

Указатели и препратки

Трифон Трифонов

Увод в програмирането,
спец. Компютърни науки, 1 поток, 2021/22 г.

6–20 декември 2021 г.

Тази презентация е достъпна под лиценза Creative Commons Признание-Некомерсиално-Споделяне на споделеното 4.0 Международен 

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

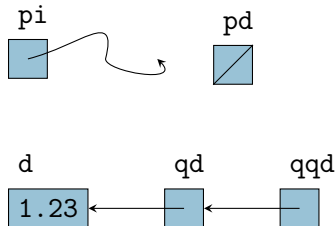
- рефериране (&<lvalue>)
- дерефериране (*<указател>)
 - унарна операция!
- сравнение (==, !=, <, >, <=, >=)
- указателна аритметика (+, -, +=, -=, ++, --)
- извеждане (<<)
- няма въвеждане! (>>)

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;`



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

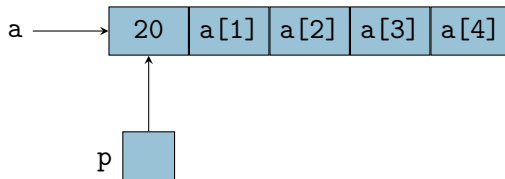
В C++ има много тясна връзка между указатели и масиви.

Факт

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

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.
- 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”?
- **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
- `(long)p` — цялото число, съответстващо на адреса сочен от `p`
- $p + i \iff (T*)((long)p + i * \text{sizeof}(T))$

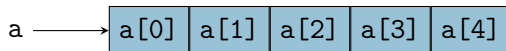
Указателна аритметика за масиви

Факт

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

Освен това, $a[i] \iff *(a + i)$

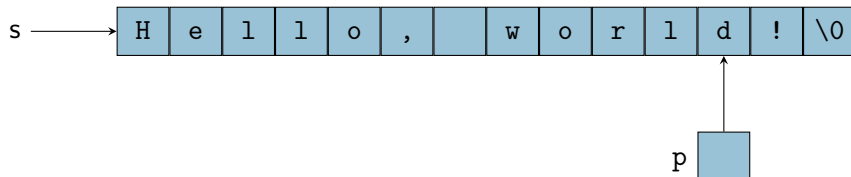
Примери:



- `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



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;  
}
```

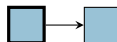
Указатели и константи

- Константен указател (който е константа)

- `<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>`

- 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`).
 Низовите константи са **указатели към константен символ** (`char const*`).

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

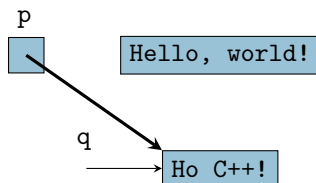
- ~~`char* q = "Hi C++!";`~~

- `char q[] = "Hi C++!";`

- `p = q;`

- `q[1] = 'o';` ~~`p[1] = 'o';`~~

- `cout << p[4];`



Указатели и многомерни масиви

- `int a[2][2][3];`
- `a` е от тип `int (*const)[2][3];`
- `a[i]` е от тип `int (*const)[3];`
- `a[i][j]` е от тип `int *const;`
- $a[i] \iff *(a+i)$
- $a[i][j] \iff *((a+i)+j)$
- $a[i][j][k] \iff *((*(a+i)+j)+k)$
- $a[1][1][1] \iff *((*(a+1)+1)+1)$

a											
a[0]						a[1]					
a[0][0]			a[0][1]			a[1][0]			a[1][1]		
a[0][0][0]	a[0][0][1]	a[0][0][2]	a[0][1][0]	a[0][1][1]	a[0][1][2]	a[1][0][0]	a[1][0][1]	a[1][0][2]	a[1][1][0]	a[1][1][1]	a[1][1][2]

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** дерефериране (`void` е празният тип)
 - `int x; void* p = &x;`
 - ~~`*p = 2; void y = *p;`~~

Препратка (reference)

- **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**
 - $T\& \iff T* \text{ const}$

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
 - 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 дерефериране с `*`
- Препратките на един и същ обект са взаимозаменяеми

Константни препратки

- `const <тип>& \iff <тип> const&`
- Представяват 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
- 5