Указатели и препратки

Трифон Трифонов

Увод в програмирането,
спец. Компютърни науки, 1 поток, 2018/19 г.

13–20 декември 2018 г.

Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.

Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.
- Интегрален нечислов тип

Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.
- Интегрален нечислов тип
- Параметризиран тип: ако `T` е тип данни, то `T*` е тип "указател към елемент от тип `T`"

Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.
- 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

Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.
- 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

Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.
- 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

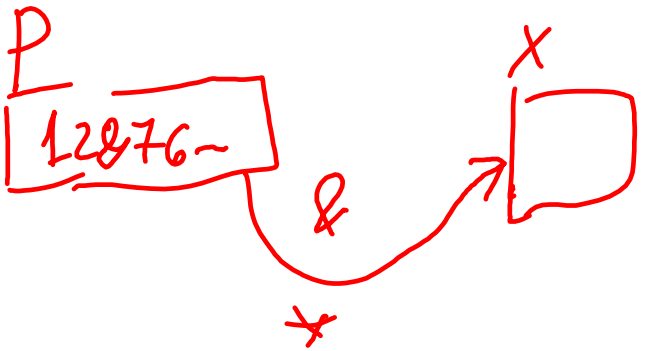
Тип указател

- **Множество от стойности:** всички възможни lvalue от даден тип и специалната стойност `nullptr`.
- 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

- реферирane (&<lvalue>)

Операции с указатели



- реферииране (&<lvalue>)
- дереферииране (*<указател>)
  - унарна операция!

Операции с указатели

- реферииране (&<lvalue>)
- дереферииране (*<указател>)
  - унарна операция!
- сравнение (==, !=, <, >, <=, >=)

Операции с указатели

- реферииране (&<lvalue>)
- дереферииране (*<указател>)
  - унарна операция!
- сравнение (==, !=, <, >, <=, >=)
- указателна аритметика (+,-,+=,-=,++,--)

Операции с указатели

- реферииране (&<lvalue>)
- дереферииране (*<указател>)
  - унарна операция!
- сравнение (==, !=, <, >, <=, >=)
- указателна аритметика (+,-,+=,-=,++,--)
- извеждане (<<)

Операции с указатели

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

Дефиниране на указателни променливи

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

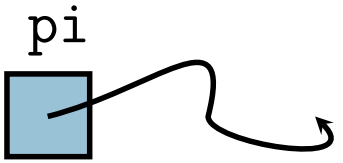
Примери:

Дефиниране на указателни променливи

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

Примери:

- `int *pi;`

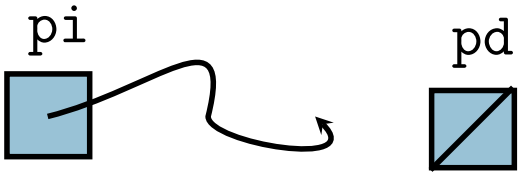


Дефиниране на указателни променливи

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

Примери:

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

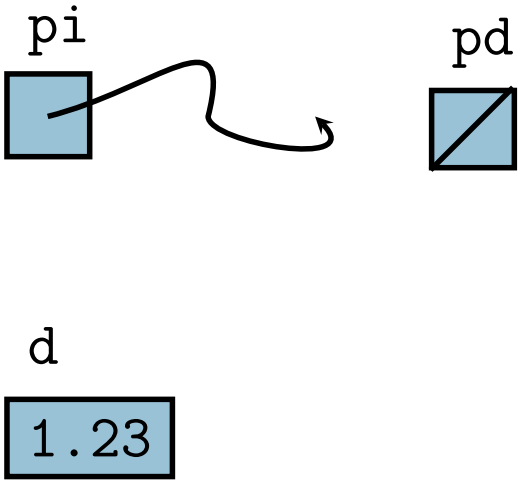


Дефиниране на указателни променливи

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

Примери:

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

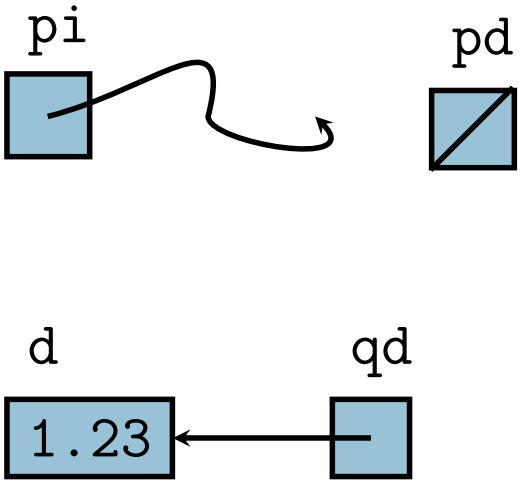


Дефиниране на указателни променливи

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

Примери:

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

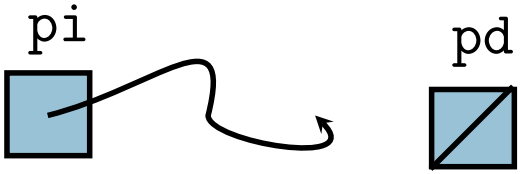


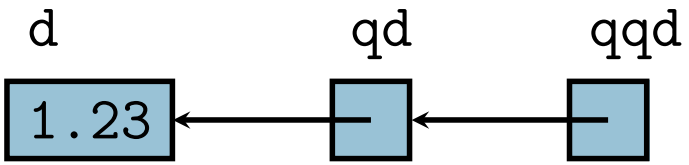
Дефиниране на указателни променливи

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

Примери:

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





Реферирane и дереферирane

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

Реферирane и дереферирane

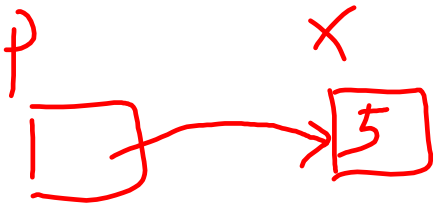
- &<име> — указател към променливата <име>
- *<указател> — мястото в паметта, сочено от <указател>

Реферирane и дереферирane

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

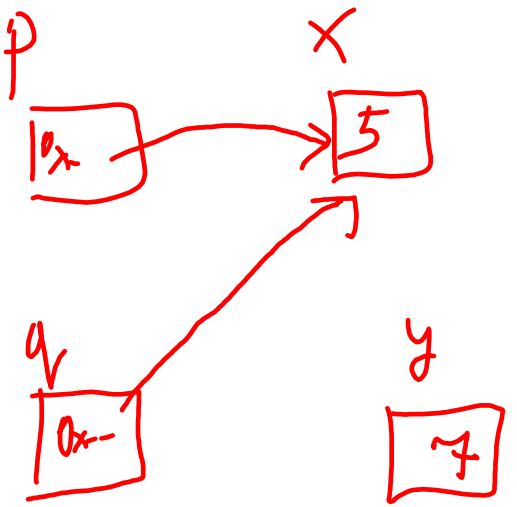
Реферирање и дереферирање

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



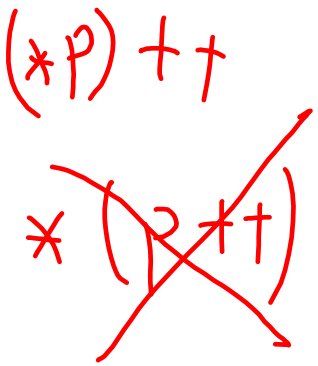
Рефериране и дерефериране

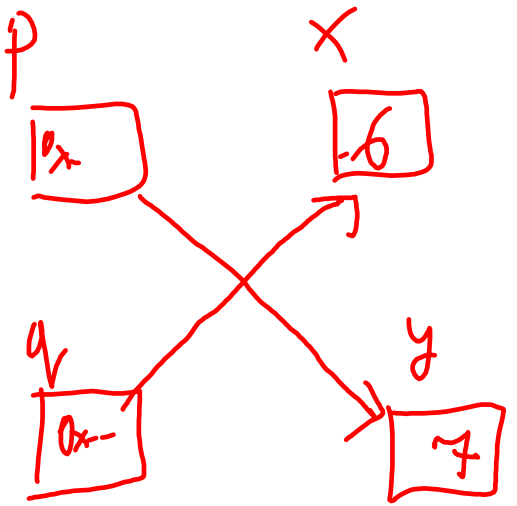
- &<име> — указател към променливата <име>
- *<указател> — мястото в паметта, сочено от <указател>
- **Примери:**
  - `int x = 5, *p = &x;`
  - `int *q = p, y = *p + 2;`



Рефериране и дерефериране

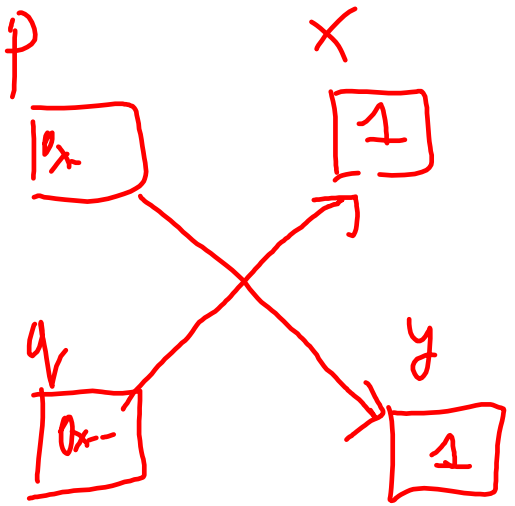
- &<име> — указател към променливата <име>
- *<указател> — мястото в паметта, сочено от <указател>
- **Примери:**
  - `int x = 5, *p = &x;`
  - `int *q = p, y = *p + 2;`
  - `*p++; p = &y;`





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;`



Реферииране и дереферииране

- &<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>!

Реферирane и дереферирane

- &<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~~

Реферирane и дереферирane

- &<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;`~~

Реферирane и дереферирane

- &<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>!

Реферирane и дереферирane

- &<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;`

Реферирane и дереферирane

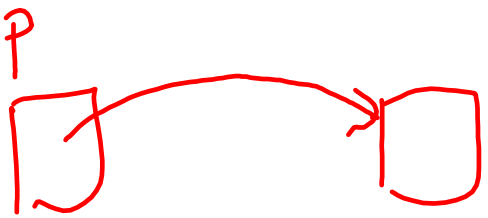
- &<име> — указател към променливата <име>
- *<указател> — мястото в паметта, сочено от <указател>
- **Примери:**
  - `int x = 5, *p = &x;`
  - `int *q = p, y = *p + 2;`
  - `*p++; p = &y;`
  - `*q = 1; *p = *q;`
- &<lvalue> връща като резултат <rvalue>!
  - ~~`&3`~~
  - ~~`&x = p;`~~
- *<rvalue> връща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

Реферииране и дереферииране

- &<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

Реферииране и дереферииране

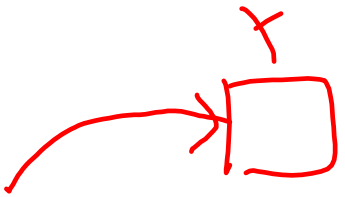
- &<име> — указател към променливата <име>
- *<указател> — мястото в паметта, сочено от <указател>
- **Примери:**
  - `int x = 5, *p = &x;`
  - `int *q = p, y = *p + 2;`
  - `*p++; p = &y;`
  - `*q = 1; *p = *q;`



- &<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)` $\Longleftrightarrow$ `p`

Реферииране и дереферииране

- &<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)` $\Longleftrightarrow$ `p`
  - `*(&x)` $\Longleftrightarrow$ `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

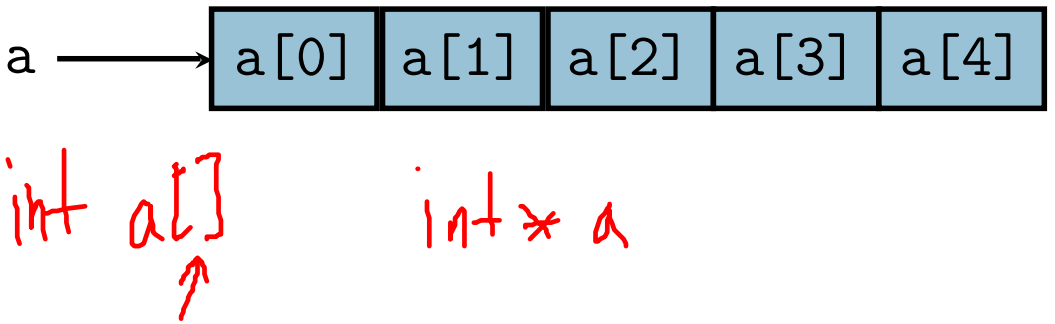
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е константен указател към първия му елемент.

Примери:

- `int a[5];`



Указатели и масиви

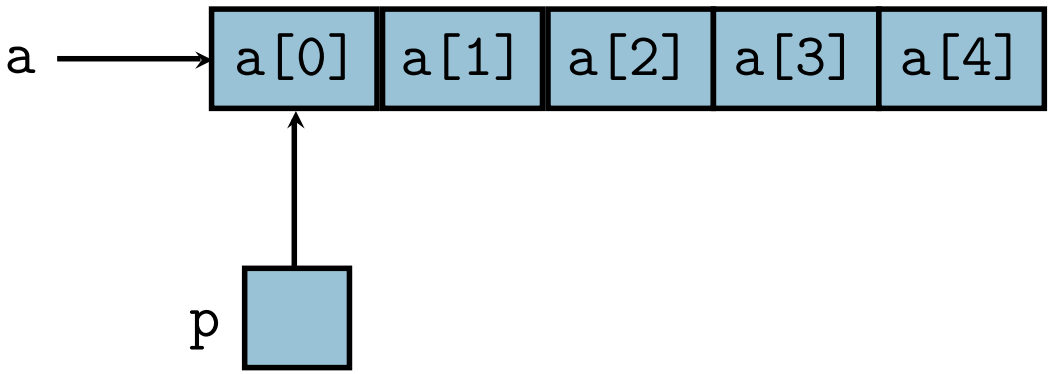
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е константен указател към първия му елемент.

Примери:

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



Указатели и масиви

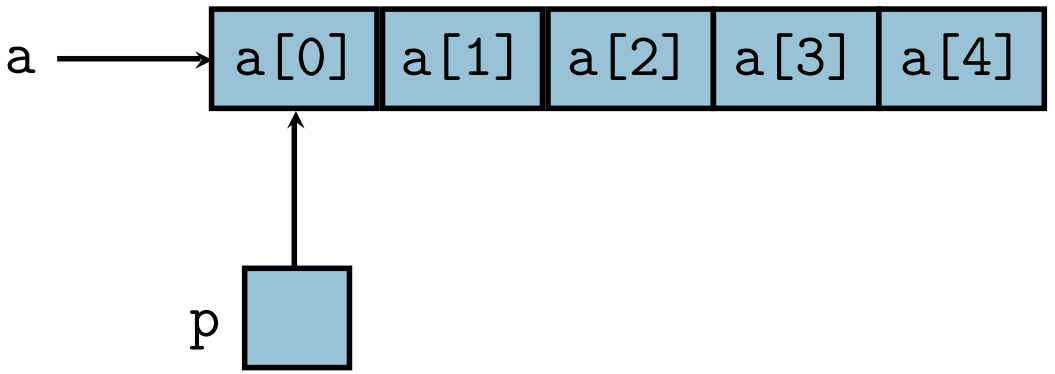
В C++ има много тясна връзка между указатели и масиви.

Факт

Името на масив е константен указател към първия му елемент.

Примери:

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



Указатели и масиви

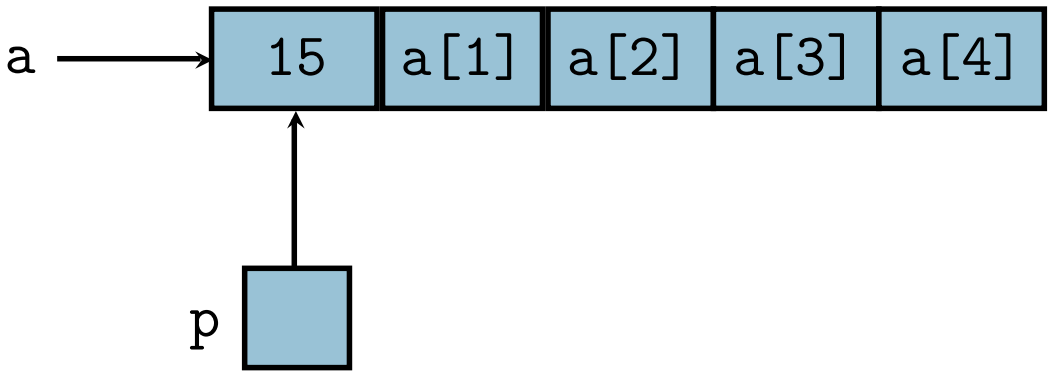
В C++ има много тясна връзка между указатели и масиви.

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

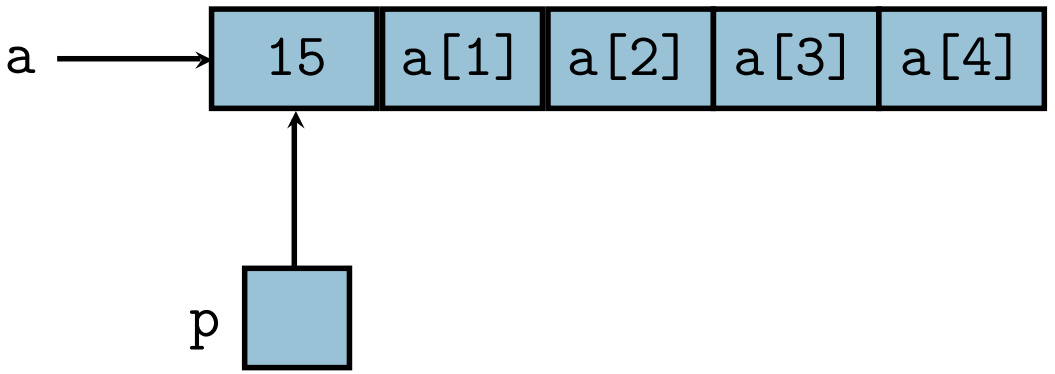
В C++ има много тясна връзка между указатели и масиви.

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

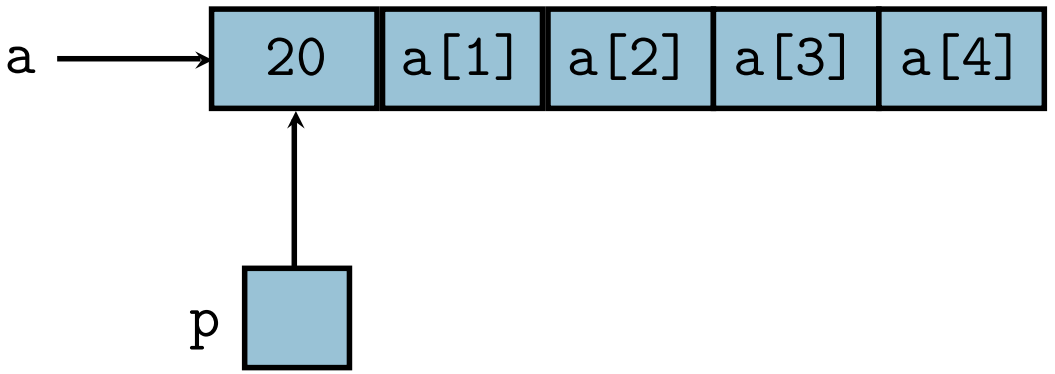
В C++ има много тясна връзка между указатели и масиви.

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:
  - <указател> [+ | -] <цяло_число>
  - <цяло_число> + <указател>
- прескачаме <цяло_число> клетки напред (+) или назад (-) от адреса, сочен от <указател>

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, заеман 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, заеман 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, заеман в паметта от <израз> или от стойност от <тип>
- Така, ако имаме `T* p;`...
- ...тогава `p + i` прескача `i * 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 * 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, заеман в паметта от <израз> или от стойност от <тип>
- Така, ако имаме `T* p;`...
- ...тогава `p + i` прескача `i * sizeof(T)` байта напред
- `(int)p` — цялото число, съответстващо на адреса сочен от `p`
- `p + i` $\Longleftrightarrow$ `(T*)((int)p + i * 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]` $\Longleftrightarrow$ `*(a + i)`

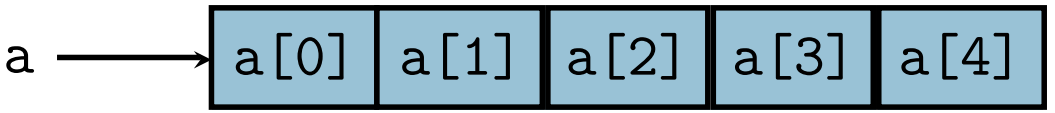
Указателна аритметика за масиви

Факт

Името на масив е константен указател към първия му елемент.
Освен това, `a[i]` $\Longleftrightarrow$ `*(a + i)`

Примери:

- `int a[5], x;`



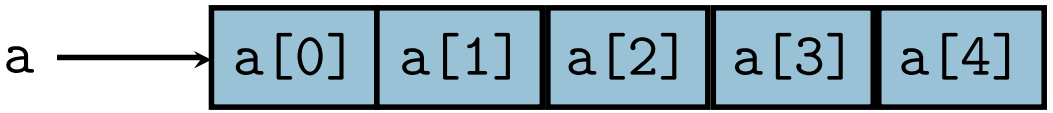
Указателна аритметика за масиви

Факт

Името на масив е константен указател към първия му елемент.
Освен това, `a[i]` $\Longleftrightarrow$ `*(a + i)`

Примери:

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

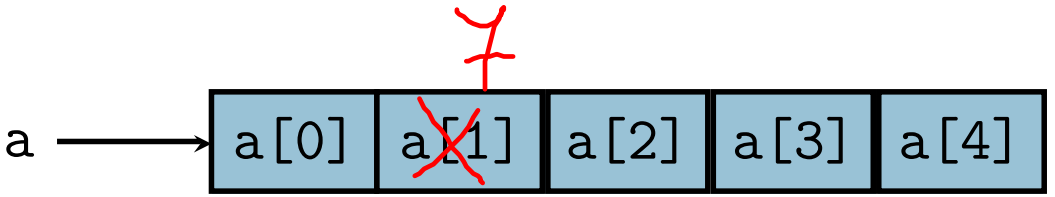


Указателна аритметика за масиви

Факт

Името на масив е константен указател към първия му елемент.
Освен това, `a[i]` $\Longleftrightarrow$ `*(a + i)`

Примери:



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

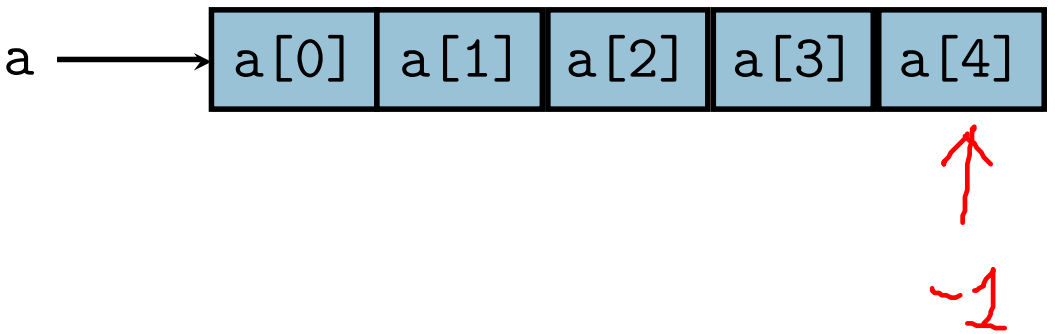
Указателна аритметика за масиви

Факт

Името на масив е константен указател към първия му елемент.
Освен това, `a[i]` $\Longleftrightarrow$ `*(a + i)`

Примери:

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

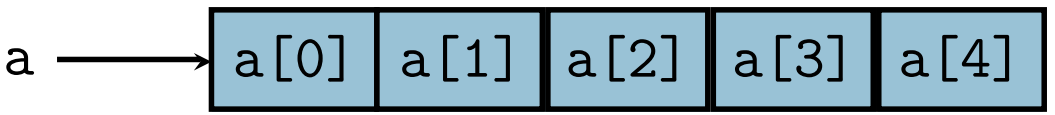


Указателна аритметика за масиви

Факт

Името на масив е константен указател към първия му елемент.
Освен това, `a[i]` $\Longleftrightarrow$ `*(a + i)`

Примери:



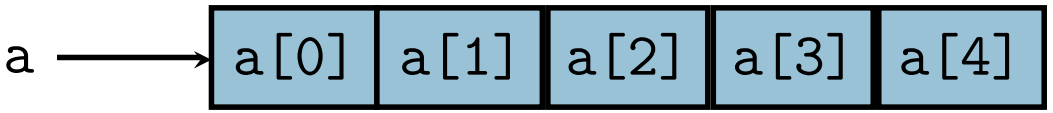
- `int a[5], x;`
- `cout << *a;`
- `*(a + 1) = 7;`
- `*(a + 4)--;`
- ~~`a++; a--; a = &x;`~~

Указателна аритметика за масиви

Факт

Името на масив е константен указател към първия му елемент.
Освен това, `a[i]` $\Longleftrightarrow$ `*(a + i)`

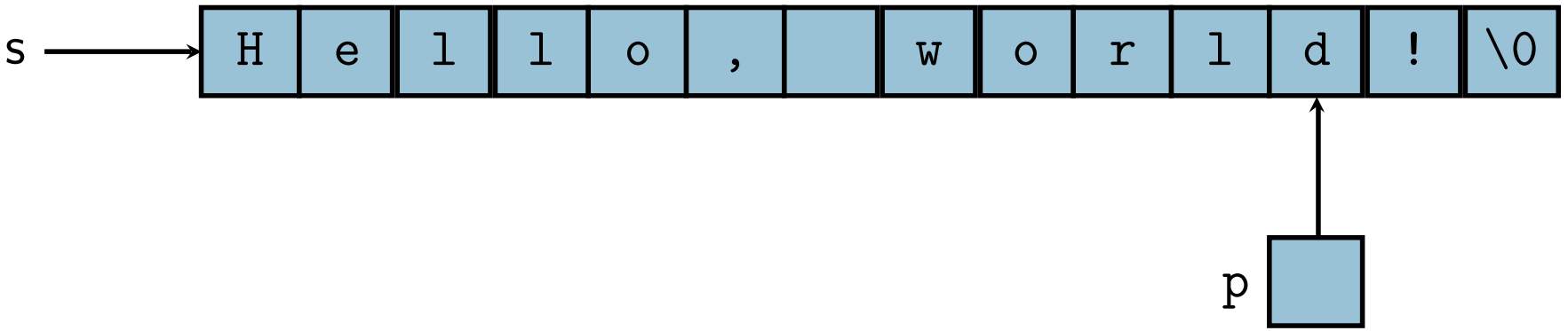
Примери:



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

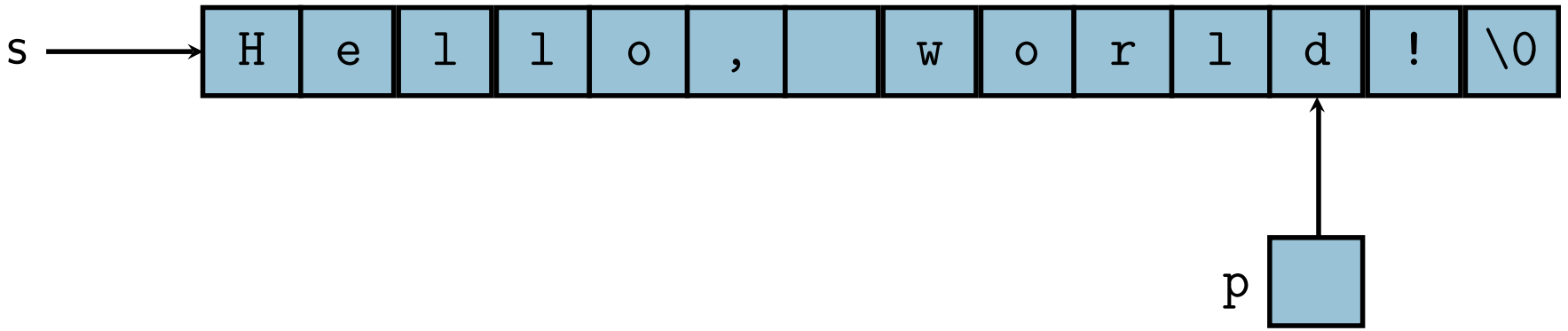
Указатели и низове

Низовете са масиви от символи



Указатели и низове

Низовете са масиви от символи

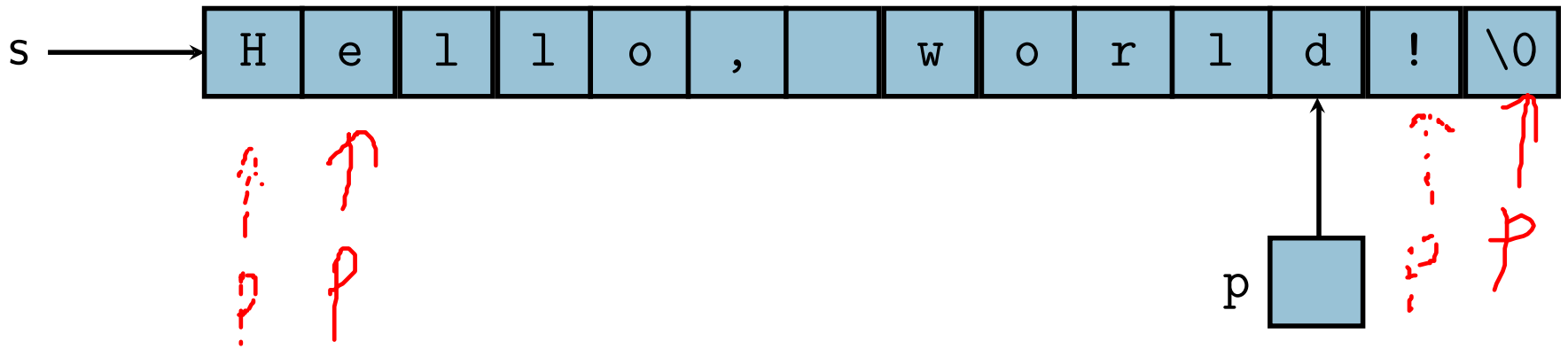


Примери:

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

Указатели и низове

Низовете са масиви от символи



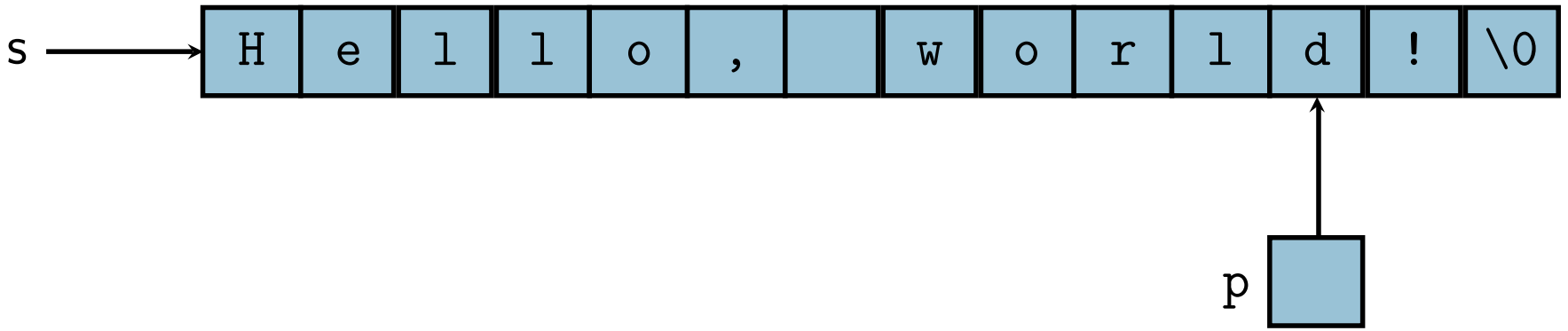
Примери:

```
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++;
}
```

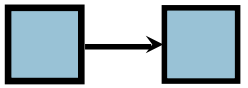
```
int strlen(char* p) {
  int n = 0;
  while (*p++) n++;
  return n;
}
```

Указатели и константи

- Константен указател (който е константа)

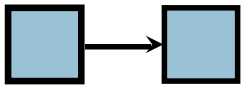
Указатели и константи

- Константен указател (който е константа)
  - <тип>* const



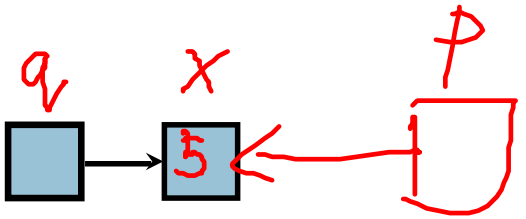
Указатели и константи

- 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;



var = const;

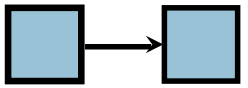
const = var;

p = q; ✓

~~q = p;~~

Указатели и константи

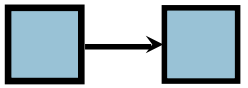
- Константен указател (който е константа)
  - <тип>* const
  - int x, *p = &x;
  - int* const q = p; ~~q = p + 2;~~ *q = 5;



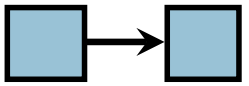
- Указател към константа (сочещ към константа)

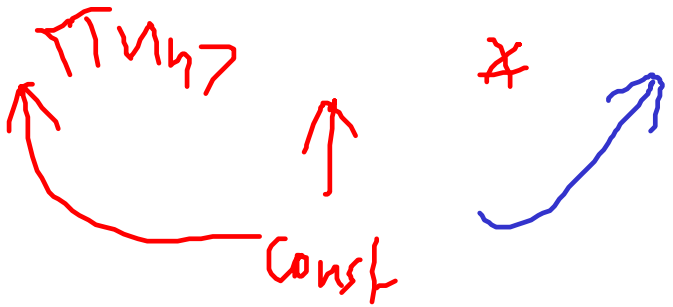
Указатели и константи

- Константен указател (който е константа)
  - <тип>* const
  - int x, *p = &x;
  - int* const q = p; ~~q = p + 2;~~ *q = 5;



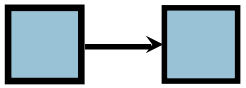
- Указател към константа (сочещ към константа)
  - const <тип>* $\Longleftrightarrow$ <тип> const*



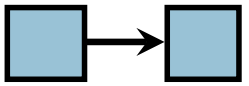


Указатели и константи

- Константен указател (който е константа)
  - <тип>* const
  - int x, *p = &x;
  - int* const q = p; ~~q = p + 2;~~ *q = 5;

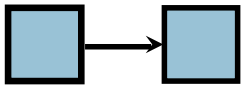


- Указател към константа (сочещ към константа)
  - const <тип>* $\Longleftrightarrow$ <тип> const*
  - int x, *p = &x;

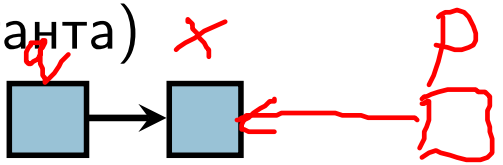


Указатели и константи

- Константен указател (който е константа)
  - <тип>* const
  - int x, *p = &x;
  - int* const q = p; ~~q = p + 2;~~ *q = 5;

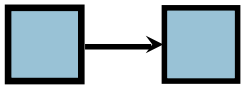


- Указател към константа (сочещ към константа)
  - const <тип>* $\Longleftrightarrow$ <тип> const*
  - int x, *p = &x;
  - int const* q = &x; q++; ~~p = q; *q = 5;~~

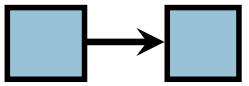


Указатели и константи

- Константен указател (който е константа)
  - <тип>* const
  - int x, *p = &x;
  - int* const q = p; ~~q = p + 2;~~ *q = 5;



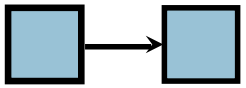
- Указател към константа (сочещ към константа)
  - const <тип>* $\Longleftrightarrow$ <тип> const*
  - int x, *p = &x;
  - int const* q = &x; q++; ~~p = q; *q = 5;~~



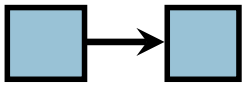
- Ако p е указател към константа, то *p е <rvalue>

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>* $\Longleftrightarrow$ <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*`).

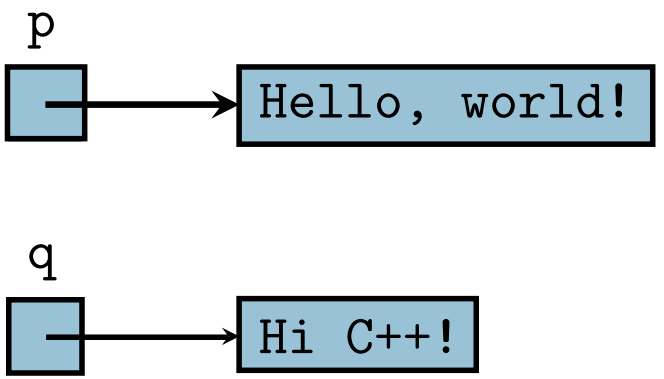
- `char const* p = "Hello, world!";`
- ~~`char* q = "Hi C++!";`~~

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++!";`

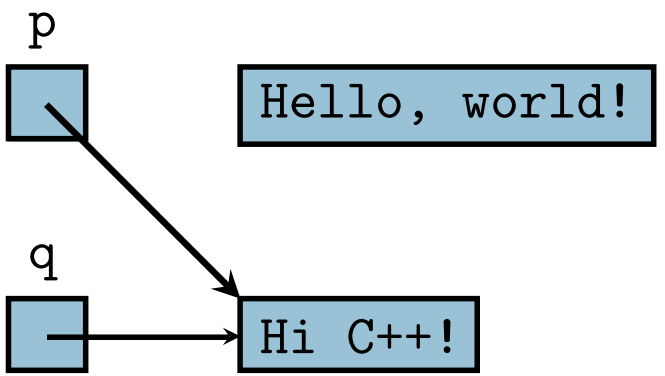


Указатели и низови константи

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;`

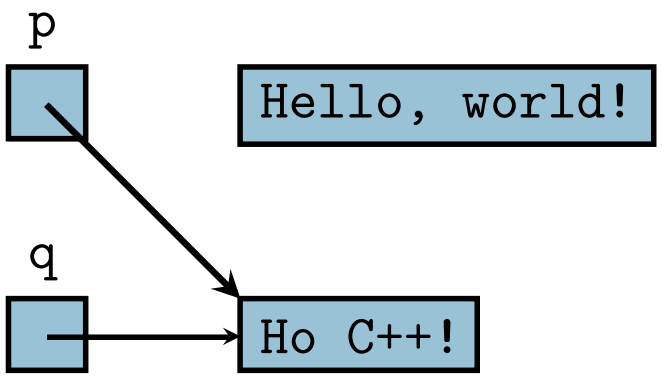


Указатели и низови константи

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';`~~

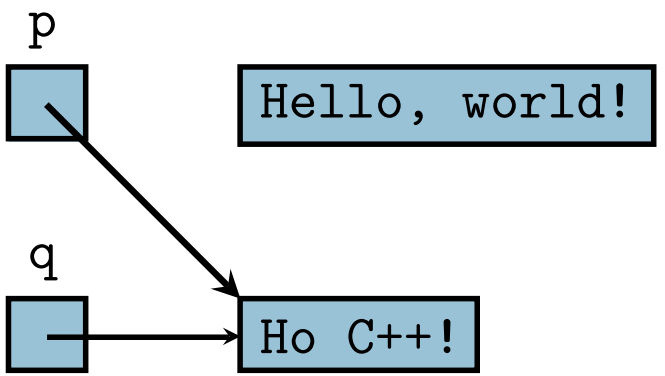


Указатели и низови константи

Факт

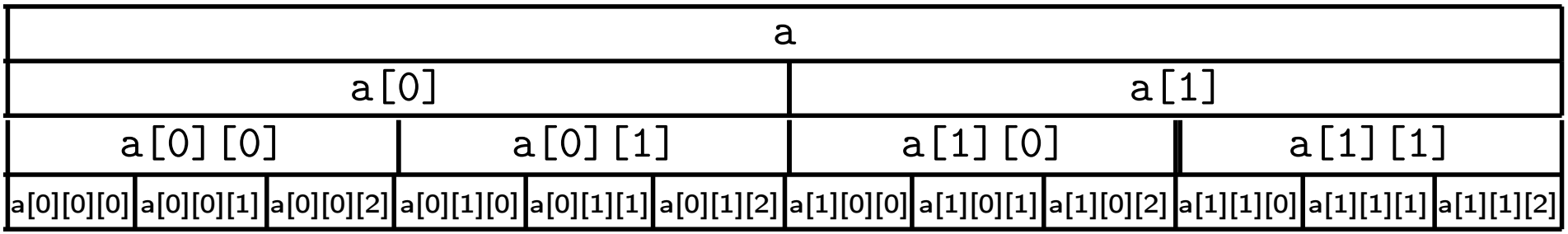
Името на низ е константен указател към първия му символ (`char* const`).
Низовите константи са указатели към константен символ (`char const*`).

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



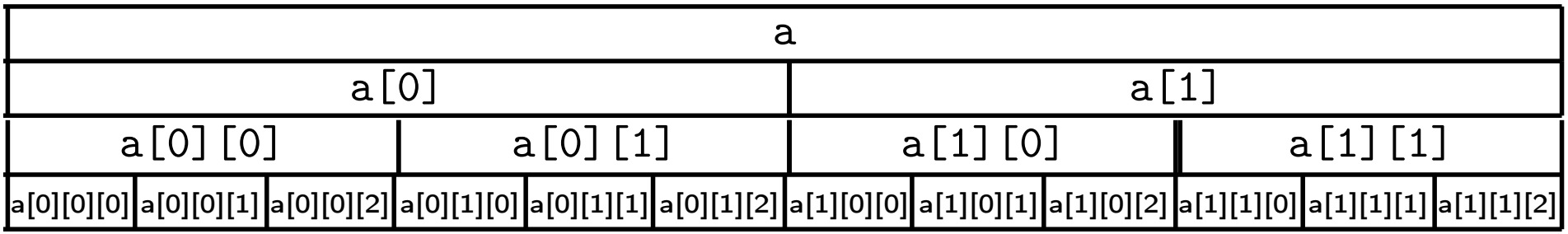
Указатели и многомерни масиви

- `int a[2][2][3];`



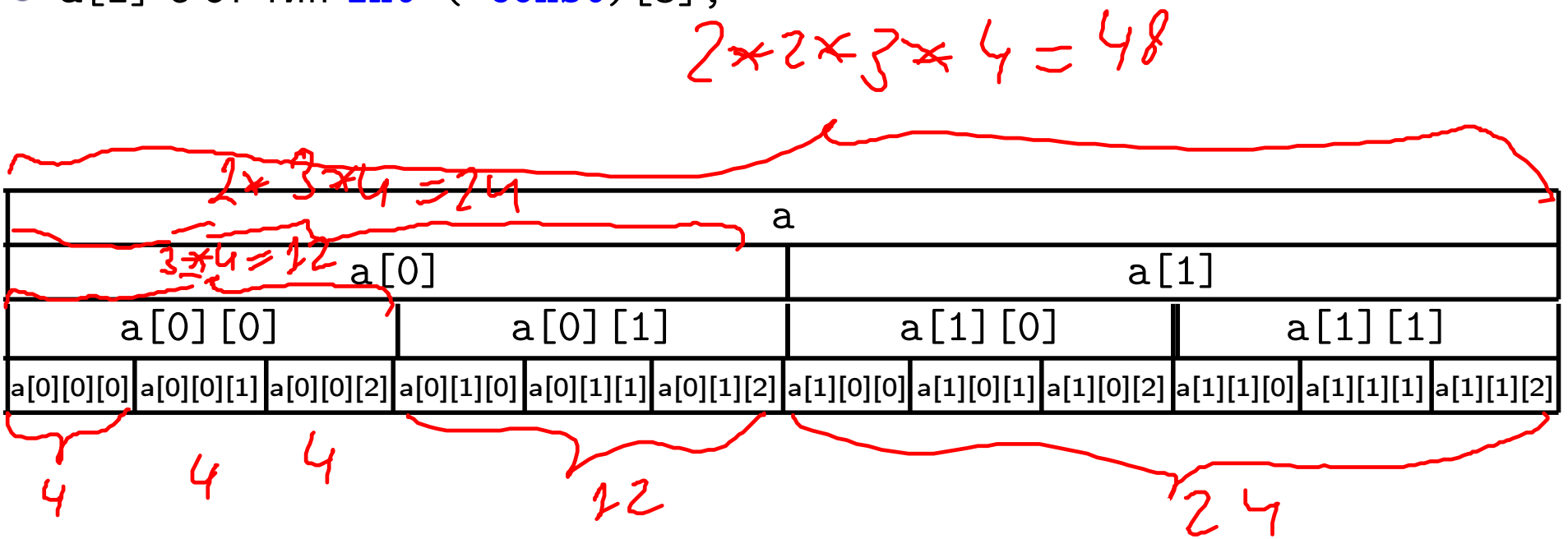
Указатели и многомерни масиви

- `int a[2][2][3];`
- a е от тип `int (*const)[2][3];`



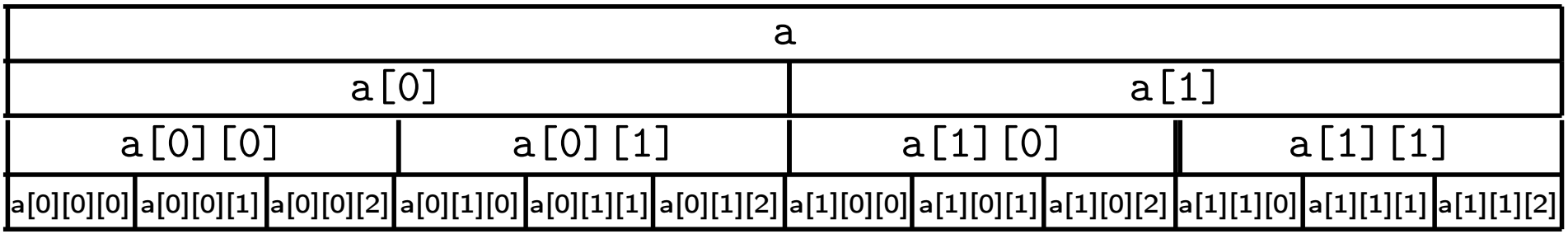
Указатели и многомерни масиви

- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`



Указатели и многомерни масиви

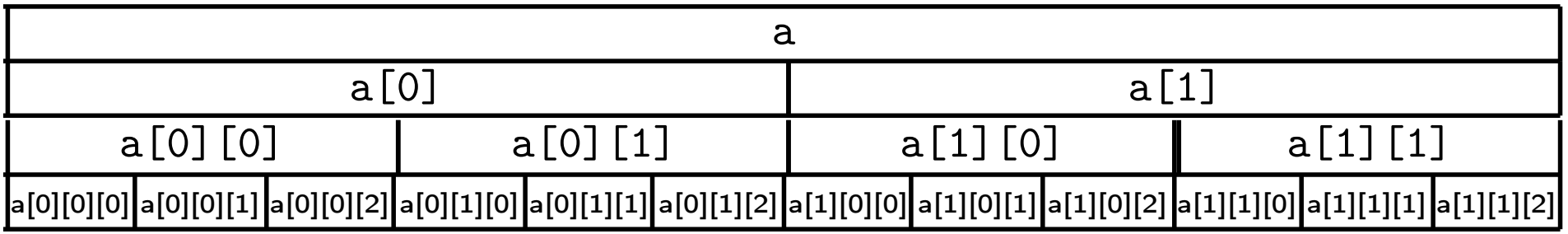
- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`
- `a[i][j]` е от тип `int *const;`



Указатели и многомерни масиви

- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`
- `a[i][j]` е от тип `int *const;`

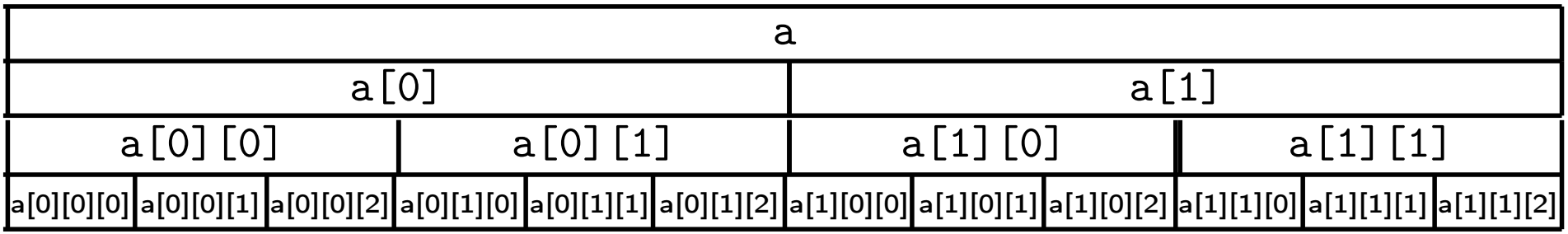
- `a[i]` $\Longleftrightarrow$ `*(a+i)`



Указатели и многомерни масиви

- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`
- `a[i][j]` е от тип `int *const;`

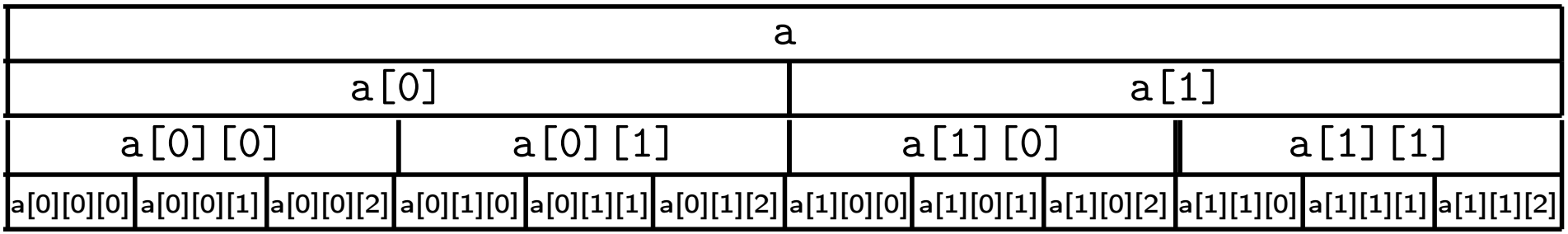
- `a[i]` $\Longleftrightarrow$ `*(a+i)`
- `a[i][j]` $\Longleftrightarrow$ `*(*(a+i)+j)`



Указатели и многомерни масиви

- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`
- `a[i][j]` е от тип `int *const;`

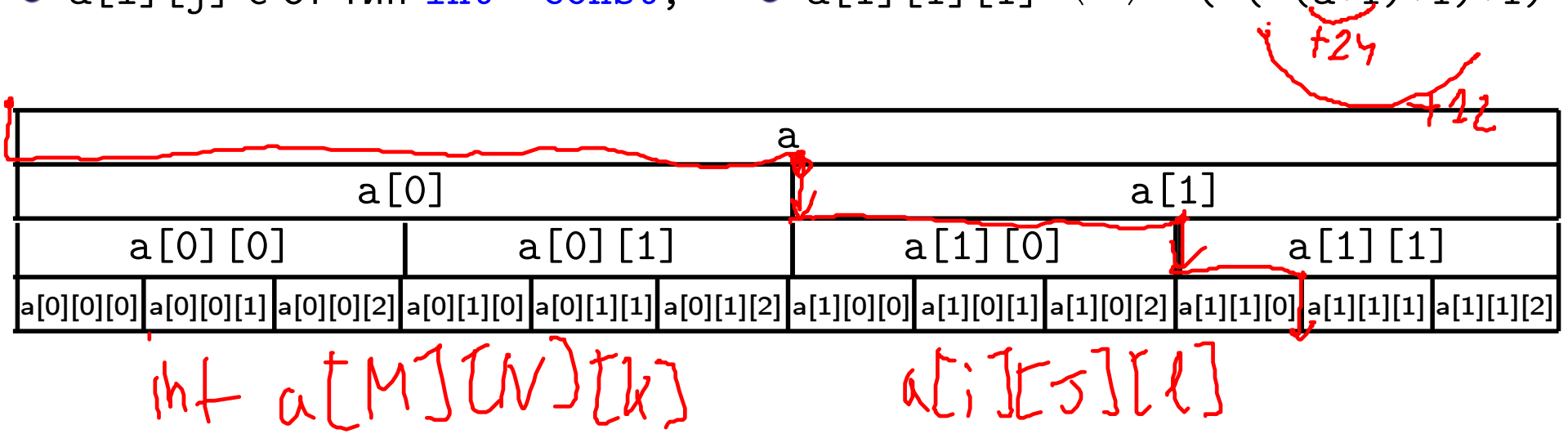
- `a[i]` $\Longleftrightarrow$ `*(a+i)`
- `a[i][j]` $\Longleftrightarrow$ `*(*(a+i)+j)`
- `a[i][j][k]` $\Longleftrightarrow$ `*(*(*(a+i)+j)+k)`



Указатели и многомерни масиви

- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`
- `a[i][j]` е от тип `int *const;`

- `a[i]` $\Longleftrightarrow$ `*(a+i)`
- `a[i][j]` $\Longleftrightarrow$ `*(*(a+i)+j)`
- `a[i][j][k]` $\Longleftrightarrow$ `*(*(*(a+i)+j)+k)`
- `a[1][1][1]` $\Longleftrightarrow$ `*(*(*(a+1)+1)+1)`



Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!

Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

Указател към неизвестен тип

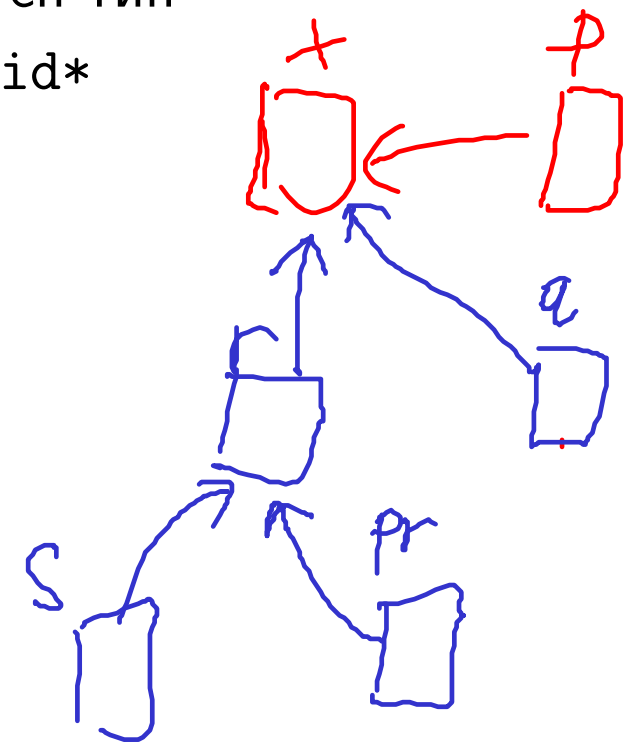
- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`

Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип
- ✓ Преобразуваме автоматично от `T*` към `void*`
  - `int x, *p; void *q = p;`
  - `void *r = &x, *pr = &r, *s = &r;`



Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

Указател към неизвестен тип

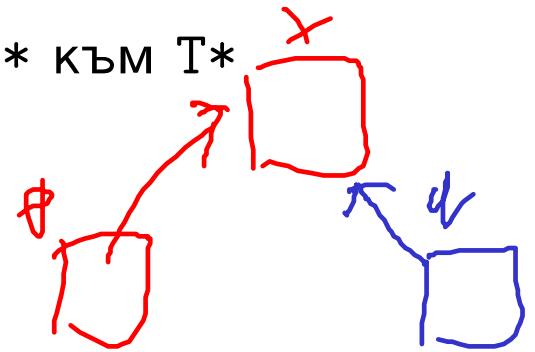
- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

- `int* p; void* q = p;`



Указател към неизвестен тип

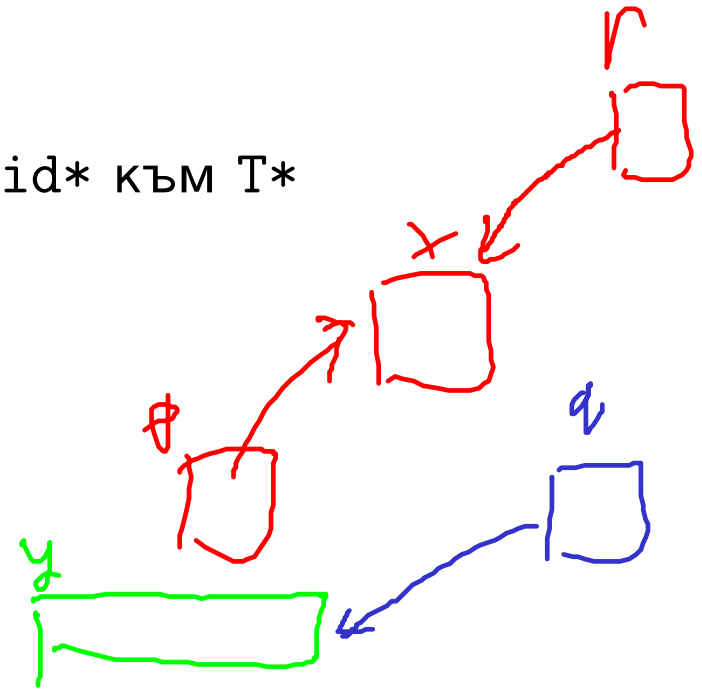
- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

- `int* p; void* q = p;`
- ~~`int* r = q;`~~



Указател към неизвестен тип

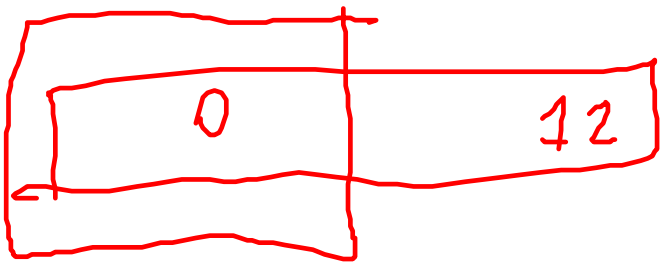
- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

- `int* p; void* q = p;`
- ~~`int* r = q;`~~
- `int* s = (int*)q;`



Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

- `int* p; void* q = p;`
- ~~`int* r = q;`~~
- `int* s = (int*)q;`

× Няма дереферииране (`void` е празният тип)

Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

- `int* p; void* q = p;`
- ~~`int* r = q;`~~
- `int* s = (int*)q;`

× Няма дереферииране (`void` е празният тип)

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

Указател към неизвестен тип

- **Проблем:** Не можем да насочваме един и същ указател към променливи от различен тип!
- **Решение:** `void*` — указател към неизвестен тип

✓ Преобразуваме автоматично от `T*` към `void*`

- `int x, *p; void *q = p;`
- `void *r = &x, *pr = &r, *s = &r;`

× Няма автоматично преобразуване от `void*` към `T*`

- `int* p; void* q = p;`
- ~~`int* r = q;`~~
- `int* s = (int*)q;`

× Няма дереферииране (`void` е празният тип)

- `int x; void* p = &x;`
- ~~`*p = 2; void y = *p;`~~

Препратка

- **Множество от стойности:** всички възможни lvalue от даден тип

Препратка

- **Множество от стойности:** всички възможни lvalue от даден тип
- Параметризиран тип: ако `T` е тип данни, то `T&` е тип "препратка към елемент от тип `T`"

Препратка

- **Множество от стойности:** всички възможни lvalue от даден тип
- Параметризиран тип: ако `T` е тип данни, то `T&` е тип "препратка към елемент от тип `T`"
- Физическо представяне:

Препратка

- **Множество от стойности:** всички възможни lvalue от даден тип
- Параметризиран тип: ако `T` е тип данни, то `T&` е тип "препратка към елемент от тип `T`"
- Физическо представяне:
  - на теория: както реши компилаторът

Препратка

- **Множество от стойности:** всички възможни lvalue от даден тип
- Параметризиран тип: ако `T` е тип данни, то `T&` е тип "препратка към елемент от тип `T`"
- Физическо представяне:
  - на теория: както реши компилаторът
  - на практика: еквивалентно на константен указател към `T`

Препратка

- **Множество от стойности:** всички възможни lvalue от даден тип
- Параметризиран тип: ако `T` е тип данни, то `T&` е тип "препратка към елемент от тип `T`"
- Физическо представяне:
  - на теория: както реши компилаторът
  - на практика: екиквалентно на константен указател към `T`
  - `T&` $\Longleftrightarrow$ `T* const`

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Пример:

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Пример:

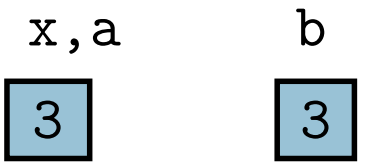
- `int x = 3;`

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Пример:

- `int x = 3;`
- `int &a = x, b = a;`



Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Пример:

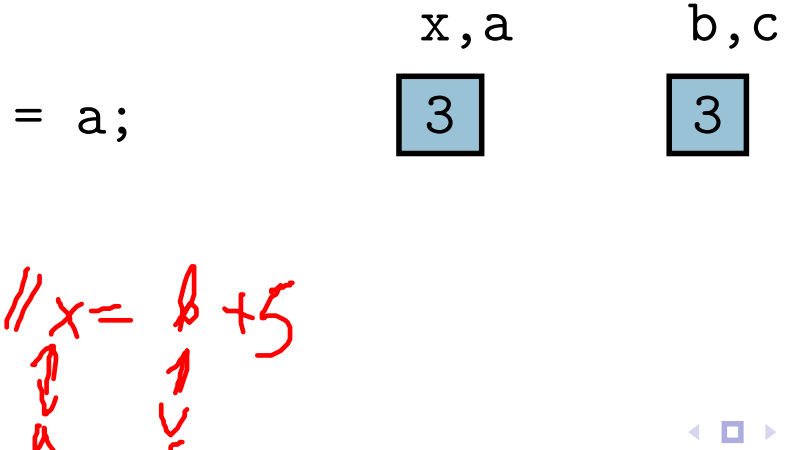
- `int x = 3;`
- `int &a = x, b = a;`
- `int &c = b;`

Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  { , &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Пример:

- `int x = 3;`
- `int &a = x, b = a;`
- `int &c = b;`
- `a = c + 5;`



Дефиниране на препратка

- <тип>& <идентификатор> = <обект>
  {, &<идентификатор> = <обект> };
- инициализацията е задължителна!
  - както и на константните указатели
- препратката не може да се пренасочва към друг обект
  - както и константните указатели

Пример:

- `int x = 3;`
- `int &a = x, b = a;`
- `int &c = b;`
- `a = c + 5;`

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран
  - Указателите могат да бъдат насочвани към различни адреси

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран
  - Указателите могат да бъдат насочвани към различни адреси
- Препратките винаги са закачени за съществуващ обект

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран
  - Указателите могат да бъдат насочвани към различни адреси
- Препратките винаги са закачени за съществуващ обект
  - Указателите могат да имат стойност `nullptr`

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран
  - Указателите могат да бъдат насочвани към различни адреси
- Препратките винаги са закачени за съществуващ обект
  - Указателите могат да имат стойност `nullptr`
- Препратката не се различава от оригинала

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран
  - Указателите могат да бъдат насочвани към различни адреси
- Препратките винаги са закачени за съществуващ обект
  - Указателите могат да имат стойност `nullptr`
- Препратката не се различава от оригинала
  - Указателят изисква изрично дереферииране с `*`

Сравнение на препратки и указатели

- Самата препратка не е обект, който може да бъде манипулиран
  - Указателите могат да бъдат насочвани към различни адреси
- Препратките винаги са закачени за съществуващ обект
  - Указателите могат да имат стойност `nullptr`
- Препратката не се различава от оригинала
  - Указателят изисква изрично дереферииране с `*`
- Препратките на един и същ обект са взаимнозаменяеми

Константни препратки

- `const<тип>&` $\Longleftrightarrow$ `<тип> const&`

Константни препратки

- `const`<тип>`&` $\Longleftrightarrow$ <тип> `const&`
- Представляват константен "изглед" на дадено място в паметта

Константни препратки

- `const`<тип>`&` $\iff$ <тип> `const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Константни препратки

- `const`<тип>`&` $\iff$ <тип> `const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

Константни препратки

- `const<тип>&` $\iff$ `<тип> const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

- `int a = 3;`

Константни препратки

- `const<тип>&` $\iff$ `<тип> const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

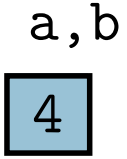
- `int a = 3;`
- `a++;`

Константни препратки

- `const<тип>&` $\iff$ `<тип> const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

- `int a = 3;`
- `a++;`
- `int& b = a;`

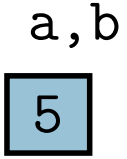


Константни препратки

- `const<тип>&` $\iff$ `<тип> const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

- `int a = 3;`
- `a++;`
- `int& b = a;`
- `b++;`

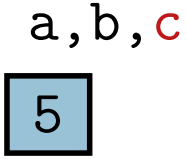


Константни препратки

- `const<тип>&` $\iff$ `<тип> const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

- `int a = 3;`
- `a++;`
- `int& b = a;`
- `b++;`
- `int const& c = b;`



Константни препратки

- `const<тип>&` $\Longleftrightarrow$ `<тип> const&`
- Представляват константен "изглед" на дадено място в паметта
- Няма разлика между константи препратки и препратки на константи

Пример:

```
int a = 3;
a++;
int& b = a;
b++;
int const& c = b;
c++;
```

a,b,c