Массивы, модульное программирование

Массивы, примеры массивов

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

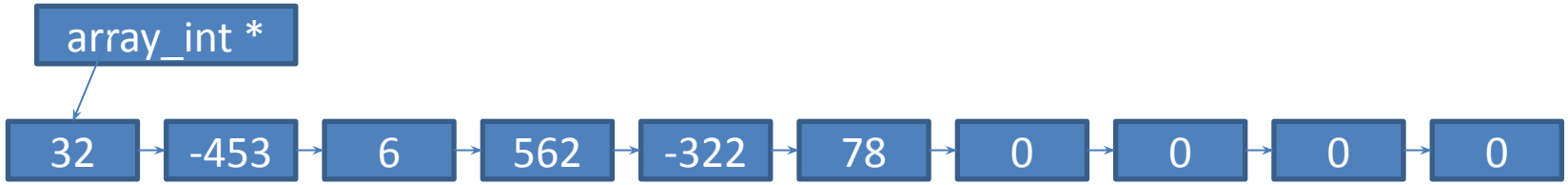
```
cout << " Array: " << array_int[i] << ' ';
```

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

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, размещена в файле <string>.

Массивы

Предварительно рассмотрим еще раз следующие объявления:

- int i; // целочисленная переменная
- int *ps; // указатель на переменную
- const int *pci; // указатель на целую переменную константу
- int *const cp =&i; // константный указатель на целую переменную

Массивы

```
#include<iostream>
using namespace std;
const char *str  = "String";
//char *str  = "String";

int main()
{
int i=0;
      while( *(str+i) != '\0')
      {
        cout << *(str+i);
        ++i;
      }
      cout<< endl;
      return 0;
}
```

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 mult(int a, int b)
{
    return a*b;
}
```

Массивы

Объявим массив указателей на функции:

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

Модульное программирование

Прежде чем начинать рассмотрение модульного программирования, следует напомнить фразу Б. Страуструпа – автора языка C++ : «Модульность – фундаментальный аспект всех успешных работающих крупных систем».

Модульное программирование

*С увеличением объема программы становится невозможным удерживать в памяти все детали. Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В C++ задача может быть разбита на простые и обозримые с помощью функций, после чего программу можно рассматривать как взаимодействие функций.

Модульное программирование

Использование функций является первым шагом к повышению абстракции программы.

Разделение программы на функции позволяет избежать избыточности кода, процесс отладки упрощается. Часто используемые функции помещаются в библиотеки.

Модульное программирование

Следующим шагом в повышении уровня абстракции программы является группировка функций и связанных с ними данными в отдельные файлы (модули), компилируемые отдельно. Получившиеся в результате компиляции объектные модули собираются в исполняемую программу с помощью компоновщика.

Модульное программирование

Модуль содержит в себе данные и функции их обработки. Для того чтобы использовать модуль, нужно знать его *интерфейс,* а не детали его реализации.

Интерфейсом модуля являются заголовки всех функций и описание доступных извне типов, переменных, констант.

Интерфейс содержится в файлах с расширением h.

Модульное программирование

Модульность в языке достигается с помощью *директив препроцессора, пространства имен, классов памяти, исключений (обработкой исключительных ситуаций) и раздельной компиляцией*. Раздельная компиляция не является свойством языка, она относится к его реализации.

Функции

Объявление и определение функций

Функция – это именованная последовательность описаний и операторов, выполняющая некое законченное действие. Функция может принимать параметры (аргументы) и возвращать значение.

Любая программа на языке C++ состоит из функций, одна из которых должна иметь имя main. С нее начинается выполнение программы.

Функции

Функция начинает выполняться в момент ее вызова.

Любая функция должна быть объявлена и определена. Объявление функции текстуально должно быть раньше ее вызова, определение может располагаться в текущем модуле или в другом.

Функции

Объявление функции (прототип, заголовок, сигнатура) задает ее имя, тип возвращаемого результата и список передаваемых параметров. Определение функции содержит, кроме объявления, тело функции, представляющее собой последовательность описаний и операторов в фигурных скобках (тело функции – это блок).

Функции

Общий формат определения функции:

[класс] тип_результата имя_функции
([список_параметров])[throw(исключения)]
{
  // тело_функции
}

Функции

Рассмотрим составные части функций:

- С помощью не обязательного модификатора «класс» можно задать область видимости функции, используя ключевые слова extern и static:

- extern - глобальная область видимости во всех модулях программы (по умолчанию);
- static – видимость в пределах текущего модуля (в том, в котором определена).

Функции

- Тип возвращаемого функцией результата может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция не возвращает результата, указывается тип void.
- Список параметров определяет величины, передаваемые функции при ее вызове. Элементы списка разделяются запятыми. Для каждого параметра указывается его тип и имя. В объявлении имена параметров можно не указывать.

Функции

В объявлении, определении и вызове одной и той же функции типы и порядок следования параметров должны совпадать. Имен параметров – произвольные идентификаторы.

Функцию можно определить как встроенную (подставляемую) с помощью модификатора inline. Этот модификатор рекомендует компилятору вместо обращения к функции помещать ее код непосредственно в точку ее вызова.

Функции

Модификатор inline обычно используется для коротких функций и носит рекомендательный характер.

Все составные функции класса (структуры) по умолчанию являются подставляемыми.

Тип возвращаемого результата и типы параметров совместно определяют тип функции.

Функции

Для вызова функции необходимо указать ее имя и передать ей необходимое количество фактических параметров.

Рассмотрим пример простой функции для вычисления факториала числа:

```
long factorial(long); // объявление, прототип
    //
long factorial(long n) // определение функции
{
    if(n==0||n==1) return 1;
    return (n *factorial (n-1));
}
```

Функции

Как уже говорилось, имя функции является указателем на ячейку памяти, начиная с которой расположен исполняемый код функции. Попытка разыменовать данный указатель приведет к получению объектного кода первой команды функции.

Функции

Все величины (переменные, объекты), объявленные внутри функции, а также ее параметры считаются локальными. При вызове функции компилятор организует стек вызова, в который заносятся эти параметры. При выходе из функции стек освобождается и связи переменных между вызовами теряются.

Если необходимо запомнит значения локальных переменных, их можно объявить с модификатором static.

Функции

Рассмотрим пример

```
void static_var(int a)
{
    int m = 0;            // в стеке периода исполнения
    cout << "n m p" << endl;
    while(a--)
    {
     static int n = 0;  // в программном сегменте
     int p = 0;
     cout << n++ << ' ' << m++ << ' ' << p++ << endl;
    }
}
```

Функции

Статическая переменная n размещается в сегменте данных программы, инициализируется один раз при первом вызове оператора. Ее значение будет сохраняться от одного вызова функции к другому.

Функции

Глобальные переменные

Переменные, объявленные вне всякого блока, называются глобальными. Они доступны всем программным объектам, в том числе видны во всех функциях. Чаще всего их используют для передачи информации между отдельными функциями. Их изменение допускается, но не рекомендуется. Подобный прием считается «дурным тоном» в программировании.

Функции

Возвращаемое значение

Механизм возврата из функции в вызывающую ее функцию реализуется оператором return:

return[выражение];

Функция может содержать несколько операторов return, что определяется потребностями алгоритма.

Функции

Этот оператор может опускаться для функций, имеющих тип возвращаемого результата void. Перед возвратом результата оно преобразуется к типу (если это возможно), указанному в прототипе функции.

Функция может вернуть в качестве результата только скалярное значение. Она не может вернуть массив или другую функцию, но указатели на них – может.

Функции

Важное замечание: нельзя возвратить из функции указатель на локальную переменную, поскольку память, выделенная на момент выполнения функции перед выходом из функции, освобождается.

Например,

```
int *f()
{
    int a =10;
    // ……
    return &a;
}
```

Функции

Параметры функции

Механизм параметров является основным способом обмена информацией меду вызываемой и вызывающей функциями. Параметры перечисленные в заголовке функции называются формальными, а записанные в операторе вызова – фактическими.

Функции

При вызове функции в первую очередь вычисляются выражения, стоящие на месте фактических параметров, затем в стеке периода исполнения им выделяется память в соответствии с типом их результатов. При передаче проверяется соответствие типов и при невозможности преобразования выдается диагностическое сообщение.

Различают два основные способы передачи параметров:

- по адресу;
- по значению.

При передаче по значению в стек исполнения заносятся копии фактических параметров, и операторы функции работают именно с копиями параметров, а не с самими параметрами. Доступа кк исходным данным у функции нет, а следовательно, нет возможности их изменить.

Функции

При передаче по адресу в стек исполнения заносится адреса фактических параметров, а функция, работая с копиями адресов, может изменить исходные значения параметров.

Необходимо отметить, что изменение фактических параметров не приветствуется.

Передача параметров по адресу реализуется через указатели или ссылки.

Функции