збир два цела броја. У главном програму унети бројеве и позвати функцију.

```
#include <iostream>
using namespace std;

int zbir (int a, int b) // funkcija se definise pre glavnog programa
{
  return a+b;             // u glavni program se vraca vrednost f-je tamo gde je pozvana
}

int main()              // glavni program
{
  int x, y, z;
  cout<<"unesi dva broja";
  cin>>x>>y;
  z=zbir(x, y);   // ovde pozivamo funkciju kao operand u izrazu
  cout<<"zbir je "<<z;
```

```
  return 0;
}
```

Функција може да се позове и као аргумент функције.

Пример: написати функцију збир којом ће се израчунати збир два цела броја. У главном програму унети три броја и позвати функцију.

```
#include <iostream>
using namespace std;

int zbir (int a, int b) // funkcija se definise pre glavnog programa
{
  return a+b;           // u glavni program se vraca vrednost f-je tamo gde je pozvana
}

int main()              // glavni program
{
  int x, y, z;
  cout<<"unesi tri broja: ";
  cin>>x>>y>>z;
  cout<<"njihov zbir je "<<zbir(x, zbir(y,z)); // ovde pozivamo funkciju kao argument
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi tri broja: 12 98 63
njihov zbir je 173
```

Параметри и аргументи функција

Параметри функције нису стварни подаци. Они само формално представљају податке који ће се навести приликом позивања функције и на основу чијих вредности ће се израчунати вредност функције. Ти подаци се називају аргументи функције. Параметри функције понекад се зову и формални аргументи. Подаци при позивању функције се тада називају стварни аргументи.

Након позива функције са стварним аргументима, праве се копије података, тако да приликом извршења функције било каква промена података не мења податак који је прослеђен као аргумент.

То се најлакше види на примеру:

```
#include <iostream>
using namespace std;

int A(int a)
{
  a = 3;
  return a;
}

int main()
{
  int a = 1;
  cout<<"pre poziva funkcije a="<<a<<endl;
  cout<<"unutar funkcije a="<<A(a)<<endl;
  cout<<"nakon poziva funkcije a="<<a<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
pre poziva funkcije a=1
unutar funkcije a=3
nakon poziva funkcije a=1
```

Потребно је нагласити и то да постоји потреба да се параметри функције не мењају унутар неке функције. Да би смо постигли ефекат непроменљивих података довољно је приликом дефинисања параметра ставити реч const.

Пример: Користећи функцију збир израчунати збир два броја.
Бројеве није потребно мењати, дакле то су константни подаци које треба само сабрати.

```
int zbir (const int a, const int b)
{
  int s=0;
    s=a+b;
  return s;
}
```

У програмском језику C++ поред локалних променљивих (које смо ми до сад користили) користе се и глобалне променљиве. Ове глобалне променљиве могу да се користе у свим функцијама и нема их потребе поново дефинисати.

Пример: Израчунати површину правоугаоника. Стране правоугаоника се уносе у главном програму, а површина се рачуна у функцији правоугаоник.

```
#include <iostream>
using namespace std;

float a, b, p;        // definisanje globalnih promenljivih

float pravougaonik()  // funkcija za izracunavanje povrsine pravougaonika
{
  return a*b;
}

int main ()  // glavni program
{
  cout<<"unesi stranice pravougaonika"<<endl;
  cin>>a>>b;
  //poziv funkcije kao argumenta funkcije cout
  cout<<"povrsina pravougaonika je P="<<pravougaonik()<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi stranice pravougaonika
12
6
povrsina pravougaonika je P=72
```

Прототипови функција

У програмском језику C++ дозвољено је да се функција само најави, а да се тело функције дефинише касније. Као и сама функција и прототип функције мора да има свој тип, име као и типове и имена параметара. Разликује се од обичне функције што се уместо тела функције пише „ ; “ (тачка зарез), а тело функције се пише иза краја програма.

Пример: Користећи функцију збир израчунати збир првих n елемената. У главном програму унети променљиву n која означава до ког броја вршимо сабирање и штампати резултат.

```
#include <iostream>
using namespace std;

int zbir (int n); // prototip funkcije

int main()
{
  int n;
  cout<<"uneti do kog broja ce se vrsiti sabiranje\n";
  cin>>n;
  cout<<"Suma prvih "<<n<<" brojeva je "<<zbir (n); // poziv funkcije
  return 0;
}
```

```
int zbir (int n) // definisanje funkcije zbir
{
  int s=0, i;
  for (i=1; i<=n; i++)
    s=s+i;
  return s;
}
```

Након компајлирања и покретања програма добијамо:

```
uneti do kog broja ce se vrsiti sabiranje
5
Suma prvih 5 brojeva je 15
```

Пример: Напиши функцију која одређује већи од два унета броја.

```
#include <iostream>
using namespace std;

int veci (int a, int b);

int main()
{
  int x, y;
  cout<<"uneti dva broja\n";
  cin>>x>>y;
  cout<<"Veci broj je "<<veci(x,y); //poziv funkcije
  return 0;
}

int veci (int a, int b)
{
  if (a>b)
    return a;
  else
    return b;
}
```

Након компајлирања и покретања програма добијамо:

```
uneti dva broja
12 48
Veci broj je 48
```

Пример: Напиши функције које претварају m у cm и dm. У главном програму унети m и конвертовати у cm и dm.

У оваквом решењу ће компајлер пријавити грешку, јер cm функција позива функцију dm која још није дефинисана.

```
#include <iostream>
using namespace std;

int cm(int x) {return dm(x)*10;}
int dm (int x){return x*10;}

int main()
{
  int n;
  cout<<"uneti duzinu u m\n";
  cin>>n;
  cout<<"duzina "<<n<<" m je "<<cm(n)<<" cm, tj. "<<dm(n)<<" dm"<<endl;
  return 0;
}
```

Након компајлирања добијамо:

```
In function 'int cm(int)':
error: 'dm' was not declared in this scope
```

Ако користимо прототипове функција компајлер неће пријавити грешку, иако је cm функција дефинисана пре функције dm.

```
#include <iostream>
using namespace std;

int cm(int x);
int dm (int x);

int main()
{
  int n;
  cout<<"uneti duzinu u m\n";
  cin>>n;
  cout<<"duzina "<<n<<" m je "<<cm(n)<<" cm, tj. "<<dm(n)<<" dm"<<endl;
  return 0;
}

int cm(int x) {return dm(x)*10;}
int dm (int x){return x*10;}
```

Након компајлирања и покретања програма добијамо:

```
uneti duzinu u m
3
duzina 3 mm je 300 cm, tj. 30 dm
```

Бочни ефекти функција

Свака функција враћа једну одређену вредност. Враћање вредности се врши помоћу наредбе return. Наравно ништа нас не спречава да искористимо улазне параметре функције за мењање вредности промењивих дефинисаних ван функције. То се може постићи само индиректним адресирањем параметара, који морају бити показивачи. Тиме се постиже да промена вредности података унутар тела функције изазива промену вредности и изван функције. Ово није нешто што одговара дефиницији функције, па зато има посебан назив „бочни ефекти“ функција.

Свака функција може да враћа једну вредност функције и да ствара бочне ефекте. Ако није потребно да врати неку вредност за такве функције обавезно се користи службена реч void.

```
void ime_funkcije (tip_par1 ime_par1, tip_par2 ime_par2, … tip_parn *ime_parn, tip_parN,
*ime_parN)
{
telo funkcije
}
```

НАПОМЕНА: Као параметаре функције обавезно треба навести почетне промењљиве и њихове типове, али уврстити и решења ове функције и то као показиваче и навести тип тих података.

Пример: Израчунати у оквиру функције правоугаоник обим и површину правоугаоника. У главном програму унети дужине страница и штампати површину и обим.

```
#include <iostream>
using namespace std;

/* pošto f-ja pravougaonik ima dva resenja moraju da se vrate kao bočni efekti.
   Zato ona ima 4 parametra. Parametri a i b su stranice i prenose se u f-ju
   pomoću vrednosti. *pP i *pO su pokazivači pomoću kojih se u f-ju prenose
   adrese promenljivih */

void pravougaonik (int a, int b, int *pO, int *pP)
{
  *pO=2*(a+b);
  *pP=a*b;
  // f-ja ne vraca vrednost, pa ne sadrzi naredbu return
}
```

```
int main() // glavni program
{
  // definišemo dve stranice i promenljive za povrsinu i obim pravougaonika
  int a, b, o, p;
  cout<<"unesi stranice pravougaonika"<<endl;
  cin>>a>>b;
  pravougaonik(a,b,&o,&p);
  /* pozivanje f-je pravougaonik prostom naredbom.
     Argumenti su stranice pravougaonika a i b. Kopijama njihovih vrednosti
     prilikom pozivanja f-je inicijalizovaće se parametri.
     Druga dva argumenta su &o i &p. Njima se inicijalizuju pokazivački
     parametri pP i pO.
     Na taj način će se vrednosti za rezulate za površinu i obim smestiti
     u promenljive p i o */
  cout<<"O="<<o<<endl<<"P="<<p<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi stranice pravougaonika
12 6
O=36
P=72
```

Пример: Написати функцију којом се израчунава отпорност и снага потрошача када су познати напон и струја.

```
#include <iostream>
using namespace std;

void el_kolo (float u, float i, float *pR, float *pP)
{
  *pR=u/i;
  *pP=u*i;
}

int main()
{
  float u, i, P, R;
  cout<<"unesi jacinu struje I:\n";
  cin>>i;
  cout<<"unesi napon U:\n";
  cin>>u;
  el_kolo (u, i, &R, &P);
  cout<<"otpornost potrosaca je: R="<<R<<endl;
  cout<<"snaga potrosaca je: P="<<P<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi jacinu struje I:
2
unesi napon U:
4
otpornost potrosaca je: R=2
snaga potrosaca je: P=8
```

Рекурзивне функције

То су функције које непосредно или посредно позивају саме себе. Уствари, сви рекурзивни проблеми своде се на неки циклус понављања, али са различитим параметрима. Рекурзивне функције позивају саме себе, али са измењеним аргументима. Сви рекурзивни проблеми могу да се реше помоћу циклуса и у пракси се ретко примењују. Обично рекурзивна решења трају предуго или заузима превише RAM меморије.

Најбољи пример за рекурзивне функције је решавање факторијела.

Вредност ф-је добија се условним изразом којим се или израчунава факторијел по формули или се узима константа 1. Израчунавање формуле подразумева поновно позивање ф-је факторијел са аргументом n-1 и то се понавља све док параметар не постане нула. Када параметар постане нула бира се константа 1. Ово је поступак за израчунавање међурезултата који се понавља све док се не добије вредност функције за провобитни позив (faktorijel (n)).

```
int faktorijel (int n)
{
    return (n>0)?(n*faktorijel(n-1)):1;
/* Ovom komandom vrsimo poredjenje dva podatka ciji je rezultat ponovno pozivanje f-je
ako je iskaz u zagradama tacan, u suprotnom rezultat je 1.*/
}
```

исти проблем може да се реши и помоћу функције на следећи начин:

```
int faktorijel (int n)
{
    int i, f;
    for(i=f=1; i<=n; f*=i++);
    return f;
}
```

Функције и низови

Параметри функција могу бити и низови. Тада се као параметар функције наводи тип и име низа након кога ставимо „[ ]“ да нагласимо да је у питању низ.

Пример: `int niz(int a[], int n);`

Напомена: Низови не могу бити вредност функције.

Пример: Написати функцију која одређује вредности елемената низа. У главном програму иницијализовати низ 1,2,3,4,5,6,7,8,9,10 и позвати функцију.

```
#include <iostream>
using namespace std;

int suma(int a[])
{
  int S=0;
  for(int i = 0; i<10; i++)
    S=S+a[i];
  return S;
}

int main()
{
  int x[10] = {1,2,3,4,5,6,7,8,9,10};
  cout<<"suma elemenata niza je "<<suma(x)<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
suma elemenata niza je 55
```

Већ смо научили да се приликом преноса по вредности, аргументи функције не мењају, јер се прави копија података. Такође смо научили да су имена низова у ствари показивачи на први елемент низа.

Поставља се питање шта се дешава са елементима низа када се он проследи као аргумент функцији.

То је најлакше видети на примеру.

Пример: Написати функцију која поставља вредност елемената низа на нулу. У главном програму иницијализовати низ 1,2,3,4,5,6,7,8,9,10 и позвати функцију.

```
#include <iostream>
using namespace std;

void nula(int a[])
{
  for(int i = 0; i<10; i++)
    a[i]=0;
}

int main()
{
  int x[10] = {1,2,3,4,5,6,7,8,9,10};
  cout<<"pre poziva funkcije niz x je "<<endl;
  for(int i = 0; i<10; i++)
    cout<<x[i]<<" ";
  cout<<endl;
  nula(x);
  cout<<"posle poziva funkcije niz x je "<<endl;
  for(int i = 0; i<10; i++)
    cout<<x[i]<<" ";
  cout<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
pre poziva funkcije niz x je
1 2 3 4 5 6 7 8 9 10
posle poziva funkcije niz x je
0 0 0 0 0 0 0 0 0 0
```

Овим смо видели да ако променимо елементе низа у функцији мењамо их свуда, па је у случају да не желимо да мењамо вредности у низу приликом дефинисања параметра ставити реч const.

Пример: Користећи функцију збир израчунати збир елемената низа a.

Елементе низа није потребно мењати, дакле то су константни подаци које треба само сабрати. Низ је дужине n, али се та дужина уноси у главном програму, као што се у главном програму уносе и елементи низа, а ми сад само пишемо функцију.

```
int zbir (const int a[], int n)
{
  int s=0, i;
  for (i=0; i<n; i++)
    s=s+a[i];
  return s;
}
```

Питања:

- Како се називају потпрограми у програмском језику C++?
- Шта садржи дефиниција функције?
- Шта садржи прототип функције?
- Ког типа могу бити функције?
- Ког типа је функција која не враћа вредност?
- Шта су параметри, а шта аргументи функције?
- Колико параметара може имати функција?
- Чиме се иницијализују параметри функције?
- Шта су локални подаци у функцији?
- Шта је тело функције?
- Како се враћа вредност функције?
- Како се позива функција?

- Које услове морају да испуне аргументи функције у односу на њене параметре?
- Шта омогућава прототип функције?
- Шта се постиже бочним ефектима функције?
- Како се низови преносе у функције?
- Да ли низ може да буде вредност функције?
- Како се постиже да функција не мења вредности параметара/аргумената?

Задаци за вежбање

1. Написати функцију за израчунавање хипотенузе троугла. У главном програму уносити дужине катета.

```
#include <iostream>
#include <cmath>
using namespace std;

float hipotenuza (float a, float b)
{
  float c;
  c=sqrt(a*a + pow (b,2));
  return c;
}

int main()
{
  float a, b, c;
  cout<<"unesi katete"<<endl;
  cin>>a>>b;
  c=hipotenuza (a, b);
  cout<<"hipotenuza je "<<c<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi katete
3 4
hipotenuza je 5
```

2. Написати функцију којом се израчунава степен неког броја.

```
#include <iostream>
using namespace std;

float stepen (float x, int y)
{
  int i;
  float P=1;
  for (i=1;i<=y;i++)
    P=P*x;
  return P;
}

int main()
{
  float a;
  int b;
  cout<<"Unesi broj koji stepenujes"<<endl;
  cin>>a;
  cout<<"Unesi stepen"<<endl;
  cin>>b;
  cout<<endl;
  cout<<a<<"^"<<b<<" = "<<stepen(a,b)<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Unesi broj koji stepenujes
3
Unesi stepen
3
3^3 = 27
```

3. Написати програм којим ће се формирати функција за израчунавање апсолутне вредности унетог броја.

```
#include <iostream>
using namespace std;

int aps (int x)
{
  if (x>0)
    return x;
  else
    return -x;
}
int main()
{
  int a;
  cout<<"a?"<<endl;
  cin>>a;
  cout<<"Apsolutna vrednost broja je "<<aps(a)<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
a?
-12
Apsolutna vrednost broja je 12
```

4. Написати програм у којем се у оквиру функције израчунава збир првих N природних бројева

```
#include <iostream>
using namespace std;

int zbir (int n)
{
  int i, S=0;
  for (i=1;i<=n;i++)
    S=S+i;
  return S;
}
int main()
{
  int m;
  cout<<"Do kog broja zelite sabiranje?"<<endl;
  cin>>m;
  cout<<"Zbir prvih "<<m<<" prirodnih brojeva je "<<zbir(m)<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Do kog broja zelite sabiranje?
10
Zbir prvih 10 prirodnih brojeva je 55
```

5. Написати функцију којом ће да се израчуна површина квадра. У главном програму унети податке за два квадра па одредити који је већи.

```
#include <iostream>
using namespace std;
```

```
int kvadar (int a, int b, int c)
{
  return 2*(a*b+a*c+b*c);
}

int main()
{
  int a, b, c, x, y, z, p1, p2;
  cout<<"unesi stranice prvog kvadra"<<endl;
  cin>>a>>b>>c;
  cout<<"unesi stranice drugog kvadra"<<endl;
  cin>>x>>y>>z;
  p1=kvadar (a, b, c);
  p2=kvadar (x, y, z);
  if (p1>p2)
    cout<<"prvi kvadar je veci P1="<<p1<<" P2="<<p2<<endl;
  else
    cout<<"drugi kvadar je veci P2="<<p2<<" P1="<<p1<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi stranice prvog kvadra
12 6 3
unesi stranice drugog kvadra
15 9 2
drugi kvadar je veci P2=366 P1=252
```

6. Написати функције којима се рачунају површине и обими једнакостраничног, једнакокраког, правоуглог и неједнакостраничног троугла. Све функције снимити у **trougao.h**.

```
float OjednakostranicniT(float a)
{
  return 3*a;
}

float PjednakostranicniT(float a)
{
  return a*a*sqrt(3)/4;
}

float OjednakokrakiT(float a, float b)
{
  return 2*a+b;
}

float PjednakokrakiT(float a, float b)
{
  float h;
  h = sqrt(a*a/4 + b*b);
  return a*h/2;
}

float OpravougliT(float a, float b)
{
  float c;
  c = sqrt(a*a + b*b);
  return a+b+c;
}

float PpravougliT(float a, float b)
{
  return a*b/2;
}
```

```
float OnejednakostranicniT(float a, float b, float c)
{
  return a+b+c;
}

float PnejednakostranicniT(float a, float b, float c)
{
  float s,P;
  s = OnejednakostranicniT(a,b,c)/2;
  P = sqrt(s*(s-a)*(s-b)*(s-c));
  return P;
}
```

7. Написати програм у којем се рачуна површина и обим једнакокраког троугла.

Користићемо функције које смо написали у претходном задатку.
То се ради тако што на жељеном месту у програму ставимо **#include"trougao.h".**
Снимити задатак као **trougao.cpp,** а датотеку **trougao.h** копирамо у исти директоријум где је снимљена датотека са главним програмом.

```
#include <iostream>
#include<cmath>
using namespace std;
#include"trougao.h"
int main()
{
  float x,y;
  cout<<"unesi stranice za jednakokraki trougao:"<<endl;
  cin>>x>>y;
  cout<<"obim je "<<OjednakokrakiT(x,y)<<endl;
  cout<<"povrsina je "<<PjednakokrakiT(x,y)<<endl;
  return 0;
}
```

8. Написати функцију којом ће се израчунати обим квадрата. У главном програму унети четири квадрата па одредити који од њих има највећи обим.

9. Написати функцију за унос елемената низа.

10. Написати функцију за приказ елемената низа

11. Написати функцију за израчунавање суме низа.

12. Написати функцију за израчунавање производа низа.

13. Написати функцију за одређивање најмањег елемента низа.

14. Написати функцију за одређивање највећег елемента низа.

15. Написати програм у коме ће се унети број елемената низа, унети елементе низа, одредити суму, производ, најмањи и највећи елемент низа и приказати тај низ. Користити хедер фајл.

16. Напиши програм који учитава низа од н елемената, и функцију у којој се рачуна сума низа.

17. Написати функцију која израчунава и враћа у главни програм решења квадратне једначине. У функцији испитати дискирминанту. Уколико је она мања од нуле дати извештај да постоје комплексно решење, а ако не вратити решења за x1 и x2. У главном програму унети коефицијенте квадратне једначине.

Решења квадратне једначине $a\cdot x^2-b\cdot x+c=0$ су: $x_{1,2}=\frac{-b\pm\sqrt{b^2-4\cdot a\cdot c}}{2\cdot a}$

Дискриминанта је $D=b^2-4\cdot a\cdot c$ и она треба да буде већа од нуле да би имали реална решења.

18. Написати функцију којом се израчунавају апсолутна и релативна грешка на основу мерене и тачне вредности.

19. Написати функцију која израчунава и враћа у главни програм дијагоналу, запремину и површину коцке када је позната страница.

20. Функција која као резултат враћа три вредности:

$$F_1=\frac{\sqrt{x}}{1+y} \qquad F_2=\sqrt{\frac{x}{1+y}} \qquad F_3=\frac{x}{\sqrt{1+y}}$$

Вишедимензионални низови

До сада смо помињали само низове који имају као елементе основне типове података (int, float,...).

Шта би се десило ако би елементи низова били други низови?

Тада би смо добили низове који имају вишедимензионе низове. Ако би имали две димензије то су онда дводимензионални, односно матрице. Ако би имали n димензија добили би n-димензиони низ.

Дефинисање n-димензионих низова се врши тако што се наводи тип, име и све димензије низа:

```
тип име [димензија 1][димензија 2]...[димензија n ];
```

Од свих вишедимензионалних низова најчешће се користе дводимензионални низови, па ћемо се највише позабавити њима, али све што се користи код њих може се користити и код низова са више димензија.

Дводимензионални низови - матрице

Дводимензионални низови се другачије зову матрице. Састоје се од одређеног броја редова (врста) и колона. Увек се прво наводи вредност за ред, а тек онда вредност за колону.

Дефинисање дводимензионалног низа врши се на следећи начин:

```
tip ime [duzina_reda] [duzina_kolone];
```

пример: `int matrica[2][3];`

Дакле, реч је о дводимензионалном низу који садржи целобројне елементе, који се зове матрица и има два реда и три колоне. Када би смо таквом дводимензионалном низу доделили вредности оне би изгледала овако:

$$matrica[2][3]=\begin{vmatrix}1&2&3\\4&5&6\end{vmatrix}$$

Елементи матрице имају своје индексе као и елементи једнодимензионалних низова. Само, код дводимезнионалних низова елементи се обележавају са два индекса, рецимо A(i, j). Где „i“ представља ред тј. врсту, а „j“ преставља колону.

У претходном примеру елемент matrica(1, 2) би имао вредност 5.

Напомена: И даље важи правило да сви индекси низа почињу од нуле!!!
Индекси почињу са 0, 0 и завршавају се са m-1, n-1.

За унос елемената матрице користе се две FOR петље.

Пример: Унети елементе за матрицу A која има две колоне и две врсте.

```
int A [2][2];
for (i=0; i<2; i++)
  for (j=0; j<2; j++)
    cin>>A[i][j];
```

Наравно, и приказ матрице се врши на исти начин:

Пример: Приказати елементе матрице A која има две колоне и две врсте.

```
int A [2][2];
for (i=0; i<2; i++)
  for (j=0; j<2; j++)
    cout<<A[i][j];
```

Иницијализација дводимензионалних низова

Иницијализација дводимензионалних низова се врши на сличан начин као и код једнодимензионалних низова.

Пример: `int A [2][3] = {{1, 2, 3}, {4, 5, 6}};`

дакле, прво се иницијализују елементи првог реда, а онда другог реда.

Уколико се не наведу сви елементи у некој димнзији онда се аутоматски дописују нуле.
Пример:

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

је исто што и:

```
int A [3][4] = {{1, 2, 3, 0}, {4, 5, 0, 0}, {6, 0, 0, 0}};
```

Пример: Иницијализација се може извршити и на следећи начин:

```
int A [4][3]={1, 2, 3, 4, 5, 6, 7};
```

а то је исто што и:

```
int A [4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 0, 0}, {0, 0, 0}};
```

Пример: Неправилно иницијализоване матрице су следеће:

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

зато што је иницијализовано више редова него што је дефинисано.

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

зато што има превише елемената у колонама,

```
int A [2][3]={1, 2, 3, 4, 5, 6, 7};
```

зато што има више елемената него што је дефинисано.

Пример: Написати програм којим ће се унети елементи матрице m x n (m<100 и n<100), а затим приказати матрицу.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100][100], m,n,i,j;

  // unos matrice

  cout<<"unesi dimenzije matrice\n";
  cin>>m>>n;                  // m je broj vrsta, n broj kolona
  cout<<endl<<"unesi elemente matrice"<<endl;
  for (i=0;i<m;i++)
  {
    for (j=0;j<n;j++)
    {
      cout<<"A("<<i<<","<<j<<")=";
      cin>>a[i][j];
    }
  }

  // ispisavanje matrice
  cout<<endl<<"prikaz matrice:"<<endl;
  for (i=0;i<m;i++)
  {
    for (j=0;j<n;j++)
      cout<<a[i][j]<<"\t";// ovo \t predstavlja koriscenje tabulatora prilikom stampe
    cout<<endl;
  }
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dimenzije matrice
3
3
```

```
unesi elemente matrice
A(0,0)=1
A(0,1)=2
A(0,2)=3
A(1,0)=4
A(1,1)=5
A(1,2)=6
A(2,0)=7
A(2,1)=8
A(2,2)=9

prikaz matrice:
1       2       3
4       5       6
7       8       9
```

Приступање елементима

Приступање елементима вишедимензионалног низа врши се преко индекса и то тако што се у угластим заградама наведе индекс елемента коме приступамо.

Пример: A[0][0] је адреса првог елемента матрице A. A[1][0] је адреса елемента који се налази у другом реду и првој колони.

A_{00}	A_{01}	A_{02}
A_{10}	A_{11}	A_{12}
A_{20}	A_{21}	A_{22}

Дакле ако желимо да приступимо неком конкретном елементу довољно је навести његове индексе и да видимо вредност тог елемента.

За матрице су карактеристичне колоне и редови. Уколико је потребно да обиђемо елементе матрице то можемо да урадимо на неколико начина. Најчешћи начин је обилажење преко колона или преко редова. И за један и за други метод користе се две for петље. За обилазак матрице по врстама потребно је у спољном циклусу мењати индекс врсте (реда) а у унутрашњем мењати индекс колоне.

→ A_{00}	→A_{01}	→ A_{02}	→ A_{03}
→ A_{10}	→ A_{11}	→ A_{12}	→ A_{13}
→ A_{20}	→ A_{21}	→ A_{22}	→ A_{23}

Уколико обилазимо матрицу по колонама у спољном циклусу мењамо индекс колоне, а у унтурашњем индекс реда (врсте).

↓ A_{00}	↓ A_{01}	↓ A_{02}	↓ A_{03}
↓ A_{10}	↓ A_{11}	↓ A_{12}	↓ A_{13}
↓ A_{20}	↓ A_{21}	↓ A_{22}	↓ A_{23}

Специјалну врсту матрице представља **квадратна матрица**. Она има исти број колона и број редова. Карактеристика тих матрица је главна и споредна дијагонала. Главна почиње у горњем лево углу и завршава се у доњем десном углу:

↘ A_{00}		
	↘ A_{11}	
		↘ A_{22}

Помоћна дијагонала је у супротном правцу:

		↙ A_{02}
	↙ A_{11}	
↙ A_{20}		

Индекси елемената на главној дијагонали су исти. За споредну дијагоналу збир првог и другог индекса сваког елемемента једнак је n-1. ($A_{i, n-1-i}$) где је n број колона и редова.

Дијагонала матрице дели матрицу на горњи и доњи троугао. Карактеристика тих елемената је следећа:

- за горњи троугао увек важи да је i<j,
- за доњи троугао увек важи да је i>j.

Ако посматрамо споредну дијагоналу онда је однос мало другачији.

- Изнад споредне дијагонале однос је: i+j<n-1,
- а испод дијагонале однос је: i+j>n-1

Пример: Дат је природан број n. Формирати квадратну матрицу A реда н којој су сви елементи на главној дијагонали једнаки 1, изнад ње 2, а испод ње 3.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[20][20],n,i,j;

  // popunjavanje matrice
  cout<<"unesi dimenzije kvadratne matrice\n";
  cin>>n;                 // n je broj vrsta i broj kolona
  for (i=0;i<n;i++)
  {
    for (j=0;j<n;j++)
    {
      if (i<j) a[i][j]=2;
      if (i==j) a[i][j]=1;
      if (i>j) a[i][j]=3;
    }
  }

  // ispisavanje matrice
  cout<<endl<<"prikaz matrice:"<<endl;
  for (i=0;i<n;i++)
  {
    for (j=0;j<n;j++)
      cout<<a[i][j]<<"\t";
    cout<<endl;
  }
  return 0;
```

```
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dimenzije kvadratne matrice
5

prikaz matrice:
1       2       2       2       2
3       1       2       2       2
3       3       1       2       2
3       3       3       1       2
3       3       3       3       1
```

Вишедимензионални низови и функције

Вишедимензионални низови могу бити параметри функција као и једнодимензионални низови, али и даље важи правило да не могу бити вредност функције.

Овде је важно напоменути да када користимо вишедимензионални низ као параметар функције не наводимо једино прву димензију низа, али морамо све остале.

Пример: `int matrica(int a[][10], int n);`

Овде видимо да је прва димензија изостављена, али смо морали да наведемо број колона.

Приликом позивања функције, када прослеђујемо вишедимензионални низ као аргумент треба навести само име низа без икаквих димензија.

Пример: `matrica(a,n);`

Из овога можемо закључити да се приликом позивања матрице као аргумента функције број колона мора поклапати са бројем колона параметра функције, док је број редова произвољан.

Напомена: Компајлер се неће бунити у случају да задате и прву димензију низа у оквиру параметара.
Нпр: `int matrica(int a[10][10], int n);`

Пример: Написати функцију за унос матрице целих бројева највећих димензија 100x100.

```
//funkcija za unos matrice celih brojeva:
void unos_matriceN (int a[][100], int m, int n)
{
  int i,j;
  cout<<"Unesi elemente matrice po redovima:\n";
  for(i=0;i<n;i++)
  {
    cout<<i+1<<". red"<<endl;
    for(j=0;j<m;j++)
      cin>>a[i][j];
  }
}
```

Пример: Написати функцију за приказ матрице целих бројева највећих димензија 100x100.

```
//funkcija za prikaz matrice celih brojeva:
void prikaz_matriceN (int a[][100], int m, int n)
{
  int i,j;
  cout<<endl;
  cout<<"Elementi matrice su: \n";
  for(i=0;i<n;i++)
  {
    for(j=0;j<m;j++)
      cout<<a[i][j]<<"\t";
    cout<<endl;
  }
}
```

Питања:

1. Шта су вишедимензионални низови?
2. Шта су врсте, а шта су колоне матрице?
3. Како се дефинишу вишедимензионални низови?
4. Како се иницијализују вишедимензионални низови?
5. Шта ако има мање елемената приликом иницијализације вишедимензионалних низова?
6. Како се приступа елементима вишедимензионалних низова?
7. Како се могу обилазити елементи матрице?
8. Зашто је специфична квадратна матрица?
9. Шта је главна дијагонала квадратне матрице?
10. Шта је споредна дијагонала квадратне матрице?
11. Шта је доњи и горњи троугао квадратне матрице?
12. Да ли матрице могу бити параметри функција?
13. Да ли матрице могу бити резултати функција?
14. Како се матрице прослеђују као аргументи функција?

Задаци за вежбање

1. Дат је природан број n. Формирати квадратну матрицу A реда н којој су сви елементи на главној дијагонали једнаки 1, изнад ње 2, а испод ње 3.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[20][20],n,i,j;
  // popunjavanje matrice
  cout<<"unesi dimenzije kvadratne matrice\n";
  cin>>n;                 // n je broj vrsta i broj kolona
  for (i=0;i<n;i++)
  {
    for (j=0;j<n;j++)
    {
      if (i<j) a[i][j]=2;
      if (i==j) a[i][j]=1;
      if (i>j) a[i][j]=3;
    }
  }
  // ispisavanje matrice
  cout<<endl<<"prikaz matrice:"<<endl;
  for (i=0;i<n;i++)
  {
    for (j=0;j<n;j++)
      cout<<a[i][j]<<"\t";
    cout<<endl;
  }
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dimenzije kvadratne matrice
5
prikaz matrice:
1       2       2       2       2
3       1       2       2       2
3       3       1       2       2
3       3       3       1       2
3       3       3       3       1
```

2. Написати програм којим ће се унети елементи матрице m x n (m<100 и n<100), а затим приказати

матрицу.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100][100], m,n,i,j;

  // unos matrice

  cout<<"unesi dimenzije matrice\n";
  cin>>m>>n;                 // m je broj vrsta, a n broj kolona
  cout<<endl<<"unesi elemente matrice"<<endl;
  for (i=0;i<m;i++)
  {
    for (j=0;j<n;j++)
    {
      cout<<"A["<<i<<","<<j<<"]=";
      cin>>a[i][j];
    }
  }

  // ispisavanje matrice
  cout<<endl<<"prikaz matrice:"<<endl;
  for (i=0;i<m;i++)
  {
    for (j=0;j<n;j++)
      cout<<a[i][j]<<"\t";
    cout<<endl;
  }
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dimenzije matrice
3
3
unesi elemente matrice
A[0,0]=1
A[0,1]=2
A[0,2]=3
A[1,0]=4
A[1,1]=5
A[1,2]=6
A[2,0]=7
A[2,1]=8
A[2,2]=9
prikaz matrice:
1       2       3
4       5       6
7       8       9
```

3. Унети елементе матрице m x n, па израчунати суму сваког реда, приказати суму сваког реда, а онда приказати и целу матрицу.

4. Унети елементе матрице m x n, па израчунати и приказати суму сваке колоне, а онда приказати и целу матрицу.

5. Унети елементе квадратне матрице n x n па израчунати суму целе матрице и суму главне дијагонале и приказати их.

6. Написати програм за израчунавање збира елемената матрице који се налазе:

a)изнад споредне дијагонале матрице

b)испод споредне дијагонале матрице

7. Направити хедер фајл за унос и штампање матрице целих бројева.

```
//funkcija za unos matrice celih brojeva:
void unos_matriceN (int a[100][100], int m, int n)
{
  int i,j;
  cout<<"Unesi elemente matrice po redovima:\n";
  for(i=0;i<m;i++)
  {
    cout<<i+1<<". red:"<<endl;
    for(j=0;j<n;j++)
    {
      cout<<"\t"<<j+1<<". kolona: ";
      cin>>a[i][j];
    }
  }
}

//funkcija za prikaz matrice celih brojeva:
void prikaz_matriceN (int a[][100], int m, int n)
{
  int i,j;
  cout<<endl;
  cout<<"Elementi matrice su: \n";
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
      cout<<a[i][j]<<"\t";
    cout<<endl;
  }
}
```

8. Написати програм којим ће се унети елементи матрице m x n, а затим приказати матрицу.

```
#include <iostream>
using namespace std;

#include "matrice.h"

int main ()
{
  int a[100][100], m, n, i, j;
  cout<<"unesi dimenzije matrice"<<endl;
  cout<<"broj redova: ";
  cin>>m;
  cout<<"broj kolona: ";
  cin>>n;
  unos_matriceN (a,m,n);
  prikaz_matriceN (a,m,n);
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dimenzije matrice
broj redova: 3
broj kolona: 2
Unesi elemente matrice po redovima:
1. red:
        1. kolona: 1
        2. kolona: 2
2. red:
        1. kolona: 3
        2. kolona: 4
3. red:
        1. kolona: 5
        2. kolona: 6

Elementi matrice su:
```

```
1       2
3       4
5       6
```

9. Унети елементе матрице MxN па одштампати нову која представља квадрате унете матрице.
10. Унети елементе матрице MxN па проверити колико елемената те матрице је дељиво са 8.
11. Унети елементе матрице димензија MxN па утврдити колико елемената је позитивно.
12. Унети елементе матрице димензија MxN па израчунати суму парних елемената и суму непарних елемената те матрице па на крају утврдити која сума има већу апсолутну вредност.

Дефиниција стринга

У програмирању се поред нумеричких израчунавања често јавља потреба за рад са текстом. У програмском језику C++ постоје знаковни подаци и они се обележавају са char. Али то су подаци који су предвиђени за један појединачни знак, а не за целе текстове. Уколико желимо да обрађујемо текст морамо да користимо низове знакова (char [ ]) или да користимо класу string која се налази у хедер фајлу стринг и омогућује лакшу манипулацију знаковним подацима.

Ниске знакова

Ако користимо ниске знакова, оно о чему треба водити рачуна је да се на крају овако дефинисаног низа је да се на крају налази празан знак. Он се додаје иза последњег корисног знака. Празан знак има вредност нула, али не може да се учитава и да се исписује, већ једноставно постоји да би означио крај текста. Обично се обележава са '0' или '\0'.

Програмер мора да води рачуна о томе да предвиди довољну дужину ниске знакова да би се стринг разликовао од низа који садржи знаковне податке. То се постиже предвиђањем места за празан знак на крају.

Стрингови су сложени типови података. Преводилац за језик C++ генерише по један цели број типа char и на крају додаје још један празан знак. Тако је стринг "zdravo" исто што и 'z' 'd' 'r' 'a' 'v' 'o' '\0'. Стандардне библиотечке функције за обраду текста препознају празан знак као крај текста.

С обзиром да се стрингови смештају у низове знаковних података они се обрађују као и остали низови. Појединачним знаковима у стрингу се приступа индексирањем или помоћу показивача. Крај се препознаје на основу празног знака чија је бројчана вредност једнака нули.

Иницијализација ниски знакова

Ниске знакова се иницијализују на следећи начин:

```
char ime[dužina] = string
```

где је **ime** – име ниске, **dužina** – цео број који обезбеђује колико знакова може садржати ниска и **string** – неки низ знакова под знацима навода.

Код иницијализације ниски мора да се води рачуна о дужини ниске.

Пример:

```
char niska[30] = "dobar dan!";
char niska[10] = "dobar dan!";
```

Прва иницијализација је добра, јер има места за празан знак, док је друга погрешна, јер нема места за крај стринга. Уколико се као у првом примеру стави већа дужина стринга преостали број елемената допуњавају се нулама тј. празним знацима.

Ако иницијализујемо ниску не наводећи димензије, тада ће се за димензију низа узети дужина ниске којом смо иницијализовали и додатни знак за крај ниске.

Пример:

```
char niska[] = "dobar dan!";
```

Дужина овакве ниске је 11 знакова.

Приликом писања исписаће се сви знакови до празног знака. У тексту могу да буду и белине (бланко знак, ентер, таб).

Ако смо ниску знакова иницијализовали не можемо јој доделити нову вредност тако што ћемо је изједначити са неким другим стрингом.

Пример:

```
char niska[] = "dobar dan!";
niska[] = "Laku noc!";  // погрешно!!!
niska = "Laku noc!";    // погрешно!!!
```

Унос и приказ ниске знакова

Могу се користити стандардне функције cin и cout, а постоје и неке друге као што су scanf и printf.

Пример: Написати програм који за унето име и презиме приказује оба податка у истом реду.

```
#include <iostream>
using namespace std;

int main ()
{
  char ime[10];
  char prezime[20];
  cout<<"Unesi ime:";
  cin>>ime;
  cout<<"Unesi prezime:";
  cin>>prezime;
  cout<<ime<<" "<<prezime<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Unesi ime:Petar
Unesi prezime:Petrovic
Petar Petrovic
```

Код ниске знакова не постоје оператори за обраду стринга, већ се за то користе библиотечке функције о којима ће бити речи на наредним часовима.

За сада можемо поменути да се са нискама може радити као и са свим осталим низовима.

Пример: Написати програм који ће у стринг уписати сва велика слова енглеског језика и онда их приказати.

```
#include <iostream>
using namespace std;

int main ()
{
  char a[26]; // definisemo nisku
  int i;
  for(i=0;i<26;i++)
          a[i]='A'+i; // dodeljujemo elementima niza slova
  cout<<"slova engleske abecede:\n";
  for(i=0;i<26;i++)
          cout<<a[i];
  cout<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
slova engleske abecede:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Пример: Написати програм који ће у стрингу заменити један знак.

```
#include <iostream>
using namespace std;
int main ()
{
  char a[]="proba";
  cout<<"originalni string "<<a<<endl;
  a[3]='j';
  cout<<"izmenjeni string "<<a<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
originalni string proba
izmenjeni string proja
```

Функције за обраду знаковних података

Функције за обраду знаковних података могу се поделити у три групе:
1. одређивање врсте појединачних знакова
2. обрада ниски (копирање, упоређивање...)
3. конвертовање ниски у нумеричке типове података

Одређивање врсте знакова

За одређивање врсте знакова потребно је користити библиотеку #include <cstring> .
Неке од функција које ћемо користити су:

isalnum (promenljiva)	Да ли је променљива слово или цифра
isalpha (promenljiva)	Да ли је променљива слово
islower (promenljiva)	Да ли је променљива мало слово
isupper (promenljiva)	Да ли је променљива велико слово
isdigit (promenljiva)	Да ли је променљива децимална цифра

За обраду ниски

За обраду ниски потребно је укључити и библиотеку #include <cstring>.

Код ових функција мора се водити рачуна да се резултујућој променљивој обезбеди довољно места како би прихватила почетне стрингове и посебно водити рачуна о последњем, празном знаку.

strcpy (niska1, niska2)	Преписује се ниска2 у ниску1
strcat (niska1, niska2)	Иза ниске1 додаје се ниска2
strlen (niska)	Одређује дужину ниске с тим што не броји празан знак
strcmp (niska1, niska2)	Резултат ове ф-је је цели број. Ако су две ниске исте по абецедном поретку вредност ове ф-је је нула, ако је ниска1 испред ниске2 онда је вредност мања од нуле, а ако је ниска2 испред ниске1 онда је већа од нуле.
strchr (niska1, znak)	Вредност ове ф-је је показивач на први елемент ниске1 која садржи знак. Уколико тај знак не постоји у ниски онда је вредност NULL.
strrchr (niska1, znak)	Вредност ове ф-је је покаивач на последњи елемент ниске која садржи знак. Уколико тај знак не постоји у ниски онда је вредност NULL.
srtstr (niska1, znak)	Вредност ове ф-је је показивач на први елемент ниске где се знак појављује као подниз. Уколико тај знак не постоји у ниски онда је вредност NULL.

Конверзија ниске у нумеричке податке

Конверзија ниску у нумеричке податке захтева укључивање бибилиотеке #include <cstdlib>.

atoi (string)	Ова ф-ја претвара ASCII стринг у цео број. Ако стринг садржи децимални број ф-ја враћа еквивалентни цели број. Испред децималног броја може бити произвољан број нула или празина или табулација, може бити и негативан броја, али ће функција да врати цео, позитиван број.
atof (string)	Конвертује стринг у реални број двоструке тачности.
atol (string)	Конвертује стринг у податак типа лонг.

Класа стринг

Све потребне ствари у вези класе стринг добијамо тако што укључимо хедер фајл <string>.

Пример: `#include<string>`
Након овога можемо користити нови тип података string.
Промењиве декларишемо као и све остале просте типове података:

Пример: `string a,b,c;`

Класа стринг нам омогућује да са овим типом података радимо као што смо навикли са основним типовима:
- можемо их поредити (== , != , < , <= , > , >=);
- можемо им додељивати вредности (=);
- можемо их спајати са другим стринговима (+=) и друго.

Дужина стринга

Дужина стринга је број знакова у стрингу. Постоје две функције које нам то омогућују:
lenght(), size().

Пример: одреди дужину стринга „012345678“.
```
string sSource("012345678");
cout << sSource.length() << endl;
```
за овај случај добијамо да је дужина стринга 9.

Приступ елементима стринга

Сваком знаку у оквиру неког стринга можемо приступити помоћу оператора [ ] или функцијом at().

Пример: одредити 6 знак у стрингу „abcdefg“.
```
string s("abcdefg");
cout << s[5] << endl;
```

Када би се ово извршило добили би као резултат слово „f“ које се налази на шестој позицији у стрингу, зато што се и овде рачуна да први знак има индекс 0. Наравно овде програмер мора да води рачуна о индексима.
У случају да не желимо да водимо рачуна о индексима боље је користити функцију at(), јер она проверава да ли тражени индекс има смисла и да ли постоји.

Пример:
```
string s("abcdefg");
cout << s.at(5) << endl;
```
и овде као резултат добијамо слово „f“.

Уметање знакова

За уметање знакова у стринг користимо функцију insert().

Пример: у стринг „aaaa“ уметни стрингове „bbbb“ и „cccc“.
```
string sString("aaaa");
cout << sString << endl;
sString.insert(2, string("bbbb"));
cout << sString << endl;
sString.insert(4, "cccc");
cout << sString << endl;
```
На крају добијамо резултат: aabbccccbbaa

Питања:

1. Који тип користимо за приказ појединачних знакова?
2. Који типови се користе за неки текст?
3. Како се иницијализују ниске знакова?
4. Како се иницијализују стрингови?
5. Коју библиотеку функција морамо користити за ниске знакова?
6. Коју библиотеку функција морамо користити за стрингове?
7. Које функције користимо за одређивање типова знакова?
8. Које функције користимо за обраду ниски знакова?
9. Које функције користимо за превођење у нумеричке податке?
10. Како се завршавају ниске знакова?
11. Да ли стринг може да мења своју дужину?
12. Како се стрингу додељује вредност?
13. На који начин приступамо знаковима у стрингу?
14. Када је логичније да користимо ниске знакова, а не стрингове?

Задаци за вежбање

1. Иницијализовати две ниске знакова „Добар дан.“ и Лаку ноћ.“, а затим их приказати.

```
#include <iostream>
using namespace std;

int main ()
{
  char niska1[30]="Dobar dan!";
  char niska2[30]="Laku noc!";
  cout<<niska1<<endl;
  cout<<niska2<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Dobar dan!
Laku noc!
```

2. Унети ниску знакова, а затим је приказати.

```
#include <iostream>
using namespace std;

int main ()
{
  char niska[100];
  cout<<"unesi neku rec: ";
  cin>>niska;
  cout<<niska<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi neku rec: neka
neka
```

3. Унети име и презиме ученика, а затим их приказати у једном реду.

```
#include <iostream>
using namespace std;

int main ()
{
  char ime[20];
  char prezime[20];
```

```
  cout<<"unesi ime: ";
  cin>>ime;
  cout<<"unesi prezime: ";
  cin>>prezime;
  cout<<ime<<" "<<prezime<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi ime: Petar
unesi prezime: Petrovic
Petar Petrovic
```

4. Одредити број знакова у ниски унетој са тастатуре.

```
#include <iostream>
#include <cstring>
using namespace std;

int main ()
{
  char niska[100];
  cout<<"unesi rec: ";
  cin>>niska;
  cout<<"uneta rec ima duzinu "<<strlen(niska)<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: Makarone
uneta rec ima duzinu 8
```

5. У ниски унетој са тастатуре претвори мала слова у велика.

```
#include <iostream>
#include <cstring>
using namespace std;

int main ()
{
  char niska[100];
  cout<<"unesi rec: ";
  cin>>niska;
  cout<<"uneta rec: "<<niska<<endl;
  for(int i =0; i<strlen(niska);i++)
    niska[i]=toupper(niska[i]);
  cout<<"uneta rec velikim slovima: "<<niska<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: MaKaRoNe
uneta rec: MaKaRoNe
uneta rec velikim slovima: MAKARONE
```

6. У ниски унетој са тастатуре претвори велика слова у мала.

```
#include <iostream>
#include <cstring>
using namespace std;

int main ()
{
  char niska[100];
  cout<<"unesi rec: ";
  cin>>niska;
  cout<<"uneta rec: "<<niska<<endl;
  for(int i =0; i<strlen(niska);i++)
    niska[i]=tolower(niska[i]);
```

```
  cout<<"uneta rec velikim slovima: "<<niska<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: CoCaCoLa
uneta rec: CoCaCoLa
uneta rec velikim slovima: cocacola
```

7. У ниски унетој са тастатуре претвори слово „а“ у „@“.

```
#include <iostream>
#include <cstring>
using namespace std;

int main ()
{
  char niska[100];
  cout<<"unesi rec: ";
  cin>>niska;
  cout<<"uneta rec: "<<niska<<endl;
  for(int i =0; i<strlen(niska);i++)
    if(niska[i]=='a')
      niska[i]='@';
  cout<<"prepravljena rec: "<<niska<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: paganini
uneta rec: paganini
prepravljena rec: p@g@nini
```

8. Реченицу поделити на речи и пребројати колико их има.

```
/*
Овде користимо функцију:
char * strtok ( char * str, const char * delimiters );
Она дели ниску знакова на делове.
str – ниска коју хоћемо да поделимо
delimiters – карактери који деле ниску

Ова функција враћа следеће вредности:
1.показивач на последњи део који је пронађен.
2.null показивач када нема више делова које може да нађе.
*/

#include <iostream>
#include <cstring>
using namespace std;

int main ()
{
  char recenica[] ="Ovo je jedna recenica, koja ima mnogo reci.";
  char *reci;
  int b=0;

  cout<<"delim recenicu na reci"<<endl;
  reci = strtok (recenica," ,.-!?");
  while (reci != NULL)
  {
    cout<<reci<<endl;
    b++;
    reci = strtok (NULL, " ,.-!?");
  }
  cout<< "Ukupan broj reci u recenici je "<<b<<endl;
```

```
    return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
delim recenicu na reci
Ovo
je
jedna
recenica
koja
ima
mnogo
reci
Ukupan broj reci u recenici je 8
```

9. У ниски унетој са тастатуре пребројати колико се пута понавља слово „е“.
10. У ниски унетој са тастатуре пребројати колико је пута унета нека цифра.
11. Иницијализовати два стринга: „Добар дан.“ и „Лаку ноћ.“, а затим их приказати.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s1("Dobar dan!");
  string s2("Laku noc!");
  cout<<s1<<endl;
  cout<<s2<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Dobar dan!
Laku noc!
```

12. Унети стринг, а затим га приказати.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s;
  cout<<"unesi neku rec: ";
  cin>>s;
  cout<<s<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi neku rec: neka
neka
```

13. Унети име и презиме ученика, а затим их приказати у једном реду.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string ime;
  string prezime;
  cout<<"unesi ime: ";
  cin>>ime;
```

```
  cout<<"unesi prezime: ";
  cin>>prezime;
  cout<<ime<<" "<<prezime<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi ime: Petar
unesi prezime: Petrovic
Petar Petrovic
```

ДРУГИ НАЧИН:

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string ime;
  cout << "unesi tvoje ime i prezime: ";
  getline (cin,ime);
  cout << ime << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi tvoje ime i prezime: Petar Petrovic
Petar Petrovic
```

14. Одредити број знакова у унетом стрингу.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s;
  cout<<"unesi rec: ";
  cin>>s;
  cout<<"uneta rec ima duzinu "<<s.length()<<endl;
  //cout<<"uneta rec ima duzinu "<<s.size()<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: Makarone
uneta rec ima duzinu 8
```

15. У унетом стрингу претвори слово „а“ у „@“.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s;
  cout<<"unesi rec: ";
  cin>>s;
  cout<<"uneta rec: "<<s<<endl;
  for(int i =0; i<s.size();i++)
    if(s[i]=='a')
      s[i]='@';
    cout<<"prepravljena rec: "<<s<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: paganini
uneta rec: paganini
prepravljena rec: p@g@nini
```

ДРУГИ НАЧИН:

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s;
  cout<<"unesi rec: ";
  cin>>s;
  cout<<"uneta rec: "<<s<<endl;
  for(int i =0; i<s.size();i++)
    if(s.at(i)=='a')
      s.at(i)='@';
    cout<<"prepravljena rec: "<<s<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi rec: paganini
uneta rec: paganini
prepravljena rec: p@g@nini
```

16. Стрингу додати стринг и 10 тачака.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s1("Kako dodati string i ");
  string s2("10 tackica stringu?");
  string s3;
  s3.append(s1);
  s3.append(s2);
  s3.append(10,'.');
  cout << s3 << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Kako dodati string i 10 tackica stringu?..........
```

17. У стринг уметнути други стринг.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s1("Ovo je recenica.");
  string s2(" jedna");
  s1.insert(6,s2);
  cout << s1 << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Ovo je jedna recenica.
```

18. У реченици избрисати једну реч.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s1("Ovo je jedna recenica.");
  cout << "ovo je nepromenjena recenica: ";
  cout << s1 << endl;
  s1.erase (7,6);
  cout << "ovo je promenjena recenica: ";
  cout << s1 << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
ovo je nepromenjena recenica: Ovo je jedna recenica.
ovo je promenjena recenica: Ovo je recenica.
```

19. У реченици заменити једну реч другом речи.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s1("Ovo je jedna recenica.");
  string s2("kratka");
  cout << "ovo je nepromenjena recenica: ";
  cout << s1 << endl;
  s1.replace(7,5,s2);
  cout << "ovo je promenjena recenica: ";
  cout << s1 << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
ovo je nepromenjena recenica: Ovo je jedna recenica.
ovo je promenjena recenica: Ovo je kratka recenica.
```

20. Проверити да ли је реченица палиндром.

Примери: `Ана, Ава, топот, невен, дуд, кук, око, потоп, радар, ротор, Ана воли Милована, а мене ни до године нема, и дивово око ово види, исо кувар траву коси, ево пола зеке за лопове, бобо а јеси ли сејао боб, и лаву кикирики кували, на себе је бесан, и јогурт ујутру гоји, Ана набра пар банана, удовица баци воду, иду људи, ево лове, у Риму умиру, маче једе јечам, Ања себе сања, сир има мирис, е сине жени се, а мене ту ни минуте нема, ево сада сове, нема Ката камен`

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s;
  int pola, duzina, i, p = 0;
  cout<<"unesi recenicu: ";
  getline (cin,s);

  // razmaci nisu pozeljni u recenici koju ispitujemo, pa ih brisemo
  for (int i=0;i<s.size();i++)
    if (s.at(i)==' ')
```

```
    {
      s.erase(i,1);
      i--;
    }
  cout<<"uneta recenica bez razmaka: "<<s<<endl;

  // poredimo znak sa desne strane sa znakom sa leve strane
  for(i =0; i<s.size()/2;i++)
    if(s.at(i)!=s.at(s.size()-1-i))
      p++; // ako se razlikuju povecavamo brojac za jedan

  if(p == 0)
    cout<<"rezultat: recenica je palindrom"<<endl;
  else
    cout<<"rezultat: recenica nije palindrom, razlikuju se "<< p <<" znak(a)."<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi recenicu: i lavu kikiriki kuvali
uneta recenica bez razmaka: ilavukikirikikuvali
rezultat: recenica je palindrom
```

21. Унети два имена и приказати их по абецедном редоследу.
22. Унети реч и приказати је у обрнутом редоследу.

Претраживање и сортирање низова

Уређивање низова представља једну од најзначајнијих група обраде података. Низови података могу бити уређени по растућем или опадајућем редоследу вредности елемената. Низ је уређен по растућем редоследу ако је сваки наредни елемент већи од претходног, а у падајућем редоследу ако је сваки елемент мањи од претходног..

Постоје различите технике претраживања и уређивања низова. Поступак не зависи од типа елемената низа.

Претраживање

Најједноставнији начин да се утврди да ли нека вредност постоји у низу јесте да се испитује сваки елемент. Претраживање се завршава онда када се пронађе тражени елемент. Поступак се зове секвенцијално претраживање. Да је тражена вредност пронађена зна се по томе што је циклус претраживања завршен пре краја низа.

ПАЖЊА: вредност функције је различита од нуле у случају успеха, а нула у случају неуспеха. То је у језику C++ уобичајени начин кодирања логичке истине.

Пример: У низу целих бројева одредити да ли постоји број 1.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100], n, i, b = 0;
  cout<<"unesi duzinu niza: ";
  cin>>n;
  cout<<"unesi niz\n";
  for (i=0; i<n; i++)
    cin>>a[i];
  // petlja za pretragu niza
  for (i=0; i<n; i++)
    if (a[i]==1)
      b++;
    if (b>0)
      cout<<"u nizu postoji broj 1 i javlja se "<<b<<" puta."<<endl;
    else
      cout<<"u nizu ne postoji broj 1"<<endl;
    return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi duzinu niza: 7
unesi niz
1 2 3 1 2 3 1
u nizu postoji broj 1 i javlja se 3 puta.
```

Сортирање

Уређивање, тј. сортирање низова је знатно сложенији посао. Да би се то обезбедило потребно је сваки елемент упоредити са свим осталим елементима и по потреби извршити њихову замену. Због тога сви алгоритми за сортирање низова имају више пролаза. У сваком се уочи неки елемент који се на неки начин упоређује са осталим. Постоје неколико алгоритма за сортирање низова, али ми радимо само онај најпростији, што никако не значи да је то и најоптималније решење.

Алгоритми који се користе за сортирање низова су:

1. метода избора
2. метода уметања
3. метода замене суседа
4. метода поделе

Ми ћемо обрадити методу избора. Ова метода заснива се на најједноставнијој идеји: изабрати најмањи елемент низа и довести га на прво место, затим други најмањи на друго место, па следећи најмањи и тако до краја низа.

За ову методу користе се две петље. Први циклус креће од нуле и иде до н-1 елемента, а друга петља почиње од другог елемента и иде до краја низа. Треба водити рачуна о томе да тај други елемент за сваки циклус има другачију вредност индекса јер се упоређују суседни елементи и зато почетна вредност за други циклус износи I+1.

Програмски то би изгледало овако:

```
for (i=0; i<n-1; i++)
for (j=i+1; j<n; j++)
```

затим се простом if наредбом упоређује вредност елемената:

```
if (a[i] > a[j])
```

и на крају имамо алгоритам за замену места елементима низа. За тај део користимо једну помоћну променљиву:

```
{
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
```

помоћној промељивој додељујемо вредност за и-ти елемент, затим и-ти елемент мењамо j-тим елементом. Дакле у том тренутку се врши промена вредности, тј. већи број мењамо мањим, а онда том мањем елементу (у овом тренутку вредност и за и-ти и за j-ти елемент је иста) додељујемо вредност већег елемента.

Напомена: Погледај пример на страни 7.

Пример:

```
a[i]=4, a[j]=3, 5, 6
```

треба променити места вредностима 3 и 4.

Првим кораком променљивој temp додељујемо вредност 4.

Други корак служи да се променљивој a[i] додели вредност 3.

Трећи корак служи да се променљивој a[j] додели вредност 4.

дакле,

```
a[i]=4
a[j]=3
```

1. `temp = 4`
2. `a[i] = 3 ali je i a[j] = 3`
3. `a[j] = temp tj. 4`

Пример: Сортирање низа од највећег до најмањег елемента.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100], n, i, j, temp;
  /* temp - promenljiva potrebna za zamenu
   n - broja clanova* niza
   i, j - brojaci
   a[n] - niz od n clanova   */
  cout<<"unesi duzinu niza: ";
  cin>>n;
  cout<<"unesi niz\n";
  for (i=0; i<n; i++)
    cin>>a[i];
  /* petlje za sortiranje niza*/
  for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
      if (a[i]< a[j])
```

```
      {
          temp = a[i];
          a[i] = a[j];
          a[j] = temp;
      }
      // prikaz niza
      for (i=0; i<n; i++)
    cout<<a[i]<<" ";
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi duzinu niza: 5
unesi niz
4 5 1 3 2
5 4 3 2 1
```

Пример: Написати функцију која претражује целобројни низ за жељену вредност.

```
void pronadji(int a[], int n)
{
  int i, b=0, m;
  cout<<"unesi vrednost koju trazim: ";
  cin>>m;
  for(i = 0; i < n; i++)
    if(a[i]==m)
      b++;
    if(b!=0)
      cout<<"pronadjeno je elemenata: "<<b<<endl;
    else
      cout<<"nisu pronadjeni elementi niza jednaki trazenoj vrednosti "<<endl;
}
```

Пример: Написати функцију која сортира целобројни низ по растућим вредностима.

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

Пример: Написати функцију која сортира целобројни низ по опадајућим вредностима.

```
void sortirajO(int a[], int n)
{
  int i,j, x;
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)
      if(a[i]<a[j])
      {
        x = a[i];
        a[i] = a[j];
        a[j] = x;
      }
}
```

Пример: написати програм који претражује или сортира унети низ.

```
#include <iostream>
using namespace std;

void pronadji(int a[], int n);
void sortirajR(int a[], int n);
void sortirajO(int a[], int n);
void unos(int a[], int n);
void prikaz(int a[], int n);

int main ()
{
  int a[100],n,x;
  // unos i prikaz niza
  cout<<"Koliko elemenata ima niz? ";
  cin>>n;
  unos(a,n);
  cout<<"nesortiran niz: ";
  prikaz(a,n);
  cout<<endl;
  //izbor pretraga ili sortiranje
  cout<<"Izaberi opciju:"<<endl;
  cout<<"1 - pretraga niza"<<endl;
  cout<<"2 - sortiranje niza po rastucem redosledu"<<endl;
  cout<<"3 - sortiranje niza po opadajucem redosledu"<<endl;
  cin>>x;
  switch(x)
  {
    case 1: pronadji(a,n); break;
    case 2: sortirajR(a,n); break;
    case 3: sortirajO(a,n); break;
    default: cout<<"Pogresan unos"<<endl;
  }
  cout<<endl;

  return 0;
}

void pronadji(int a[], int n)
{
  int i, b=0, m;
  cout<<"unesi vrednost koju trazim: ";
  cin>>m;
  for(i = 0; i < n; i++)
    if(a[i]==m)
      b++;
    if(b!=0)
      cout<<"pronadjeno je elemenata: "<<b<<endl;
    else
      cout<<"nisu pronadjeni elementi niza jednaki trazenoj vrednosti "<<endl;
}

void sortirajR(int a[], int n)
{
  int i,j, x;
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)
      if(a[i]>a[j])
      {
        x = a[i];
        a[i] = a[j];
        a[j] = x;
      }
      //prikaz niza
      cout<<"sortiran niz: ";
    prikaz(a,n);
}
```

```
void sortirajO(int a[], int n)
{
  int i,j, x;
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)
      if(a[i]<a[j])
      {
        x = a[i];
        a[i] = a[j];
        a[j] = x;
      }
      //prikaz niza
      cout<<"sortiran niz: ";
    prikaz(a,n);
}

void unos(int a[], int n)
{
  int i;
  cout<<"Unesi "<<n<<" elemenata niza\n";
  for (i=0; i<n; i++)
  {
    cout<<"a["<<i<<"]=";
    cin>>a[i];
  }
}

void prikaz(int a[], int n)
{
  int i;
  for (i=0; i<n; i++)
    cout<<a[i]<<" ";
  cout<<endl;
}
```

Након компајлирања и извршења програма добијамо:

```
Koliko elemenata ima niz? 5
Unesi 5 elemenata niza
a[0]=3
a[1]=6
a[2]=9
a[3]=11
a[4]=1
nesortiran niz: 3 6 9 11 1

Izaberi opciju:
1 - pretraga niza
2 - sortiranje niza po rastucem redosledu
3 - sortiranje niza po opadajucem redosledu
2
sortiran niz: 1 3 6 9 11
```

Питања:

1. Шта значи уређивање низова?
2. Како се низови могу уредити?
3. Који су могући резултати претраживања низа?
4. Које све методе постоје за сортирање низова?
5. Која је основна идеја методе избора приликом сортирања низова?

Задаци за вежбање

1. У низу целих бројева одредити да ли постоји тражени број.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100], n, i, x;
  bool nadjen = false;
  cout<<"unesi duzinu niza: ";
  cin>>n;
  cout<<"unesi niz\n";
  for (i=0; i<n; i++)
    cin>>a[i];
  cout<<"unesi trazeni broj: ";
  cin>>x;
  /* petlje za pretragu niza*/
  for (i=0; i<n; i++)
    if (a[i]==x)
      nadjen=true;
    if (nadjen)
      cout<<"u nizu postoji broj "<<x<<endl;
    else
      cout<<"u nizu ne postoji broj "<<x<<endl;
    return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi duzinu niza: 3
unesi niz
1 2 3
unesi trazeni broj: 2
u nizu postoji broj 2
```

2. Сортирање низа од највећег до најмањег елемента.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100], n, i, j, temp;
  /* temp - promenljiva potrebna za zamenu
   *  n - broja clanova* niza
   *  i, j - brojaci
   *  a[n] - niz od n clanova   */
  cout<<"unesi duzinu niza: ";
  cin>>n;
  cout<<"unesi niz\n";
  for (i=0; i<n; i++)
    cin>>a[i];
  /* petlje za sortiranje niza*/
  for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
      if (a[i] < a[j])
      {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }
      /* stampanje niza */
      for (i=0; i<n; i++)
        cout<<a[i]<<" ";
      cout<<endl;
      return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi duzinu niza: 5
unesi niz
4 37 12 58 2
58 37 12 4 2
```

3. Сортирање низа од најмањег до највећег елемента.

```
#include <iostream>
using namespace std;

int main ()
{
  int a[100], n, i, j, temp;
  /* temp - promenljiva potrebna za zamenu
   *  n - broja clanova* niza
   *  i, j - brojaci
   *  a[n] - niz od n clanova   */
  cout<<"unesi duzinu niza: ";
  cin>>n;
  cout<<"unesi niz\n";
  for (i=0; i<n; i++)
    cin>>a[i];
  /* petlje za sortiranje niza*/
  for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
      if (a[i] > a[j])
      {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }
      /* stampanje niza */
      for (i=0; i<n; i++)
        cout<<a[i]<<" ";
      cout<<endl;
      return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi duzinu niza: 5
unesi niz
4 37 12 58 2
2 4 12 37 58
```

4. Написати програм који претражује или сортира унети низ.

5. Сортирати низ целих бројева помоћу функције sort().

функција за сортирање се налази у библиотеци „algorithm“: sort(A,A+n) , где је:
sort - име функције
A - показивач на почетак низа (име низа)
A+n - показивач на последњи елемент низа (почетак низа + број елемената)

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
  int A[100],i,n;
  cout<<"Koliko ima elemenata u nizu? ";
  cin>>n;

  cout << "Unesi elemente niza: "<< endl;
  for (i = 0; i < n; i++)
    cin >> A[i];
```

```
  sort(A, A+n);

  cout << "Sortirani niz: " ;
  for (i = 0; i < n; i++)
    cout << A[i] << " ";
  cout << endl;

  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Koliko ima elemenata u nizu? 5
Unesi elemente niza:
4 37 12 58 2
Sortirani niz: 2 4 12 37 58
```

6. Сортирати низ стрингова помоћу функције sort().
7. Унети имена и презимена 10 ученика и приказати их по абецедном редоследу.
8. Сортирати низ реалних бројева помоћу функције sort().

Структуре

Структуре су сложени типови података и састоје се од одређеног броја елемената. Елементи структуре се називају поља. Поља могу бити различитих типова и обележавају се идентификаторима. Могу бити прости или сложени типови података, односно елементи структуре могу бити друге структуре или низ, као што може бити и цели, реални или знаковни податак.

Разлика између структуре и низа је у томе што низове чине елементи истог типа, а сваку структуру могу да чине елементи различитог типа. Структура је начин да се организују сложени подаци, посебно код дугачких програма, због тога што у многим ситуацијама омогућавају групи промењљивих да буду третиране као једна, уместо свака за себе.

Дефинисање структуре

Свака структура се пре употребе мора дефинисати, а онда се касније у програму декларишу промењљиве тог типа на уобичајени начин.

Дефинисање структуре се врши на следећи начин:

```
struct ime_strukture
{
tip ime_promenljive_1;
tip ime_promenljive_2:
...
};
```

Име структуре је идентификатор и мора се користити при позивању на овај тип.

Идентификатори поља важе само у њој и као такви могу се користи само при позиву структуре у којој су дефинисани. (Простије речено ако дефинишемо промењљиву X типа интеџер у структури то не значи да можемо да је користимо у програму као независну промењљиву, већ само као део структуре!!!).

Пример: Датум се састоји од следећих елемената: дан, месец и година. Месец унети као реч (јануар, фебруар...). Ако их дефинишемо као структуру то има следећи изглед:

```
struct datum          /*struct - rezervisana rec, datum je ime strukture*/
{
  int dan;            /*element polja*/
  char mesec[10];     /*element polja*/
  int godina;         /*element polja*/
};
```

Приступање елеметима структуре

Елементима поља структуре се приступа изразом следеће конструкције:

```
ime_strukture.član
```

Пример: Датум се састоји од следећих елемената: дан, месец и година. Унети и приказати датум.

```
#include <iostream>
using namespace std;

int main ()
{
  struct datum
  {
    int dan;
    char mesec[10];
    int godina;
  };
```

```
  datum d; // promenljiva d je tipa strukrure datum
  cout<<"unesi dan\n";
  cin>>d.dan;
  cout<<"unesi mesec\n";
  cin>>d.mesec;
  cout<<"unesi godinu\n";
  cin>>d.godina;
  cout<<d.dan<<". "<<d.mesec<<" "<<d.godina<<" g.\n";
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dan
1
unesi mesec
septembar
unesi godinu
2010
1. septembar 2010 g.
```

Иницијализација структуре

Структурама може да се додели и почетна вредност, да се иницијализује, али се касније у програму приступа сваком идентификатору поља (члану) посебно.

Пример: Иницијализовати структуру датум вредности 2.јануар 1986.г. Подсетимо се да се сваки знаковни податак иницијализује тако што се реч стави између знакова навода.

```
#include <iostream>
using namespace std;

int main ()
{
  struct datum
  {
    int dan;
    char mesec[10];
    int godina;
  };
  datum d = {2, "januar", 1986};
    cout<<d.dan<<". "<<d.mesec<<" "<<d.godina<<" g.\n";
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
2. januar 1986 g.
```

Пример: Написати програм који ће дефинисати структуру са називом ученик, која садржи следеће елементе: име, презиме и разред. Направити две променљиве типа ученик: Марко (чије податке уносимо са тастатуре) и Петар (чије податке ћемо иницијализовати).

```
#include <iostream>
using namespace std;

int main ()
{
  struct ucenik
  {
    char ime[15];
    char prezime [15];
    int razred;
  };
  /*Sada kada imamo strukturu mozemo da napravimo dve promenljive tipa ucenik
   gde cemo prvoj odmah dodeliti neke vrednosti (za Petra), a drugoj (Marko)
   podatke unosimo sa tastature*/
```

```
  ucenik Petar ={"Petar","Petrovic",3}, Marko;

  cout<<"unesi podatke za ucenika \n";

  /* S obzirom da ovo nije inicijalizovan ucenik moramo pristupiti svakom clanu
   polja strukture i dodeliti im vrednosti. Promenjivoj 'ime' ne mozemo
   pristupiti ako navedemo samo naziv promenljive, vec moramo pozvati i ime
   strukture kojoj pripada i tom prilikom koristi se operator '.' tacka */

  cout<<"ime:\n";
  cin>>Marko.ime;
  cout<<"prezime:\n";
  cin>>Marko.prezime;
  cout<<"razred\n" ;
  cin>>Marko.razred;

  /*prikaz podataka */

  cout<<Petar.ime<<endl<<Petar.prezime<<endl<<Petar.razred<<endl;
  cout<<Marko.ime<<endl<<Marko.prezime<<endl<<Marko.razred<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi podatke za ucenika
ime:
Marko
prezime:
Markovic
razred
4
Petar
Petrovic
3
Marko
Markovic
4
```

Пример: Требају нам следећи подаци о особама: име, презиме, и године старости. Унети податке за неколико особа и сортирати по годинама старости.

```
//Ako ne koristimo strukture, vec nizove:
#include <iostream>
using namespace std;

int main ()
{
  string ime[100], prezime[100], temp;
  int godine[100], n, i, j, x;

  cout<<"Koliko ima osoba? ";
  cin>>n;

  cout<<"Unesi podatke za osobe"<<endl;
  for(i =0; i<n; i++)
  {
    cin>>ime[i];
    cin>>prezime[i];
    cin>>godine[i];
  }

  //sortiramo nizove
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)
      if(godine[i]>godine[j])
      {
```

```
        //sortiramo imena
        temp = ime[i];
        ime[i] = ime[j];
        ime[j] = temp;

        //sortiramo prezimena
        temp = prezime[i];
        prezime[i] = prezime[j];
        prezime[j] = temp;

        //sortiramo godine
        x = godine[i];
        godine[i] = godine[j];
        godine[j] = x;
      }

  //prikaz podataka
  for(i =0; i<n; i++)
    cout<<prezime[i]<<" "<<ime[i]<<" "<<godine[i]<<endl;

  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Koliko ima osoba? 2
Unesi podatke za osobe
Petar
Petrovic
15
Marko
Markovic
17
Petrovic Petar 15
Markovic Marko 17
```

```
//Ako koristimo strukture:
#include <iostream>
using namespace std;

int main ()
{
  struct osoba{string ime,prezime;int godine;};
  int n, i, j;
  osoba X[100],temp;

  cout<<"Koliko ima osoba? ";
  cin>>n;

  cout<<"Unesi podatke za osobe"<<endl;
  for(i =0; i<n; i++)
  {
    cin>>X[i].ime;
    cin>>X[i].prezime;
    cin>>X[i].godine;
  }

  //sortiramo nizove
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)
      if(X[i].godine>X[j].godine)
      {
        //sortiramo osobe
        temp = X[i];
        X[i] = X[j];
        X[j] = temp;
```

```
        }

    //prikaz podataka
    for(i =0; i<n; i++)
      cout<<X[i].prezime<<" "<<X[i].ime<<" "<<X[i].godine<<endl;

    return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Koliko ima osoba? 2
Unesi podatke za osobe
Petar Petrovic 15
Marko Markovic 17
Petrovic Petar 15
Markovic Marko 17
```

Пример: Ако су дате координате тачака А(4,4), Б(8,2), С(4,2), које су темена правоуглог троугла написати програм који ће рачунати његову површину. У програму користити структуре за дефинисање тачака и троугла.

```
#include <iostream>
using namespace std;

int main ()
{
  //Najpre definisemo strukturu tacka
  struct tacka{int x,y;};

  //zatim strukturu trougao
  struct trougao{tacka A, B, C;};

  //inicijalizujemo trougao
  trougao T = {{4,4}, {8,2},{4,2}};

  //treba nam promenjiva za povrsinu
  float P;

  //Racunamo povrsinu
  P=((T.B.x-T.A.x)*(T.A.y-T.C.y))/2;

  cout<<"P="<<P<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
P=4
```

Пример: Написати програм који испитује да ли се нека тачка налази унутар правоугаоника. Правоугаоник је одређен доњом левом тачком и горњом десном тачком (дијагоналом). Координате за тачке које одређују правоугаоник иницијализовати, а за тачку коју испитујемо унети са тастатуре.

```
#include <iostream>
using namespace std;

int main ()
{
  //Najpre definisemo strukturu tacka
  struct tacka{int x,y;};

  //zatim strukturu pravougaonik
  struct pravougaonik{tacka A, D;};

  //definisemo pravougaonik i tacku
  pravougaonik P;
  tacka Z;
```

```
  //trebaju nam podaci
  cout<<"Unesi koordinate za tacke pravougaonika"<<endl;
  cout<<"x1=";
  cin>>P.A.x;
  cout<<"y1=";
  cin>>P.A.y;
  cout<<"x2=";
  cin>>P.D.x;
  cout<<"y2=";
  cin>>P.D.y;
  cout<<"Unesi koordinate za tacku"<<endl;
  cout<<"x=";
  cin>>Z.x;
  cout<<"y=";
  cin>>Z.y;

  //proveravamo koordinate pravougaonika i postavljamo kako nama odgovaraju
  if(P.A.x>=P.D.x)
  {
    int temp;
    temp=P.D.x;
    P.D.x=P.A.x;
    P.A.x=temp;
  }
  if(P.A.y>=P.D.y)
  {
    int temp;
    temp=P.D.y;
    P.D.y=P.A.y;
    P.A.y=temp;
  }

  //proveravamo da li je tacka u pravougaoniku
  if(Z.x>=P.A.x && Z.x<=P.D.x && Z.y>=P.A.y && Z.y<=P.D.y)
    cout<<"tacka se nalazi unutar pravougaonika\n";
  else
    cout<<"tacka se ne nalazi unutar pravougaonika\n";

  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Unesi koordinate za tacke pravougaonika
x1=1
y1=2
x2=7
y2=9
Unesi koordinate za tacku
x=4
y=8
tacka se nalazi unutar pravougaonika
```

Структуре и функције

Структуре су кориснички дефинисани типови података и као такви они се користе као и сви остали прости типови података, што значи да они могу бити кориштени у функцијама.

Структуре могу да буду параметри функција. Она се сматра једним податком, па се, без обзира на сложеност, преноси у функцију помоћу вредности.

Из истог разлога функција може да буде и резултат функције (зато што се сматра да је то један податак).

Величина сложеног податка није битна, јер се у функцију преносе сва поља из структуре. Ако желимо могуће је помоћу функције сизеоф() проверити стварно заузеће меморије сложеног податка.

Пример: Требају нам следећи подаци о особи: име, презиме, и године старости. Унети податке за

једну особу и приказати колико меморије заузима.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  struct osoba
  {
    string ime;
    string prezime;
    int godine;
  };
  osoba neko={"Janko","Jankovic",22};
  cout << "podatak zauzima "<<sizeof(neko)<<" bajtova." << endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
podatak zauzima 24 bajtova.
```

Ово нам показује релно заузеће меморије сложеног податка.

Као што је напоменуто програмер није обавезан да води рачуна о томе колика је величина, јер ће се сви подаци пренети у или из функције. Овде треба ипак напоменути да је понекад величина података значајна, па је у погледу потрошње меморије много боље пренети вредност по референци него по вредности.

Када користимо структуру као параметар функције потребно је да пре прве употребе сложеног податка тај исти дефинишемо.

Пример: Требају нам следећи подаци о особи: име, презиме, и године старости. Унети податке за једну особу и приказати их у функцији.

```
#include <iostream>
#include <string>
using namespace std;

struct osoba // definišemo pre upotrebe u funkciji
{
  string ime, prezime;
  int godine;
};

void prikazi(osoba x); // prototip funkcije

int main()
{
  osoba neko={"Janko","Jankovic",22};
  prikazi(neko);
  return 0;
}
void prikazi(osoba x)
{
  cout<<"podaci o osobi:"<<endl;
  cout<<"ime: "<<x.ime<<endl;
  cout<<"prezime: "<<x.prezime<<endl;
  cout<<"godine: "<<x.godine<<endl;
}
```

Након компајлирања и извршења програма добијамо:

```
podaci o osobi:
ime: Janko
prezime: Jankovic
godine: 22
```

Сложени подаци се могу користити и као резултати функције. Наравно и овде важи правило да се

морају дефинисати пре било ког позива.

Коришћење сложених података као резултата функције нам олакшава рад, и смањује потребу да користимо бочне ефекте функција.

Пример: Написати програм којим ће да се саберу два комплексна броја. Комплексне бројеве приказати као структуру, а операцију сабирања решити у функцији. У главном програму унети бројеве и приказати резултате.

```
#include <iostream>
using namespace std;

struct KB {float re, im;};    // struktura za kompleksne brojeve

KB sabiranje ( KB z1, KB z2); // prototip funkcije

int main ()
{
  KB z1, z2,z3;
  cout<<"unesi realni i imaginarni deo za prvi broj\n";
  cin>>z1.re>>z1.im;
  cout<<"unesi realni i imaginarni deo za drugi broj\n";
  cin>>z2.re>>z2.im;
  z3 = sabiranje(z1, z2);
  cout<<"z3="<<z3.re<<"+i"<<z3.im<<endl;
  return 0;
}

KB sabiranje ( KB z1, KB z2)
{
  KB z;
  z.re=z1.re+z2.re;
  z.im=z1.im+z2.im;
  return z;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi realni i imaginarni deo za prvi broj
11 22
unesi realni i imaginarni deo za drugi broj
12 23
z3=23+i45
```

Пример: Написати програм којим ће се рачунати обим троугла. Троугао је представљен са три тачке, а свака тачка има координате x и y. Обим троугла решити у функцији. У главном програму унети бројеве и приказати резултате.

```
#include <iostream>
#include<cmath>
using namespace std;

struct tacka {float x, y;};      // struktura za tacku

struct trougao {tacka A, B, C;}; // struktura za trougao

float rastojanje(tacka A, tacka B);

float obim (trougao T);

int main ()
{
  trougao ABC = {{2.2},{6.2},{6.5}};
  cout<<"obim trougla je "<<obim(ABC)<<endl;
  return 0;
}
```

```
float rastojanje(tacka A, tacka B)
{
  return sqrt(pow(A.x-B.x,2)+pow(A.y-B.y,2));
}

float obim (trougao T)
{
  float O,a,b,c;
  a=rastojanje(T.A,T.B);
  b=rastojanje(T.B,T.C);
  c=rastojanje(T.A,T.C);
  O=a+b+c;
  return O;
}
```

Након компајлирања и извршења програма добијамо:

```
obim trougla je 8.6
```

Структуре и показивачи

Већ смо рекли да су структуре кориснички дефинисани типови података и да имају своја поља. Пољима структуре се приступа помоћу оператора „ . “.

Пример: Структура особа има: име, презиме, и године старости.

```
struct osoba
{
  string ime;
  string prezime;
  int godine;
};
osoba neko;
neko.ime = „petar“;
```

Када користимо структуре они се понашају као и сви остали типови података, па је сасвим нормално да можемо дефинисати и показивач на неки податак.

Пример: Структура особа има: име, презиме, и године старости.

```
struct osoba
{
  string ime;
  string prezime;
  int godine;
};
osoba a, *pa;
```

Проблем који постоји код структура је да оне имају своја поља, па је приступ тим пољима преко показивача мало другачији него код осталих типова података.

Ако би желели да приступимо пољу неке структуре A, помоћу показивача, то би изгледало овако:

```
(*A).poljeN
```

Заграде око показивача су обавезне, јер је приоритет оператора „ . “ већи него приоритет оператора „ * “.

Управо због веома честе употребе показивача на структуре, направљен је нови оператор „->“ којим се приступа пољима структуре унутар показивача.

Тако да су изрази: `(*A).poljeN` и `pa->poljeN`, потпуно равноправни.

Пример: Структура тачка се састоји од координата x и y. Иницијализовати тачку (3.2, 4.3) и показивач на дату тачку. Преко показивача приказати координате тачке.

```
#include <iostream>
using namespace std;
```

```
int main ()
{
  struct tacka {float x,y;};
  tacka a={3.2,4.3}, *pa;
  pa=&a;
  cout<<(*pa).x<<endl;
  cout<<pa->y<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
3.2
4.3
```

Показивачи се могу користити и у функцијама.

Пример: Структура ученик се састоји од имена, презимена и разреда. Написати функције за унос и приказ података типа ученик. У главном програму дефинисати промењиву типа ученик и позвати обе функције.

```
#include <iostream>
#include <string>
using namespace std;

struct ucenik{string ime,prezime;int razred;};

void unos(ucenik *pu)
{
  cout<<"unesi ime, prezime i razred"<<endl;
   cin>>pu->ime;
   cin>>pu->prezime;
   cin>>pu->razred;
}

void prikaz(ucenik u)
{
  cout<<u.ime<<endl;
  cout<<u.prezime<<endl;
  cout<<u.razred<<endl;
}

int main ()
{
  ucenik X;
  unos(&X);
  prikaz(X);
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi ime, prezime i razred
Petar Petrovic 2
Petar
Petrovic
2
```

Питања:

1. Шта су структуре?
2. За шта користимо структуре?
3. Како се дефинишу структуре?
4. Ког типа могу да буду поља у структурама?
5. Да ли структуре могу да буду попља структура?
6. Да ли низови могу да буду поља структура?
7. Како се дефинишу промењиве типа дате структуре?

8. Како се иницијализују структуре?
9. Да ли структура може бити параметар функције?
10. Да ли структура може бити резултат функције?
11. Да ли постоје низови структура?
12. Како се одређује величина податка неке структуре?
13. Како се дефинише показивач на структуру?
14. Како се помоћу показивача приступа пољима структуре?

Задаци за вежбање

1. Датум се састоји од следећих елемената: дан, месец и година. Унети и приказати датум ако је могућ.

```
#include <iostream>
using namespace std;

int main ()
{
  //struktura za datum
  struct datum
  {
    int dan;
    int mesec;
    int godina;
  };

  //broj dana po mesecima
  int mesec[12]={31,28,31,30,31,30,31,31,30,31,30,31};

  //promenjiva tipa datum
  datum d;

  //korisnik unosi datum
  cout<<"unesi dan\n";
  cin>>d.dan;
  cout<<"unesi mesec\n";
  cin>>d.mesec;
  cout<<"unesi godinu\n";
  cin>>d.godina;

  //prestupna godina - svaka 4 i koja nije kraj veka
  if(d.godina%4==0 && d.godina%100!=0)
    mesec[1]=29;

  //ispitujemo da li je moguc datum
  if(d.dan<1 || d.dan>mesec[d.mesec-1] || d.mesec<1 || d.mesec>12 || d.godina<0)
    cout<<"nemoguc datum!!!"<<endl;
  else
    cout<<d.dan<<". "<<d.mesec<<". "<<d.godina<<" g.\n";
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi dan
29
unesi mesec
2
unesi godinu
2015
nemoguc datum!!!
```

2. Време се састоји од следећих елемената: сат, минут и секунда. Унети и приказати време ако је могуће.

```
#include <iostream>
using namespace std;
```

```
int main ()
{
  //struktura za datum
  struct vreme
  {
    int h;
    int m;
    int s;
  };

  //promenjiva tipa vreme
  vreme v;

  //korisnik unosi vreme koje se odmah ispituje.
  do
  {
    cout<<"unesi sate\n";
    cin>>v.h;
  }while(v.h<0);
  do
  {
    cout<<"unesi minute\n";
    cin>>v.m;
  }while(v.m<0 || v.m>59);
  do
  {
    cout<<"unesi sekunde\n";
    cin>>v.s;
  }while(v.s<0 || v.s>59);

  cout<<v.h<<":";
  if(v.m<10)
    cout<<"0";
  cout<<v.m<<":";
  if(v.m<10)
    cout<<"0";
  cout<<v.s<<endl;
  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
unesi sate
2
unesi minute
68
unesi minute
32
unesi sekunde
55
2:32:55
```

3. Требају нам следећи подаци о ученицима: име, презиме, и разред. Унети податке за неколико ученика и сортирати по разреду.

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  struct ucenik{string ime,prezime;int razred;};
  int n, i, j;
  ucenik X[100],temp;

  cout<<"Koliko ima ucenika? ";
  cin>>n;
```

```
  cout<<"Unesi podatke za ucenike"<<endl;
  for(i =0; i<n; i++)
  {
    cin>>X[i].ime;
    cin>>X[i].prezime;
    cin>>X[i].razred;
  }

  //sortiramo niz
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)
      if(X[i].razred>X[j].razred)
      {
        //rotiramo podatke
        temp = X[i];
        X[i] = X[j];
        X[j] = temp;
      }

      //prikaz podataka
      for(i =0; i<n; i++)
        cout<<X[i].prezime<<" "<<X[i].ime<<" "<<X[i].razred<<endl;

      return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Unesi podatke za ucenike
Petar Petrovic 2
Marko Markovic 4
Janko Jankovic 1
Jankovic Janko 1
Petrovic Petar 2
Markovic Marko 4
```

4. Написати програм који испитује да ли се нека тачка налази унутар правоугаоника. Правоугаоник је одређен доњом левом тачком и горњом десном тачком (дијагоналом). Координате за тачке које одређују правоугаоник иницијализовати, а за тачку коју испитујемо унети са тастатуре.

```
#include <iostream>
using namespace std;

int main ()
{
  //Najpre definisemo strukturu tacka
  struct tacka{int x,y;};

  //zatim strukturu pravougaonik
  struct pravougaonik{tacka A, D;};

  //definisemo pravougaonik i tacku
  pravougaonik P;
  tacka Z;

  //trebaju nam podaci
  cout<<"Unesi koordinate za tacke pravougaonika"<<endl;
  cout<<"x1=";
  cin>>P.A.x;
  cout<<"y1=";
  cin>>P.A.y;
  cout<<"x2=";
  cin>>P.D.x;
  cout<<"y2=";
  cin>>P.D.y;
  cout<<"Unesi koordinate za tacku"<<endl;
```

```
  cout<<"x=";
  cin>>Z.x;
  cout<<"y=";
  cin>>Z.y;

  //proveravamo koordinate pravougaonika i postavljamo kako nama odgovaraju
  if(P.A.x>=P.D.x)
  {
    int temp;
    temp=P.D.x;
    P.D.x=P.A.x;
    P.A.x=temp;
  }
  if(P.A.y>=P.D.y)
  {
    int temp;
    temp=P.D.y;
    P.D.y=P.A.y;
    P.A.y=temp;
  }

  //proveravamo da li je tacka u pravougaoniku
  if(Z.x>=P.A.x && Z.x<=P.D.x && Z.y>=P.A.y && Z.y<=P.D.y)
    cout<<"tacka se nalazi unutar pravougaonika\n";
  else
    cout<<"tacka se ne nalazi unutar pravougaonika\n";

  return 0;
}
```

Након компајлирања и извршења програма добијамо:

```
Unesi koordinate za tacke pravougaonika
x1=1
y1=2
x2=7
y2=9
Unesi koordinate za tacku
x=11
y=6
tacka se ne nalazi unutar pravougaonika
```

Датотеке

Улаз и излаз треба да омогући комуникацију програма са спољним светом ради уношења податка за обраду и приказивање или складиштење резултата. Као извор података може бити неки од улазних/излазних уређаја (тастатура, екран, штампач), али може да буде и датотека која се налази на неком од уређаја спољашње меморије.

У зависности од тога како складиштимо датотеке оне могу бити текстуалне и бинарне.

Текстуалне датотеке састоје се од низа знакова које у себи садрже и знак за прелазак у нови ред. За ове датотеке мора да се користи конверзија када се подаци смештају у меморију или се исчитавају из ње.

Бинарне датотеке се састоје из низа бајтова и садржај им је истоветан као кад се смешатају у меморију. Ове датотеке су практичније за смештај бројчаних података, јер заузимају много мање меморије.

По организацији датотеке можемо поделити на: секвенцијалне, релативне и индексне.

Секвенцијалне датотеке су карактеристичне по томе што је приступ могућ само по оном редоследу који је у датотеци. Иде се редом, знак по знак. И такав приступ се зове *секвенцијални приступ*.

Релативне датотеке су мало напредније и подацима у овим датотекама могуће је приступити на основу редног броја записа у датотеци. Овакав приступ се зове *директан приступ*.

Индексне датотеке омогућавају произвољно приступање и то помоћу дела записа који се третира као кључ тог записа. Овакав приступ се зове приступ *помоћу кључа* или *индекс-секвецијални* приступ.

Кад је језик C++ у питању постоји само једна врста датотека, а то су секвенционалне датотеке. Редови су ствар логичке, а не физичке организације. То значи да су датотеке само дугачки низови знакова. Никада се аутоматски не умеће знак за нови ред, нити функције за улаз њега региструју. То практично значи да је могуће да се један ред у датотеци пише у више редова или да се више редова пишу у једном реду. Поделу датотеке у записе мора да изврши сам програмер.

Улазна и излазна функционалност није дефинисана као део основног C++ језика, већ је обезбеђена кроз стандардне библиотеке (и на тај начин борави у std namespace). У претходним примерима смо увек укључивали **iostream** библиотеку у заглавље и користили cin и cout наредбе да урадимо једноставне I/O операције. Сада ћемо да погледамо iostream библиотека детаљније.

Коришћењем iostream хедера долазимо до великог броја класа одговорних за I/O операције.

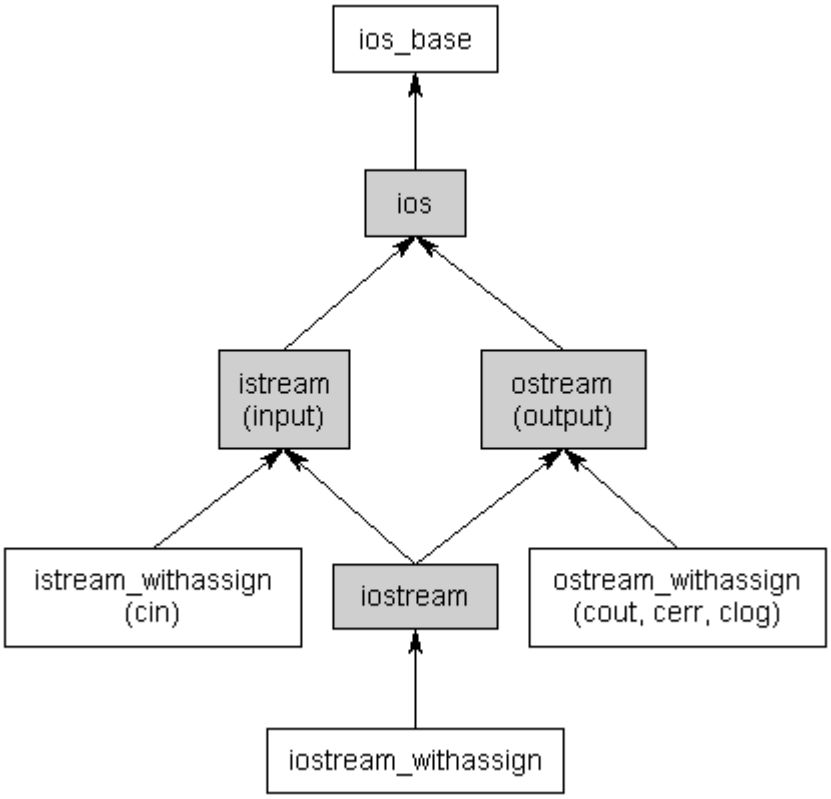


График који показује зависност функција у библиотекама

Секвенцијални приступ датотекама подразумева да имамо два различита типа приступа: улазне и излазне секвенце.

Улазне секвенце постоје у класи **istream** и за приступ таквим подацима користимо оператор **„>>“ (extraction operator)**.

Излазне секвенце постоје у класи **ostream** и за приступ таквим подацима користимо оператор **„<<“ (insertion operator)**.

Класа **iostream** омогућује рад и са улазним и излазним секвенцама.

Када је реч о датотекама постоје класе:

- **ofstream –** изведена из класе **istream**, која омогућује упис у датотеке,
- **ifstream –** изведена из класе **ostream**, која омогућује читање из датотеке,
- **fstream –** изведена из класе **iostream**, која омогућује упис и читање у/из датотеке.

При раду са датотекама постоје следеће радње:

- отварање датотеке,
- приступ датотеци,
- испитивање стања датотеке,
- затварање датотеке.

Отварање датотеке служи за успостављање везе програма са датотеком.

Приступање датотеци подразумева „комуникацију“ са датотеком: уписивање или читање података.

Испитивање стања датотеке се утврђује да ли је она уопште постоји на уређају спољне меморије.

Затварање датотеке је завршна радња којом се прекида веза имеђу датотеке и програма.

Приликом рада са датотекама користимо библиотеку функцију **fstream,** јер она садржи у себи потребне рутине за рад са датотекама.

Отварање датотека

Прва операција коју морамо урадити јесте да повежемо наш програм са реалном датотеком која нам је потребна за рад. Ово је могуће урадити помоћу функције:

```
open (filename, mode);
```

где је **filename** – име реалне датотеке, а **mode** – опциони параметар који може бити:

ios::in	Улазне операције
ios::out	Излазне операције
ios::binary	Рад са бинарним датотекама
ios::ate	Отвара датотеку и поставља почетну позицију на крај датотеке
ios::app	Све операције се врше тако што се додају на крају и при томе се чувају сви претходни подаци из датотеке
ios::trunc	Ако је датотека отворена за излазне операције њен садржај се брише пре уписа нових података

По потреби можемо комбиновати све ове параметре користећи bitwise operator OR („|“).

Пример: Ако желимо да отворимо бинарну датотеку за додавање садржаја, тада користимо следећи опциони параметар:

```
ios::out | ios::app | ios::binary
```

Свака функција **open** класа **ofstream**, **ifstream** и **fstream** има подразумевани режим који се користи уколико се датотека отвори без другог аргумента:

Класа	Подразумевани параметар
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

Затварање датотека

Када смо завршили са нашим улазним и излазним операцијама над датотеком требамо је затворити, тако да оперативни систем буде обавештен и да ти ресурси поново постану доступни. За то користимо функцију која затвара датотеку:

```
close()
```

Након ове наредбе можемо поново користити постојећи објекат да отворимо ту или неку другу датотеку.

У случају да заборавимо да затворимо датотеку пре краја програма, то ће бити урађено аутоматски.

Пример: У датотеку пример.txt уписати стринг „стринг“.

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
  ofstream MojFajl ("primer.txt");
  if (MojFajl.is_open())
  {
    MojFajl << "string";
    MojFajl.close();
  }
  else
    cout << "ne mogu da otvorim datoteku"<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Садржај датотеке primer.txt је следећи:
string
```

НАПОМЕНА: Овај резултат се не види као сви остали резултати у програму већ је потребно отворити датотеку која се налази на хард диску и потребно је отворити тај фајл као било који други документ који се чува на спољашној меморији.

Пример: Из датотеке пример.txt прочитати стринг.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main ()
{
  string S;
  ifstream MojFajl("primer.txt");
  if (MojFajl.is_open())
  {
    MojFajl >> S;
    MojFajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  cout << S << endl;
```

```
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
string
```

```
Други начин:
------------
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main ()
{
  string S;
  ifstream MojFajl("primer.txt");
  if(!MojFajl)
  {
    cout << "Ne mogu da otvorim datoteku!" << endl;
    return 1;
  }
  MojFajl >> S;
    MojFajl.close();

  if(!MojFajl.good())
  {
    cout << "Postoji greska u vezi sa datotekom!" << endl;
    return 1;
  }
  cout << S << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
string
```

Функције за рад са датотекама

У оквиру библиотеке за рад са датотекама постоји велики број функција. Неке функције смо већ користили, али постоје и нек које ћемо поменути први пут.

open()	Отвара датотеку чије име је наведено као аргумент за читање или додавање.
	Пример:
	`#include <iostream>`
	`#include <fstream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  fstream fajl;`
	`  fajl.open ("test.txt", std::fstream::in | std::fstream::out);`
	`  fajl << "string";`
	`  fajl.close();`
	`  return 0;`
	`}`

close()	Затвара отворену датотеку након што заврши све започете радње над њом.
	Пример:
	`#include <iostream>`
	`#include <fstream>`
	`using namespace std;`
	
	`int main()`
	`{`
	`  fstream fajl;`
	`  fajl.open ("test.txt", std::fstream::in | std::fstream::out);`
	`  fajl << "string";`

	`fajl.close();`
	`return 0;`
	`}`

is_open()	Проверава да ли је датотека отворена или не.
	Пример:
	`#include <iostream>`
	`#include <fstream>`
	`using namespace std;`
	`int main()`
	`{`
	`ofstream fajl("test.txt");`
	`if (fajl.is_open())`
	`{`
	`fajl << "string" << endl;`
	`fajl.close();`
	`cout << "Podaci upisani u datoteku!" << endl;`
	`}`
	`else`
	`cout << "Ne mogu da otvorim datoteku" << endl;`
	`return 0;`
	`}`

get()	Из датотеке чита један или више знакова.
	Пример:
	`#include <iostream>`
	`#include <fstream>`
	`using namespace std;`
	`int main()`
	`{`
	`char a[10];`
	`char b;`
	`ifstream fajl("test.txt");`
	`if (fajl.is_open())`
	`{`
	`b = fajl.get(); //cita jedan znak;`
	`fajl.get(a,10); //cita 10 znakova;`
	`fajl.close();`
	`}`
	`else`
	`cout << "Ne mogu da otvorim datoteku" << endl;`
	`return 0;`
	`}`

getline()	Из датотеке чита задати број знакова.
	Пример:
	`#include <iostream>`
	`#include <fstream>`
	`using namespace std;`
	`int main()`
	`{`
	`char a[100];`
	`ifstream fajl("test.txt");`
	`if (fajl.is_open())`
	`{`
	`fajl.getline(a,100); //cita 10 znakova;`
	`fajl.close();`
	`}`
	`else`
	`cout << "Ne mogu da otvorim datoteku" << endl;`
	`return 0;`
	`}`

eof()	Враћа вредност тачно ако је се са читањем датотеке дошло до краја. Пример: `#include <iostream>`

```
#include <fstream>
using namespace std;
int main()
{
  char x;
  ifstream fajl("test.cpp");
  if (fajl.is_open())
  {
    while(!fajl.eof())
    {
      x = fajl.get();
      cout << x;
    }
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

precision()

Поставља број цифара за приказ реалних бројева.

Пример:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char x;
  ofstream fajl("primer.txt");
  if (fajl.is_open())
  {
    fajl.precision(3); // upisujemo samo 3.14 u datoteku
    fajl << 3.14259 << endl;
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

width()

Поставља најмању ширину поља за упис податка.

Пример:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char x;
  ofstream fajl("primer.txt");
  if (fajl.is_open())
  {
    fajl.width(10); // upisujemo podatak u polje sirine 10 znakova
    // poravnjanje na levu stranu
    fajl << left << "Pi = ";
    fajl.width(8); // upisujemo podatak u polje sirine 8 znakova
    // poravnjanje na desnu stranu
    fajl << right << 3.14259;
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Задаци са датотекама

Пример: У датотеку „програмирање.тхт“ уписати презимена, имена и оцене из програмирања ученика једног одељења.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  struct ucenik{string ime,prezime; int ocena;};
  ucenik X[40];
  int n=0,i;
  cout<< "koliko ima ucenika?" << endl;
  cin >> n;
  for( i=0; i<n; i++)
  {
    cout << "prezime? ";
    cin >> X[i].prezime;
    cout << "ime? ";
    cin >> X[i].ime;
    cout << "ocena? ";
    cin >> X[i].ocena;
  }
  ofstream fajl("programiranje.txt");
  if (fajl.is_open())
  {
    for( i=0; i<n; i++)
    {
      fajl.width(15);
      fajl <<left<<X[i].prezime;
      fajl.width(10);
      fajl<<X[i].ime;
      fajl.width(2);
      fajl<<right<<X[i].ocena<< endl;
    }
    fajl.close();
    cout << "podaci su upisani" << endl;
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
koliko ima ucenika?
3
prezime? Petrovic
ime? Petar
ocena? 4
prezime? Markovic
ime? Marko
ocena? 1
prezime? Jankovic
ime? Janko
ocena? 0
podaci su upisani
```

Садржај датотеке programiranje.txt је следећи:

```
Petrovic        Petar       4
Markovic        Marko       1
Jankovic        Janko       0
```

Пример: Из датотеке „програмирање.txt“ учитати презимена, имена и оцене из програмирања ученика једног одељења.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  struct ucenik{string ime,prezime; int ocena;};
  ucenik X[100];
  int i=0;
  ifstream fajl("programiranje.txt");
  if (fajl.is_open())
  {
    while(!fajl.eof())
    {
      fajl >> X[i].prezime>> X[i].ime>> X[i].ocena;
      i++;
    }
    i--;
    for(int j=0; j<i;j++)
    {
      cout.width(15);
      cout <<left<<X[j].prezime;
      cout.width(10);
      cout<<X[j].ime;
      cout.width(2);
      cout<<right<<X[j].ocena<< endl;
    }
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Petrovic       Petar      4
Markovic       Marko      1
Jankovic       Janko      0
```

Пример: Из датотеке „програмирање.txt“ учитати презимена, имена и оцене из програмирања ученика једног одељења и одредити колико има недовољних оцена и приказати ученике са недовољним оценама.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  struct ucenik{string ime,prezime; int ocena;};
  ucenik X[100];
  int i=0;
  ifstream fajl("programiranje.txt");
  if (fajl.is_open())
  {
    while(!fajl.eof())
    {
      fajl >> X[i].prezime>> X[i].ime>> X[i].ocena;
      i++;
    }
    i--;
    int n=0;
    cout << "Ucenici sa nedovoljnim ocenama:" << endl;
```

```
    for(int j=0; j<i;j++)
    {
      if(X[j].ocena==1)
      {
    cout.width(15);
    cout <<left<<X[j].prezime;
    cout.width(10);
    cout<<X[j].ime;
    cout.width(2);
    cout<<right<<X[j].ocena<< endl;
    n++;
      }
    }
    cout << endl;
    cout << "ukupno sa nedovoljnim ocenama: " << n << endl;
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Ucenici sa nedovoljnim ocenama:
Markovic        Marko       1
```

Питања:

1. Које врсте датотека постоје према начину ускладиштавања?
2. Шта су то текстуалне датотеке?
3. Шта су бинарне датотеке?
4. Које врсте датотека постоје по организацији?
5. Шта су секвенцијалне датотеке?
6. Шта су релативне датотеке?
7. Шта су индексне датотеке?
8. Која је разлика између текстуалних и бинарних датотека?
9. Које су основне радње са датотекама?
10. Како се отвара датотека?
11. Како се затвара датотека?
12. Којим функцијама се врши отварање и затварање датотека?
13. Којом функцијом се чита један знак из датотеке?
14. Који оператори се користе у раду са датотекама?
15. Којим функцијама се чита више знакова из датотеке?
16. Како се форматирају текстуалне датотеке?
17. Када користимо ofstream?
18. Када користимо ifstream?
19. Како се додају подаци на крају већ постојеће датотеке?
20. Шта се дешава ако не затворим датотеку и изађемо из програма?
21. Како знамо да ли је датотека отворена?
22. Како знамо да ли смо стигли до краја датотеке приликом читања података из ње?
23. Да ли можемо у једном програму радити са више датотека?
24. Да ли можемо у једном програму писати и читати истовремено из исте датотеке?
25. Чему служи left и right приликом уписа у датотеку?

Задаци за вежбање

1. Уписати велика слова енглеске абецеде у датотеку „пример.тхт“.

```
#include <iostream>
#include <fstream>                  //biblioteka za rad sa datotekama
using namespace std;
```

```
int main()
{
  ofstream fajl("primer.txt");   //otvaramo datoteku primer.txt
  char ch;
  int i;
  if (fajl.is_open())            //proveravamo da li je datoteka otvorena
  {
    for(i=0;i<26;i++)
    {
      ch = 'A' + i;              //A->65, B->66, ... ,Z->91
      fajl << ch;                //upisujemo slovo u fajl
    }
    fajl.close();                //zatvaramo fajl
    cout << "Podaci upisani u datoteku!" << endl;  //poruka da je uspelo
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl; //poruka da ima greske
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Podaci upisani u datoteku!

Садржај датотеке primer.txt је следећи:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

2. Прочитати и приказати све знакове из датотеке „пример.txt“.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  ifstream fajl("primer.txt");
  char ch;
  int i;
  if (fajl.is_open())
  {
    while(!fajl.eof())    // ispitujemo da li je kraj datoteke
    {
      fajl >> ch;         // mogla je i naredba: fajl.get(ch);
      if(!fajl.eof())
        cout <<ch;
    }
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

3. Уписати слово, цео број и реални број у датотеку „пример.txt“.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  ofstream fajl("primer.txt");
  char x = 'a';
  int y = 2;
```

```
  float z = 3.3;
  if (fajl.is_open())
  {
    fajl << x <<endl;        // upisujemo slovo
    fajl << y <<endl;        // upisujemo ceo broj
    fajl << z <<endl;        // upisujemo realan broj
    fajl.close();
    cout << "Podaci upisani u datoteku!" << endl;
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Podaci upisani u datoteku!

Садржај датотеке primer.txt је следећи:
a
2
3.3
```

4. Прочитати слово, цео број и реални број из датотеке „пример.тхт“.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  ifstream fajl("primer.txt");
  char x = 'a';
  int y = 2;
  float z = 3.3;
  if (fajl.is_open())
  {
    fajl >> x;        // citamo slovo
    fajl >> y;        // citamo ceo broj
    fajl >> z;        // citamo realan broj
    fajl.close();
    cout << x << " " << y << " " << z << endl;
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
a 2 3.3
```

5. Унети два цела броја и сабрати их, а затим њихове вредности и резултат уписати у датотеку чије име задаје корисник.

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  string f;
  int a, b, c;
  cout << "unesi dva broja:" << endl;
  cin >> a >> b;
  c = a + b;
  cout << "unesi ime fajla u koji da upisem brojeve i rezultat:" << endl;
  cin >> f;
```

```
  /* ofstream prihvata samo niske znakova, tako da moramo da string pretvorimo u nisku!
   * To se radi pomocu funkcije c_str(). */
  ofstream fajl(f.c_str());
  if (fajl.is_open())
  {
    fajl << "a=" << a << endl;
    fajl << "b=" << b << endl;
    fajl << "c=a+b" << endl;
    fajl << "c=" << c << endl;
    fajl.close();
    cout << "Podaci upisani u datoteku!" << endl;
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
unesi dva broja:
2
3
unesi ime fajla u koji da upisem brojeve i rezultat:
primer.txt
Podaci upisani u datoteku!

Садржај датотеке primer.txt је следећи:
a=2
b=3
c=a+b
c=5
```

6. У датотеку „пример.txt“ уписати стринг „primer“.

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
  ofstream MojFajl ("primer.txt");
  if (MojFajl.is_open())
  {
    MojFajl << "primer";
    MojFajl.close();
    cout << "Podaci upisani u datoteku!" << endl;
  }
  else
    cout << "ne mogu da otvorim datoteku"<<endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Podaci upisani u datoteku!

Садржај датотеке primer.txt је следећи:
primer
```

7. Из датотеке „пример.txt“ прочитати стринг.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main ()
{
```

```
    string S;
    ifstream MojFajl("primer.txt");
    if (MojFajl.is_open())
    {
      MojFajl >> S;
      MojFajl.close();
    }
    else
      cout << "Ne mogu da otvorim datoteku" << endl;
    cout << S << endl;
    return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
primer
```

8. Написати програм који ће помоћу наставнику приликом оцењивања. У програму унети следеће податке: Име и презиме ученика, оцену на тесту (Ако ученик није радио тест унети 0). Податке снимити у датотеку „test.txt“.

Пример:

```
Milojević Miloje   5
Marković Marko     3
Jović Jovan        0
Petrović Petar     1
```

9. Направити структуру ученик са следећим подацима: презиме, име, низ од 15 оцена, оцена из владања и просечна оцена, број оправданих, неоправданих и укупних изостанака. Направити низ стрингова предмети који ће садржати називе предмета за ученика. У програму унети податке за неколико ученика и у датотеци „ucenici.txt“ уписати податке као у примеру испод:

```
Prezime Ime
Predmet 1     ocena1
Predmet 2     ocena2
.
.
.
Predmet 15    ocena15
Vladanje      ocena

Prosek        prosek ocena

Izostanci:
Opravdani     broj
Neopravdani   broj
Ukupno        broj
```

10. Унети низ целих бројева и након тога у датотеку „сортирано.txt“ уписати сортирани низ по растућем редоследу.

11. Унети низ целих бројева и након тога у датотеку „минмакс.txt“ уписати најмањи и највећи број у низу.

12. У датотеку „пример.txt“ уписати презимена, имена и плате радника.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  struct osoba{ string ime,prezime; float plata;};
  osoba X[100];
  int n=0,i;
  cout<< "koliko ima radnika?" << endl;
```

```
  cin >> n;
  for( i=0; i<n; i++)
  {
    cout << "prezime? ";
    cin >> X[i].prezime;
    cout << "ime? ";
    cin >> X[i].ime;
    cout << "plata? ";
    cin >> X[i].plata;
  }
  ofstream fajl("primer.txt");
  if (fajl.is_open())
  {
    for( i=0; i<n; i++)
    {
      fajl.width(10);
      fajl <<left<<X[i].prezime;
      fajl.width(15);
      fajl<<X[i].ime;
      fajl.width(2);
      fajl<<right<<X[i].plata<< endl;
    }
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
koliko ima radnika?
3
prezime? Petrovic
ime? Petar
plata? 140000
prezime? Makovic
ime? Marko
plata? 65489.2
prezime? Jankovic
ime? Janko
plata? 36789.4

Садржај датотеке primer.txt је следећи:
Petrovic  Petar          140000
Makovic   Marko          65489.2
Jankovic  Janko          36789.4
```

13. Из датотеке „пример.txt“ учитати презимена, имена и плате радника и приказати их на екрану.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  struct osoba{ string ime,prezime; float plata;};
  osoba X[100];
  int i=0;
  ifstream fajl("primer.txt");
  if (fajl.is_open())
  {
    while(!fajl.eof())
    {
      fajl >> X[i].prezime>> X[i].ime>> X[i].plata;
      i++;
    }
```

```
    i--;
    for(int j=0; j<i;j++)
    {
      cout.width(10);
      cout <<left<<X[j].prezime;
      cout.width(15);
      cout<<X[j].ime;
      cout.width(10);
      cout.precision(2);
      cout<<right<<fixed<<X[j].plata<< endl;
    }
    fajl.close();
  }
  else
    cout << "Ne mogu da otvorim datoteku" << endl;
  return 0;
}
```

Након компајлирања и покретања програма добијамо:

```
Petrovic  Petar          140000.00
Makovic   Marko           65489.20
Jankovic  Janko           36789.40
```