Покажувачи

Типовите на податоци што се користат во програмирањето можат да бидат:

- **статички** - се оние чија големина е однапред дефинирана (најчесто на почетокот на програмата). Тие се сместени на фиксни локации во меморијата
- **динамички** - се оние чија големина и/или структура се менува во текот на извршувањето на програмата. Тие не се сместуваат на фиксна локација во меморијата, туку на локација која во моментот на нивното креирање е слободна.

Динамичките типови на податоци можат да бидат:

- **со променлива големина** (низа битови, низа знаци, општа низа, множество, куп, ред и датотека)
- **со променлива структура** (листа, дрво, покажувач и објект).

При преведувањето на програмата потребно е преведувачот да ги знае сите променливи и нивниот тип. Променливите кои се создаваат за време на извршување на програмата, а потоа се бришат се нарекуваат **динамички променливи**. Динамичките променливи кои добиваат податоци од тип покажувач се наречени **променливи од тип покажувач или само покажувачи**. При креирањето, динамичките променливи се сместуваат во слободната внатрешна (динамичка) меморија, наречена heap-меморија.

Користењето на променливи од тип покажувач нуди две битни погодности во однос на меморијата:

1. Се проширува меморискиот простор што може да се користи за податоци во една програма
2. Со користење на покажувачките променливи во динамичката меморија, програмата може да се извршува со помала количина неопходна меморија. На пример, програмата може да поседува две многу сложени структури на податоци што не се користат истовремено. Ако овие две структури се декларираат како глобални променливи, тогаш тие се наоѓаат во сегментот за податоци и завземаат меморија за цело време на извршувањето на програмата, без оглед дали се користат во моментот или не. Но, ако се дефинирани преку покажувачи (динамички), тие се наоѓаат во динамичката меморија и ќе бидат избришани од неа по престанокот на нивното користење

Декларација на покажувачи

Покажувачот наместо податок содржи локација т.е. адреса на податокот што се наоѓа во динамичката меморија. Покажувачите се декларираат на следниот начин:

тип *име_на_покажувач;

каде што: **тип**- е типот на променлива на кои ќе покажува покажувачот, а " * " е знак за декларација на покажувач т.е покажувачка променлива.

Пример: Со int *pok; се декларира покажувачот (покажувачката променлива) pok, кој може да се користи само за да покажува на променливи од тип int.

Забелешка: важно е да се каже дека, при креирање на покажувачи (int *pok), позицијата на која се наоѓа знакот '*' нема никакво влијание на извршувањето на програмата. Имено, знакот '*' може да се наоѓа веднаш по податочниот тип (int* pok), помеѓу податочниот тип и името на покажувачот (int * pok) или до името на покажувачот (int *pok) - сите наредби ќе имаат ист ефект.

Адресен оператор &

Операторот & се нарекува адресен оператор. Тој е унарен оператор, кој ја враќа адресата на операндот по него.

Пример:

```
int y=5;
int *yPok;
yPok=&y;  // на покажувачот yPok му се доделува адресата на променливата y
```

Оператор за дереференцирање *

Променливата yPok (од предходниот пример) може да се означи и преку покажувачот yPok со *yPok. Во овој случај операторот * се нарекува **индиректен оператор** или **оператор за дереференцирање – тогаш тој ја враќа вредноста на променливата на која покажува неговиот операнд**.

Пример 1: int* ipa; int a=40; ipa=&a; cout<<*ipa; // ќе се печати 40
*ipa=120 ; cout<<a; // ќе се печати 120, бидејќи вредноста на а е промената преку покажувачот ipa

Пример 2:

```
int  a, b;   //a е од тип  'int', b е од  тип  'int'
int *a, *b;  // a и  b се покажувачи кон тип на податок од  'int'
int *a, b;   //a е покажувач , b е обична променлива од тип  'int'
int a, *b;   //a е обична променлива од тип  'int', b е покажувач
```

Иницијализација на покажувачи

Покажувачите треба да се иницијализираат или при нивна декларација или во некоја наредба за доделување.
Пример со командата ipa=&a се иницијализира покажувачот ipa – што значи дека покажувачот веќе има некаква вредност, во овој случај адресата каде што е сместена променливата а.

Еден покажувач може да се искористи повеќе пати (во една иста програма) и притоа (во различни моменти) да покажува кон различни променливи од ист тип и мемориски локации.
Доколку сакаме експлицитно да изразиме дека одреден покажувач не покажува кон ништо, тој може да биде иницијализиран на 0 (NULL). Стандардот гарантира дека не постои податок во компјутерската меморија со адреса 0.
NULL покажувач не е исто што и неиницијализиран покажувач!

Пример за користење на операторите & и *

```
#include <iostream>
 using namespace std;
 int main()
 {
    int x;
    int *pok;
    x=7;
    pok=&x;
    cout<<" Adresata na x e" << &x << "\n Vrednosta na pok e " << pok;
    cout<<"Vrednosta na x e "<< x << "\n Vrednosta na *pok e "<< *pok;
    return 0;
 }
```

Забелешка: Операторите * и & се комплементни т.е. &*aPok=*&aPok

Пример за еден покажувач кога се користи да покажува кон различни променливи од ист тип:

```
#include <iostream>
using namespace std;
int main()
{
float  c = 8.0123, d = 1.12345;
float  *pok;
pok = &c;
(*pok) += 2.0;
cout << (*pok) << endl;   //печати  10.0123
рок = &d;
(*pok)--;
cout << (*рок) << endl;  // печати 0.12345
return 0;
}
```

Разликата помеѓу операторите '&' и '*' може да се дефинираме како:

- **со '&' се чита "адреса на" и ја враќа адресата на одреден податок. На пример, "&x" ја враќа адресата на x.**
- **со '*' се чита "вредноста покажувана од" и ја враќа вредноста која се чува на одредена локација. На пример, "*p" служи за пристап до вредноста покажувана од покажувачот p.**

Пример:

```
#include <iostream>
using namespace std;
int main()
{
int a = 5, b = 2;
int *pa = &a, *pb;   // иницијализација на покажувач  pa, и декларација   на покажувач  pb
pb = &b;  //иницијализација на покажувач  pb
*pa = 3;
cout << a << " " << *pa << " " << *pb << endl;   //  печати  3 3 2
*pb = -1;
cout << b << " " << *pb << endl;    //  печати   -1 -1
*pa = *pb;
cout << a << " " << b << endl;   //   печати  -1 -1
return 0;
}
```

Големината на еден покажувач зависи од архитектурата на компјутерскиот систем на кој се извршува програмата и од оперативниот систем: 32-битен компјутерски систем ќе користи 32-битни мемориски адреси (и, соодветно, покажувачите ќе зафаќаат 4 бајти податочен простор), додека 64-битен компјутерски систем ќе користи 64-битни мемориски адреси (и, соодветно, покажувачите ќе зафаќаат 8 бајти податочен простор). Ова на никој начин не влијае на однесувањето на покажувачите: тие и понатаму можат да се искористат за пристап до податокот на кој покажуваат - без разлика на неговата големина.

Алокација на меморија

Доколку е потребно да се креира низа од N елементи, каде што N не може да се специфицира пред самото извршување на програмата, односно N може да се открие единствено за време на извршување на програмата, тогаш решението на овој проблем е со т.н. динамичко алоцирање на меморија. Во C++, тоа се прави со користење на операторите **new** (доколку сакаме да алоцираме простор за само еден елемент) или **new[N]** (доколку сакаме да алоцираме простор за N елементи). Резултатот од овие операции е покажувач до блокот меморија кој бил резервиран.

Многу е важно, по користењето на резервираната меморија (откако истата повеќе не е потребна), да се направи нејзино бришење. На тој начин, оперативниот систем потоа може да ја додели таа меморија на процесите кои навистина имаат потреба од неа. Во C++, ова се прави со користење на операторите **delete** (за еден елемент) и **delete[]** (за повеќе елементи).

Наредбата new има форма : покажувач= new тип на податок;

Со неа во меморијата се креира променлива од соодветен тип и на покажувач му се доделува адресата на креираната променлива. Во спротивно покажувачот добива вредност NULL.

Пример, ако е деклариран покажувач int *ip; тогаш со наредбата **ip=new int;** во меморијата се креира неименувана променлива и нејзината адреса му се доделува на покажувачот ip, а со **delete ip;** се ослободува меморијата на променливата на која покажувал покажувачот ip, при што тој останува недефиниран.

Пример:

```
#include <iostream>
using namespace std;
int main()
{
int *рок;
```

```
рок = new int;
*рок = 2;
cout << *рок << endl; //печати  2
delete pok; //важно!!!
int N = 100;
parr = new int[N];  //низа од 100 елементи
рок[5] = 3;
cout << рок[5] << endl;   //печати 3
delete [] рок;
return 0;
}
```

Поради фактот што секој компјутерски систем има ограничено количество на меморија, понекогаш е можно да не успее динамичкото алоцирање на меморија (new int[N]). Притоа, доколку системот не успее да резервира доволно количество меморија, програмата ќе прекине со извршување и ќе јави грешка (runtime error).

Доколку сакаме самите да го решиме проблемот со евентуалниот недостаток на меморија, тогаш треба да го искористиме **специјалниот параметар (nothrow)** - **дефиниран во датотеката <new>,** и самите да провериме дали алокацијата на меморија завршила успешно - доколку се случил проблем, покажувачот ќе има вредност 0 (NULL).

Пример на програма:

```
#include <iostream>
#include <new>
using namespace std;
int main()
{
int N = 2000000000;
int *рок = new (nothrow) int[N];
if (рокr == 0)
{
cout << "Greshka: Nema dovolno memorija!" << endl;
}
else
{
  naredbi so nizata;
}
delete [] pok;
return 0;
}
```

Покажувачи и низи

Доколку е декларирана и иницијализирана низата: **int a[5]={1,2,4,8,16};** Со наредбата: p=a; на покажувачот p му се доделува вредноста на адресата на првиот елемент на низата a[] т.е. адресата на a[0], додека вредноста на првиот елемент од низата може да се добие со *a или a[0].

Пример 1:

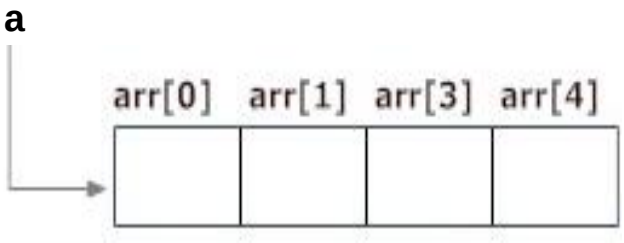
```
#include <iostream>

using namespace std;
const int MAX = 3;
int main ()
{
   int  niza[MAX] = {10, 100, 200};
   int *рок[MAX];
   for (int i = 0; i < MAX; i++)
   {
      рок[i] = &niza[i]; // запишување на адресите на елементите од niza во низа со покажувачи pok
   }
```

```
  for (int i = 0; i < MAX; i++)
  {
    cout << pok[i]<<"  "<<*pok[i] << endl;   //печатење на адресите и елементите од низата
  }
  return 0;
}
```

Пример 2:

```
#include <iostream>
using namespace std;
const int MAX = 4;
int main ()
{
  char *ime[MAX] = {"Petar", "Nikola", "Tamara", "Stefan", "Monika"};
  for (int i = 0; i < MAX; i++)
  {
    cout << ime[i] << endl;
  }
  return 0;
}
```

<u>Забелешка:</u> Врската меѓу покажувачи и низи, може да се опише со следното:
Ако е дефинирана низа и покажувач: int arr[4]; int *a; a=arr;



Во овој случај покажувачот a , покажува на првиот член од низата arr т.е на arr[0], па за секој елемент од низата важи следното:
&arr[i] е еднакво со (a+i) , а вредноста на arr[i] е еднакво со *(a+i).

Пример 3:

```
#include <iostream>
using namespace std;
int main ()
{
  double niza[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
  double *p;
  p = niza;
  for ( int i = 0; i < 5; i++ )
  {
     cout << *(p + i) << endl;  // се печатат елементите од niza
  }
  return 0;
}
```

Пример 4:

```
#include <iostream>
using namespace std;
int main ()
{
  int number[5];
  int * p, n;
  p = number;   *p = 10;
  p++;  *p = 20;
  p = &number[2];   *p = 30;
  p = number + 3;   *p = 40;
  p = number;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << number[n] << ", "; // Се печати 10, 20, 30, 40, 50,
 return 0;
}
```

Пример 5:

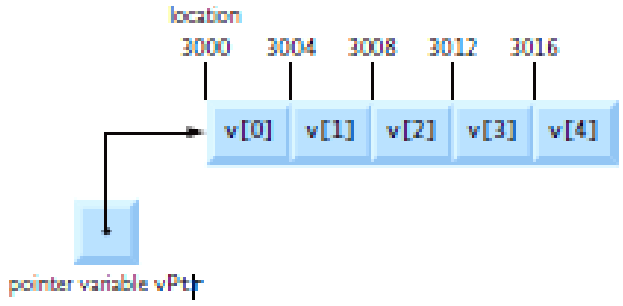
```
#include <iostream>
using namespace std;
int main ()
{
   int  niza[5] = {10, 100, 200, 3000, 5000};
   int  i, *pok;
   pok = niza;
   for ( i = 0; i < 5; i++)
   {
      cout<<"Adresata na niza ["<<i<<"]="<< pok;
      cout<<"Vrednosta na niza ["<<i<<"]="<< *pok;
      pok++;  // придвижување на покажувачот на следна локација
   }
   return  0;
}
```

Аритметика со покажувачи

Постојат следните аритметички оператори за покажувачи: ++, --, +, -, +=, -=

Пример:

int v[5]; int *vPtr ; vPtr = v; // vPtr покажува на локација 3000



vPtr += 2; // ќе се оди на локација 3008 (бидејќи кај int адресите се зголемуваат за 4 т.е. 3000 + 2 * 4)
Ако покажувачот е на локација 3016 со наредбата vPtr -= 4; тогаш покажувачот го враќаме на локација 3000.
Со изразите ++vPtr и vPtr++ се носи покажувачот до следниот елемент во низата.
Со изразите --vPtr и vPtr-- се носи покажувачот до предходниот елемент во низата.
Ако се дефинираат два покажувачи кон низата т.е. vPtr1 и vPtr2, при што vPtr1 е на локација 3000, а vPtr2 е на локација 3008 тогаш со наредбата vPtr2-vPtr1 ќе се добие 2.

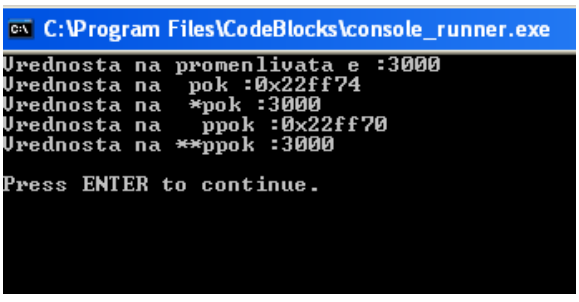
<u>Забелешка:</u> Адресите каде се зачувуваат податоците во динамичка меморија се претставени со хексадекаден број пр: bff5a400 , bff5a3f6 и за колку ќе се зголемува (намалува) вредноста на адресата зависи од типот на податокот т.е.: кај тип на податок char – адресите се зголемуваат за 1, кај тип на податок short int – адресите се зголемуваат за 2, кај тип на податок int – адресите се зголемуваат за 4, кај тип на податок float – адресите се зголемуваат за 4, кај тип на податок double – адресите се зголемуваат за 8 и.т.н.

Пример 6:

```
#include <iostream>
using namespace std;
int main ()
{
  int  niza[5] = {10, 100, 200, 3000, 5000};
  int  i, *pok;
  pok = niza;  i=0;
  while ( pok <= &niza[4] )  // споредување на покажувач
  {
    cout<<"Adresata na niza ["<<i<<"]="<< pok;
    cout<<"Vrednosta na niza ["<<i<<"]="<< *pok;
    pok++;  // придвижување на покажувачот на следна локација
    i++;
  }
  return  0;
}
```

<u>Покажувач кон покажувач</u>: Покажувач на покажувач е форма на повеќекратно покажување кон некоја променлива. Кога ќе дефинираме покажувач на покажувачот, првиот покажувачот ја содржи адреса на вториот покажувач, што ја кажува локацијата која ја содржи вистинската вредност на променливата како што е прикажано на сликата:


```
#include <iostream>
using namespace std;
int main ()
{
  int  v;
  int  *рок;
  int  **ррок;
  v = 3000;
  рок = &v;
  ррок= &рок;
  cout << "Vrednosta na promenlivata e :" << v << endl;
  cout << "Vrednosta na  pok :" << pok << endl;
  cout << "Vrednosta na  *pok :" << *pok << endl;
  cout << "Vrednosta na   ppok :" << ppok << endl;
  cout << "Vrednosta na **ppok :" << **ppok << endl;
  return 0;
}
```



Покажувачи и функции

1) Повикување на функција со аргументи (вредности):

```
#include <iostream>
using namespace std;
int  cub( int n )
 {
    return n * n * n;
 }
 int main()
  {
     int br=5;
     cout<< “Vrednosta na kub na brojot  “<<br<<”  e:”<<cub(br);
    return 0;
 }
```

2) Повикување на функција преку референца со покажувач аргумент:

```
#include <iostream>
using namespace std;
void   cub( int *n )
 {
    *n=(*n) * (*n)  * (*n);
 }
 int main()
  {
     int br=5;
     cub(&br);
     cout<< “Vrednosta na kub na brojot   e:”<<br;
    return 0;
 }
```

Пример:

```
#include <iostream>
using namespace std;
float  Average(float  *arr)
{
  int  i;
  float   avg, sum=0;
  for (i = 0; i < 5; i++)
    {
       sum += arr[i];
    }
   avg = sum / 5;
   return avg;
 }
int main ()
{
   float  niza[5] = {10.5, 2.7, 3.1, 17.4, 50.23};
   float  av;
   av = Average( niza ) ;
   cout << "Srednata vrednost e: " << av << endl;
   return 0;
}
```