

Указатели и псевдоними

Трифон Трифонов

Увод в програмирането,
спец. Компютърни науки, 1 поток,
спец. Софтуерно инженерство,
2016/17 г.

7 декември 2016 г.

Тип указател

- **Множество от стойности:** всички възможни `lvalue` от даден тип и специалната стойност `NULL`.
- Интегрален **нечислов** тип
- Параметризиран тип: ако `T` е тип данни, то `T*` е тип “указател към елемент от тип `T`”
- Физическо представяне: цяло число, указващо адреса на указваната `lvalue` в паметта
- Стойностите от тип “указател” са с размера на машинната дума
 - 32 бита (4 байта) за 32-битови процесорни архитектури
 - 64 бита (8 байта) за 64-битови процесорни архитектури

Операции с указатели

- рефериране (&<lvalue>)
- дерефериране (*<указател>)
 - унарна операция!
- сравнение (==, !=, <, >, <=, >=)
- указателна аритметика (+, -, +=, -=, ++, --)
- извеждане (<<)
- няма въвеждане! (>>)

Дефиниране на указателни променливи

`<тип> *<име> [= <израз>] { , *<име> [= <израз>] } ;`

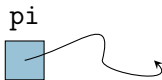
Примери:

Дефиниране на указателни променливи

`<тип> *<име> [= <израз>] { , *<име> [= <израз>] };`

Примери:

- `int *pi;`



Дефиниране на указателни променливи

`<тип> *<име> [= <израз>] { , *<име> [= <израз>] };`

Примери:

- `int *pi;`
- `double *pd = NULL;`

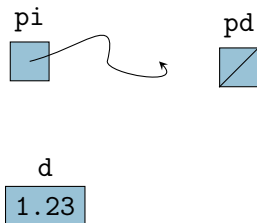


Дефиниране на указателни променливи

```
<тип> *<име> [ = <израз> ] { , *<име> [ = <израз> ] };
```

Примери:

- `int *pi;`
- `double *pd = NULL;`
- `double d = 1.23;`

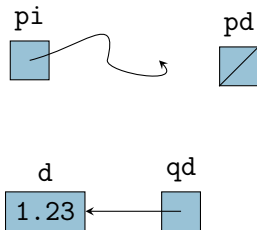


Дефиниране на указателни променливи

```
<тип> *<име> [ = <израз> ] { , *<име> [ = <израз> ] };
```

Примери:

- `int *pi;`
- `double *pd = NULL;`
- `double d = 1.23;`
- `double *qd = &d;`

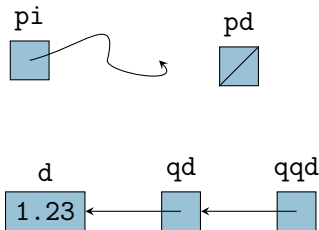


Дефиниране на указателни променливи

```
<тип> *<име> [ = <израз> ] { , *<име> [ = <израз> ] };
```

Примери:

- `int *pi;`
- `double *pd = NULL;`
- `double d = 1.23;`
- `double *qd = &d;`
- `double **qqd = &qd;`



Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`

Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`

Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`

Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - ~~`&x = p;`~~

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - `&x = p;`
- `*<rvalue>` връща като резултат `<lvalue>!`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - ~~`&x = p;`~~
- `*<rvalue>` връща като резултат `<lvalue>!`
 - `*p = x;`

Рефериране и дерефериране

- `<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - `&x = p;`
- `*<rvalue>` връща като резултат `<lvalue>!`
 - `*p = x;`
 - `**qqd = 3.15;`

Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - ~~`&x = p;`~~
- `*<rvalue>` връща като резултат `<lvalue>!`
 - `*p = x;`
 - `**qqd = 3.15;`
- операциите са дуални една на друга и се унищожават взаимно

Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - ~~`&x = p;`~~
- `*<rvalue>` връща като резултат `<lvalue>!`
 - `*p = x;`
 - `**qqd = 3.15;`
- операциите са дуални една на друга и се унищожават взаимно
 - $\&(*p) \iff p$

Рефериране и дерефериране

- `&<име>` — указател към променливата `<име>`
- `*<указател>` — мястото в паметта, сочено от `<указател>`
- **Примери:**
 - `int x = 5, *p = &x;`
 - `int *q = p, y = *p + 2;`
 - `*p++; p = &y;`
 - `*q = 1; *p = *q;`
- `&<lvalue>` връща като резултат `<rvalue>!`
 - `&3`
 - ~~`&x = p;`~~
- `*<rvalue>` връща като резултат `<lvalue>!`
 - `*p = x;`
 - `**qqd = 3.15;`
- операциите са дуални една на друга и се унищожават взаимно
 - `&(*p) \iff p`
 - `*(&x) \iff x`

Указатели и масиви

В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Указатели и масиви

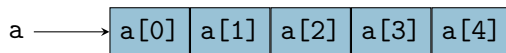
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Примери:

- `int a[5];`



Указатели и масиви

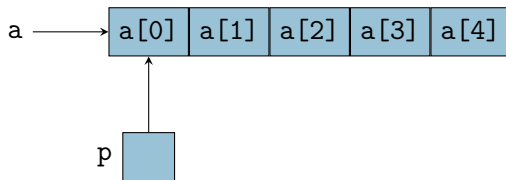
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Примери:

- `int a[5];`
- `int* p = a;`



Указатели и масиви

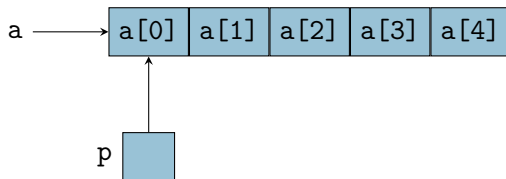
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Примери:

- `int a[5];`
- `int* p = a;`
- `*p = 15;`



Указатели и масиви

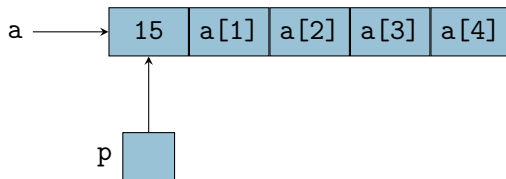
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Примери:

- `int a[5];`
- `int* p = a;`
- `*p = 15;`
- `cout << a[0];`



Указатели и масиви

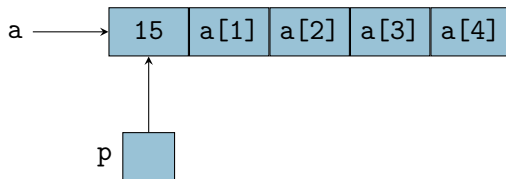
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Примери:

- `int a[5];`
- `int* p = a;`
- `*p = 15;`
- `cout << a[0];`
- `*a = 20; a = p;`



Указатели и масиви

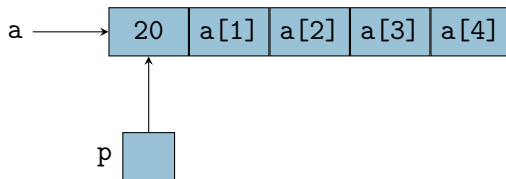
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е **константен указател** към първия му елемент.

Примери:

- `int a[5];`
- `int* p = a;`
- `*p = 15;`
- `cout << a[0];`
- `*a = 20; a = p;`



Указателна аритметика

- Указателната аритметика позволява по дадена отправна точка в паметта (указател) да реферираме съседни на нея клетки.

Указателна аритметика

- Указателната аритметика позволява по дадена отправна точка в паметта (указател) да реферираме съседни на нея клетки.
- За целта трябва да укажем колко клетки напред или назад в паметта искаме да прескочим.

Указателна аритметика

- Указателната аритметика позволява по дадена отправна точка в паметта (указател) да реферираме съседни на нея клетки.
- За целта трябва да укажем колко клетки напред или назад в паметта искаме да прескочим.
- Синтаксис:

Указателна аритметика

- Указателната аритметика позволява по дадена отправна точка в паметта (указател) да реферираме съседни на нея клетки.
- За целта трябва да укажем колко клетки напред или назад в паметта искаме да прескочим.
- Синтаксис:
 - `<указател> [+ | -] <цяло_число>`

Указателна аритметика

- Указателната аритметика позволява по дадена отправна точка в паметта (указател) да реферираме съседни на нея клетки.
- За целта трябва да укажем колко клетки напред или назад в паметта искаме да прескочим.
- Синтаксис:
 - `<указател> [+ | -] <цяло_число>`
 - `<цяло_число> + <указател>`

Указателна аритметика

- Указателната аритметика позволява по дадена отправна точка в паметта (указател) да реферираме съседни на нея клетки.
- За целта трябва да укажем колко клетки напред или назад в паметта искаме да прескочим.
- Синтаксис:
 - $\langle \text{указател} \rangle [+ \mid -] \langle \text{цяло_число} \rangle$
 - $\langle \text{цяло_число} \rangle + \langle \text{указател} \rangle$
- прескачаме $\langle \text{цяло_число} \rangle$ клетки напред (+) или назад (-) от адреса, сочен от $\langle \text{указател} \rangle$

Големина на тип

- Но... какво означава “прескачаме n клетки”?

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- Зависи от типа, който указваме!

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;
 - $p + 2$ означава “прескочи 16 байта напред”, ако `double* p`;

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;
 - $p + 2$ означава “прескочи 16 байта напред”, ако `double* p`;
- `sizeof(<тип>|<израз>)` — размера в байтове, заемаан в паметта от <израз> или от стойност от <тип>

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;
 - $p + 2$ означава “прескочи 16 байта напред”, ако `double* p`;
- `sizeof(<тип>|<израз>)` — размера в байтове, заемаан в паметта от <израз> или от стойност от <тип>
- Така, ако имаме `T* p`;

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;
 - $p + 2$ означава “прескочи 16 байта напред”, ако `double* p`;
- `sizeof(<тип>|<израз>)` — размера в байтове, заемаан в паметта от <израз> или от стойност от <тип>
- Така, ако имаме `T* p`;
- ...тогава $p + i$ прескача $i * \text{sizeof}(T)$ байта напред

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;
 - $p + 2$ означава “прескочи 16 байта напред”, ако `double* p`;
- `sizeof(<тип>|<израз>)` — размера в байтове, заеман в паметта от <израз> или от стойност от <тип>
- Така, ако имаме `T* p`;
- ...тогава $p + i$ прескача $i * \text{sizeof}(T)$ байта напред
- `(int)p` — цялото число, съответстващо на адреса сочен от `p`

Големина на тип

- Но... какво означава “прескачаме n клетки”?
- **Зависи от типа, който указваме!**
 - $p + 2$ означава “прескочи 2 байта напред”, ако `char* p`;
 - $p + 2$ означава “прескочи 8 байта напред”, ако `int* p`;
 - $p + 2$ означава “прескочи 16 байта напред”, ако `double* p`;
- `sizeof(<тип>|<израз>)` — размера в байтове, заемаан в паметта от <израз> или от стойност от <тип>
- Така, ако имаме `T* p`;
- ...тогава $p + i$ прескача $i * \text{sizeof}(T)$ байта напред
- `(int)p` — цялото число, съответстващо на адреса сочен от `p`
- $p + i \iff (T*)((int)p + i * \text{sizeof}(T))$

Указателна аритметика за масиви

Факт

Името на масив е **константен указател** към първия му елемент.

Указателна аритметика за масиви

Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

Указателна аритметика за масиви

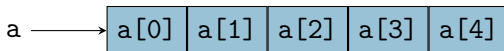
Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

Примери:

- `int a[5], x;`



Указателна аритметика за масиви

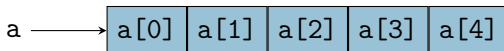
Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

Примери:

- `int a[5], x;`
- `cout << *a;`



Указателна аритметика за масиви

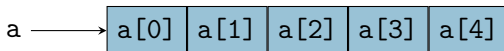
Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

Примери:

- `int a[5], x;`
- `cout << *a;`
- `*(a + 1) = 7;`



Указателна аритметика за масиви

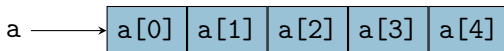
Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

Примери:

- `int a[5], x;`
- `cout << *a;`
- `*(a + 1) = 7;`
- `*(a + 4)--;`



Указателна аритметика за масиви

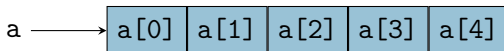
Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

Примери:

- `int a[5], x;`
- `cout << *a;`
- `*(a + 1) = 7;`
- `*(a + 4)--;`
- ~~`a++; a--; a = &x;`~~



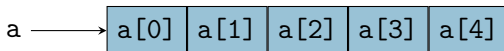
Указателна аритметика за масиви

Факт

Името на масив е **константен указател** към първия му елемент.

Освен това, $a[i] \iff *(a + i)$

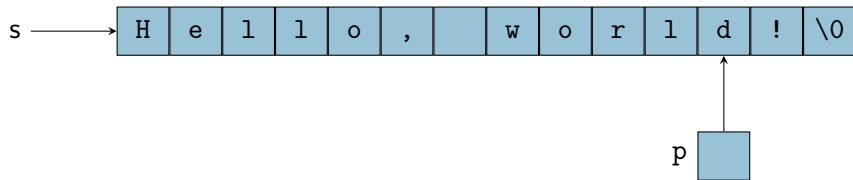
Примери:



- `int a[5], x;`
- `cout << *a;`
- `*(a + 1) = 7;`
- `*(a + 4)--;`
- ~~`a++; a--; a = &x;`~~
- Странно, но вярно: $a[i] \iff *(a+i) \iff *(i+a) \iff i[a]$

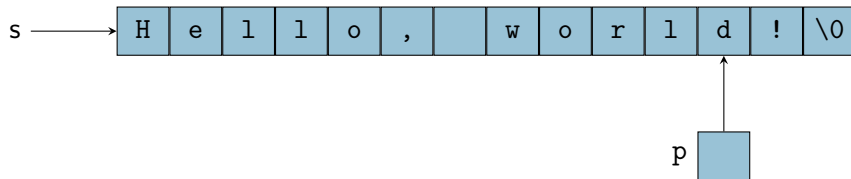
Указатели и низове

Низовете са масиви от символи



Указатели и низове

Низовете са масиви от символи

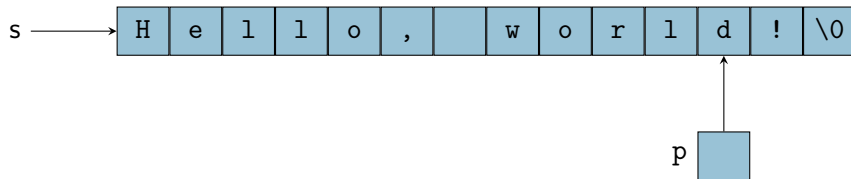


Примери:

```
char s[] = "Hello, world!";
```

Указатели и низове

Низовете са масиви от символи



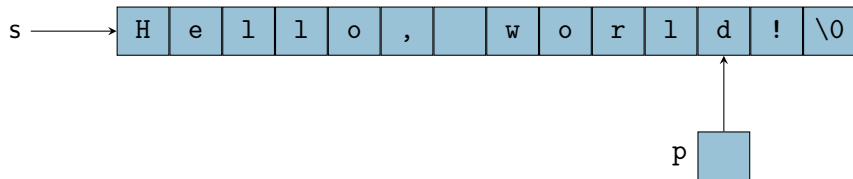
Примери:

```
char s[] = "Hello, world!";
```

```
void print(char* p) {  
    while (*p) cout << *p++;  
}
```

Указатели и низове

Низовете са масиви от символи



Примери:

```
char s[] = "Hello, world!";
```

```
void print(char* p) {
    while (*p) cout << *p++;
}
```

```
int strlen(char* p) {
    int n = 0;
    while (*p++) n++;
    return n;
}
```

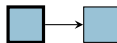
Указатели и константи

- Константен указател (който е константа)

Указатели и константи

- Константен указател (който е константа)

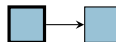
- `<тип>* const`



Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`
- `int x, *p = &x;`



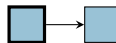
Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

- `int x, *p = &x;`

- `int* const q = p; q = p + 2; *q = 5;`



Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

- `int x, *p = &x;`

- `int* const q = p; q = p + 2; *q = 5;`



- Указател към константа (сочещ към константа)

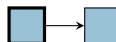
Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

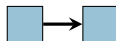
- `int x, *p = &x;`

- `int* const q = p; q = p + 2; *q = 5;`



- Указател към константа (сочещ към константа)

- `const <тип>* \iff <тип> const*`



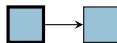
Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

- `int x, *p = &x;`

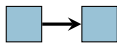
- `int* const q = p; q = p + 2; *q = 5;`



- Указател към константа (сочещ към константа)

- `const <тип>* ⇔ <тип> const*`

- `int x, *p = &x;`



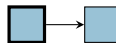
Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

- `int x, *p = &x;`

- `int* const q = p; q = p + 2; *q = 5;`

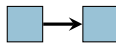


- Указател към константа (сочещ към константа)

- `const <тип>* \iff <тип> const*`

- `int x, *p = &x;`

- `int const* q = &x; q++; p = q; *q = 5;`



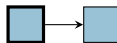
Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

- `int x, *p = &x;`

- `int* const q = p; q = p + 2; *q = 5;`

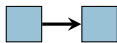


- Указател към константа (сочещ към константа)

- `const <тип>* \iff <тип> const*`

- `int x, *p = &x;`

- `int const* q = &x; q++; p = q; *q = 5;`



- Ако `p` е указател към константа, то `*p` е `<rvalue>`

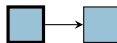
Указатели и константи

- Константен указател (който е константа)

- `<тип>* const`

- `int x, *p = &x;`

- `int* const q = p; q = p + 2; *q = 5;`

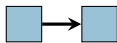


- Указател към константа (сочещ към константа)

- `const <тип>* ⇔ <тип> const*`

- `int x, *p = &x;`

- `int const* q = &x; q++; p = q; *q = 5;`



- Ако `p` е указател към константа, то `*p` е `<rvalue>`

- Ако `x` е константа, то `&x` е указател към константа

Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).

Указатели и низови константи

Факт

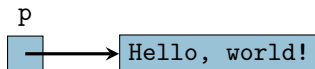
Името на низ е **константен указател** към първия му символ (`char* const`).
Низовите константи са **указатели към константен символ** (`char const*`).

Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).
Низовите константи са **указатели към константен символ** (`char const*`).

- `char const* p = "Hello, world!";`



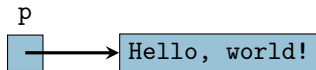
Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).
Низовите константи са **указатели към константен символ** (`char const*`).

- `char const* p = "Hello, world!";`

- ~~`char* q = "Hi C++!";`~~

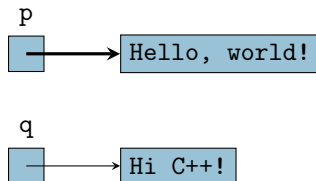


Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).
 Низовите константи са **указатели към константен символ** (`char const*`).

- `char const* p = "Hello, world!";`
- ~~`char* q = "Hi C++!";`~~
- `char q[] = "Hi C++!";`

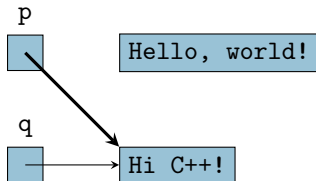


Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).
 Низовите константи са **указатели към константен символ** (`char const*`).

- `char const* p = "Hello, world!";`
- ~~`char* q = "Hi C++!";`~~
- `char q[] = "Hi C++!";`
- `p = q;`

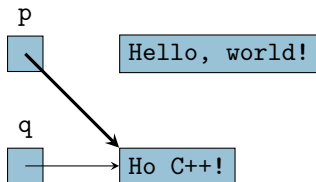


Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).
 Низовите константи са **указатели към константен символ** (`char const*`).

- `char const* p = "Hello, world!";`
- ~~`char* q = "Hi C++!";`~~
- `char q[] = "Hi C++!";`
- `p = q;`
- `q[1] = 'o';` ~~`p[1] = 'o';`~~



Указатели и низови константи

Факт

Името на низ е **константен указател** към първия му символ (`char* const`).
 Низовите константи са **указатели към константен символ** (`char const*`).

- `char const* p = "Hello, world!";`
- ~~`char* q = "Hi C++!";`~~
- `char q[] = "Hi C++!";`
- `p = q;`
- `q[1] = 'o';` ~~`p[1] = 'o';`~~
- `cout << p[4];`

