Указатели и массивы

Указатели

Указатели

Инициализация указателей

При объявлении указателя надо стремиться выполнить его инициализацию, то есть присвоить ему некоторое начальное значение. Использование неинициализированного указателя – распространенный источник ошибок в программе. Инициализатор записывается после имени указателя после знака равенства или в круглых скобках.

Указатели

Существуют следующие способы инициализации указателей:

1. Присваивание указателю адреса существующего объекта:

   - *с помощью операции получения адреса &*

```
int var_int = 6338;
int *ptr_int = &var_int;    // или
int *ptr_int(&var_int);
```

Указатели

Еще один пример:

```
struct Test
{
    int test_int;
    char test_char;
    double tetst_double;
} test={10, 'A', 4.78};
```

Указатели

```
int main()
{
    Test *ptr_Test = &test;
    cout << ptr_Test -> test_int << ' '
    << ptr_Test -> test_char << ' '
    << ptr_Test -> tetst_double << endl;
    system("pause");
    return 0;
}
```

Указатели

- *с помощью значения другого инициализированного указателя*

```
float var_float = 3.55f;
float *ptr_float_1 = &var_float;
float *ptr_float_2 = ptr_float_1;
```

Указатели

- с помощью имени массива

```
unsigned array_int[] = {33, 51, 78, 4, 15};

int main()
{
    unsigned *ptr_array = &array_int;     // & - не обязательно
    cout << *(ptr_array + 3) << endl;
    system("pause");
    return 0;
}
```

Указатели

Известно, что имя массива является указателем на его первый элемент, поэтому следующие выражения эквивалентны:

```
cout << *ptr_array << endl;
cout << *array_int << endl;
```

В обоих случаях получим значение первого элемента массива.

Указатели

- с помощью имени функции

```
void func(int,char){ // ……};
void (*ptr_fun)(int,char) = &func;
```

Знак операции взятия адреса & здесь также не обязателен, так как имя функции трактуется как адрес ячейки памяти, начиная с которого находится объектный код данной функции.

Указатели

2. Присваивание указателю адреса области памяти в явном виде:

```
char *vp = (char *)0xB8000000;

int main()
{
    cout << *vp << endl;        // ???????
    system("pause");
    return 0;
}
```

Указатели

3. Присваивание пустого указателя:

```
short *ptr_short = NULL;
long *ptr_long = 0;
```

4. Выделение участка динамической памяти:

   - *с помощью операции new:*

```
int *p_i = new int;
```

Указатели

Здесь в динамической памяти выделяется место для хранения величины типа int.

```
int *p_j = new int(100);
```

Здесь помимо выделение памяти, заносится значение 100.

```
int *p_array new int[10];
```

Здесь выделяется место в динамической области памяти для хранения массива целых чисел.

Указатели

- с помощью функции malloc:

```
int *p_i = (int *)malloc(sizeof(int));
```

Функция malloc заимствована из языка C, тем не менее, она работает.

При работе с динамической областью памяти необходимо следить за тем, чтобы указатель не вышел за пределы области видимости. В этом случае память отведенная под указатель освобождается, указатель обнуляется, а переменная становится недоступной.

Указатели

На программистском сленге это означает появление «мусора» в памяти.

Полезный совет при попытке выделения памяти в динамической области. Проверяйте значение указателя после выполнения операции new или функции malloc на равенство нулю. Если указатель нулевой, то операционной системе не удалось найти достаточного свободного объема. Это предотвратит последующие ошибки.

Указатели

```
int main()
{
    float *ptr_float = new float(67.44f);
    if(!ptr_float)
    {
      cout << " Недостаточно памяти " << endl;
      exit(1);
    }
    else cout << " Ok " << endl;

    system("pause");
    return 0;
}
```

Указатели

Операции над указателями

Как уже было сказано, указатели – переменные, хранящие адреса ячеек памяти, то есть, величины, относящиеся к без знаковому целому типу. Не сложно догадаться, что к величинам данного типа применимы арифметические операции, но не все. Кроме того над ними допускаются и другие операции.

Указатели

Определим основные операции допустимые над указателями:

- операция разадресации (разыменования), косвенное обращение к объекту (*);
- присваивание;
- сложение с константой;
- вычитание константы;
- инкремент (--);
- декремент (++);

Указатели

- сравнение;
- приведение типов.

При работе с указателями очень часто используется операция получения (взятия) адреса (&).

Основной операцией над указателями является операция разадресации или разыменования, которая предназначена для доступа к величине, на которую указывает указатель.

Указатели

Эта операция симметрична, то есть с ее помощью можно получить значение объекта или изменить его.

Рассмотрим пример:

```
int main()
{
    typedef unsigned long int UINT;
//
    UINT *ptr_UINT = new UINT(12L);
//  объявление и инициализация указателя на объект UINT
    cout << " Величина UINT " << *ptr_UINT << endl;
//  получение значения по указателю
    return 0;
}
```

Указатели

Эта операция применима только к указателям на объект какого-либо типа и на тип void. К указателям на функцию она не имеет смысла.

Арифметические операции над указателями, в частности, сложение с константой, вычитание константы, инкремент, декремент допустимы, но не над всеми видами указателей.

Указатели

Например, при работе с массивами они допустимы, а с указателями на обычные переменные или с указателями на функции, лишены смысла. Кроме перечисленных арифметических операций допускается использование операции вычитания указателей. Все остальные, то есть сложение, умножение или деление указателей не допускаются.

Указатели

Рассмотрим пример использования арифметических операций:

```
float array_float = {3.2, -44.6, -0.073, 12, 5.0};
// ….
float summa_array(float arr[])
{
    float rez = 0;
    for(int i =0; i<5; i++)
    rez += *(arr+i);    //сложение указателя с константой
    return rez;
}
```

Указатели

Операции инкременты, декремента, вычитание константы рассмотреть самостоятельно.

Важную операцию представляет операция преобразования указателей. Преобразование указателей в C++ допускается двумя способами:

- унаследованным от языка C. Общий формат оператора преобразования следующий:

  (тип *) имя_указателя;, или

  тип * (имя_указателя);.

Указатели

- в стиле языка C++, используя операции static_cast, dynamic_cast, reinterpret_cast, например,

static_cast<float *> (ptr_int);

Операции преобразования широко используются при преобразовании в иерархии родственных классов в условиях наследования.

Указатели

Обычно операция преобразования используется при выполнении операции присваивания. Присваивание без явного преобразования допускается если в левой части выражения используется указатель на тип void или типы указателей совпадают. Значение 0 неявно приводится к указателю на любой тип.

Присваивание указателей на объекты указателям на функции и наоборот не допускается.

Ссылки

Ссылки

Ссылка (ссылочный тип данных) – синоним имени, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который не надо разыменовывать. Формат объявления ссылки следующий:

```
тип_ссылки &имя_ссылки = инициализация;
```

Ссылки

Пример объявления ссылки:

```
int var_int = 57;
int  &ref_int = var_int;
const char &ref_char = '\n';
```

Следует запомнить следующие правила определения и использования ссылок:

- переменная-ссылка должна явно инициализироваться при ее описании, кроме случаев, когда она является параметром функции, описана как extern или ссылается на поле данных класса;

Ссылки

- 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 результатов

Ссылки

Ссылки позволяют использовать в функциях переменные, передаваемые по адресу, без операции разыменования, что упрощает процесс программирования.

Ссылки, в отличие от указателей не занимают место в памяти и являются по сути другим именем объекта. Операции над ссылкой приводит к изменению величины, на которую она ссылается.

Массивы

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин (объектов) необходимо выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Конечная именованная область памяти, содержащая однотипные элементы, называется *массивом.*

Массивы

Общий формат описания массива:

тип_массива имя_массива[размерность] = {инициализация};

Массивы в C++ имеют ряд особенностей:

- нумерация (индексация) элементов начинается с нуля;
- компилятор не отслеживает границ массива.

Массивы

Примеры объявлений массивов:

```
double array_double[20];
// объявление массива из 20 чисел типа
   double
int array_int[10] = {34, 86, -53, 1024, 0, -778};
// объявление массива из 10 целых чисел с
   инициализацией
int array_int[] = {553, 749, -503, 46, 120, 59};
// тоже, но без указания размерности
```

Массивы

Рассмотрим пример:

```
int array_int[10] = {32, -453, 6, 562, -322, 78};
int main()
{
    for(int i=0; i<=5; i++)
     cout << " Array: " << array_int[i] << ' ';
    cout << endl;
    return 0;
}
```

Массивы

Здесь объявлен массив целых чисел array_int[10] и инициализирован значениями. Инициализация массива осуществлена не полностью, только первые 6 элементов. Все остальные заполняются нулями целого типа. Если при объявлении не указана размерность, инициализация массива обязательна.

Массивы

Обращение к элементам массива можно осуществлять двумя способами:

- с помощью операции индексирования – [n], как в приведенном выше примере;
- через указатель.

Как уже было сказано, имя массива компилятором понимается как указатель на его первый элемент.

Массивы

Поэтому выражение

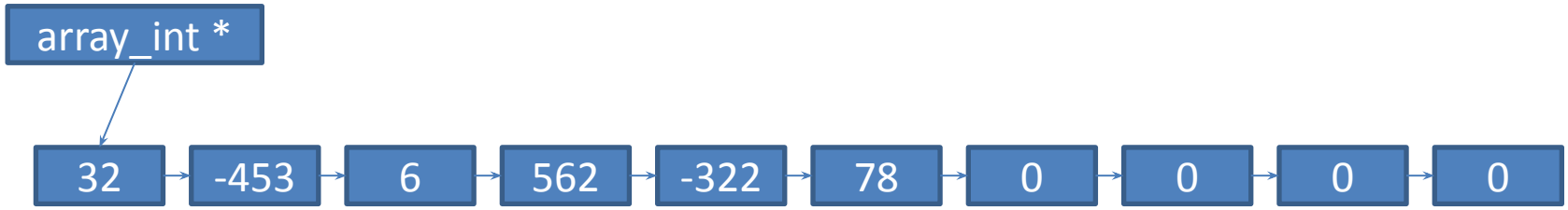
```
cout << " Array: " << array_int[i] << ' ';
```

можно записать в следующем виде:

```
cout << " Array: " << *(array_int+i) << ' ';
```

В памяти машины все элементы массива будут расположены в последовательных ячейках ОЗУ.

Массивы



Размерность массива принято задавать с помощью именованных констант, например:

const int n_str=10, n_stb=15;,

поскольку при таком подходе значение константы достаточно скорректировать только в одном месте.

Массивы

Многомерные массивы

Многомерные массивы задаются указанием каждого измерения в отдельных квадратных скобках, например,

```
int matr[6][8];.
```

Здесь задается двумерный массив целых чисел, состоящий из 6 строк и 8 столбцов.

Массивы

Инициализация многомерного массива также допускается, например,

int arr_int[3][3] = {{1,2,3}, {2,3,4}, {3,4,5}}; или же:

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

Для доступа к многомерному массиву можно использовать операцию индексирования -

arr_int[2,1] или посредством указателя - *(*(arr_int+2)+1).

Массивы

Многомерные массивы размещаются в памяти так, что при переходе к следующему элементу, быстрее всех изменяется последний индекс.

Массивы можно объявлять в динамической области памяти, например,

```
const int nstr =5, nstb =6;
```

Массивы

```
int **array_int = new int *[nstr];
for(int i=0; i<nstr; i++)
    array_int[i] = new int[nstb];
```

Массивы

Строки

Строка – это массив символов, заканчивающийся нуль-символом (‘\0’).

По положению нуль-символа компилятор определяет конец строки. В отличие от обычного массива строка занимает на один элемент больше (под нуль-символ).

Массивы

Рассмотрим простой пример:

```
char str[10] = "Hello!";
int main()
{
    int i=0;
    while(str[i] != '\0')
    {
      cout << str[i];
      i++;
    }
    cout << endl;
    return 0;
}
```

Массивы

Строку можно задать как указатель на константную величину:

```
char *str = "Hello!";
```

Изменение элементов этой строки не допускается.

Операции над строками можно осуществлять через операторы цикла, кроме того, много операций определено в стандартной библиотеке.

Стандартная библиотека размещена в файле <string>.

Массивы

До сих пор мы говорили о массивах, содержащих объекты стандартные типы. А можно ли создавать массивы объектов пользовательского типа?

Рассмотрим пример простой структуры:

```
struct Student
{
    string Name;
    int Age;
};
```

Массивы

Объявим массив типа Student:

```
Student arr_Student[10];
```

Воспользуемся этим массивом

```
arr_Student[0].Name = "Иван";
arr_Student[0].Age = 20;
arr_Student[1].Name = "Маша";
arr_Student[1].Age = 19;
```

Массивы

Заметим, что отличия в обращении к элементам такого массива есть, в частности, используется операция доступа к полям структуры (‘ . ’).

Еще один вариант обращения – через указатель (имя массива – указатель на его первый элемент):

```
(arr_Student+2)->Name = "Вася";
(arr_Student+2)->Age = 19;
```

Массивы

И здесь есть отличие – использование операции доступа ‘ -> ’ производит автоматическое разыменование указателя и поэтому символ звездочки перед указателем не ставится.

Массивы

Следующий пример связан с объявлением массива указателей на функции.

Предположим, что есть ряд одинаковых функций, выполняющих разные действия:

```
int add(int a, int b)
{
   return a+b;
}
```

Массивы

```
int sub(int a, int b)
{
    return a-b;
}
int mul(int a, int b)
{
    return a*b;
}
```

Массивы

Объявим массив указателей на функции:

```
typedef int (*PF)(int,int);
PF ptr_fun[5] = {&add, &sub, &mul,0,0};
```

Теперь можно вызывать функции, обращаясь к элементам массива:

```
int v_int_1 = 10, v_int_2 = 5;
cout << (ptr_fun)[0](v_int_1, v_int_2) << endl;
cout << (ptr_fun)[1](v_int_1, v_int_2) << endl;
cout << (ptr_fun)[2](v_int_1, v_int_2) << endl;
```

Массивы

Результат посмотреть обязательно.

Следующий вариант вызова функции в работу – через указатель:

```
cout << (*(ptr_fun+1))(v_int_1, v_int_2) << endl;
```

Массивы

Массивы

Массивы

Массивы

Массивы

Массивы