

Лекция 7.

Указатели, динамични променливи,
псевдоними

*Има ли друг начин за достъп до
програмните елементи?*

Адреси и достъп до програмните елементи

Начини за достъп до (за идентификация на) програмните елементи:

- I начин:** Чрез **името** им – използван до момента
- II начин:** Чрез указване на **адреса** им – мястото, където са разположени в паметта

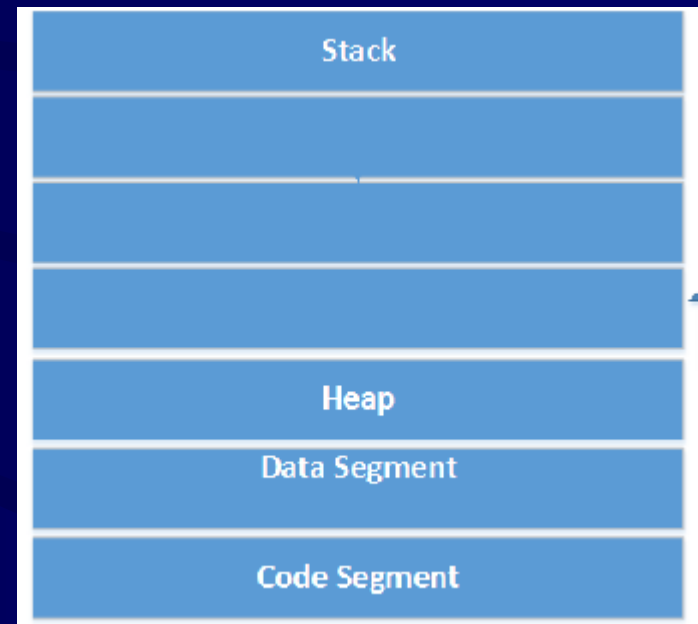
За достъп по II начин в C++ могат да се използват **2 възможности:**

- тип **указател**
- **псевдоними**

Организация на паметта

Паметта за всяка програма логически се разделя на четири части :


- код(машинен),
- данни- глобални и статичните променливи,
- т.нар. стек(stack) – локалните променливи и параметрите на функциите
- и т.нар. динамична памет(heap)



Тип указател: АМ – динамични променливи; ФМ в С++

АМ: Според жизнения цикъл в програмите има **два вида променливи**:

- Чийто жизнен цикъл се **управлява автоматично**:
 - създават се (памет за тях се заделя) при срещането на тяхната декларация в **стека** и се унищожават (паметта се освобождава) обикновено в края на блока, в който са декларирани
 - средството за идентификация е името на променливата (към стойността ѝ можем да се обръщаме чрез името ѝ)
- Чийто жизнен цикъл се **управлява от програмиста** т. нар. **динамични променливи**:
 - могат да се създават и унищожават по всяко време
 - паметта, в която се разполагат се нарича **динамична памет** (*heap*, “*free store*”)
 - средството за идентификация на динамичните променливи е адреса им (към тях се обръщаме чрез техните адреси)

 Основното предназначение на тип указател в ЕП е управлението на динамичните променливи. Освен това те служат за достъп чрез адрес и до други програмни елементи

Физически модел в С++ – заема 4 байта

Тип указател в C++: 3М

- **Синтаксис** (непълнен) за деклариране на променлива от тип указател (*pointer*):

<тип> * <име> ;


,където <тип> основен или структурен тип на C++, включително и `void` и се нарича **базов тип** на указателя (свързан с указателя); <име> е името на променливата от тип указател, за която се казва, че **сочи към** стойност от указания <тип>

- **Примери:**


```
void* vp;    // vp е от тип: произволен-указател
int* pi;     // pi е от тип: указател-към-int
float * pf;  // pf: указател-към-float
char *s;     // s: указател-към-char
```

 Независимо, че ‘*’ може да бъде написана навсякъде, тя се отнася към името на променливата. Ето защо в следващата декларация само `pa` е указател:


```
int* pa, pb;
```

- **Множество от стойности:**
 - ако <тип> не е `void`: Съвкупност от адреси в паметта (от динамичната памет и стека), където са разположени стойности от базовия тип на указателя
 - за тип `void`: Множество от произволни адреси от паметта (на променливи, входни точки на функции и т.н.)
 -  в C++ има стандартна константа от тип указател `NULL`, чийто смисъл е, че указателят сочи “никъде” (всеки константен целочислен израз със стойност 0 също се счита за `==NULL`)

Тип указател в C++: ЗМ – операция “присвояване на стойност”

 След декларирането на променлива-указател тя няма смислена стойност. Съществуват два начина за присвояване на стойност, в това число и за инициализация (присвояване на начална стойност):

- Чрез оператора за препратка (адресът-на, *reference operator, address-of*):
 - Синтаксис: `& <операнд>` – Семантика: Връща адреса на `<операнд>`, `<операнд>` може да бъде променлива от основен тип, тип масив, `struct`, `class`, или `union` или име на предварително декларирана функция
 - Примери:

```
int* p;  
int c;  
p = &c; // задава p да сочи-към c
```
- Чрез оператор `new` ( служи за създаване на динамични променливи):
 - Синтаксис: `new <тип>`, `<тип>` не може да е тип-функция
 - Семантика: Заделя място в динамичната памет за стойност от типа `<тип>` и връща адреса на тази памет (ако няма свободна памет, връща `NULL` – вж. Примера на слайд 16)
 - Примери:

```
p = new int; //заделя памет за int и задава p да сочи-към нея  
float * pf=new float ; // pf: указател-към-float  
char *s =new char;
```

Тип указатель в C++

```
int a = 2;  
int *pa = &a;  
cout << "Value of pointer pa is: " << pa << endl;
```

	стойность	адрес
a	2	0x230004

pa	0x230004	0x230220

Тип указател в C++: ЗМ – операция “*”

Операторът за достъп до променливата, към която сочи указател
(оператор за косвен достъп, *dereference operator*, *indirection operator*)
действа обратно на оператора &, т.е.

$d = *(&c);$ е еквивалентно на $d = c;$

- **Синтаксис:** `* <операнд>`
, <операнд> трябва да е стойност-указател
- **Семантика:** Връща стойността, към която сочи указателят
- **Примери:**

(1) достъп за ‘четене’ (автоматична променлива)

```
int* p; int c=2, d;
```

```
p = &c;
```

```
d = *p; //d има същата стойност като c
```

(2) достъп за ‘запис’ (динамична променлива)


```
int *pSomething = new int;; // дефинира указател
```

```
*pSomething = 15; // присвоява 15 на променливата към  
// която сочи указателят
```

 С променливата, към която сочи указател могат да се извършват всички възможни операции за базовия тип на този указател


Тип указател в C++: ЗМ – операция delete

Динамичната памет заделена чрез оператора `new` може да бъде **освободена** по всяко време на изпълнение на програмата чрез оператора `delete`

- **Синтаксис:** `delete <операнд>` , <операнд> е променлива-указател
- **Семантика** ( служи за освобождаване на динамични променливи) : Освобождава **динамичната памет** заета от стойността (динамичната променлива), към която сочи указателят
- **Примери:**

```
delete p;      //освобождава памет за int, където сочи p
delete pf;     // освобождава памет за float
delete s;      // освобождава памет за char
```

 след освобождаване на указателя неговата стойност е неопределена

 трябва да се внимава да не останат динамични променливи, които не се използват, а не са освободени

Тип указател в C++: ЗМ – операции за присвояване, въвеждане, извеждане и преобразуване на типовете

- **Операция за присвояване:**

Правило 1: На променлива-указател с базов тип различен от `void` може да се присвои само указател със същия базов тип

Правило 2: На променлива-указател с базов тип `void` може да се присвои указател с произволен базов тип

- **Въвеждане на указатели** – не е реализирано

- **Извеждане на указатели:**

Понякога с цел отстраняване на грешки в програмата е полезно да се изведат стойностите на указателите (обикновено се използва `hex` формат):

```
cout << "ptrA е адресът " << hex << ptrA << endl;
```

- **Преобразуване на типовете указатели** – възможно е, тъй като те са прости типове: чрез операцията претипизиране и чрез специалните функции за преобразуване (вж. Лекция 3)

Тип указател в C++: ЗМ – сравнение между присвояване на указатели и на променливите към които сочат

Нека е извършено:

```
double r1= 7.8, r2= 3.14;
```

```
double *R1=&r1, *R2=&r2;
```

- **R2= R1;** // присвояване на указатели

Преди присвояването

R1 • → 7.8

R2 • → 3.14

След присвояването

R1 • → 7.8

R2 • ↗ 3.14

- ***R2=*R1;** // присвояване на променливите към които сочат

Преди присвояването

R1 • → 7.8

R2 • → 3.14

След присвояването

R1 • → 7.8

R2 • → 7.8

Тип указател в C++: 3М – релации

С указателите могат да се извършват всички сравнения:

Правило 1: Указател с базов тип различен от `void` може да се сравнява само с указател със същия базов тип

Правило 2: Указател с базов тип `void` може да се сравнява с указател с произволен базов тип

Правило 3: Гарантирано е, че ако два указателя имат един и същи базов тип и сочат към един и същи обект, то те са равни

Пример:

```
double *R1=&r1;
```

```
double *R3=&r1;
```

```
cout << (R1==R3); //отпечатва 1
```

Тип указател в C++: Особенности

- За да може да се работи с даден указател на него трябва да му е присвоена стойност (адрес) по един от известните начини, т.е. да бъде **инициализиран**. В случай, че указателят не е инициализиран или е вече освободен, указателят се нарича **неинициализиран**
- C++ осигурява и друг начин за резервиране и освобождаване на памет за указатели чрез стандартните функции `malloc`, `calloc`, `free`, които служат за заделяне и освобождаване на блок от байтове (`stdlib.h` и `malloc.h`)

Указатели и масиви

В C++ както и в C указателите са тясно свързани с масивите

- **Правило за обработка на едномерни масиви чрез указатели:**
Всяка променлива от тип едномерен масив представлява всъщност указател с базов тип еднакъв с типа на елементите на масива, който сочи към първия елемент на масива:

```
char chArray[10];  
char *pch = chArray; // Указател към първия  
                      // елемент на масива
```

 Обратното присвояване е забранено:

```
chArray = pch; // !!! Грешка
```

 Поради горното, ф-ята `sizeof(chArray)` връща 4, а не 40

Аритметика с указатели

Аритметиката с указатели позволява да се извършват операциите + и - между указатели и цели числа

- **Правило:** Това е възможно само ако указателят сочи към елемент на масив, а цялото число дава отместване (*offset*), което е в рамките на този масив
- **Събиране/ Изваждане :**

<указател> + <цяло>

<указател> - <цяло>

Резултатът е указател от типа на <указател>, който сочи към друг елемент на масива, който има индекс по-голям/по-малък с <цяло> от индекса на първоначалния елемент

```
short IntArray[10]; // Масив от short (всеки по 2 байта)
short *pIntArray = IntArray;
for( int i = 0; i < 10; ++i ) {
    *pIntArray = i;
    cout << *pIntArray << "\n";
    pIntArray = pIntArray + 1;
}
```



За предпочитане е да се използва синтаксисът:

```
pIntArray++ или pIntArray += 1
pIntArray-- или pIntArray -= 1
```

Масивите като динамични променливи

- **Синтаксис** за деклариране

 Обърнете внимание на разликата:

```
float (*raf)[25]; //указател към масив от float
```

```
float *arf [25]; //масив от указатели към float
```

- **Резервиране на място в динамичната памет чрез оператор new**

Когато чрез new се резервира памет за масив, резултатът е указател към първия елемент на масива и резултатният тип запазва всички освен най-лявата размерност на масива:

(1) Правилно

```
float (*cp)[25][10];
```

```
cp = new float[10][25][10];
```

(2) Неправилно

```
float *fp;
```

```
fp = new float[10][25][10];
```

- **Освобождаване на място в динамичната памет чрез оператор delete**

- **Синтаксис:** `delete[] <указател>`

, където <указател> е указател към масив

- **Семантика:** Освобождава мястото в динамичната памет заето от масива

Тип указател в C++: Пример за недостиг на памет при резервиране на динамичната памет за масив


```
int *pi = new int[10000000000000000000];  
if( pi == 0 )  
{  
    cerr << "Insufficient memory" << endl;  
    return;  
}
```

Псевдоними в C++: 3М

Псевдонимите (**алтернативно име, препратка, *reference***) са средство за осъществяване на достъп към променливи по адрес като се използва **поудобния синтаксис за достъп по име**

- **Синтаксис** за деклариране на променлива-псевдоним:

`<тип> & <име> = <променлива>;`

- <име> е псевдонимът на <променлива>
- <тип> е кой да е основен или структурен тип на C++ (**базов тип** на псевдонима)
- <променлива> инициализира псевдонима и е предварително декларирана променлива от базовия тип или такъв, който може да се преобразува към него
- псевдонимът съдържа адреса на <променлива>, но синтактично се държи точно като нея
-  не могат да се декларират указатели към псевдоними и масиви от псевдоними, както и псевдоним на масив, за който явно не е зададена размерност

- **Примери:**

```
/*1*/  int i = 7;      int &ri = i; // ri е псевдоним на i
/*2*/  int ai[3], (& air)[3]=ai; //air е псевдоним на масива ai
/*3*/  long lVar;
       long& LongRef1 = lVar;      // Не се изисква преобразуване
       long& LongRef2 = i;          // !!!Грешка
       const long& LongRef3 = i    // Правилно
/*4*/  double* d=new double; double*& dr=d; // Псевд. на указател
```

- **Множество от стойности:**

16- или 32-битова клетка, която съдържа адреса на променлива от базовия тип или такъв, който може да се преобразува към него

Псевдоними в C++: ЗМ – операции

- **Инициализация:**

- инициализацията на псевдонимите е задължителна. Дефинирането на псевдоним без инициализация води до грешка при компилиране
- инициализиращата променлива може да бъде всяка променлива, чийто адрес може да бъде преобразуван към указател от типа на псевдонима (напр. `char &` – `char *`)
- променливите от тип псевдоним винаги сочат към обекта, с който са инициализирани и не могат да се пренасочват

- С псевдоним могат да се извършват **всички операции**, които са допустими за променливата, за която той се отнася

- Всички операции прилагани върху даден псевдоним в действителност се извършват върху променливата, за която той се отнася

```
/*1*/ int x = 10; // x
      int& r = x; // псевдоним на x
      r = r + 2;  //прибавя 2 към x, и така x е 12
/*2*/ int ai[3], (& air)[3]=ai; //air е псевдоним на ai
      air[2]=1;  // присвоява стойност на 3-я елемент на ai
/*3*/ double* d=new double;
      double*& dr=d; // псевдоним на указател
      *dr=2.02;      // присвоява стойност на дин. пром.
```

🔊 Обърнете внимание на разликата между инициализация и присвояване на псевдоним

🔊 Основното предназначение на псевдонимите е за деклариране на параметри и резултат на функции

Псевдонимы в C++: ФМ

Заемат 4 байта