**Массивы и функции**

**Типичные ошибки оператора if:**

1. Отсутствие фигурных скобок
2. Незнание операторов (использование присваивания вместо сравнения)
3. Проверка диапазона записана неправильно

**Массивы**

Тип\_элемента имя [размер];

Размер – константное выражение, т.е. изменить нельзя

Индексы элементов массива – целые числа, начиная с 0

Int a[100]

a[0] a[10] a[99]

Инициализация массива

Int p[ ] = {0, 1, 2, 3, 4, 5}

К элементам массива можно обратиться, указав имя [индекс]

Контроль выхода за пределы массива не производится

Как работать с массивами:

For (int i = 0; i < 100; i++)

**Многомерные массивы**

Const int m = 4, n = 5;

Float math [ m ] [ n ];

Int mas [ 3 ] [ 2 ] = { {1,1}, {0,2}, {1,0} };

Правый индекс меняется первым

**Указатели**

При определении переменной компилятор выделяет ей адрес памяти (имеется в виду сегмент данных) все обращения к этой переменной заменяются на адрес этой памяти.

Программист может определить собственные переменные для хранения адресов памяти.

В С++ есть 3 вида указателей:

1. На объект (на переменную)
2. На функцию
3. На void

Указатель – это не самостоятельный тип, он всегда связан с каким-то другим конкретным типом.

**Указатель на объект**

С ним всегда связана информация о том, на какой тип данных он указывает, причём тип данных может быть и простым и составным.

Тип может быть только объявлен, но ещё не определён. Следовательно, например, в структуре может присутствовать указатель на структуру того же типа.

Можно определить указатель на указатель.

**Указатель на void**

Применяется в случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определён (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов)

Указателю на void можно присвоить значение указателя любого типа.

Так же можно сравнивать указатель на void с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.

**Указатель на функцию**

Содержит адрес в сегменте кода, по которым располагается исполняемый код функции, т.е. адрес, по которому передаётся управление при вызове функции.

Используется для:

1. Косвенного вызова функции (не через её имя, а через обращение к переменной, хранящей её адрес)
2. Передачи имени функции в другую функцию в качестве параметра

Тип указателя на функцию звучит так:

«Указатель функции, возвращающий значение заданного типа и имеющий аргументы заданного типа»

Тип (+имя) (список\_аргум),

Int (+fun) (double, double)

Указатель может быть константой или переменной, а так же указывать на константу или переменную.

**Инициализация указателя**

Указатели чаще всего используются при работе с кучей.

Доступ к участкам памяти из кучи возможен только через указатели.

Время их жизни от распределения памяти до её освобождения.

В С++ два способа работы с динамической памятью:

1. Malloc   
   free
2. New  
   delete

При определении указателя необходимо помнить о его первоначальной инициализации.

Если использовать инициализатор, то синтаксис такой же, как и для любых других переменных (либо «=» либо скобки)

Если инициализатора нет, то надо помнить, что первоначальное присваивание надо обязательно где-то выполнить.

Непреднамеренное использование неинициализированных указателей – распространённый источник ошибок в программе.

Способы инициализации указателей:

1. Присваивание указателю адреса существующего объекта  
   а) с помощью операции получения адреса  
   int a = 5;  
   int \* p = &a;  
   int \* p (&a);  
   б) int \* r = p;  
   в) int b [10];  
   int \* t = b;  
   в переменную t записывается адрес начала массива  
   void f (int a) { /\*……\*/}  
   void (\*pf) (int);  
   pf = f;
2. Присваивание указателю адреса области памяти в явном виде.  
   char \* vp = (char\*)0xB8000000
3. Присваивание пустого значения.  
   int \* x = null  
   int \* y = 0
4. Выделение участка динамической памяти и присваивание её адреса указателю.  
   а) с помощью функции new  
   int \* n = new int;  
   Функция обращается в кучи и берёт столько памяти, сколько занимает указанный тип; адрес этого участка записывается в переменную n.  
   Адрес под саму переменную n (для 32-разрядной архитектуры это 4 байта) выделяется на этапе компиляции, указатель может быть локальным или глобальным (расположен может быть либо в стеке, либо в сегменте данных)  
   int \* n = new int;  
   int \* m = new int (10);  
   Выполняется то же самое, что и в первой строке, Но выделенная область памяти инициализируется значением 10.  
   int \*q = new int [10]  
   Выделяется область под 10 величин типа int (для массива из 10 элементов)  
   Адрес начала этого участка записывается в переменную q и эта переменная может трактоваться как имя массива.  
   Исключительная ситуация – это ошибка в программе (нет памяти, деление на 0 и т.д.)

Освобождение памяти происходит следующим образом:

Delete n;  
delete m;  
delete [ ] q;

Если выделяли память функцией new, то освобождаем функцией delete.

Освобождение памяти из под переменной массива (если использовались квадратные скобки) – используется операция delete и квадратные скобки. Размерность массива не указывается. Если при освобождении памяти из под массива забыли квадратные скобки, то никакого сообщения выдаваться не будет; помеченным как свободный будет только 1 элемент массива; остальные будут не доступны для дальнейших операций.

Такие ячейки памяти называются мусором; может происходить утечка памяти.

Другие причины возникновения мусора:

1. Если переменная указатель выходит из области своего действия, а память и под самой динамической переменной не освобождена
2. Когда инициализированному указателю присваивается значение другого указателя

После освобождения памяти переменная указатель сохраняется (с учётом времени её жизни) и может инициализироваться повторно.