# Лекция 2

Тема лекции: Комментарии С ++. Вступление в потоки i/o. Oбласть видение и объявления переменных. Встроенные функции. Управление памятью с помощью NEW и DELETE

Вступление в С ++. Необъектные свойства С ++

  В названии языка С ++ используется оператор инкремента языка С. Дословно С ++ - следующий шаг по С. Изобретатель ее - Бьерн Страуструп назвал этот язык "улучшенная С". Язык С ++ сделала значительный вклад в понятие класса - особой структуры, членами которой могут быть как данные, так и функции. Кроме классов, в С ++ добавлены несколько новых ключевых слов и операторов, встроенные и перегружены функции, перегрузка операторов, новая техника управления памятью и другие средства. Программы на С и С ++ очень похожи. В них примерно одинаковый синтаксис, те же виды циклов, типы данных, указатели и др. Для того, чтобы отметить компилятору, какой из языков он должен обработать, программам, написанным на С ++ присваивается расширение .CPP. Заголовочный файлам можно давать расширение .HPP, хотя компилятор не оказывает различий между .H и .HPP.

Пример программы на C ++:

#include <iostream. h>

main ()

{

 cout << "Welcome to C ++! \n ";

 return 0;

}

В этой программе, во-первых, включается заголовочный файл iostream.h вместо stdio.h. Во-вторых, используется стандартный поток вывода объектов cout (character out или "вывода символов") библиотеки потоков ввода / вывода для вывода текста на экран. Для того, чтобы направить информацию на экран, библиотека потоков ввода / вывода использует оператор С ++ << (направить в). Можно также использовать составной оператор потоков ввода / вывода. Если FirstName и LastName имеют тип указателей на строку char \*, то и выражение

cout << LastName << "," << FirstName << '\ n';

выводит два имени, разделенных запятой и заканчиваются переходом на следующую строку. Возможно написание каждой части составного оператора iostream с новой строки. К примеру:

cout << LastName

<< ","

<< FirstName

<< '\ N';

Библиотека потоков ввода / вывода способна выводить различные типы данных, включая встроенные (int, long, double), а также типы, созданные программистом. Если count имеет тип long, тогда выражение

cout << "count ==" << count << '\ n';

выведет строку, завершается значением count и переходом на следующую строку.

В С ++ можно использовать методы stdio.h и библиотеки потоков ввода / вывода вместе, а также использовать printf () и аналогичные функции. Но функция printf () может манипулировать только встроенными типами данных и своими форматами вывода, которые нельзя запрограммировать.

Основные отличия С и С ++

• В С ++ были введены дополнительные ключевые слова: catch, class, delete, friend, inline, new, operator, private, protected, public, template, this, throw, try, virtual.

• C ++ требует, чтобы прототипы всех функций были определены перед их вызовом. В С ++ осуществляется более строгая проверка типов в выражениях.

• Значение в выражениях должны быть или одинакового типа, или легко преобразующими на соответствующий тип. Там, где С выдает предупреждение о несоответствии типов, С ++ выдает ошибку компиляции.

• В С ++ очень ослаблена потребность в typedef-объявлениях. Например, можно объявить структуру типа

struc mystruct {...};

и потом объявлять переменные как

mystruct x;

• В С нужно использовать typedef для структуры mystruct (или struct mystruct x).

• В С символьные константы имеют тип int, и выражение sizeof ( 'x') эквивалентно sizeof (int). В С ++ символьные константы имеют тип char. И sizeof ( 'x') эквивалентно sizeof (char).

Этот список не является полным.

# Комментарий С ++

В программе С ++ "//" (двойная черта) означает комментарий. Занимает он одну строку. Можно использовать и комментарий С. Внутрь оператора можно вставить только комментарий С. Наприлад:

result = count / \* Вложенный комментарий \* / + 100;

В одной и той же программе одновременно могут использоваться комментарии С и С ++.

# Вступление в потоков ввода / вывода

Ввода и вывода в С ++ не является частью языка, а обеспечивается внешними библиотечными модулями. Поэтому программисты могут создавать свои функции и средства ввода / вывода. Потоки cout и cin имеют много возможностей. Для просмотра символ р, можно написать:

cout << p; // или

cout.put (p)

Функция put () - член потока cout. Поток cout - пример класса, имеет относящиеся к нему функции. Нужно вызвать функцию-член только со ссылкой на объект, подобный cout. Без этого выражение

put (p) попытается вызвать независимую функцию put (). оператор

cout.put (p) вызывает функцию put (), принадлежащей cout.

Для того, чтобы прочитать символ из стандартного потока ввода, нужно использовать поток ввода cin, что стоит перед оператором >> (взять с). оператор

cin >> p;

читает один символ из стандартного потока ввода и присваивает его переменной p. Можно также вызвать функцию-член get с cin:

cin.get (p)

Библиотека потоков ввода / вывода может выводить символьные строки. Например, следующее объявление присваивает указателю на строку адрес строки символов:

char \* s = "символы"; оператор

cout << s; отправит данная строка в системный стандартный поток вывода, обычно на дисплей. Для чтения строки производится обратный процесс.

# Область видения и объявления переменных

Неоднозначности области видение переменных могут возникать в программах, написанных как на С так и на С ++. Например, если существует глобальная переменная int COUNT Функция может объявить переменную с тем же именем, не вызвав при этом сообщение об ошибке

int COUNT; // Глобальная переменная

void AnyFunction ()

{

 int COUNT; ....

}

Область видение локальной переменной COUNT действует в пределах объявленной функции. В этой функции невозможно доступ к глобальной переменной COUNT. Для решения этой проблемы в С ++ используется оператор расширения области видение: "::". Перепишем функцию в таком виде:

int COUNT;

void AnyFunction ()

{

int COUNT;

   COUNT = 2;

   :: COUNT = 3;

}

Здесь локальной переменной COUNT присваивается значение 2, а глобальной - 3. Иллюстрируется еще одно свойство С ++: объявления в С ++ можно вводить в любой точке программы, а не только глобально или в начале функции, как у С. Переменные, объявлены внутри блока-оператора, существуют только в пределах этого блока.

К примеру:

if (ex)

{

int COUNT = 0 ...... ..

}

cout << COUNT << '\ n';

При компиляции последнюю строчку вызывает сообщение об ошибке, поскольку переменная COUNT объявлена ​​вне области видение оператора if, но она не существует вне ее. Бывает полезно одновременно объявлять и инициализировать переменную, управляющий циклом for. К примеру:

void f ()

{

for (int i = 0; i <max; i ++)

..........  }

Такое объявление действует не только в пределах видимости цикла for. Инициализация управляющей переменной выполняется до того, как начинается цикл. В данном примере i попадает в области видение f () и может использоваться любыми операторами, следующих за оператором for.

# К**онстанты**

Объявление переменных с ключевым словом const защищает их от изменения во время выполнения программы. К примеру:

const int count = 1234;

Компилятор выдаст сообщение об ошибке в операторе

count ++;

Рекомендуется использовать объявления const вместо символьных констант, созданных с помощью #define. символьная константа

#define MAX 100

определяет макрос с именем MAX, связан с текстом 100 (не целым значением 100). Если MAX используется следующим образом:

for (int i = 0; i <MAX; i ++)

...;

то компилятор заменит MAX на текстовые цифры 100. Если объявить MAX так:

const int MAX = 100;

то цикл for останется таким же, а объявления MAX действительным значением константы даст следующие преимущества:

• Компилятор сможет выполнить проверку типов с MAX более строго. С ++ распознает, что константа MAX имеет тип int, в то время как компилятор ничего не знает о типе текстового макроса MAX.

• Turbo Debugger распознает истинное значение константы MAX, но не распознает символьные константы, созданные с помощью #define.

# встроенные функции

На вызов функций тратится время. Многочисленные вызовы ухудшают общую скорость программы. Решение этой проблемы - встроенные функции. Чтобы объявить функцию встроенной, нужно использовать ключевое слово inline. К примеру:

inline int func (a, b)

Обычно встроенные функции объявляются в заголовочный файлах и включаются туда, где необходимо их использования. Встроенную функцию необходимо полностью определить до того, как можно будет ее использовать и включение ее текста в заголовочный файл - простейший способ удовлетворения данного требования. В скомпилированном коде программы Borland C ++ не вызывает встроенную функцию в строке, где она записана. Вместо этого компилятор вставляет тело функции непосредственно к телу программы. Но, если код встроенной функции слишком большой, то компилятор может отказаться встраивать функцию в поток кода и вместо этого сгенерирует обычный вызов. Также при компиляции программ Turbo Debugger все встроенные функции будут преобразованы в обычные функции, вызываемые.

# Управление памятью с помощью операторов new и delete

  Можно использовать одинаковую технику управления памятью в программах на С и на С ++. Но С ++ предлагает альтернативные операторы управления памятью new и delete. Использование этих операторов вместо стандартных функций управления памятью позволяет при необходимости самостоятельно обрабатывать ситуацию управления памятью. Чтобы выделить память для переменной, имеет тип double и присвоить ее адрес указателю, можно написать:

double \* dp = new double;

Указатель dp используется так же, как и любой другой указатель на память, выделенную функцией malloc () или подобной функцией. Закончив использования динамической переменной, удалите ее следующим образом:

delete dp;

Удаление указателя означает освобождение выделенной ранее памяти, адрес которой хранится в dp, для последующего использования. Для освобождения памяти, выделенной с помощью new, необходимо использовать только delete.

# Выявление ошибок связанных с нехваткой памяти

При создании динамических переменных необходимо выяснить, достаточно свободной памяти. Если операция new завершилась успешно, результатом ее является указатель на зарезервированную память необходимого размера. В случае неудачного завершения, возможны следующие ситуации:

• new возбуждает исключительную ситуацию.

• new возвращает ноль.

Чтобы не возбуждать исключительную ситуацию, нужно включить заголовочный файл new.h в программу и добавить в main () или в любое другое место к использованию new, оператор:

set\_new\_handler (0);

При завершении программы вся выделенная память автоматически освобождается. Однако, нужно удалять динамические переменные после окончания их использования. Также желательно удалять указатель, даже если он равен нулю. Удаление того же указателя повторно или повторное использование адресованной им памяти может вызвать серьезные ошибки. Удалив указатель, ни в коем случае нельзя использовать его ни для каких целей, кроме адресации памяти, заново выделенной new. Оператор new можно использовать для динамического выделения памяти для строки любого размера. К примеру:

#include <iostream. h>

#define SIZE 80

main ()

{

char \* sp;

sp = new char [SIZE]; // Выделяем символьный массив

cout << "String? "; // Размером SIZE

cin.getline (sp, SIZE) // Присваиваем адрес 1-го байта sp

cout << "You entered:" << sp << '\ n';

delete [] sp;

return 0;

}

Можно оператор delete написать без скобок. Скобки необходимы лишь в том случае, когда удаляется массив классовых объектов. В С ++ new и delete также могут использоваться для создания и освобождения динамических массивов любого типа.