*Если вы считаете, что C++ труден, попытайтесь выучить английский.*

*Бьёрн Страуструп*

Путь тернистый …

или программирование в Visual C++

И вот наступил тот момент , когда вы, уже достаточно мотивированы вашими преподавателями, преисполнились желаниям изучить и создать, подошли к своему компу…а что собственно делать?

И тут нужно сказать, что разработка программ на языке С++ ведется с помощью специальных комплексов программ, которые называются *системами программирования* и позволяют создавать программы на определенной *реализации* языка. Системы программирования даже одного производителя имеют различные *версии*, которые отражают развитие технологии программирования и эволюции среды выполнения программ. Это позволит максимально использовать *стандартные средства* языка для того, чтобы снизить затраты на модификацию программ при изменении среды выполнения или при переходе на другую версию языка.

Поэтому , прежде всего, нужно определить, какой именно комплекс вы хотите установить у себя на компьютере.

Мы будем работать в Visual Studio C++ версии 6.0., и если вы решили технические проблемы – присоединяйтесь.

Первый вопрос, который вас занимает, а зачем необходим этот язык программирования? С одной стороны, раз требуют изучить, значит нужно…Но…

Вот эта железяка, на которой вы играете, работает только, если в ней есть программы. Программа внутри железяки представляет собой машинный код, который состоит он из нулей и единиц. Описать в таком виде свою задачу трудно. Очень трудно. Язык программирования дает вам возможность задать свои требования в гораздо более удобном виде. Язык программирования выбирается в соответствии с классом решаемых задач. Существуют и универсальные языки программирования, С++ как раз таков.

Процесс создания программ включает четыре этапа:

1.Написание и редактирование исходного текста программы с сохранением ее в виде *исходного файла* или *модуля.*

Именно на этом этапе вы должны написать ваши требования к тому, какие действия будет выполнять программа в виде конструкций языка С++.

2.Компиляция программы и получение ее на определенном промежуточном языке с сохранением в виде *объектного файла* или *модуля*.

Происходит преобразование конструкций языка в машинные коды. И на этом этапе выявляются синтаксические ошибки. Если конструкция языка записано вами неверно, то преобразование произойти не может, и вы получаете сообщение об ошибке.

3.Построение *исполнимого файла* или *модуля* путем объединения (компоновки) полученного объектного модуля программы с другими объектными модулями стандартных и специальных библиотек.

4.Отладка программы с помощью специального средства (*отладчика*), облегчающего обнаружение ошибок.

Соответственно, чтобы все это выполнить, в современных системах программирования должны присутствовать такие компоненты:

-интегрированная среда программирования;

-редактор связей (компоновщик);

-библиотеки заголовочных файлов:

-стандартные и специальные библиотеки;

-библиотеки примеров программ;

-программы-утилиты;

-файлы документации.

Интегрированная среда разработки

(Integrated Development Environment) – это полностью самодостаточная среда, предназначенная для создания, компиляции компоновки и тестирования программ на С++.

Редактор представляет собой интерактивную среду, в которой создается и редактируется исходный код С++.

Наряду с обычными средствами вроде вырезания и вставки фрагментов, редактор обеспечивает цветовое выделение различных элементов языка. Редактор автоматически распознает ключевые конструкции языка С++ и окрашивает их в соответствии с их значением.

Компилятор преобразует исходный код в объектный код, обнаруживает и извещает об ошибках в процессе компиляции. Компилятор может обнаружить широкий диапазон ошибок, связанных с некорректным или нераспознаваемым программным кодом , а также структурные ошибки.

Выходной объектный код, созданный компилятором, помещается в объектные файлы. Файлы с объектным кодом имеют расширение .obj.

Компоновщик комбинирует вместе различные модули, сгенерированные компилятором из файлов исходного кода, добавляет необходимые модули из библиотек и сшивает в одно исполняемое целое.

Библиотека – это коллекция предварительно написанных процедур , которые поддерживают и расширяют язык С++, предоставляя для использования стандартные, профессионально разработанные единицы кода для выполнения часто встречающихся операций.

В процессе разработки часто возникает ситуация, когда необходима дополнительная информация о Visual C++ Библиотека разработчика Microsoft Development Network (MSDN) содержит материалы обо всех возможностях Visual C++, а также всю сопутствующую информацию.

Схема получения исполнимого модуля программы в интегрированной среде показана на рисунке. Исходный модуль (ИМ) программы подготавливается с помощью встроенного или внешнего текстового редактора и размещается в файле с расширением *срр.* После этого ИМ обрабатывается процессором и, в случае необходимости, к исходному тексту программы присоединяются подключаемые файлы (ПФ). В дальнейшем модернизированный исходный модуль (ИМ\*) обрабатывается компилятором. Выявленные синтаксические ошибки устраняются, и безошибочно откомпилированный объектный модуль(ОМ) помещается в файл с расширением *obj.* Затем ОМ обрабатывается компоновщиком, который дополняет программу нужными библиотечными функциями из библиотечных файлов (БФ). Полученный модуль называется исполнимым модулем (ИсМ) и помещается в файл с расширением *ехе* , который в дальнейшем исполняется.

Программы – утилиты – это вспомогательные программы, которые могут потребоваться при создании программ. В качестве примера программы-утилиты следует назвать программу - библиотекарь, которая позволяет объединять объектные модули в один файл, называемый статической библиотекой. Такой файл имеет расширение lib. Примером программы - утилиты является автономный отладчик. С помощью отладчика можно, в частности, выполнять программу в пошаговом режиме и контролировать содержимое всех переменных программы.

*Выше голову!*

*Жизнь хочет вас по ней погладить.*

***Станислав Ежи Лец***

Тайны логистики или

Данные переменные и вычисления

Обратимся теперь к основам программирования на С++. Наша первая цель - написать простую программу на С++ в традиционной форме : ввод, обработка, вывод.

Азы строительного мастерства или

Структура программы С++

Основы языка будем познавать, создавая консольное приложение.

И так, для создания в Microsoft Visual C++ 6.0 самого простого типа проекта вам необходимо выполнить следующие действия:

* Запустим Visual C++. В меню выбераем «File => New => Projects».
* В появившемся окне диалога в поле Project name задаем имя проекта: PRIMER.
* Укажем в поле Location месторасположение папки с проектом. Среда Microsoft Visual C++ 6.0 автоматически создаст папку проекта.
* Выберем тип проекта Win32 Console Application и нажмем кнопку OK (рис.1).
* Согласимся с выбором An empty project, нажав Finish (рис. 2).
* Подтвердм выбор, нажав OK (в окне New project).

Вот, мы и создали проект простого консольного приложения.

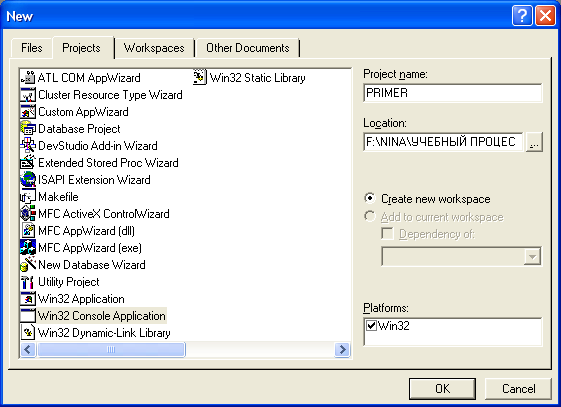


рис.1.

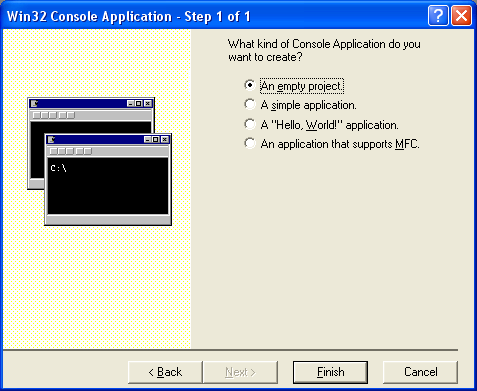


рис.2.

* В меню выберите File => New => Files. В появившемся окне диалога (рис. 3) выберите C++ Source File.  Поставьте птичку «Add to project». В поле File name задайте имя main и нажмите кнопку OK.

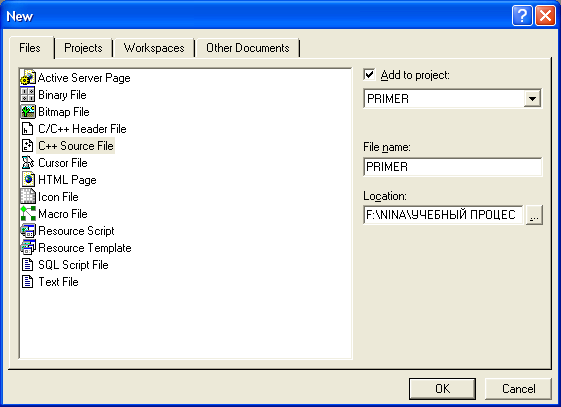


рис.3.

После этого ваш экран примет вид представленный на рис. 4. В заголовке окна пишется имя проекта и, в квадратных скобках, текущий открытый файл. В нашем случае это «First» и «main.cpp».

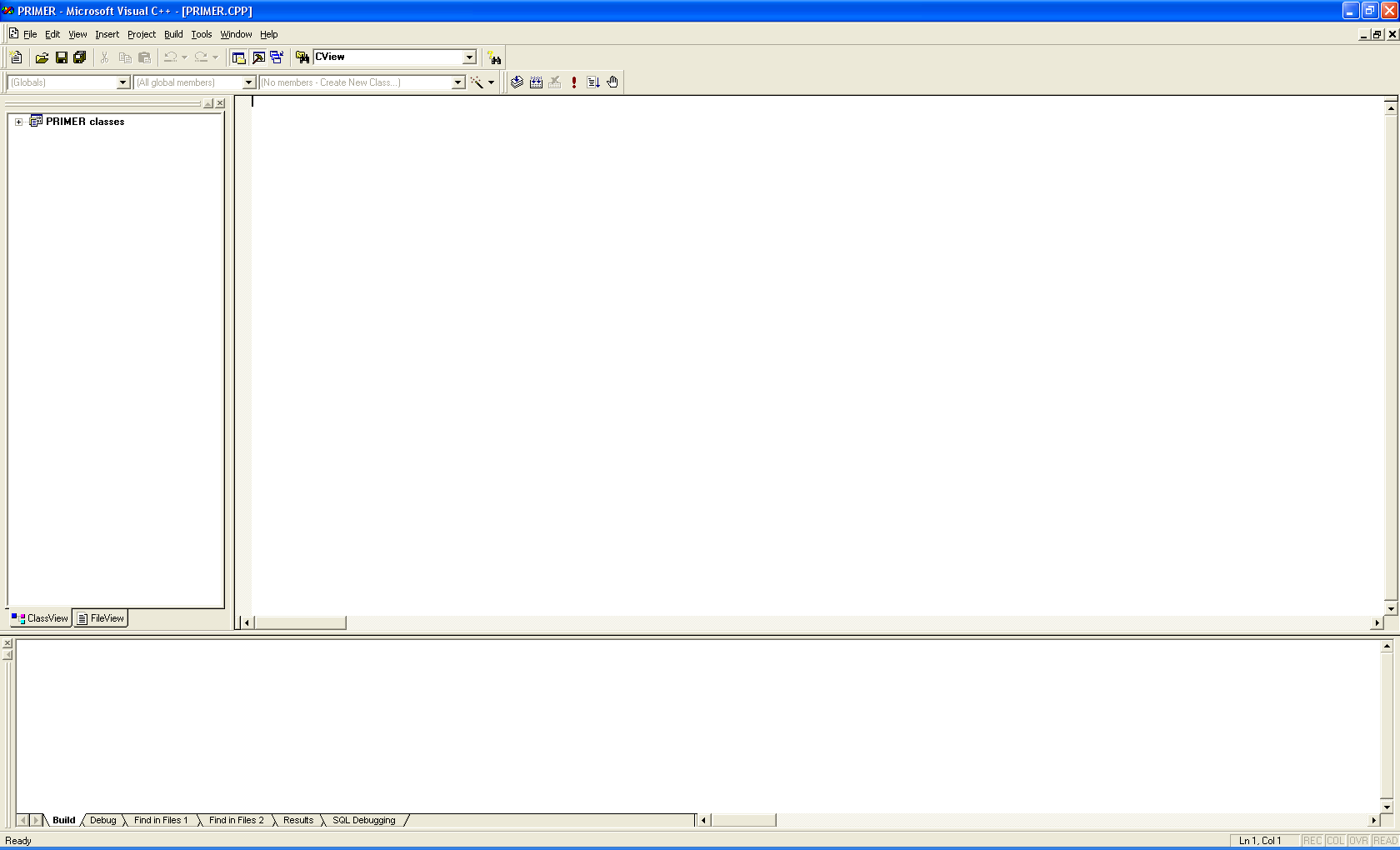


рис.4.

Вертикальная область слева называется окном рабочего пространства (Workspace). Справа от него располагается окно редактора кода, в которое выводится содержимое текущего файла. В данный момент файл main.cpp пуст.

В дальнейшем мы будем очень часто и охотно пользоваться окном Workspace, так как оно позволят легко осуществлять навигацию по различным файлам проекта, а также функциям, классам и переменным.

Программы, запускаемые как консольное приложение под Visual C++, читают данные из командной строки и туда же выводят результаты. Определим , каким образом структурированы консольные программы.

Программа на С++ состоит из одной или более функций.

Каждая программа С++ содержит функцию main(). Обычно программы С++ любого размера состоят из нескольких функций - функции main(), с которой начинается выполнение программы и некоторого количества других функций.

Функция – это просто блок кода с уникальным именем , которое используется для запуска ее на выполнение.

Типичная программа командной строки может быть структурирована, как показано на рис.5.



рис.5

А теперь рассмотрим самую простую программу с одной целью – определить, из каких компонентов, строительных блоков, она состоит.

Программа из листинга 1 достаточно проста и состоит из одной функции main().

// Листинг 1

//Простой пример программы

#include < iostream.h>

int main()

{

int apples, oranges; //Объявление двух целоч.пер

int fruit; //… и еще одной

apples = 5; //Присваивание нач. значений

oranges = 6;

fruit = apples + oranges; //Получить сумму

cout << endl; //Новая строка

cout << “ we have” << fruit << “ fruit”; //Вывод

cout << endl; //Новая строка

return 0; //выход из программы

}

Вот так будет выглядеть исходный код вашего примера в окне редактора рис.6.

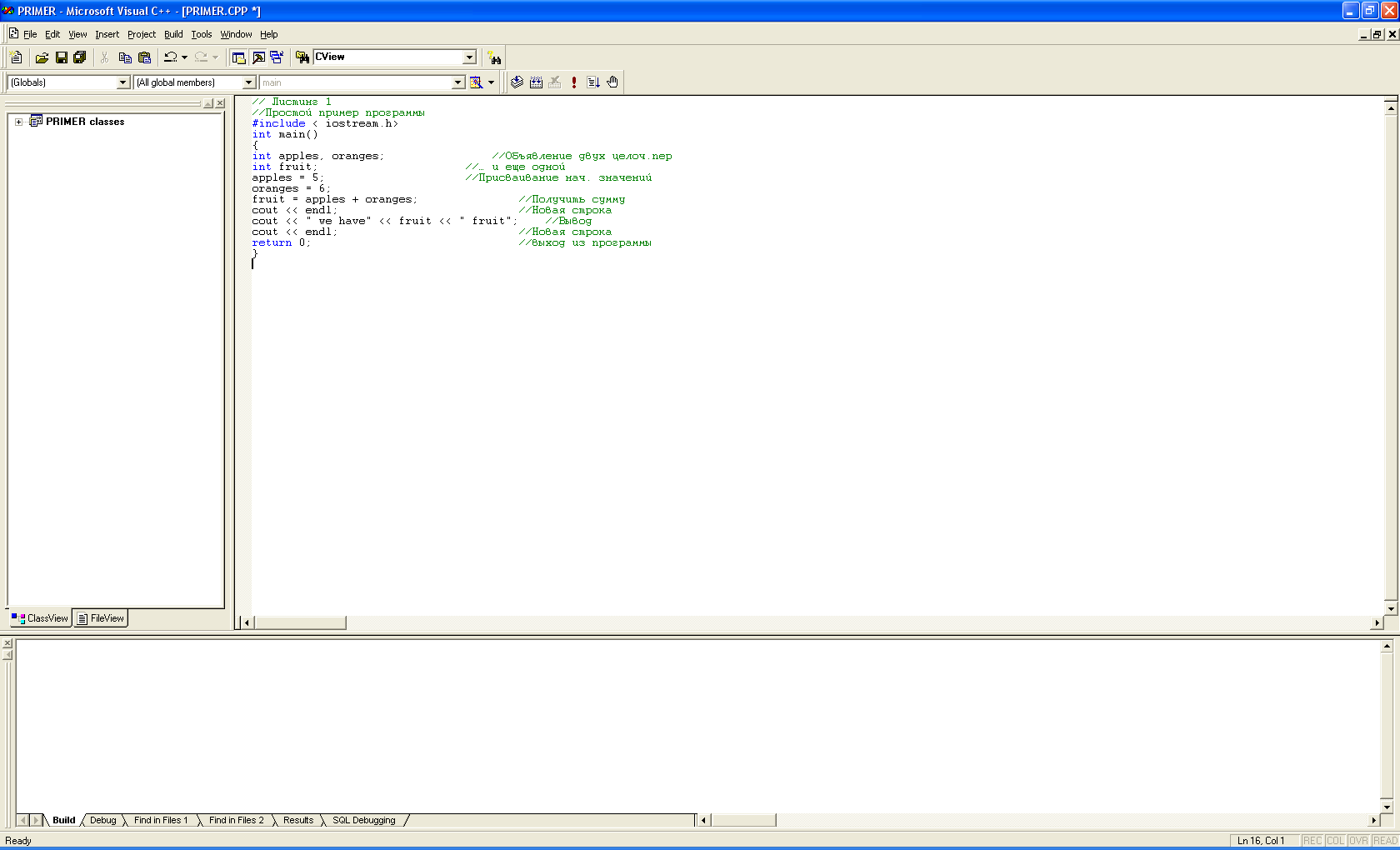


рис.6

Эта программа содержит следующие элементы:

* Комментарии, предваряемые двумя косыми чертами - //
* Директива препроцессора *#include*
* Заголовок функции int *main()*
* Тело функции, ограниченное символами { }
* Оператор , в котором для вывода сообщений на экран используется объект *cout*
* Оператор *return* , завершающий выполнение функции main()

**Послание в вечность или**

**Комментарии в языке С++**

В языке С++ комментарии обозначаются двойной наклонной чертой (//).

Комментарий - это написанное программистом примечание, которое служит для идентификации разделов программы, или пояснения некоторых аспектов кода программы. Вы пишите их для себя или для ваших последователей.

Компилятор комментарии игнорирует.

Комментарием в языке С++ считается текст от символов // и до конца строки.

Может находиться в отдельной строке , или в той же строке, что и код программы.

В программе на С++ можно использовать комментарии на языке С , которые заключены между символами /\* и \*/.

Поскольку комментарий из языка С завершается не символом конца строки, а символом \*/ , он может занимать несколько строк.

Комментарий необходимая часть вашей программы, как напоминание о том что и зачем вы делали. Помогает и как шпаргалка на экзамене по программированию.

**Не бойтесь давать указания или**

Директива *#include* – файлы заголовков

Вслед за начальным комментарием в программе находится директива *#include*

Она называется директивой, поскольку указывает компилятору сделать что-то, в данном случае, вставить содержимое файла *< iostream.h>* в программу перед ее компиляцией. Он называется файлом заголовков, потому, что обычно появляется в

начале программного файла. Заголовочный файл *< iostream.h>* включает определения, необходимые , для использования операторов ввода- вывода С++.

Существует множество заголовочных файлов, редактор распознает их и окрашивает в голубой цвет.

Директивы препроцессора – это команды, выполняемые на фазе предварительной обработки компилятора, которая выполняется перед тем, как исходный код будет скомпилирован в объектный код.

При создании исполняемого кода, программой на языке С++ используется препроцессор. Это программа, которая обрабатывает исходный файл перед основной компиляцией, запускается автоматически при компиляции программы.

Эта директива указывает препроцессору добавить в программу содержимое файла *iostream*. Не забывайте давать указания!

Файлы, такие как *iostream*, называются включаемыми или заголовочными. Заголовочные файлы , традиционно , имеют расширение *h*.

**Наше все или**

Функция *main()*

Учебная программа, представленная в листинге 1, имеет следующую структуру

int main()

{

операторы

return 0;

}

Эти строки означают, что перед нами функция с именем *main()* , и она содержит описание работы этой функции.

Вся совокупность приведенных выше строк , составляет определение функции.

Это определение состоит из двух частей:

заголовка функции и тела функции.

Заголовок функции в сжатом виде определяет интерфейс между функцией и остальной частью программы, а тело функции содержит инструкции для компьютера, т.е. определяет, что именно делает функция.

В языке С++ , каждая завершенная инструкция называется оператором. Каждый оператор должен заканчиваться точкой с запятой.

Заключительный оператор в функции *return ();* называется оператором возврата и завершает функцию.

Синтаксис языка С++ требует, чтобы определение функции *main()* начиналось со следующего заголовка: *int main().*

В общем случае , который будет рассмотрен далее, функция С++ активизируется или вызывается другой функцией, а заголовок функции описывает интерфейс между нею и той функцией, которая ее вызывает.

Слово, стоящее перед именем функции, называется возвращенным типом функции, оно описывает информацию, передаваемую из функции обратно в ту функцию, которая ее вызвала.

Информация в круглых скобках, следующая за именем функции называется списком аргументов или списком параметров*.* Этот список содержит описание информации , передаваемой из вызывающей в вызываемую функцию.

*main()*, как правило , вызывается кодом начальной загрузки, который добавляется в программу компилятором для связи программы с операционной системой.

Можно использовать

*int main(void)*

*void main()*

*void main(void)*

Обычно программа на С++ должна содержать функцию *main().* Выполнение программы всегда начинается с нее.

**Строительные блоки или**

Операторы программы

Каждый из операторов программы, образующих тело функции *main()* , завершается точкой с запятой. Этот символ отмечает конец оператора.

Один оператор может распространяться на несколько строк.

Оператор программы – это базовый элемент, определяющий то, что делает программа.

Оператор - определение действия , которое должен выполнить компьютер. Действие функции всегда выражается набором операторов, каждый из которых завершается точкой с запятой.

В приведенном примере в теле функции *main()*  представлены операторы:

объявления переменных,

операторы присваивания,

операторы вывода,

выход из программы.

**Чарующая пустота или**

Пробелы

Пробел – термин, используемый в С++ для обозначения символов пробела, табуляции, новой строки, новой страницы и комментариев. Пробелы служат разделителями одной части оператора от другой и позволяют компилятору определять, где заканчивается один элемент оператора и начинается другой.

В других случаях, пробелы игнорируются и никакого влияния не оказывают.

Поэтому можно включать в программу сколько угодно пробелов для повышения ее читабельности.

Правила оформления текста программы, направленные на облегчение понимания смысла и повышение наглядности.

Ну очень полезны. Они не всегда побеждают в борьбе с ленью, но если ими пользоваться, сокращают время на разработку программ. Не приходится тратить время на бесплодные воспоминания и необоснованные переделки.

* разделять логические части программы пустыми строками
* разделять операнды и операции пробелами
* для каждой фигурной скобки отводить отдельную строку
* в каждой строке должно быть не больше одного оператора
* ограничить длину строки 60-70 символами
* отступами отражать вложенность операторов и блоков
* длинные операторы располагать в нескольких блоках
* составлять алгоритм так, чтобы определение одной функции занимало, как правило, не более одного экрана текста
* стремиться использовать типовые заголовки фрагментов программ, типовую структуру блока и программы в целом

И, наконец, завершим нашу программу из листинга 1. Предполагаем, что код уже набран, нажимаем клавишу F7 для компиляции. Внизу экрана появится окно, в которое выводятся результаты компиляции (рис. 7).

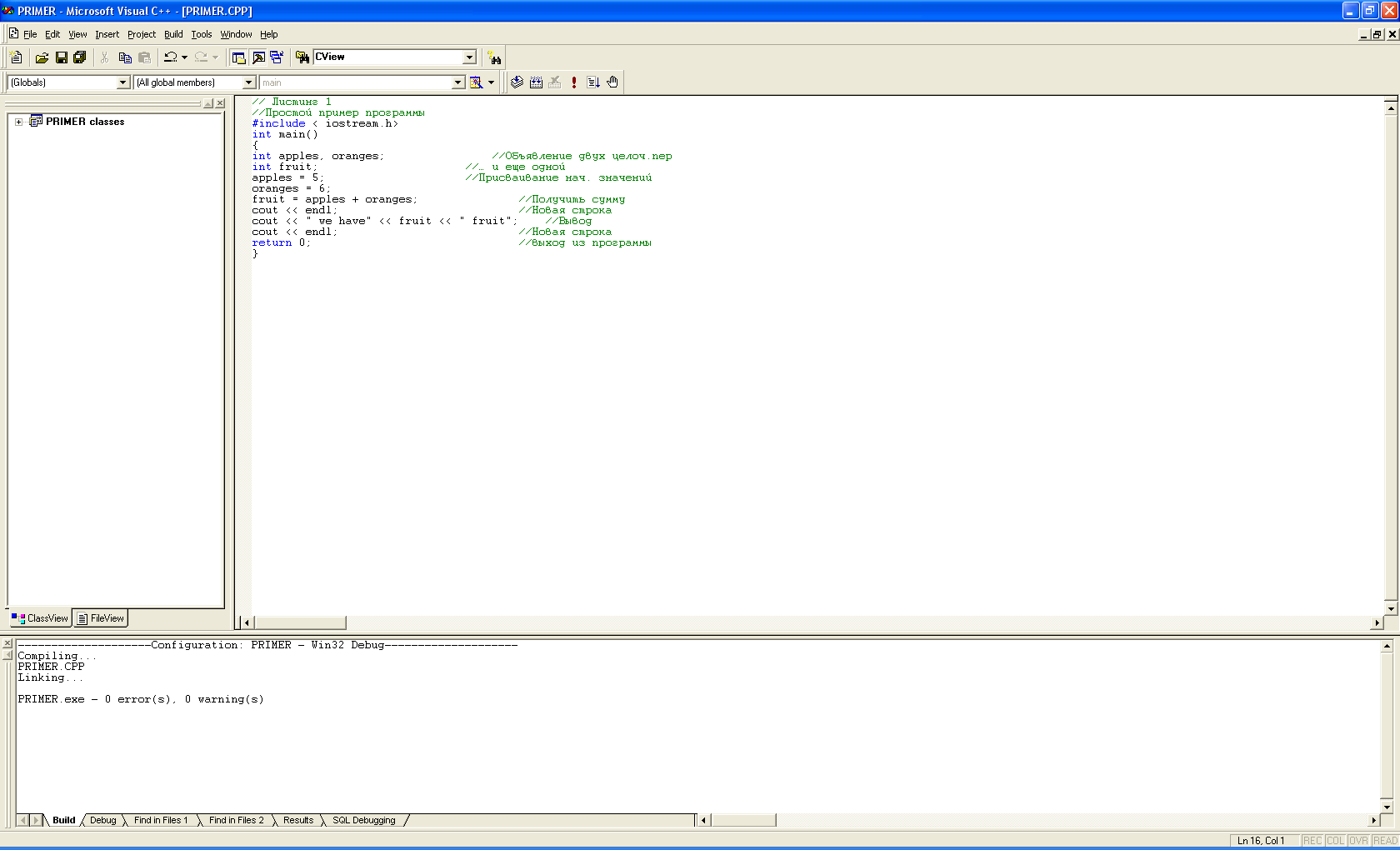


рис.7

Если достигнута безошибочная компиляция этой программыто нажимаем комбинации клавиш Ctrl + F5 и получаем результат на рис.8.

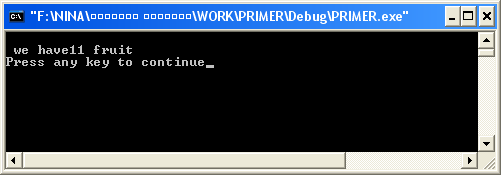


рис.8

*Где искать залежи мудрости?*

*Да там, где ее зарыли.*

***Станислав Ежи Лец***

**Как сохранить или**

Определение переменных

Зачем пишут программы?

Для того, чтобы манипулировать некоторыми данными и получать некоторые результаты.

А где хранить эти данные и как их находить, когда они нужны?

Для хранения и обращения к элементу данных необходим фрагмент памяти, к которому можно обращаться по некоторому осмысленному имени.

Каждый фрагмент памяти, определенный таким образом, называется переменной.

Каждая переменная сохраняет данные определенного вида, и тип сохраняемых данных фиксируется при объявлении переменных в программе,

т.е мы сначала должны определить, какого типа данные будут размещаться в переменной, а потом заявить о своем желании, объявив переменную.

В объявлении, как мы видели в примере, указывается и имя переменной.

И далее обращение к переменной в программе будет осуществляться по этому имени.

От типа переменной зависит размер и организация фрагмента памяти, который мы называем переменной.

Конкретное значение, хранимое переменной в каждый отдельный момент времени, определяется операторами программы, и , как правило, ее значение меняется много раз в процессе вычислений, выполняемых программой.

Если в тексте программы используется оператор

*int Х=5;*

То он означает, что в памяти выделен фрагмент для хранения данных целого типа, обратиться в программе к этому фрагменту можно по имени Х, и в данный момент в этом фрагменте записано целое число 5,что может еще многократно измениться.

**Как корыто назовете… или**

Именование переменных

Имя, которое присваивается переменной, называется идентификатором, или – именем переменной.

Имена переменных могут включить

буквы A – z (в верхнем или нижнем регистре),

цифры 0-9,

знаки подчеркивания.

Имена переменных должны начинаться либо с буквы, либо с подчеркивания.

Хотя имена переменных в Visual C++ могут иметь длину до 2048 символов, желательно ограничиться именами не длиннее 30 символов.

Желательно также избегать имен, начинающихся с подчеркивания , т.к. возможны конфликты со стандартными системными переменными, имеющими такую же форму.

Примеры имен:

*Price*

*discount*

*value*

*COUNT*

Символы верхнего и нижнего регистра различаются. Пробельные символы не могут появляться в именах.

Обычное соглашение, принятое в С++, заключается в том, что имена, начинающиеся с заглавных букв, резервируются для именования классов, а с прописных – для именования переменных.

Имя переменной желательно давать осмысленно, памятуя об их назначении. Это поможет избежать многих ошибок при написании кода.

В С++ присутствуют зарезервированные слова, которые называют ключевыми словами и имеют специальное значение внутри языка. Эти языковые конструкции при вводе кода обычно подсвечиваются. Не следует такими словами называть свои переменные.

Примеры:

*int, return,do, while…*

**Основы перфекционизма или**

Объявление переменных

Объявление переменной – это оператор программы, который определяет (специфицирует) имя переменной определенного типа.

Пример:

*int value;*

Этот оператор объявляет переменную по имени *value* для хранения целых чисел. Тип данных, который может быть сохранен в переменной *value* , указан

ключевым словом *int*, поэтому *value* вы можете применять для хранения данных типа *int*, только целого типа.

Объявление переменной всегда оканчивается точкой с запятой. Возможно в одном объявлении указать несколько имен переменных, но лучше - в отдельных операторах. Необходимо определить переменную перед тем, как переменная будет задействована в вашей программе.

Объявляя переменную, можно присвоить ей начальное значение.

Объявление переменной, которое присваивает ей начальное значение, называется инициализацией.

Примеры:

*int value = 0;*

*int count = 10;*

*int number (5);*

Если не присвоить переменной начального значения, то, как правило , она содержит произвольный мусор, который находился в том участке памяти, который для нее выделился. Избавимся от него - инициализируем переменные при их объявлении.

**Огласите весь список или**

Фундаментальные типы данных

Разновидность информации, которую может содержать переменная, называется ее типом данных. Все данные и переменные в программе должны относиться к определенному типу. Стандарт ISO/ANSI представляет набор фундаментальных типов данных, определенных ключевыми словами.

**Комментарий.** Стандарт ISO/ANSI определен в документе ISO/IEC 14882, опубликованном Американским национальным институтом стандартизации (ANSI). Стандарт ISO/ANSI C++ описывает устойчивую версию C++, которая существует с 1998 года и поддерживается компиляторами большинства аппаратных компьютерных платформ и операционных систем. Программы, написанные на ISO /ANSI C++, относительно легко могут быть перенесены с одной платформы на другую, хотя используемые ими библиотечные функции, в частности, связанные с

построением графического интерфейса, являются главным фактором, определяющим, насколько легко или трудно такой перенос осуществить. Стандарт C++ ISO/ANSI - это главный инструмент, который выбирают профессиональные разработчики программ, поскольку он широко поддерживается и потому, что на сегодняшний день он является одним из наиболее мощных доступных языков программирования.

Фундаментальные типы данных называются так потому, что хранят значения типов, представляющих фундаментальные данные в компьютере.

Фундаментальные типы подразделяются на три категории –

* типы, содержащие целые числа,
* типы, содержащие не целочисленные значения,
* тип *void*, указывающий на пустое множество значений или отсутствие типа.

Целочисленные переменные

Целочисленные переменные – это переменные , которые содержат только значения целых чисел. Целочисленные переменные можно объявлять с ключевым словом *int*.

Переменная типа *int* занимает 4 байта памяти, и может хранить как положительные, так и отрицательные целые значения.

Верхний и нижний пределы значений переменных типа int соответствуют максимальному и минимальному двоичному значению со знаком, которое может быть представлено 32 битами. Верхний предел переменной типа int – 2 в степени 31 +1, что равно 2 147 483 647, а нижний предел - -2 в 31 степени, что соответствует -2 147 483 648.

Пример:

*int toeCount = 10;*

В Visual C++ ключевое слово *short* также определяет целые переменные, но занимащие 2 байта в памяти. Ключевое слово *short* эквивалентно *short int*, и вы можете определить две переменных типа short так:

*short feetPerPerson = 2;*

*short int feetPerYard = 3;*

В С++ также предусмотрен другой целочисленный тип – *long*, который также можно записывать как *long int*.

Вот как объявляются переменные типа *long*:

*long bigNumber = 10000000L;*

*long largeValue = 0L;*

Целочисленные переменные, объявленные как *long*, в VisualC++ занимают 4 байта, и могут принимать значения от -2 147 483 648 до 2 147 483 647. Тот же диапазон могут принимать переменные типа *int*.

В различных типах компьютеров размер переменных может различаться.

Общие принципы:

Размерность данных типа short не меньше 16 разрядов.

Размерность данных типа int не меньше размера типа short.

Размерность данных типа long не меньше 32 разрядов и не меньше размера данных типа int.

Символьные типы данных

Тип данных *char* служит двум целям. Он специфицирует однобайтную переменную, в которой можно сохранять целые числа в пределах определенного диапазона значений, или же код отдельного символа ASCII( American Standard code for Information Interchange)

Пример:

*char letter = ‘A’;*

Этот код объявляет переменную по имени *letter*, инициализированную константой ‘*А*’. Поскольку символ ‘*А*’ в кодировке ASCII представлен десятичным значением 65, вы могли бы написать этот оператор так:

*char letter = 65; // Эквивалент символа А*

Обратите внимание, что стандарт ISO/ANSI C++ не требует, чтобы тип *char* представлял однобайтные целые со знаком. Это определяет выбор реализации компилятора.

Для инициализации переменных *char* (как и других целых типов) вы также можете использовать шестнадцатеричные константы.

Модификаторы целочисленных типов

Переменные целочисленных типов *char, int, short* или *long* сохраняют по умолчанию целые значения со знаком (*signed*), поэтому вы можете применять их как для сохранения положительных, так и отрицательных значений, так как для этих типов по умолчанию принят модификатор типа *signed*. Когда вы пишете *int* или *long*, то это означает *signed int* или *signed long* соответственно.

Диапазон значений, которые могут быть сохранены в переменной типа char ,находится в пределах от -128 до +127.

Если вы уверены, что вам не нужно хранить в переменной отрицательные значения (например, если собираетесь записывать в нее количество миль, которые вы проезжаете за рулем в неделю), то вы можете специфицировать переменную как *unsigned*:

*unsigned long mileage = OUL;*

Минимальное значение, которое можно записать в переменную *mileage*, равно нуля, а максимальное – 4 294 967 295 (то есть в 2 в степени 32 – 1). Один бит, которые в переменных signed служит для указания знака, в *unsigned* является частью числового значения. Как следствие, переменные *unsigned* имеют больший диапазон положительных значений, но не могут представлять отрицательных.

Булевский тип

Булевские переменные – это такие переменные, которые могут хранить только два значения: *true* или *false*. Тип таких логических переменных называется *bool*.

Имя переменной типа *bool* объявляется так:

*bool testRestult;*

*bool colorIsRed = true;*

Типы с плавающей точкой

Числовые переменные, не относящиеся к целым, хранятся как числа с плавающей точкой. Число с плавающей точкой может быть выражено в виде десятичного значения наподобие 112,5 либо в экспоненциальном виде, таком как 1,125Е2, где десятичная часть умножается на 10 в степени, указанной после Е

(экспонента). Таким образом, последнее число – это 1,125х10 в степени 2, что равно 112,5.

Константы с плавающей точкой должны включать десятичную точку либо экспоненту, либо и то, и другое. Если вы записываете числовое значение без них, то получаете целое.

Пример:

*double in\_to\_mm = 25,4*

Переменная типа *double* занимает 8 байт памяти и сохраняет значения, точность которых определяется примерно 15 десятичными знаками. Диапазон их значений намного шире, чем можно выразить 15-ю знаками – начиная от 1,7х10 в степени -308 и заканчивая 1,7х10 в степени 308, положительные и отрицательные.

Если вам не требуется 15-значаня точность и не нужны очень большие диапазоны значений, которые обеспечивает тип *double*, можете воспользоваться ключевым словом *float* для объявления переменных с плавающей точкой, занимающих 4 байта.

Пример:

*float pi = 3.14159f;*

Этот оператор определяет переменную pi с начальным значением 3,13159. Символ *f* в конце константы указывает, что она имеет тип *float*. Без *f* константа имела бы тип *double*. Переменные, объявленные с типом *float*, имеют точность примерно в 7 десятичных знаков и допускают значения от 3,4х10 в степени -38 до 3,4х10 в степени 38, положительные и отрицательные.

Литералы

В С++ фиксированные значения любого рода называются литералами. Литера – это значение определенного типа, поэтому *23, 3.14159, 9.5f* и *true* – примеры литералов типа *int, double, float, bool* соответственно. Литерал *“Samuel Beckett”* - это пример строкового литерала, но обсуждение этого типа мы отложим.

Фундаментальные типы ISO/ANSI C++

В таблице представлен список всех фундаментальных типов ISO/ANSI C++, а также диапазоны их допустимых значений в Visual C++

|  |  |  |
| --- | --- | --- |
| ТИП | РАЗМЕР В БАЙТАХ | ДИАПАЗОН ЗНАЧЕНИЙ |
| bool | 1 | true или false |
| char | 1 | от -128 до +127 |
| signed char | 1 | от -128 до +127 |
| unsigned char | 1 | от 0 до 255 |
| short | 2 | от -32768 до +32767 |
| unsigned short | 2 | от 0 до 65535 |
| int | 4 | от -2147483648 до 2147483647 |
| unsigned int | 4 | от 0 до 4294967296 |
| long | 4 | от -2147483648 до 2147483647 |
| unsigned long | 4 | от 0 до 4294967296 |
| float | 4 | +-3,4х10+-38примерно с 7- значной точностью |
| double | 8 | +-1,7х10+-308примерно с 15-значной точностью |
|  |  |  |

*— Так просто? И ты не прибегал к волшебству?*

*— Единственное, к чему я прибегал, — так это к здравому смыслу.*

*В конечном итоге всегда оказывается, что он гораздо более надежен.*

**Терри Пратчетт**

Туда и обратно или

Базовые операции ввода-вывода

Чтобы ваш программный продукт обрабатывал какие-либо данные, нужно их описать, как вы уже сделали ранее, объявив переменные, и, наконец, как-то ввести их в вашу программу, чтобы манипулировать ими. Естественным, после выполнения программы, будет желание посмотреть на результат . Для этого, ввода и вывода информации существуют специальные операторы. Посмотрим , как все это происходит.

Ввод с клавиатуры

Вы получаете ввод с клавиатуры через поток *cin*, используя для этого операцию извлечения из потока >>. Чтобы прочитать два целых значения с клавиатуры, в переменные *num1* и *num2*, можно написать следующий оператор:

*cin >> num1 >> num2*;

Операция извлечения >> «указывает» направление, куда передаются данные – в данном случае, из *cin* в каждую из двух переменных по очереди. Любые ведущие пробелы пропускаются, и , первое целое значение, введенное с клавиатуры, поступает в переменную *num1*. Между следующими друг за другом значениями должны быть какие-нибудь пробельные символы, чтобы их можно было разделить. Операция потокового ввода завершается, когда вы нажимаете клавишу <*Enter*>, и выполнение программы продолжается со следующего оператора.

Потоковый ввод и его операции автоматически распознают переменные и данные любого фундаментального типа.

Пример:

*int num1 = 0, num2 = 0;*

*double factor = 0.0;*

*cin >> num1 >> factor >> num2;*

Вывод в командную строку

Поток вывода называется *cout*, и для передачи данных в него используется операция вставки <<. Эта операция также «указывает» направление движения данных. Вы уже применяли ее для вывода текстовой строки, заключенной в кавычки.

// Листинг 2

// Упражнение по выводу

# include <iostream.h>

int main()

{

int num1 = 1234, num2 = 5678;

cout <<endl; // начать новую строку

cout << num<< num2; //вывести две переменные

cout << endl; //завершить строку

return 0; //завершить программу

}

Описание полученных результатов

Первый оператор в теле *main()* объявляет и инициализирует две целочисленных переменных – *num1* и *num2*.

Когда вы скомпилируете и запустите приведенный выше код, то получите на экране следующий вывод:

12345678

Форматирование вывода

Проблему, связанную с отсутствием пробелов между значениями, можно исправить довольно просто – вставив пробел в поток вывода между двумя значениями.

cout << num1 << num3; // Вывести два значения

Просто подставьте вместо него оператор:

cout << num1 << ‘ ‘ << num2; // Вывести два значения

Конечно, если у вас несколько строк вывода, и вы хотите выровнять колонки, то вам понадобятся какие-то дополнительные возможности, поскольку вы не знаете, сколько знаков будет в каждом значении. С этой ситуацией можно справиться, используя то, что называется манипуляторами.

Манипулятор модифицирует способ управления выводом данных в поток (или вводом из потока).

Манипуляторы определены в заголовочном файле <*iomanip.h*>, поэтому вам понадобится для него добавить директиву *#include*.

Манипулятор, который вам необходим *setw(n)*. Он выводит значение, следующее за ним, выравнивая его в поле пробелов шириной *n*, т.е *setw(6)* представит следующее за ним значение в поле шириной 6 пробелов.

Пример.

// Листинг 3

// Упражнение по выводу

# include <iostream.h>

# include <iomanip.h>

int main()

{

int num1 = 1234, num2 = 5678;

cout <<endl; // начать новую строку

//вывести два значения

cout << setw(6) << num1< setw(6) <<< num2;

cout << endl; //завершить строку

return 0; //завершить программу

}

Описание полученных результатов.

Манипулятор *setw()* работает только с единственным выходным значением, которое следует непосредственно за его вставкой в поток.

Вы должны вставлять манипуляторы непосредственно, перед каждым значением, которое хотите выровнять в пределах поля определенной ширины. Если вы вставите только один *setw()*, он воздействует лишь на первое значение, отправленное в выходной поток вслед за ним.

Перечень манипуляторов:

*endl* - уже знакомы, перевод строки,

*ends* - конец строки (нулевой байт),

*flush* - опустошить и вывести все промежуточные буферы,

*dec* - выводить числа в десятичном виде (по умолчанию),

*hex* - выводить числа в шестнадцатеричном виде,

*oct* - выводить числа в восьмеричном виде,

*setbase(n)* - установить систему счисления для вывода чисел (n=0,2,8,10,16; 0 и 10 равнозначны),

*setw(n)* - установить ширину поля вывода,

*setfill(n)* - установить символ-заполнитель до необходимой ширины поля вывода,

*setprecision(n)* - установить точность вывода дробей (количество знаков после запятой).

Пример

// Листинг 4

// Упражнение по выводу

#include <iostream.h>

#include <iomanip.h>

double x;

void main()

{

double x;

cin >> x;

cout<<setw(10)<<setfill('s')<<setprecision(3)<<x/133

<<endl;

}

Описание полученных результатов.

С манипуляторами смысл таков: в поле вывода шириной 10 символов вывести результат деления переменной x на 133 с тремя знаками после запятой. Оставшиеся промежутки забить символами "s", затем сделать перевод строки. Как видите, такая мешанина из манипуляторов нормально работает

Управляющие последовательности

Когда вы пишите символьную строку, заключенную в двойные кавычки, то можно включить в нее специальные символы, называемые управляющими последовательностями. Они позволяют поместить в строку символы, которые не могут быть представлены иным образом, за счет того, что они избегают обычного процесса интерпретации символов.

Управляющая последовательность начинается с символа обратного слеша \ , который заставляет компилятор интерпретировать следующий за ним символ особым образом.

Пример

*cout << endl<< “\tOutput after the tabulation”;*

*cout << “\n\ tOutput after the tabulation”;*

Комбинация *\t* сдвигает следующий за ней текст первую позицию табуляции.

*\n* – управляющая последовательность символа новой строки.

Некоторые управляющие последовательности.

|  |  |
| --- | --- |
| **Управляющая последовательность** | **Что делает** |
| \a | Выдает звуковой сигнал |
| \n | Символ новой строки |
| \’ | Одиночная кавычка |
| \\ | Обратный слеш |
| \b | Забой |
| \t | Символ табуляции |
| \” | Двойная кавычка |
| \? | Знак вопроса |

Если необходимо включить обратный слеш или двойную кавычку в строку , необходимо использовать соответствующую управляющую последовательность, чтобы представить их.

Можно применять символы специфицированные управляющими последовательностями в инициализации переменных типа char

Пример

*char Tab = ‘\t’; //инициализация символом табуляции*

Использование управляющих последовательностей

Пример.

// Листинг 5

// Применение управляющих последовательностей

# include <iostream.h>

# include <iomanip.h>

int main()

{

char newline = ‘\n’; //Управл последов –символ новой строки

cout << newline; //начать новую строку

cout << “\”I\’m a student\”, he said”;

cout << “\n\t The program\’s over, it\’s time take make a beep beep\a\a”;

cout << newline;

return 0;

}

После компиляции получим следующий вывод

”I’m a student”, he said

The program’s over, it’s time take make a beep beep

Описание полученных результатов.

Первая строка определяет переменную *newline* и инициализирует ее символом новой строки, в виде управляющей последовательности. Можно применить *newline* вместо *endl* из стандартной библиотеки.

Далее следует вывод строки, которая применяет управляющие последовательности для представления символов двойной и одинарной кавычек.

Следующая строка начинается с управляющей последовательности – символа новой строки, за которой идет управляющая последовательность символа табуляции. Строка также заканчивается двумя экземплярами управляющей последовательности, выдающей звуковой сигнал.