Лекция 3. Массивы, указатели и ссылки

Александр Смаль

CS центр

12 сентября 2017

Санкт-Петербург

Указатели

- Указатель — это переменная, хранящая адрес некоторой ячейки памяти.
- Указатели являются типизированными.

```
int   i = 3; // переменная типа int
int * p = 0; // указатель на переменную типа int
```

- Нулевому указателю (которому присвоено значение 0) не соответствует никакая ячейка памяти.
- Оператор взятия адреса переменной &.
- Оператор разыменования *.

```
p  = &i; // указатель p указывает на переменную i
*p = 10; // изменяется ячейка по адресу p, т.е. i
```

Передача параметров по указателю

Рассмотрим функцию, меняющую параметры местами:

```
void swap (int a, int b) {
    int t = a;
    a = b;
    b = t;
}

int main() {
    int k = 10, m = 20;
    swap (k, m);
    cout << k << ' ' << m << endl; // 10 20
    return 0;
}
```

swap изменяет локальные копии переменных k и m.

Передача параметров по указателю

Вместо значений типа `int` будем передавать указатели.

```
void swap (int * a, int * b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int main() {
    int k = 10, m = 20;
    swap (&k, &m);
    cout << k << ' ' << m << endl; // 20 10
    return 0;
}
```

`swap` изменяет переменные `k` и `m` по указателям на них.

Массивы

- Массив — это набор однотипных элементов, расположенных в памяти друг за другом, доступ к которым осуществляется по индексу.
- C++ позволяет определять массивы на стеке.

```
// массив 1 2 3 4 5 0 0 0 0 0
int m[10] = {1, 2, 3, 4, 5};
```

- Индексация массива начинается с `0`, последний элемент массива длины `n` имеет индекс `n - 1`.

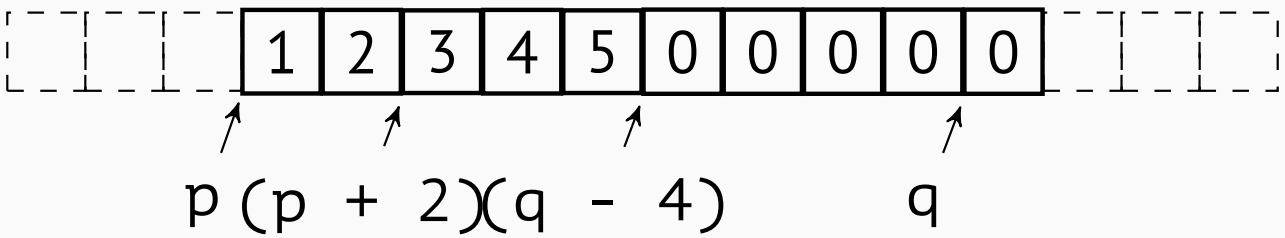
```
for (int i = 0; i < 10; ++i)
    cout << m[i] << ' ';
cout << endl;
```

Связь массивов и указателей

- Указатели позволяют передвигаться по массивам.
- Для этого используется арифметика указателей:

```
int m[10] = {1, 2, 3, 4, 5};
int * p = &m[0]; // адрес начала массива
int * q = &m[9]; // адрес последнего элемента
```

  - `(p + k)` — сдвиг на k ячеек типа `int` вправо.
  - `(p - k)` — сдвиг на k ячеек типа `int` влево.
  - `(q - p)` — количество ячеек между указателями.
  - `p[k]` эквивалентно `*(p + k)`.



Примеры

Заполнение массива:

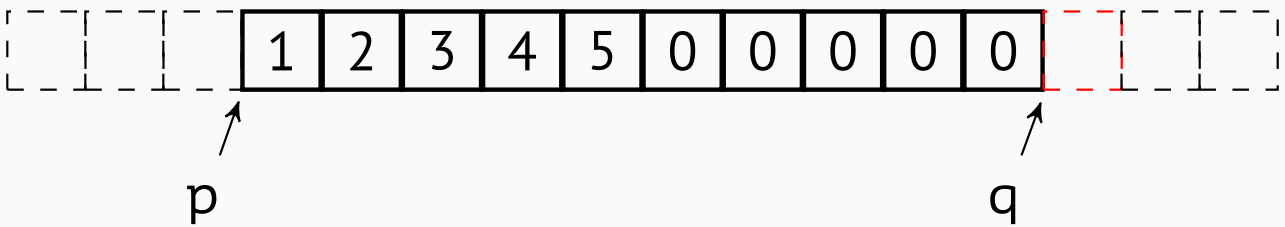
```
int m[10] = {}; // изначально заполнен нулями
//              &m[0]          &m[9]
for (int * p = m ; p <= m + 9; ++p )
    *p = (p - m) + 1;
// Массив заполнен числами от 1 до 10
```

Передача массива в функцию:

```
int max_element (int * m, int size) {
    int max = *m;
    for (int i = 1; i < size; ++i)
        if (m[i] > max)
            max = m[i];
    return max;
}
```

Два способа передачи массива

```
bool contains(int * m, int size, int value) {
    for (int i = 0; i != size; ++i)
        if (m[i] == value)
            return true;
    return false;
}
bool contains(int * p, int * q, int value) {
    for (; p != q; ++p)
        if (*p == value)
            return true;
    return false;
}
```



Возрат указателя из функции

Функция для поиска максимума в массиве:

```
int max_element (int * p, int * q) {
    int max = *p;
    for (; p != q; ++p)
        if (*p > max)
            max = *p;

    return max;
}
```

```
    int m[10] = {...};
    int max = max_element(m, m + 10);
    cout << "Maximum = " << max << endl;
```

Возрат указателя из функции

Функция для поиска максимума в массиве:

```
int * max_element (int * p, int * q) {
    int * pmax = p;
    for (; p != q; ++p)
        if (*p > *pmax)
            pmax = p;

    return pmax;
}
```

```
    int m[10] = {...};
    int * pmax = max_element(m, m + 10);
    cout << "Maximum = " << *pmax << endl;
```

Возрат значения через указатель

Функция для поиска максимума в массиве:

```
bool max_element (int * p, int * q, int * res) {
    if (p == q)
        return false;
    *res = *p;
    for (; p != q; ++p)
        if (*p > *res)
            *res = *p;
    return true;
}
```

```
    int m[10] = {...};
    int max = 0;
    if (max_element(m, m + 10, &max))
        cout << "Maximum = " << max << endl;
```

Возрат значения через указатель на указатель

Функция для поиска максимума в массиве:

```
bool max_element (int * p, int * q, int ** res) {
    if (p == q)
        return false;
    *res = p;
    for (; p != q; ++p)
        if (*p > **res)
            *res = p;
    return true;
}
```

```
    int m[10] = {...};
    int * pmax = 0;
    if (max_element(m, m + 10, &pmax))
        cout << "Maximum = " << *pmax << endl;
```

Недостатки указателей

- Использование указателей синтаксически загрязняет код и усложняет его понимание. (Приходится использовать операторы * и &.)
- Указатели могут быть неинициализированными (некорректный код).
- Указатель может быть нулевым (корректный код), а значит указатель нужно проверять на равенство нулю.
- Арифметика указателей может сделать из корректного указателя некорректный (легко промахнуться).

Ссылки

- Для того, чтобы исправить некоторые недостатки указателей, в C++ введены ссылки.
- Ссылки являются “красивой обёрткой” над указателями:

```
void swap (int & a, int & b) {
    int t = b;
    b = a;
    a = t;
}

int main() {
    int k = 10, m = 20;
    swap (k, m);
    cout << k << ' ' << m << endl; // 20 10
    return 0;
}
```

Различия ссылок и указателей

- Ссылка не может быть неинициализированной.

```
int * p; // OK
int & l; // ошибка
```

- У ссылки нет нулевого значения.

```
int * p = 0; // OK
int & l = 0; // ошибка
```

- Ссылку нельзя переприсвоить:

```
int a = 10, b = 20;
int * p = &a; // p указывает на a
p = &b;       // p указывает на b
int & l = a;  // l ссылается на a
l = b;        // a присваивается значение b
```

Различия ссылок и указателей

- Нельзя получить адрес ссылки или ссылку на ссылку.

```
int a = 10;
int * p = &a;  // p указывает на a
int ** pp = &p;// pp указывает на переменную p
int & l = a;   // l ссылается на a
int * pl = &l; // pl указывает на переменную a
int && ll = l; // ошибка
```

- Нельзя создавать массивы ссылок.

```
int * mp[10] = {};  // массив указателей на int
int & ml[10] = {};  // ошибка
```

- Для ссылок нет арифметики.

lvalue и rvalue

- Выражения в C++ можно разделить на два типа:
  1. **lvalue** — выражения, значения которых являются *ссылкой* на переменную/элемента массива, а значит могут быть указаны слева от оператора =.
  2. **rvalue** — выражения, значения которых являются временными и не соответствуют никакой переменной/элементу массива.
- Указатели и ссылки могут указывать только на lvalue.

```
int a = 10, b = 20;
int m[10] = {1,2,3,4,5,5,4,3,2,1};
int & l1 = a;          // OK
int & l2 = a + b;      // ошибка
int & l3 = *(m + a / 2);          // OK
int & l4 = *(m + a / 2) + 1;      // ошибка
int & l5 = (a + b > 10) ? a : b;  // OK
```

Время жизни переменной

Следует следить за временем жизни переменных.

```
int * foo() {
    int a = 10;
    return &a;
}

int & bar() {
    int b = 20;
    return b;
}
```

```
    int * p = foo();
    int & l = bar();
```