

Увод в програмирането

Типове указател и псевдоним

ФМИ, специалност „Софтуерно инженерство“

Тип указател ...

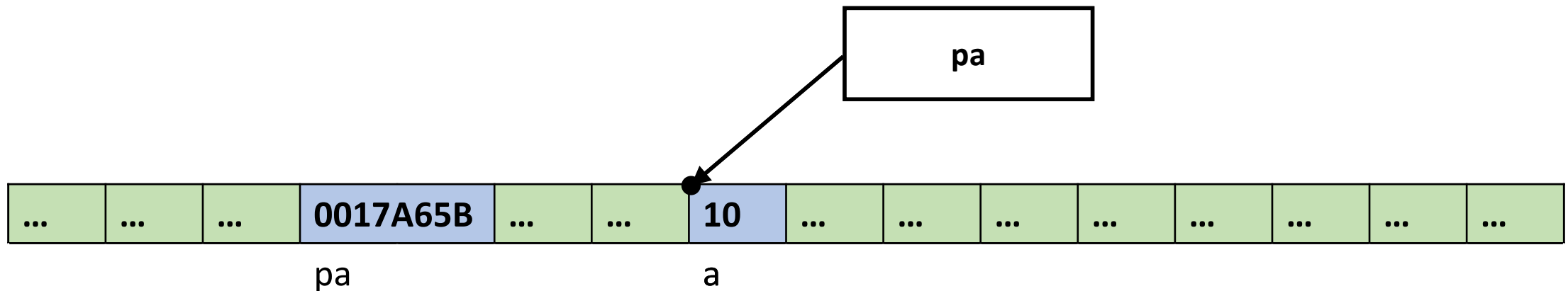
T* pa;

- Дефинира указател с име **pa** от тип **T**
- Указателят практически е (не)обикновена променлива, чиято стойност се интерпретира като адрес в паметта
- T определя типа на данните, които указателят адресира, а също и начина на интерпретацията им.

Указател

- Променлива, стойността на която се интерпретира като адрес в паметта на някакъв друг обект

```
int a = 10;  
int *pa = &a;
```

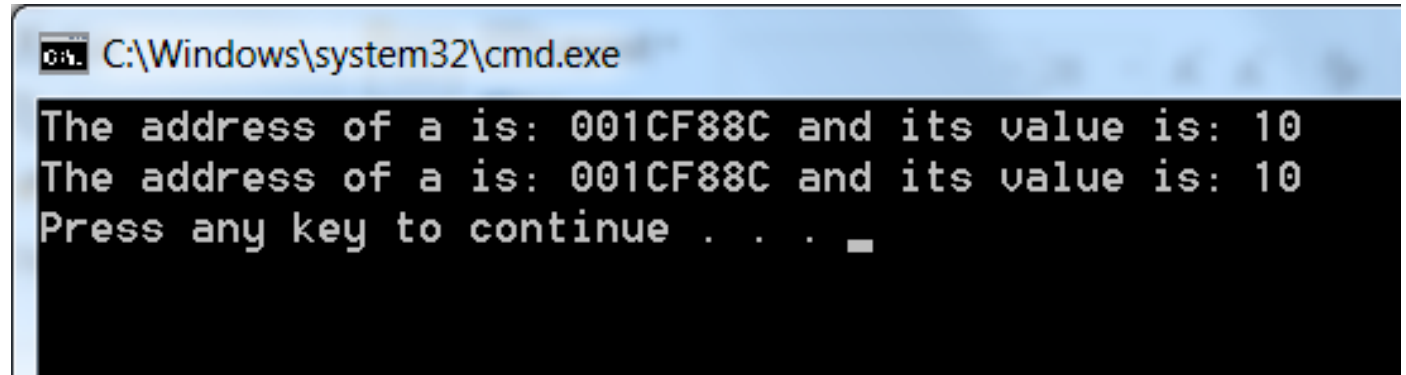


Указатели и адреса

```
int main()
{
    int a = 10;
    int *pa = &a;

    cout << "The address of a is: " << pa
          << " and its value is: " << *pa << endl;
    cout << "The address of a is: "
          << &a << " and its value is: " << a << endl;

    return 0;
}
```

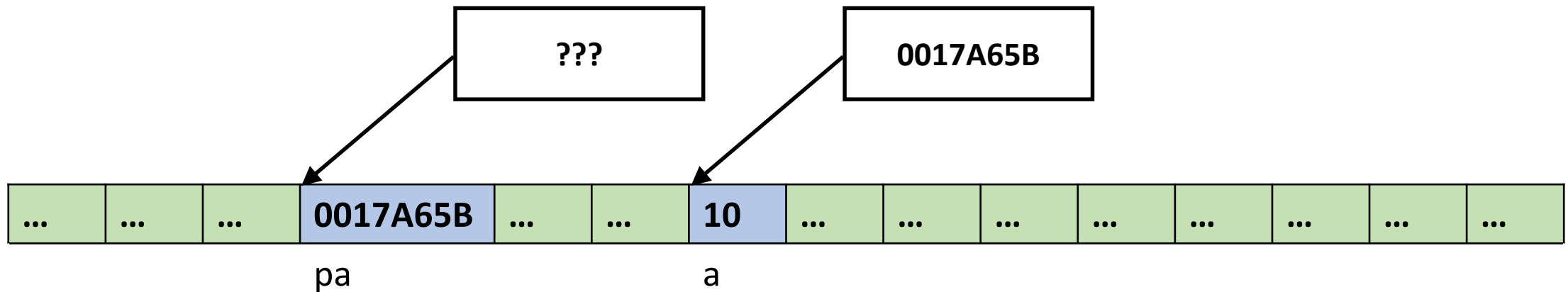


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the C++ program: "The address of a is: 001CF88C and its value is: 10" on the first line, and "The address of a is: 001CF88C and its value is: 10" on the second line. Below the output, it says "Press any key to continue . . . _".

Указател

- Променлива, стойността на която се интерпретира като адрес в паметта на някакъв друг обект

```
int a = 10;  
int *pa = &a;
```



```
int main()
```

```
{
```

```
    int a = 10;
```

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

```
    int **ppa = &pa;
```

```
    cout << "The address of    a is: "  
          << *ppa << " and its value is: "  
          << **ppa << endl << endl;
```

```
    cout << "The address of  pa is: " << ppa  
          << " and its value is: " << pa << endl << endl;
```

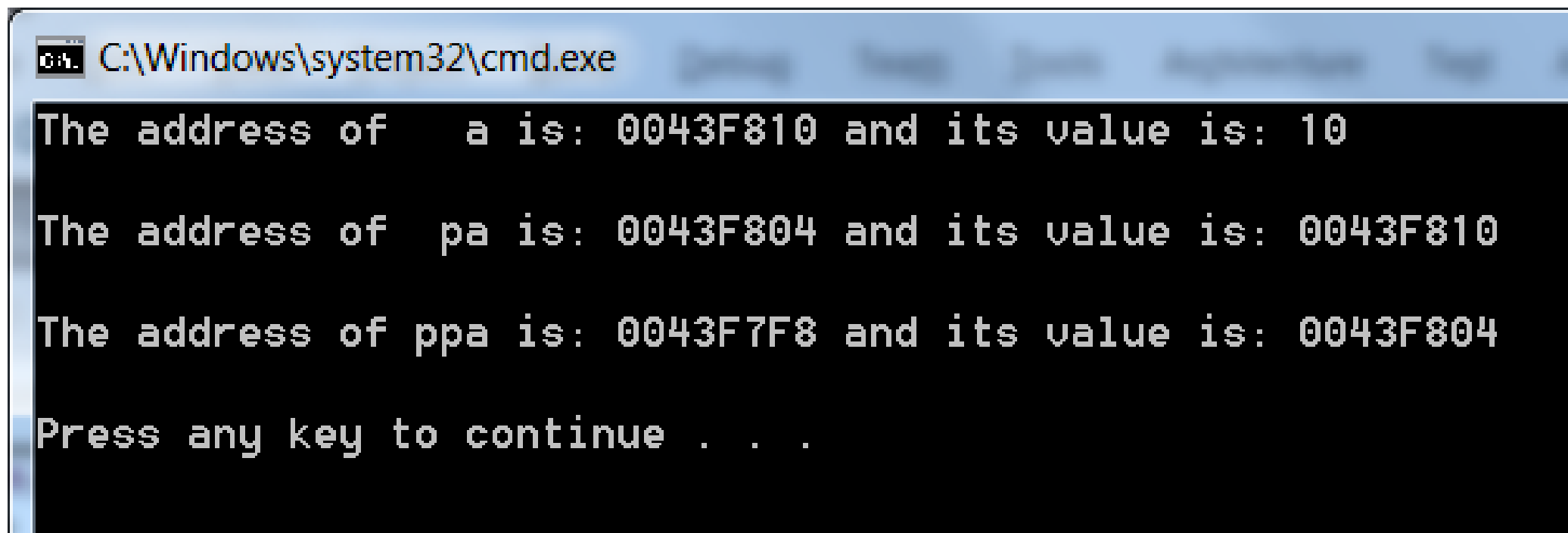
```
    cout << "The address of ppa is: " << &ppa  
          << " and its value is: " << ppa << endl << endl;
```

```
    return 0;
```

```
}
```

Указател към
указател

Указател към указател



```
C:\Windows\system32\cmd.exe

The address of  a is: 0043F810 and its value is: 10

The address of  pa is: 0043F804 and its value is: 0043F810

The address of ppa is: 0043F7F8 and its value is: 0043F804

Press any key to continue . . .
```

Типове указатели

```
int i = 100;  
double d = 1.5;  
unsigned long long l = 200;
```

```
// Указател към даден тип T се дефинира, като към T  
// се добави символът звезда.  
int * pi = &i; // Казваме "pi е указател от тип int"  
double * pd = &d;  
unsigned long long * pl = &l;
```



```
// При дефиниране на няколко променливи на един ред  
// има особеност!
```

```
// Три променливи от тип int  
int var1, var2, var3;
```

```
// Грешка: pointer 1 е указател,  
// но pointer2 е променлива от тип int!  
int* pointer1, pointer2;
```

```
// Звездата се поставя пред всяка променлива,  
// която бихме искали да бъде указатели. Това позволява  
// на един ред да се дефинират променливи и указатели от  
// един и същ тип.
```

```
int var4, var5, *pointer3, var6, *pointer4;
```

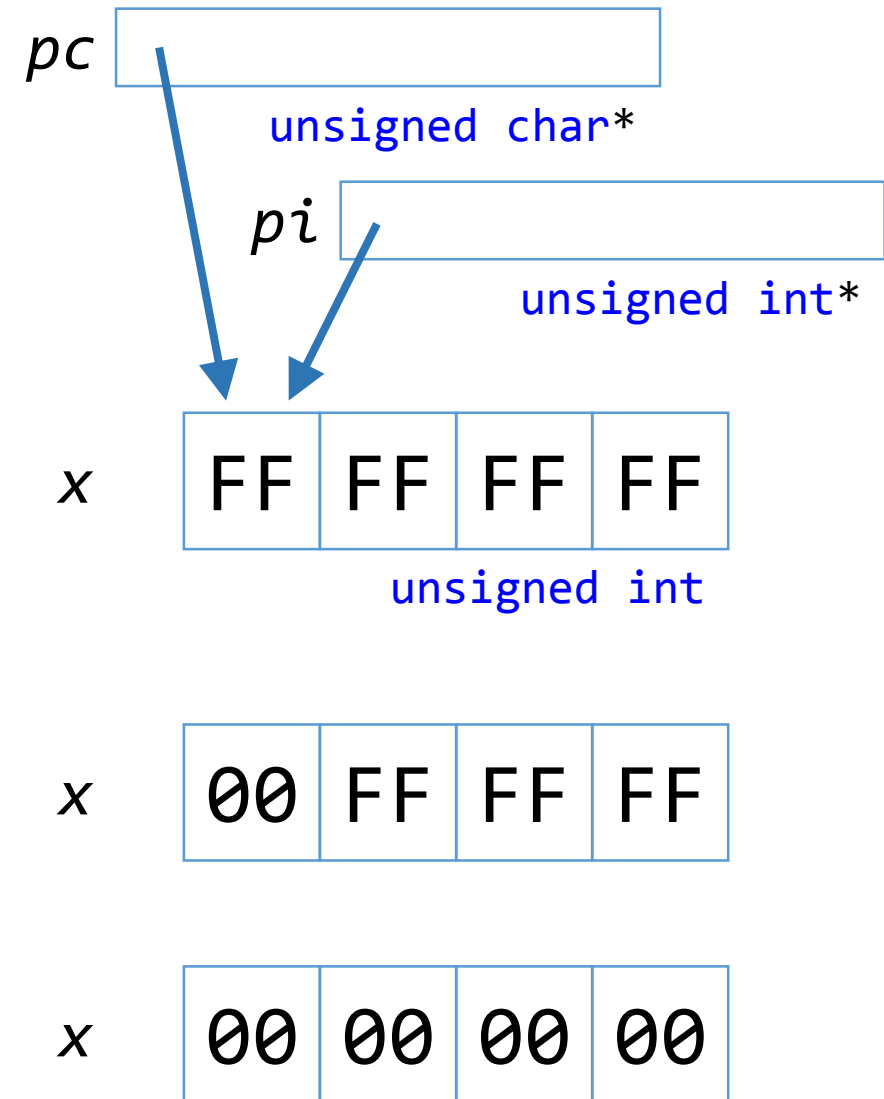
Работа с указатели

```
unsigned int x = 0xFFFFFFFF;  
unsigned int *pi = &x;  
unsigned char *pc = (unsigned char*)&x;
```

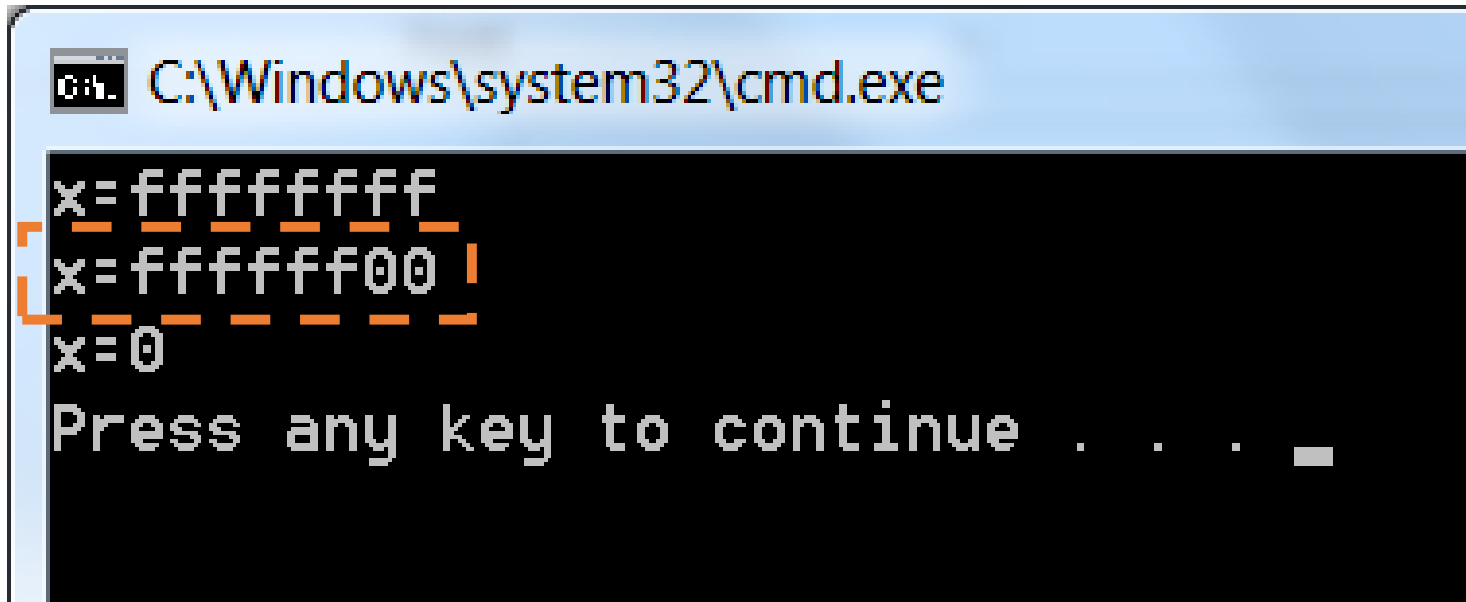
```
cout << "x=" << hex << x << endl;
```

```
*pc = 0;  
cout << "x=" << hex << x << endl;
```

```
*pi = 0;  
cout << "x=" << hex << x << endl;
```



Little Endian vs Big Endian



A screenshot of a Windows command prompt window. The title bar shows the path `C:\Windows\system32\cmd.exe`. The command prompt displays the following text:

```
x=ffffffff
x=ffffff00
x=0
Press any key to continue . . .
```

Orange dashed brackets are drawn around the first three lines of output, highlighting the memory addresses.

x

00	FF	FF	FF
----	----	----	----

Инициализиране на указателите!!!!

// p1 не е инициализиран

int *p1;

*p1 = 500; // Какво ли ще се случи?

std::cout << *p1; // Какво ли ще се случи?

error C4700: uninitialized local variable 'p1' used

```
int *p1 = 0;  
int *p2 = NULL;
```

```
// null-pointer assignment  
*p1 = 500;           // грешка!  
*p2 = 500;           // грешка!  
std::cout << *p1;    // грешка!
```

```
int x;  
p2 = &x;    // Инициализираме p2  
*p2 = 500;  // Използваме го за нещо  
p2 = NULL; // Ако вече не ни трябва, нулираме указателя
```

nullptr vs NULL

```
#define NULL 0;
```

nullptr е винаги указател!

```
int func(int* pa) {  
    if (pa == NULL)  
        return -1;  
  
    return *pa;  
}
```

```
int func(int a) {  
    return a;  
}
```



func(NULL);

void указател

- Използва се, когато е важна стойността на променливата от тип указател (т.е. адресът на данните към които сочи), а не нейното съдържание.
- Този случай е предвиден с цел една и съща променлива - указател да може в различни моменти да сочи към данни от различен тип.
- При опит да се използва съдържанието на променливата от тип указател, ще се предизвика грешка.
- Съдържанието на променлива - указател към тип `void` може да се извлече само след преобразуване на типа на указателя (`void*`) до типа на съдържанието.

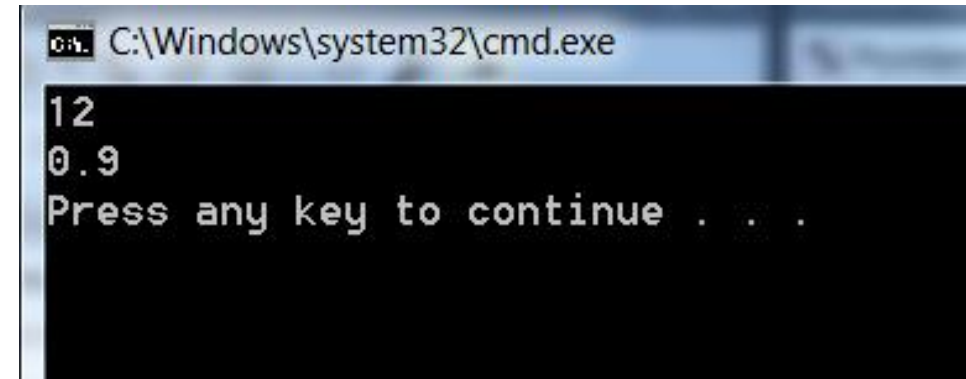

```
int main()
{
    long a = 12;
    double d = 0.9;

    void *pv;

    pv = &a;
    cout << *(int*)pv << endl;

    pv = &d;
    cout << *(double*)pv << endl;

    return 0;
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window has a black background with white text. It displays the output of a program: the number '12' on the first line, '0.9' on the second line, and the prompt 'Press any key to continue . . .' on the third line.

Тип указател и константи

- В дефиницията:

T* **const** <идентификатор>;

<идентификатор> е константен указател към тип T и не може да бъде променяна стойността му.

- В дефиницията:

const **T*** <идентификатор>;

<идентификатор> е указател към константа от тип T и не може да бъде променяно съдържанието му.

Константни указатели и указатели към константи

```
int var = 100, anotherVar = 100;
```

```
// Тези указатели може да се пренасочат  
// впоследствие
```

```
int *p1 = &var;
```

```
const int *p2a = &var;
```

```
int const *p2b = &var; // еквивалентно
```

```
p1 = &anotherVar; // OK
```

```
p2a = &anotherVar; // OK
```

```
p2b = &anotherVar; // OK
```

```
*p1 = 1000; // OK
```

```
*p2a = 1000; // грешка!
```

```
// Тези указатели се инициализират
```

```
// още при създаването им и по-късно не
```

```
// могат да сочат към друга променлива
```

```
int * const p3 = &var;
```

```
const int * const p4 = &var;
```

```
p3 = &anotherVar // грешка!
```

```
p4 = &anotherVar // грешка!
```

```
*p3 = 222222222;
```

```
*p4 = 2222; // грешка
```

Псевдоним (reference)

```
int a;  
int &ra = a;
```

- **ra** е псевдоним (alias, reference) на променливата **a**
- Свързва се с адреса на променливата **a**
- Чрез псевдонимите се задават алтернативни имена на обекти в общия смисъл на думата (променливи, константи и др.).

Псевдоним (reference)

- **Дефиницията на променлива от тип псевдоним задължително е с инициализация – дефинирана променлива от същия тип като на базовия тип на типа псевдоним.**
- Променливата от тип псевдоним не може да получи нова стойност
 - Променя се променливата, към която е псевдонимът

```
int ii = 0;  
int& rr = ii;  
rr++;  
int* pp = &rr;
```

Псевдоним (reference)

- **&**<име> —указател към променливата <име>
- *****<указател> —мястото в паметта,сочено от <указател>
- ***** и **&** всъщност са операции в езика с висок приоритет
 - Дуални са една на друга и се унищожават взаимно
- Операторът **&** не може да се прилага върху константи и изрази
 - **&100** и **&(i+5)** са недопустими обръщения

Предаване на аргументи на функция по адрес

```
void swap(int* p, int* q) {  
    int tmp = *p;  
    *p = *q;  
    *q = tmp;  
}
```

```
int main() {  
    int a = 5, b = 8;  
    swap(&a, &b);  
    cout << a << ' ' << b << endl;  
}
```

```
void swap(int& x, int& y) {  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

```
int main() {  
    int a = 5, b = 8;  
    swap(a, b);  
    cout << a << ' ' << b << endl;  
}
```

Адресна аритметика

- Допустими операции с указателите
 - рефериране (<lvalue>)
 - дерефериране (*<указател>)
 - указателна аритметика (+,-,+=,-=,++,--)
 - сравнение (==, !=, <, >, <=, >=)
 - изход (<<)
- няма вход! (>>)

Приоритет на операторите

Оператор	Име	Асоциативност
++ --	Postfix Increment Postfix Decrement	Left to Right Left to Right
++ -- *, &	Prefix Increment Prefix Decrement Dereference, Reference	Right to Left Right to Left Right to Left
* /	Multiplication Division	Left to Right Left to Right
+ - %	Addition Subtraction Modulus	Left to Right Left to Right Left to Right
< >	Less Than Greater Than	Left to Right Left to Right
&&	Logical And	Left to Right
	Logical Or	Left to Right
?:	Conditional	Right to Left
=	Assignment	Right to Left
,	Comma	Left to Right

- За подготовката на тази презентация са използвани слайдове на:
 - Доц. Александър Григоров
 - Доц. Атанас Семерджиев
 - Доц. Трифон Трифонов