# лекция 3

**Тема лекции**: Использование динамических строк и массивов. Перегрузка функций. Аргументы функций по умолчанию

# Многомерные динамические массивы

Для объявления указателя на двух или трехмерный массив (или большей размерности) нужно указать количество элементов во второй и последующих позициях. Например, для выделения пространства для массива 10х20 целых чисел, нужно начать с объявления:

int (\* matrix) [20];

Оно определяет matrix как указатель на массив из 20 целых. Далее, выделяем память для матрицы размером 10х20 и присваиваем ей адрес переменной matrix:

matrix = new int [10][20];

Для создания куба 8 \* 8 \* 8 надо написать:

int (\* matrix) [8] [8];

matrix = new int [8] [8] [8];

При выделении пространства для многомерного массива первый индекс может быть переменной, а второй и последующие должны быть константами.

Удаление многомерных массивов любой размерности происходит так, как будто они только одно измерение:

delete [] matrix;

**перегрузка функций**

 В С ++ функции могут иметь одинаковое имя до тех пор, пока они отличаются хотя бы одним параметром. Функции "перегружены", поскольку их имена одинаковые, а выполняемая ими работа разная.

#include <iostream. h>

int square (int a)

double square (double a)

long double square (long double a)

main ()

{

int x = 10;

double y = 15.7;

long double z = 30.75;

cout << square (x) << '\ n';

cout << square (y) << '\ n';

cout << square (z) << '\ n';

return 0;

}

int square (int a)

{

return a \* a;

}

double square (double a)

{

return a \* a;}

long double square (long double a)

{

return a \* a;}

**Аргументы функций по умолчанию**

 Важное свойство С ++ - возможность передачи аргументов в функцию по умолчанию. Она используется, если при вызове функции необходима только часть параметров, а не все.

Допустим, нам требуется, что возвращает сумму четырех целых значений, подобная такой:

int sum (int a, int b, int c, int d)

{

return a + b + c + d;

}

В данном случае для вызова функции только с двумя аргументами необходимо дополнить нулями два отсутствуют параметры:

cout << sum (v1, v2,0,0)

Но чтобы сделать такую ​​запись необходимо разобраться в документации по функции sum () для того, чтобы определить, какими значениями дополнить неиспользованные параметры. Аргументы функций по умолчанию созданы для защиты от использования ложных значений параметров при вызове функций. Они дополняют необходимые значения для незаданных аргументов.

Для объявления значение аргумента по умолчанию в прототипе функции нужно завершить параметр функции знаком равенства и требуемым значением.

int sum (int a, int b, int c = 0, int d = 0);

Значение по умолчанию должны идти последними в списке параметров функции и могут появляться только в прототипе функции. Теперь доступны такие операторы:

cout << sum (1,2) // A = 1; b = 2; c = 0 d = 0

cout << sum (1,2,3);

cout << sum (1,2,3,4)

посылки

В следующих строках:

int i = 12;

int \* p = & i;

объявляется целая переменная и, инициализирована значением 12 и указатель на целое p, что адресует i. Согласно этим объявлений такие два оператора выводят значение i:

cout << i;

cout << \* p;

Выражение \* p указывает компилятору, что нужно использовать значения, адресуется указателем г.. Использование указателей в качестве доступа к переменным содержит в себе опасность - если оператор изменил адрес, хранящийся в указателе, то этот указатель больше не ссылается на то же значение. Если такое изменение было внесено нечаянно, то результатом этого может быть серьезная ошибка. С ++ предлагает альтернативу для более безопасного доступа к переменным через ссылки. Операция & означает ссылку. Посылка на значение всегда ссылается на это значение.

Посылки в качестве переменных. Как переменная посылка ссылается на другое значение.

Пример: четыре средства адресации и вывода целого значения.

#include <iostream. h>

main (){

int ivar = 1234; // Переменной присваивается значение

int \* iрtr = & ivar; // Указателе присваивается адрес ivar

int & iref = ivar; // Посылка ассоциируется с ivar

int \* p = & iref; // Указателе присваивается адрес iref

cout << ivar;

cout << \* iptr;

cout << iref;

сout << \* p;

return 0;

}

Здесь объявляются четыре переменные. Первая - простое число ivar, что инициализирует значение 1234. Следующая - указатель на целое с именем iptr, которому присвоен адрес ivar. Третья объявлена ​​как переменная-посылка. строка

int & iref = ivar;

объявляет iref как посылку на целое значение. Переменной iref присваивается ivar не как значение переменной, а как адрес в памяти. iref указывает на ivar подобно указателе. оператор

cout << iref;

выводит значение ivar. Это происходит благодаря использованию iref в качестве посылки на место расположения ivar в памяти. Четвертое объявления создает еще один указатель - p, которому присваивается адрес, хранящийся в iref. Применение указателя в выражении \* p дает доступ к значению ivar.

При использовании посылок надо помнить правило: однажды инициализируя посылку, ей нельзя присвоить другое значение.

Пример: добавим в предварительную программу объявления:

int anotherInt;

iref = anotherInt;

Это присвоение не заставит iref ссылаться на anotherInt. Это выражение присвоит значение anotherInt объекта на который ссылается iref, то есть ivar. Посылки должны быть инициализированы при их объявлении. Например, можно написать так:

int & iref;

В отличие от указателей, которые могут быть объявлены в неинициализированных состоянии или установленные в ноль, посылки всегда ссылаются на объект. Для посылок не существует аналога нулевого указателя.

Посылки нельзя инициализировать значением в четырех особых случаях:

• при объявлении их с ключевым словом extern;

• при использовании в качестве параметров функции;

• при использовании в качестве типа, возвращаемым значением функции;

• в объявлениях классов.

Не существует операторов, непосредственно совершают действия над поселком. Все операции происходят только над соответствующими объектами. Предположим, добавим оператор

iref ++;

Компилятор сгенерирует код, инкрементирует ivar - переменную, на которую ссылается iref. Оператор не совершит никаких действий непосредственно над iref.

ссылки также могут ссылаться на константы. Присвоить значение непосредственно ссылкам нельзя. объявления

int & iref = 1234;

является ошибкой, так как 1234 не является объектом, на который iref может ссылаться.

Можно указать, константную ссылку на объект так:

int x = 1234;

const int & iref = x;

Этот прием эффективно преобразует переменное на константу, которую нельзя подвергнуть изменениям. Значение переменной x можно изменить явно, но присвоить x новое значение через ссылку нельзя, потому что iref - константа.

ссылки в качестве параметров функций. Сами по себе переменные-ссылки используются редко. Вместо того, чтобы использовать ссылки для доступа к другой переменной, легче использовать саму эту переменную В качестве параметров функции ссылки имеют более широкое применение. Посылки полезные в функциях, которые возвращают два или более значений вводимых данных.

Пример использования посылок в качестве параметров функции: вычисление суммарного налога и розничной цены по известной общей стоимости покупки и налогового тарифа.

#include <iostream.h>

#include <stdlib.h>

#include <stdio.h>

double GetDouble (const char \* prompt)

void GetData (double &paid, double &rate)

void Calculate (const double paid, const double rate, double & list, double & tax)

int main ()

{

double paid, rate, list, tax;

GetData (paid, rate)

Calculate (paid, rate, list, tax)

printf ( "List prise = $% 8.2f \ n", tax)

return 0;

}

double GetDouble (const char \* prompt)

{

char s [20]; // Строка для ввода

printf (prompt) // Отображение строки запроса

scanf ( "% 20s", s); // Вывести данные как строка

return atof (s) // Вернуть введенные данные

}

// Меняются значение paid и rate

void GetData (double & paid, double & rate)

{

paid = GetDouble ( "Price paid?”);

rate = GetDouble ( "Tax rate (ex :. 06)?»);

}

// Меняются значение list и tax

void Calculate (const double paid, const double rate, double & list, double & tax)

{

list = paid / (1 + rate)

tax = paid - list;

}

В программе определены три функции.

Первая - GetDouble (const char \* prompt) запрашивает и возвращает истинное значение двойной точности.

Вторая - GetData (double & paid, double & rate) демонстрирует использование параметров-посылок для возвращения двух значений. К функции GetData () передаются адреса параметров вместо их значений.

Рассмотрим тело функции GetData (). В нем дважды вызывается GetDouble () с аргументом-строкой, должен отображаться, и полученные значения присваиваются параметрам-посылкам paid и rate. Поскольку paid и rate - посылки, значение присваиваются объектам, переданным функции GetDouble ().

Функция вычисляет цену и налог к ​​уплате, основываясь на действительной цене ставки налога. Здесь paid и rate объявлены константами и поэтому функция их не меняет. Передаются их значение. Параметры list и tax - входные параметры функции. Они меняются функции и, следовательно, передаются посылкой. И поскольку параметры - это посылки, в действительности используются переменные, переданные функции Calculate (). Операция присвоение происходит за посылкой, в результате чего изменяются действительны переменные, когда в функции main () происходит вызов функции Calculate ().

Аргументы, которые передаются по значению, невозможно отличить от аргументов, передаваемых по ссылкам. При использовании параметров-посылок необходимо хорошее документирование программ.

ссылки и указатели в качестве параметров тесно связаны. ссылки могут интерпретироваться компилятором С ++ как указатели. Рассмотрим такую ​​функцию:

void f (int \* ip)

{

  \* Ip = 12;

}

Внутри этой функции осуществляется доступ к переданному аргументу, адрес которого хранится в указателе ip. Выражение f (& ivar) передает адрес ivar. Значение 12 присваивается переменной ivar. Аналогичная функция, использует параметры-посылки выглядит так:

void f (int & ir)

{

ir = 12;

}

Указатель \* ip заменен посылкой ir, которой функция присваивает значение 12. Выражение f (ivar)

присваивает значение объекта-ссылке: передает ivar за посылкой к функции f (). И поскольку ir ссылается на ivar, на самом деле значение 12 присваивается ivar.

Посылки в качестве результатов функций. Функции могут возвращать посылки на объекты при условии, что эти объекты существуют. Например, объявим функцию:

double & ref (double d)

Ей необходим один действительный аргумент двойной точности. Возвращает она посылку на объект того же типа, объявленный в другом месте. Это можно использовать при сокрытии деталей реализации внутренней структуры данных, позволяет проводить в ней изменения без соответствующих изменений в коде.