**Лекция № 11**

**(2 часа)**

**Тема: Указатели и ссылки**

План лекции

1. Указатель.
2. Ссылка.

Указатель

Указатели в C/C++ сложная вещь для новичков. Без указателей программировать серьезные программы на С/С++ просто нельзя. Кроме того, указатели помогут понять как устроена память в компьютере, и как с ней работает процессор.

*Указатель – переменная, значением которой является адрес ячейки памяти.* То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. *Указатель может ссылаться на объект, переменную или функцию. Для этого нужно знать адрес переменной или функции.*

***//объявление указателя***

***/\*тип данных\*/ \* /\*имя указателя\*/;***

Принцип объявления указателей такой же, как и принцип объявления переменных. Отличие заключается только в том, что перед именем ставится символ звёздочки \*. Визуально указатели отличаются от переменных только одним символом.

*При объявлении указателей компилятор выделяет несколько байт памяти, в зависимости от типа данных отводимых для хранения некоторой информации в памяти.*

*int var1; //переменная типа int*

*int\* pointer1; // указатель на переменную типа int*

*float\* pointer2; //указатель на переменную типа float*

Указатель - это тоже переменная. И компьютер так же выделяет для нее память. Но в ней хранится адрес памяти. Что бы это проверить, сделаем следующее:

int\* pointer;

cout<<pointer;

Программа объявляет указатель, а потом выводит его содержимое. Пока что, указатель указывает на случайный адрес, так как мы только создали, но не объявили ему на что указывать.

Примечание: при создании любых переменных(и указателей в том числе) в них находится мусор, так как компьютер не занимается очисткой области памяти, которая отводиться для переменных.

При создании указателей советуется делать так:

int\* pointer = NULL;

NULL - это так называемый «нулевой» указатель. Это поможет избежать ошибок. Так как в случайном адресе может храниться все что угодно.

*Чтобы узнать адрес конкретной переменной в С++ существует унарная операция взятия адреса &.* Такая операция извлекает адрес объявленных переменных, для того, чтобы его присвоить указателю.

***int \*ptrvar; // объявление указателя***

***ptrvar = &var; // инициализация указателя***

*Чтобы получить значение, записанное в некоторой области, на которое ссылается указатель нужно воспользоваться операцией разыменования указателя \*.* Необходимо поставить звёздочку перед именем и получим доступ к значению указателя. Разработаем программу, которая будет использовать указатели.

***int var = 123; // инициализация переменной var числом 123***

***int \*ptrvar = &var; // указатель на переменную var (присвоили адрес переменной указателю)***

***cout << "&var = " << &var << endl;// адрес переменной var содержащийся в памяти, извлечённый операцией взятия адреса***

***cout << "ptrvar = " << ptrvar << endl;// адрес переменной var, является значением указателя ptrvar***

***cout << "var = " << var << endl; // значение в переменной var***

***cout << "\*ptrvar = " << \*ptrvar << endl; // вывод значения содержащегося в переменной var через указатель, операцией разименования указателя***

Указатели на указатели

*Указатели могут ссылаться на другие указатели. При этом в ячейках памяти, на которые будут ссылаться первые указатели, будут содержаться не значения, а адреса вторых указателей. Число символов \* при объявлении указателя показывает порядок указателя. Чтобы получить доступ к значению, на которое ссылается указатель его необходимо разыменовывать соответствующее количество раз.*

***int var = 123; // инициализация переменной var числом 123***

***int \*ptrvar = &var; // указатель на переменную var***

***int \*\*ptr\_ptrvar = &ptrvar; // указатель на указатель на переменную var***

***int \*\*\*ptr\_ptr\_ptrvar = &ptr\_ptrvar;***

***cout << " var\t\t= " << var << endl;***

***cout << " \*ptrvar\t= " << \*ptrvar << endl;***

***cout << " \*\*ptr\_ptrvar = " << \*\*ptr\_ptrvar << endl; // два раза разименовываем указатель, так как он второго порядка***

***cout << " \*\*\*ptr\_ptrvar = " << \*\*\*ptr\_ptr\_ptrvar << endl; // указатель третьего порядка***

Указатель типа void

*Указатель типа void (или еще «общий указатель») — это специальный тип указателя, который может указывать на объекты любого типа данных! Объявляется он как обычный указатель, только вместо типа данных используется ключевое слово void:*

*void \*ptr; // ptr - это указатель типа void*

Указатель типа void может указывать на объекты любого типа данных:

*int nResult;*

*float fResult;*

*void \*ptr;*

*ptr = &nResult; // допустимо*

*ptr = &fResult; // допустимо*

Однако, поскольку указатель void сам не знает, на объект какого типа он будет указывать, разыменовать его напрямую не получится! Вам сначала нужно будет явно преобразовать указатель типа void с помощью оператора cast в другой тип данных, затем уже разыменовывать.

***int value = 7;***

***void \*voidPtr = &value;***

***//cout << \*voidPtr << endl; // запрещено: нельзя разыменовывать указатель типа void***

***int \*intPtr = static\_cast<int\*>(voidPtr); // однако, если мы конвертируем наш указатель void в указатель int***

***cout << \*intPtr << endl; // то мы сможем его разыменовать, подобно обычному указателю***

Ссылка

*Ссылка представляет собой синоним имени, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который всегда разыменовывается.* Формат объявления ссылки:

***тип & имя;***

где тип — это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним имя является именем переменной ссылочного типа, например:

int коl;

int& pal = kol; // ссылка pal - альтернативное имя для коl

Правило использования ссылок

* *Переменная-ссылка должна явно инициализироваться при ее описании, кроме случаев, когда она является параметром функции.*
* *После инициализации ссылке не может быть присвоена другая переменная.*
* *Тип ссылки должен совпадать с типом величины, на которую она ссылается.*
* *Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки.*

Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений.

*Ссылки позволяют использовать переменные, передаваемые по адресу, без операции разадресации, что улучшает читаемость программы.*

*Ссылка, в отличие от указателя, не занимает дополнительного пространства в памяти и является просто другим именем величины. Операция над ссылкой приводит к изменению величины, на которую она ссылается.*