**ЛЕКЦИЯ 7. УКАЗАТЕЛИ, ССЫЛКИ И МАССИВЫ**

Оглавление

[1 Указатели. Основные понятия 1](#_Toc148880656)

[2 Операции с указателями. 4](#_Toc148880657)

[3 Указатели и ссылки. 6](#_Toc148880658)

[4 Константы и указатели 8](#_Toc148880659)

[4.1 Указатели на константы 8](#_Toc148880660)

[4.2 Константный указатель 9](#_Toc148880661)

[4.3 Константный указатель на константу 9](#_Toc148880662)

[5 Указатели и массивы 10](#_Toc148880663)

[5.1 Указатель на массив 10](#_Toc148880664)

[5.2 Отличие массива от указателя 10](#_Toc148880665)

[5.3 Многомерные массивы и указатели на многомерные массивы. 11](#_Toc148880666)

[6 Ссылки и массивы 13](#_Toc148880667)

# Указатели. Основные понятия

**Указатели** представляют собой переменные, значением которых служат адреса других объектов (переменных, констант, указателей) или функций. Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

Для определения указателя надо указать тип объекта, на который указывает указатель, и символ звездочки **\***.

**тип \*имя\_указателя; // тип** –тип данных, адрес которых хранит указатель

**// имя\_указателя** – имя\_переменной

Для получения адреса переменной применяется операция **&**.

**Пример 1**

**#include <iostream>**

**int main() {**

**short c = 12, s = 2;**

**int d = 10;**

**short\* pc = &c; // получаем адрес переменной с типа short**

**int\* pd = &d; // получаем адрес переменной d типа int**

**short\* ps = &s; // получаем адрес переменной s типа short**

**std::cout << "Variable c: address=" << pc << "\t value=" << \*pc <<**

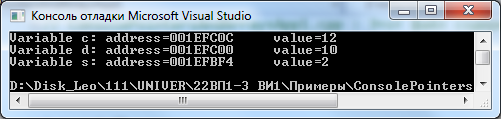
**std::endl;**

**std::cout << "Variable d: address=" << pd << "\t value=" << \*pd <<**

**std::endl;**

**std::cout << "Variable s: address=" << ps << "\t value=" << \*ps <<**

**std::endl;**

** return 0;**

**}**

Так как указатель хранит адрес, то можно по этому адресу получить хранящееся там значение. Для этого применяется операция \* или **операция разыменования**.

**Пример 2**

**#include <iostream>**

**int main() {**

**int x = 10;**

**int\* p;**

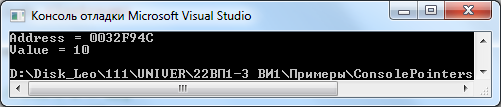
**p = &x;**

**std::cout << "Address = " << p << std::endl;**

**std::cout << "Value = " << \*p << std::endl;**

**return 0;**

**}**



Значение, которое получено в результате операции разыменования, можно присвоить другой переменной. Используя указатель, можно менять значение по адресу, который хранится в указателе.

**Пример 3**

**#include <iostream>**

**int main() {**

**int x = 10;**

**int\* p = &x;**

**int y = \*p;**

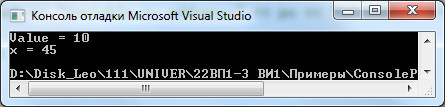
**std::cout << "Value = " << y << std::endl; // 10**

**\*p = 45;**

**std::cout << "x = " << x << std::endl; // 45**

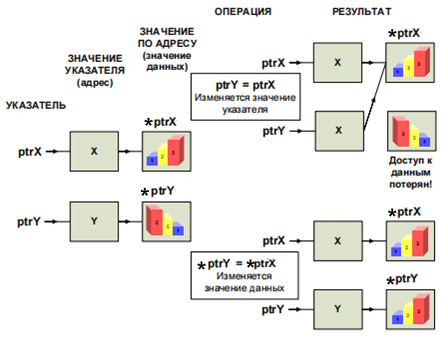
**return 0;**

**}**



Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

При работе с указателями надо отличать операции с самим указателем и операции со значением по адресу, на который указывает указатель.



**Пример 4**

**int a = 10;**

**int\* pa = &a;**

**int b = \*pa + 20; // операция со значением,**

**//на который указывает указатель**

**pa++; // операция с самим указателем**

**std::cout << "b: " << b << std::endl; // 30**

**Нулевой указатель** (null pointer) - это указатель, который не указывает ни на какой объект. Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение:

**int\* p1 = nullptr;**

**int\* p2 = NULL;**

**int\* p3 = 0;**

# Операции с указателями.

Указатели могут участвовать в арифметических операциях: сложение, вычитание, инкремент, декремент. Однако сами операции производятся немного иначе, чем с числами. И многое здесь зависит от типа указателя. В случае с указателем типа **int** увеличение/уменьшение на единицу означает изменение адреса на 4. Аналогично, для указателя типа **short** эти операции изменяли бы адрес на 2, а для указателя типа **char** на 1. Аналогично указатель будет изменяться при прибавлении/вычитании не единицы, а какого-то другого числа.

Например, добавление к указателю типа **double** числа 2: **pd = pd + 2;** означает, что мы хотим перейти на два объекта **double** вперед, что подразумевает изменение адреса на 2 \* 8 = 16 байт.

Кроме того, можно вычитать из одного указателя другой указатель.

**Пример 1**

**#include <iostream>**

**int main() {**

**int a = 10;**

**int b = 23;**

**int\* pa = &a;**

**int\* pb = &b;**

**int c = pa - pb;**

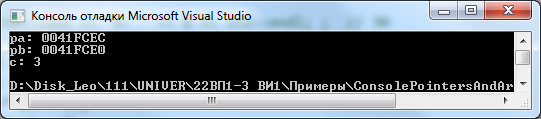
**std::cout << "pa: " << pa << std::endl;**

**std::cout << "pb: " << pb << std::endl;**

**std::cout << "c: " << c << std::endl;**

**return 0;**

**}**



**Пример 2**

**#include <iostream>**

**int main() {**

**int a = 10;**

**int\* pa = &a;**

**std::cout << "pa: address=" << pa << "\tvalue=" << \*pa << std::endl;**

**int b = \*pa++; // инкремент адреса указателя**

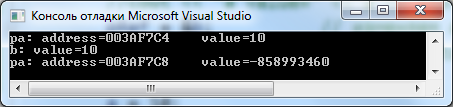
**std::cout << "b: value=" << b << std::endl;**

**std::cout << "pa: address=" << pa << "\tvalue=" << \*pa << std::endl**

**return 0;**

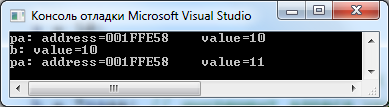
**}**

В выражении **b = \*pa++** : так как инкремент постфиксный, с помощью операции разыменования возвращается значение, которое было до инкремента – то есть число 10. И это число 10 присваивается переменной **b**. К указателю добавляется единица (то есть к адресу добавляется 4, так как указатель типа int). И в этом случае результат работы будет следующий:



Изменим выражение: **b = (\*pa)++**

Скобки изменяют порядок операций. Здесь сначала выполняется операция разыменования и получение значения, затем это значение увеличивается на 1. Теперь по адресу в указателе находится число 11. И затем так как инкремент постфиксный, переменная b получает значение, которое было до инкремента, то есть опять число 10. Таким образом, в отличие от предыдущего случая все операции производятся над значением по адресу, который хранит указатель, но не над самим указателем. И, следовательно, изменится результат работы:

****

Аналогично будет с префиксным инкрементом: **b = ++\*pa**

В данном случае сначала с помощью операции разыменования получаем значение по адресу из указателя pa, к этому значению прибавляется единица. То есть теперь значение по адресу, который хранится в указателе, равно 11. Затем результат операции присваивается переменной b:

**pa: address=0x60fe94 value=10**

**b: value=11**

**pa: address=0x60fe94 value=11**

Изменим выражение: **b = \*++pa**

Теперь сначала изменяет адрес в указателе, затем мы получаем по этому адресу значение и присваиваем его переменной b. Полученное значение в этом случае может быть неопределенным:

**pa: address=0x60fe94 value=10**

**b: value=6356864**

**pa: address=0x60fe98 value=6356864**

# Указатели и ссылки.

**Ссылка** – это синоним имени, указанного при инициализации ссылки.

Ссылка – это указатель, который всегда разыменовывается.

Объявление ссылки:

**тип &имя\_ссылки; // тип** –тип данных, на которую указывает ссылка

**// имя\_ссылки** – имя\_переменной

Например:

**int i;**

**int& rfc = i; // rfc – сылка на целую переменную i**

**Различайте указатели и ссылки.**

Указатели и ссылки позволяют неявно ссылаться на другие объекты. Как же тогда решить, когда применять указатели, а когда ссылки?

Во-первых, запомните, что не существуетнулевых ссылок. Ссылка всегдадолжнассылаться на какой-то объект**.** Если переменная обеспечивает доступ к объекту, которого может и не быть, то следует использовать указатель. При небходимости ему можно присвоить нулевое знаение. Это подразумевает, что в программе значение указателя следует проверять на равенство нулю, а значение ссылки нет.

Другое важное различие между указателем и ссылкой заключается в том, что указателю можно присваивать различные значения для доступа к разным объектам. Ссылка же всегда указывает на один и тот же объект, заданный при ее инициализации.

**Пример 1**

**#include <iostream>**

**using std::cout;**

**using std::endl;**

**int main(){**

**int i =10, j =20;**

**int& rfc = i; cout<<rfc <<endl; //результат 10**

**int\* ptr = &i; cout<<\*ptr<<endl; //10**

**ptr = &j; cout<<\*ptr<<endl;//20, изменилось**

**// значение указателя**

**cout<< i <<endl; //10, значение i не изменилось;**

**rfc = j; cout<<rfc <<endl; //20, изменилось значение данных**

**cout<< i <<endl; //20, значение i изменилось,**

**//так как rfc по-прежнему ссылается на i;**

**return 0;**

**}**

Так как ссылка не является объектом, то нельзя определить указатель на ссылку, однако **можно определить ссылку на указатель**. Через подобную ссылку можно изменять значение, на которое указывает указатель или изменять адрес самого указателя:

**Пример 2**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int a = 10;**

**int b = 6;**

**int\* p = 0; // указатель**

**int\*& pRef = p; // ссылка на указатель**

**pRef = &a; // через ссылку указателю p присваивается адрес**

**//переменной a**

**std::cout << "p value=" << \*p << std::endl; // 10**

**\*pRef = 70; // изменяем значение по адресу, на который**

**//указывает указатель**

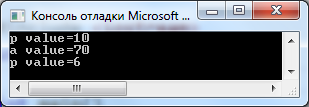
**std::cout << "a value=" << a << std::endl; // 70**

**pRef = &b; // изменяем адрес, на который указывает указатель**

**std::cout << "p value=" << \*p << std::endl; // 6**

**return 0;**

**}**

****

# Константы и указатели

* 1. Указатели на константы

Указатели могут указывать как на переменные, так и на константы. Чтобы определить **указатель на константу**, он тоже должен объявляться с ключевым словом **const**:

**#include <iostream>**

**using namespace std;**

**int main() {**

**const int A = 10;**

**const int\* pa = &A;**

**std::cout << "address=" << pa << "\tvalue=" << \*pa << std::endl;**

**return 0;**

**}**

Здесь указатель **pa** указывает на константу **a**. Поэтому даже если мы захотим изменить значение по адресу, который хранится в указателе, мы не сможем это сделать, например так: **\*pa = 34;** В этом случае мы просто получим ошибку во время компиляции.

Возможна также ситуация, когда указатель на константу на самом деле указывает на переменную:

**#include <iostream>**

**using namespace std;**

**int main(){**

**int a = 10;**

**const int\* pa = &a;**

**cout << "value=" << \*pa << endl; // 10**

**a = 22;**

**cout << "value=" << \*pa << endl; // 22**

**//\*pa = 34; // так делать нельзя**

**return 0;**

**}**

В этом случае переменную отдельно мы сможем изменять, однако по-прежнему изменить ее значение через указатель мы не сможем.

Через указатель на константу мы не можем изменять значение переменной/константы. Но мы можем присвоить указателю адрес любой другой переменной или константы:

**const int a = 10;**

**const int\* pa = &a; // указатель указывает на константу a**

**const int b = 45;**

**pa = &b; // указатель указывает на константу b**

**cout << "value=" << \*pa << endl; // 45**

**cout << "value=" << a << endl; // 10 - константа a не изменяется**

* 1. Константный указатель

**От указателей на константы надо отличать константные указатели**. Они не могут изменять адрес, который в них хранится, но могут изменять значение по этому адресу.

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int a = 10;**

**int\* const pa = &a;**

**cout << "value=" << \*pa << endl; // 10**

**\*pa = 22; // меняем значение**

**cout << "value=" << \*pa << endl; // 22**

**int b = 45;**

**// pa = &b; так нельзя сделать**

**return 0;**

**}**

* 1. Константный указатель на константу

Объединение обоих предыдущих случаев - **константный указатель на константу**, который не позволяет менять ни хранимый в нем адрес, ни значение по этому адресу:

**int a = 10;**

**const int\* const pa = &a;**

**//\*pa = 22; так сделать нельзя**

**int b = 45;**

**// pa = &b; так сделать нельзя**

# Указатели и массивы

* 1. Указатель на массив

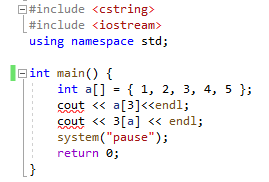
Пусть есть массив

**int A[5] = {1, 2, 3, 4, 5};**

Результат выполнения данного кода одинаков:

**void main() {**

**int A[5] = {1, 2, 3, 4, 5};**

**int \*p = A;**

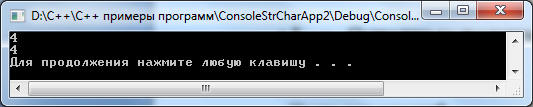
**cout << A[3] << endl; //4**

**cout << \*(A + 3) << endl; //4**

**cout << \*(p + 3) << endl; //4**

**}**

Если A[i] == \*(A + i), то от смены слагаемых местами ничего не должно поменяться, т. е. A[i] == \*(A + i) == \*(i + A) == i[A].



**Объявление указателя на массив**

**int(\*a)[2]; // Это указатель на массив из 2 целых чисел**

* 1. Отличие массива от указателя

Важно понимать, указатели – это не массивы!

Массив – непосредственно указывает на первый элемент, указатель – переменная, которая хранит адрес первого элемента.

Тогда почему возможна следующая ситуация?

**void main() {**

**int A[5] = { 1, 2, 3, 4, 5 };**

**int\* p = A;**

**cout << \*(A + 1) << endl; //2**

**cout << \*(p + 1) << endl; //2**

**}**

Это правильный код, который будет работать. Дело в том, что компилятор подменяет массив на указатель.

* 1. Многомерные массивы и указатели на многомерные массивы.

Если объявлено

**int x[5][7];**

то **x** – это не массив длины 5 неких указателей, **x** – это единый монолитный блок размером 5 x 7, размещённый на стеке:

**sizeof (x) равен 5 \* 7 \* sizeof (int).**

Элементы располагаются в памяти так:

**x[0][0], x[0][1], x[0][2], x[0][3], x[0][4], x[0][5], x[0][6],**

**x[1][0],**  **x[1][1],**  **x[1][2],**  **x[1][3]** и так далее.

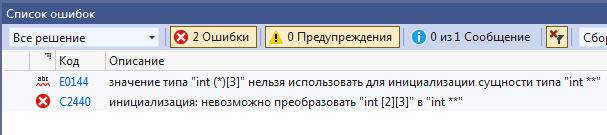
Теперь рассмотрим такой пример

**void main() {**

**int A[][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};**

**int \*\*p = A;**

**}**

****

Дело в том, что правило подмены массива на указатель не рекурсивное. Поэтому при определении многомерного массива нужно указывать размер явно, а пустыми оставлять можно только первые скобки.

Этот пример можно переписать. Мы получили указатель на первую строку. Далее вывели третий элемент.

**void main() {**

**int A[][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};**

**int \*p = A[0];//Указатель на первую строку в массиве**

**cout<< p[2]; //вывод третьего элемента**

К элементам массива можно обратиться и через массив указателей **\*p[3]:**

**cout<< \*(\*( A + 1 ) + 2);**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int x[][3] = { 1, 2, 3, 4, 5, 6 }; //массив**

**int\* p = x[0];**

**cout << "p[2] : " << p[2] << endl;**

**int(\*d)[3] = x; //указатель на массив**

**cout << "d[0][2] : " << d[0][2] << endl;**

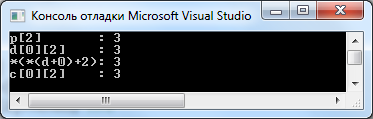
**cout << "\*(\*(d+0)+2): " << \*(\*(d+0)+2) << endl;**

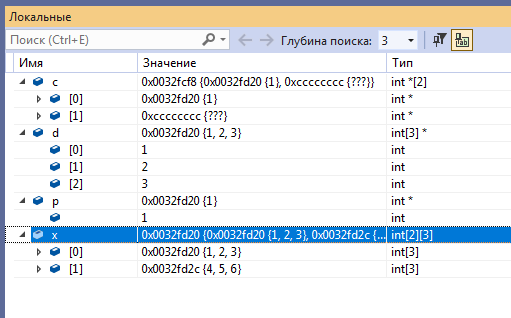
**int\* c[2]; //массив указателей**

**c[0] = x[0];**

**cout << "c[0][2] : " << c[0][2] << endl;**

**}**





# Ссылки и массивы

**Пример 1**

**int numbers[]{ 1, 2, 3, 4, 5 };**

**// меняем число на его квадрат**

**for (auto n : numbers)**

**{**

**n = n \* n;**

**}**

**// смотрим результат**

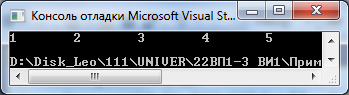
**for (auto n : numbers)**

**{**

**std::cout << n << "\t";**

**}**

**std::cout << std::endl;**



В первом цикле при переборе массива помещаем каждый элемент массива в переменную n и изменяем ее значение на квадрат числа. Однако это приведет только к изменению этой переменной n, но никак не элементов массива numbers.

**Пример 2**

**int numbers[]{ 1, 2, 3, 4, 5 };**

**// n - ссылка на элемент массива**

**for (auto& n : numbers)**

**{**

**n = n \* n;**

**}**

**// смотрим результат**

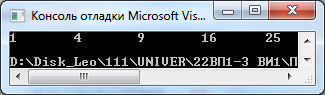
**for (auto n : numbers)**

**{**

**std::cout << n << "\t";**

**}**

**std::cout << std::endl;**



Теперь в первом цикле переменная n представляет ссылку на элемент массива. Использование ссылки позволяет оптимизировать работу с циклом, поскольку теперь значение элемента массива не копируется в переменную n, и через ссылку можно изменить значение соответствующего элемента:

**Пример3**

Если нежелательно изменять элементы массива: надо сделать ссылку константной

**int numbers[]{ 1, 2, 3, 4, 5 };**

**// n - константная ссылка на элемент массива**

**for (const auto& n : numbers)**

**{**

**std::cout << n << "\t";**

**}**

**std::cout << std::endl;**

Здесь не получиться изменять значение элементов массива, но с помощью ссылок оптимизируется перебор массива, так как элементы массива не копируются в переменную n.