1 Введение в Visual Studio

Мы будем изучать программирование на языке C++ в среде Visual Studio 2010.

Для начала запустим Visual Studio 2010. Обычно при первом запуске она спрашивает, на каком языке мы будем программировать. Нужно выбрать Visual C++. Затем Visual Studio будет производить первоначальную настройку; здесь придется подождать (как правило, несколько минут — не удивляйтесь).

Далее, для написания программы на C++ нужно создать проект. Для этого достаточно щелкнуть мышью на соответствующей ссылке на стартовой странице («New Project...»). В появившемся окне нужно выбрать тип проекта (Win32 Console Application), а также задать его имя и расположение (в какой хотите папке, но на Вашем сетевом диске W:), после чего нажать «OK».

Наконец, нужно определить свойства проекта. Для этого в появившемся окне нужно нажать «Next» и в появившемся после этого окне сбросить флаг «Precompiled header», установить флаг «Empty project» и нажать «Finish».

Если все было сделано правильно, создастся пустой (не содержащий файлов) проект и откроется в обозревателе проектов (Solution Explorer, при обычных настройках — левая верхняя часть окна). Для добавления в проект файлов, содержащих текст программы, нужно щелкнуть правой кнопкой мыши по заголовку проекта (имя проекта жирным шрифтом на голубом фоне) и в появившемся контекстном меню выбрать пункт Add, затем New item... и в появившемся окне задать тип добавляемого файла (C++ File(.cpp), файл с программой на C++) и его имя. Имя не должно содержать точку — тогда расширение .cpp будет добавлено автоматически. Если же в имени нужно использовать точку, то расширение .cpp надо добавить явно. После этого нужно щелкнуть на кнопке Add, и создается указанный файл, добавляется в проект и открывается в редакторе. Теперь можно набирать программу.

Для запуска программы нужно воспользоваться меню «Debug», пункт «Start without debugging» и в появившемся окне нажать «Yes». Программа компилируется, компонуется, и если не было ошибок, запускается, для чего открывается новое текстовое окно терминала (как для ввода команд в командной строке). После завершения работы программы выдается сообщение «Press any key to continue...», и после нажатия любой клавиши окно терминала закрывается.

Важное замечание состоит в том, что если программа запущена, то перекомпилировать ее невозможно — нужно сначала дождаться завершения работы программы или принудительно закрыть ее, и только потом компилировать новую версию.

Также, для каждой программы нужно создавать свой проект. Для этого после завершения работы над программой нужно закрыть проект, выбрав пункт «Close solution» меню «File», и затем создать для следующей программы новый проект, щелкнув на первой кнопке из панели инструментов (с подсказкой New Project (Ctrl+Shift+N)).

2 Общая структура программы

Любая программа на C++ состоит из нескольких текстовых файлов. По смыслу эти файлы делятся на две категории: так называемые заголовочные файлы и файлы с исходными текстами. Первые обычно имеют расширения «.h», «.hh», «.hpp», «.H» (последнее актуально на операционных системах, на которых большие и маленькие буквы в именах файлов различаются — на Windows это не так). Файлы с исходными текстами обычно имеют расширения «.cc», «.cpp», «.cxx», «.C». Впоследствии мы рассмотрим разделение всего текста программы на несколько файлов более подробно, а сейчас в качестве примера возьмем простейшую программу на C++, состоящую из всего одного текстового файла:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    cout<<"Hello, world!"<<endl;
    return EXIT_SUCCESS;
}
```

Разберем эту программу более подробно. Первые две строки — это так называемые директивы препроцессора. Препроцессор — это небольшая программа, которая обрабатывает текст всякой программы на C или C++ непосредственно перед компиляцией. Препроцессор позволяет включать в программу другие файлы, определять и использовать макроопределения, включать в программу или пропускать определенные ее фрагменты (так называемая условная компиляция). Данные директивы требуют включить в программу файлы iostream и cstdlib. Это означает, что перед компиляцией из нашего текстового файла будет построен новый файл, в котором сначала будет представлено содержимое файла iostream (вместо первой директивы #include), затем — файла cstdlib (вместо второй директивы), и, наконец, оставшиеся шесть строк нашей программы. Именно этот новый файл и будет скомпилирован для получения окончательной программы на машинном языке, которую можно запустить. Включаемые файлы содержат информацию, необходимую для правильной компиляции нашей программы. Какую именно — узнаем несколько позже. Далее, очевидно, что мы этих файлов не писали — они предоставляются нам разработчиками системы программирования на языке C++ — в нашем случае Visual Studio.

Следующая строка требует использовать пространство имен std. Что такое пространство имен и зачем его использовать — узнаем опять же несколько позже. Пока примем необходимость данной строки как аксиому и продолжим рассмотрение программы.

Наконец, оставшаяся часть программы — это определение функции main. В C++ функция — это поименованная часть программы, на которую можно сослаться, указав ее имя. Это то же самое, что в Pascal функция или процедура (в

C++ для обоих понятий используется один и тот же термин). В других языках программирования встречается также термин «подпрограмма». В каждой программе должна быть ровно одна функция main, поскольку запуск программы как раз и состоит в выполнении функции main.

Ключевое слово int, стоящее перед именем функции, означает тип целых чисел (аналог integer в Pascal). Любая программа работает с информацией разного рода — числами, строками, более сложными объектами. Тип объекта показывает, каким образом информация, относящаяся к объекту, хранится в памяти компьютера, и какие операции над этими объектами можно выполнять. Например, целое число 1 и вещественное число 1.0 математически представляют собой одно и то же, но в памяти компьютера хранятся совершенно по-разному, потому что первое считается относящимся к целому типу (int), а второе — к вещественному (double). Они различаются компилятором при помощи следующего нехитрого правила: если в записи числа нет символа «.», число считается целым, а если есть (даже при том, что дробная часть может быть равна нулю) — вещественным.

Тип, стоящий перед именем функции, указывает тип результата данной функции. Некоторые функции могут вырабатывать в процессе своей деятельности определенный результат. Например, функция sin, вычисляющая синус, вырабатывает вещественное число, так что тип ее результата будет double. Функция main всегда должна вырабатывать целое число; в некоторых учебниках до сих пор иногда встречается конструкция void main, что указывает на отсутствие у функции main результата, но это противоречит стандартам как C, так и C++. По этим стандартам, поведение такой функции считается неопределенным, т. е. она вполне имеет право делать все, что угодно, в том числе отформатировать Вам жесткий диск.

Пустые скобки после имени функции указывают на отсутствие у данной функции параметров. Параметры — это некоторая дополнительная информация, необходимая для работы функции. Например, у все той же функции sin должен быть один вещественный параметр, содержащий значение того угла, синус которого она будет вычислять. В отличие от Pascal, даже если у функции нет параметров, эти скобки (пустые!) нельзя выкинуть. Правила о постановке символа «;» после заголовка функции в C++ сложнее, чем в Pascal. Этот символ ставится только в том случае, если мы опускаем тело функции; если же тело функции присутствует (как в нашей программе), то точку с запятой после заголовка ставить нельзя.

Фигурные скобки в C++ ограничивают тело функции, т. е. собственно тот фрагмент программы, которому назначается имя при определении функции. Фигурные скобки в данном случае нужны даже тогда, когда тело функции состоит из всего одного оператора (оператор — единица действия в императивных языках программирования, т. е. таких, как Fortran, Pascal, C, C++, Ada, Java и т. п.).

В нашем случае тело функции main состоит из двух операторов. Первый из них — оператор, сделанный из выражения, смысл его состоит в вычислении этого выражения, а синтаксис — выражение и затем точка с запятой. В отличие от Pascal, точка с запятой не разделяет операторы, а завершает оператор, сделанный из выражения, и ряд других разновидностей операторов. В частности, в C++ допустим пустой оператор, состоящий только из точки с запятой (оператор, сделанный из пустого выражения, который ничего не делает; зачем это бывает нужно — обсудим чуть позже).

Данное выражение состоит из двух операций вывода (<<), примененных к следующим операндам:

cout — поток вывода, связанный с экраном. Вывод в этот поток означает, как правило, вывод на экран.

"Hello, world!" — строковая константа. В отличие от Pascal, в C++ различаются строковые и символьные (тип которых — один символ, char, как в Pascal) константы.

endl — так называемый манипулятор вывода. Вывод его в поток означает перевод строки и сброс буферов, т. е. запись информации из промежуточных буферов в файл или на экран, в соответствии с тем, с каким объектом связан данный поток.

Здесь продемонстрирован прием программирования, называемый сцеплением. Дело в том, что операция вывода в поток возвращает сам поток, который можно затем использовать в дальнейших операциях вывода. При этом окончательный результат последней операции вывода (все тот же поток) в данном случае просто игнорируется. В отличие от Pascal, в C++ можно игнорировать результат выражения.

Наконец, оператор return возвращает результат функции main. В отличие от Pascal, основная функция программы должна возвращать целое число в качестве результата. Это число используется в операционной системе для указания того, успешно ли завершилась программа или в процессе работы возникли ошибки.

Для этого в файле cstdlib определены две константы — EXIT_SUCCESS и EXIT_FAILURE. Смысл этих констант ясен из их названия. Их использование переносимо, т. е. должно работать везде, где система программирования на C/C++ соответствует стандарту. Вообще говоря, программа может возвращать и другие значения, по которым можно судить, какие именно ошибки имели место, но это, как правило, непереносимо, т. е. привязывает программу к определенной операционной системе.

Здесь можно сделать еще одно замечание. В отличие от Pascal, языки C и C++ чувствительны к регистру букв, т. е. различают большие и маленькие буквы. Так что если написать return exit_success; — такое работать не будет.

Задачи.

1. Напишите программу, выводящую на экран строку «Here was I».

2. Напишите программу, выводящую на экран две строки «Line one» и «Line two» на разных строках, используя имя потока cout только один раз.

Скажем здесь пару слов и о строковых константах в C++. Мы уже видели обычную разновидность строковых констант, доставшуюся языку C++ в наследство от C. Она позволяет представлять в программе любые строки (ну хорошо, почти любые), однако иногда это бывает весьма неудобно. Например, чтобы представить в такой строке символ двойной кавычки, нужно предварить его символом обратной косой черты; соответственно, чтобы представить сам символ обратной косой черты, нужно написать его дважды. Иногда, в особенности для представления регулярных

выражений, где требуется обычно много раз писать символ обратной косой черты, это приводит к очень громоздкой и нечитаемой записи. Чтобы ее упростить, новый стандарт C++ предлагает новый способ записи строковых констант, который называется «необработанные строковые константы» (raw string literals). Суть его в том, что записи начала и конца строковой константы усложняются, но зато все символы внутри строки пишутся как есть, без необходимости предварять некоторые из них символом обратной косой черты. Синтаксис этого способа записи строковых констант таков:

```
R"ограничитель(строка)ограничитель"
```

Здесь строка — текст строковой константы, а ограничитель — некоторая специальная строка (почти любая, в том числе может быть пустой).

Завершим этот раздел, сказав пару слов о таком явлении в языках программирования, как комментарии. Это некоторая дополнительная информация, предназначенная для людей, читающих программу. Компилятор, преобразующий программу в машинно-исполняемый вид, обычно их просто игнорирует. В C++ бывают комментарии двух видов: комментарии, начинающиеся с текста //, заканчиваются концом текущей строки. Если же комментарии начинаются с текста /*, то они заканчиваются текстом */. При этом комментарии одного и того же типа не могут быть вложенными один в другой (разных типов — могут).

Иногда в языках программирования встречаются и такие комментарии, которые влияют на компиляцию: например, в языке Turbo Pascal при помощи такого рода комментариев можно устанавливать различные режимы компиляции и другие опции компилятора. Например, при помощи определенного вида комментариев можно управлять тем, проверяется ли переполнение чисел при арифметических вычислениях. В стандартном C++ такого рода комментарии не приняты, однако различные расширения языка вполне могут их использовать.

Наконец, в последнее время широкое распространение получили так называемые документирующие комментарии. Их смысл в том, что существуют специальные программы, извлекающие такие комментарии из текстов программ и порождающие по ним документацию к программам автоматически. В качестве примера для C++ можно назвать программу doxygen.

3 Выражения и переменные

В программах на C++ можно вычислять выражения. Синтаксис выражений, т. е. способ их записи, похож на принятый в математике. В выражениях можно использовать целые числа (как в математике), а также вещественные числа в обычном виде, как десятичные дроби, только вместо десятичной запятой используется точка. Например, запись 2.731 означает «2 целых 731 тысячная». Можно также использовать вещественные числа в так называемом экспоненциальном или научном формате, когда к числу сзади дописывается буква «e» и за ней целое число (вся эта запись не должна содержать пробелы), которое означает, что исходное число надо умножить на 10 в указанной степени. Например, число 1.23e2 означает просто 123, а 1e-6 означает одну миллионную.

В большинстве языков программирования, и C++ не является исключением, нельзя использовать в выражениях слишком большие целые или вещественные числа. Более того, диапазон используемых чисел зависит от компьютера, операционной системы и компилятора. В пределах этого диапазона, т. е. если исходные числа и все промежуточные результаты попадают в этот диапазон, арифметические действия с целыми числами выполняются точно. Если же исходные числа или промежуточные результаты выходят за пределы этого диапазона, результат будет неверным, причем система не сообщает об этом программисту. Для вещественных чисел используется специальное значение, означающее «бесконечно большое число».

Даже если все исходные числа и промежуточные результаты в выражении с вещественными числами лежат в допустимых пределах, все равно вычисления с вещественными числами выполняются лишь приближенно. Это происходит по ряду причин. Во-первых, даже такое, казалось бы, нехитрое вещественное число, как одна десятая, не может быть представлено в компьютере точно, поскольку в памяти компьютера вещественные числа представляются конечными двоичными дробями, а одна десятая так представлена быть не может (чтобы получить одну десятую, нужно поделить не только на 2, но и на 5, а этого уже конечные двоичные дроби не умеют). Далее, число двоичных цифр в этих дробях ограничено, поэтому неизбежно необходимо округление результатов (например, при умножении точный результат должен содержать примерно в два раза больше цифр, чем каждый из операндов).

В выражениях также можно использовать обычные арифметические операции, такие как сложение, вычитание, умножение и деление. Для умножения правила несколько отличаются от математических: в качестве значка для умножения используется звездочка, и пропускать его нельзя. Деление имеет одну особенность: если оно используется для целых чисел, т. е. оба его операнда имеют целый тип, результат будет целым (дробная часть отбрасывается), так что выражение 1/3 имеет достаточно странный результат 0. Если нам нужно получить вещественный результат, нужно явно преобразовать один из операндов к вещественному типу, например так: 1.0/3 .

Кроме того, для целых чисел имеется дополнительная операция % (остаток от деления), которая дает целый результат. Также, разумеется, можно использовать скобки, но только круглые (смысл квадратных скобок будет ясен несколько позже).

Как и в большинстве других языков программирования, в C++ имеются переменные. В C++ переменная — это поименованная область памяти, в которой можно хранить промежуточные результаты вычислений или введенные в программу извне данные.

Каждая переменная имеет тип, имя и значение. Тип переменной определяет способ трактовки той информации, которая содержится в переменной. Например, одно и то же содержимое памяти, если рассматривать его как целое

число, имеет совсем другое значение по сравнению с тем, как если бы его рассматривать как вещественное число (целые и вещественные числа записываются в память совершенно по-разному).

C++ имеет три типа для вещественных чисел (в порядке увеличения диапазона значений и точности): float, double и long double (да, именно так — с пробелом!). Целых типов очень много: char, short int, int, long int, иногда (не везде он есть, но по новому стандарту должен быть) long long int. Кроме того, к любому целому типу можно добавить ключевое слово unsigned — оно означает, что в переменной такого типа хранятся только неотрицательные числа, что позволяет сдвинуть диапазон значений соответствующего типа так, чтобы можно было представить большие числа (обычно диапазон примерно от -x до x, где x зависит от типа; для unsigned диапазон будет от 0 до 2x).

Логический тип называется bool. Тип для хранения символов — char (да, этот тип считается целочисленным и переменные такого типа могут участвовать в арифметических выражениях). Существуют в C++ и строки (тип string), однако они не входят в сам язык, а реализуются на базовом языке как часть стандартной библиотеки, поэтому, чтобы ими пользоваться, нужно включить в свою программу файл string при помощи директивы #include.

Именем переменной может быть любой идентификатор (т. е. последовательность букв и цифр, начинающаяся с буквы, без пробелов), не совпадающая ни с одним ключевым словом. В C++ большие и маленькие буквы различаются.

Всякая переменная должна быть объявлена перед использованием. Переменные объявляются следующим образом:

```
тип имя, ..., имя;
```

(точка с запятой в конце декларации обязательна).

Можно также совместить объявление переменной с инициализацией, т. е. присваиванием ей начального значения. Например, следующая декларация объявляет целую переменную i и инициализирует ее начальным значением 0:

```
int i = 0;
```

Если в одной декларации объявляется несколько переменных, можно инициализировать некоторые из них, но для инициализации нельзя использовать сцепление, т. е. каждую переменную нужно инициализировать отдельно.

Также, новый стандарт C++ предоставляет программисту любопытную возможность определять переменные, указывая в качестве типа ключевое слово auto. Оно означает, что тип переменной должен определяться тем выражением, которое используется для ее инициализации, т. е. как ее начальное значение. В Visual Studio 2010 реализовано не очень много языковых средств C++, предложенных новым стандартом, но это там есть.

Используя вышеуказанную возможность определять тип переменной по начальному значению, предыдущий пример можно переписать так:

```
auto i = 0;
```

Понятно, что в таких простых случаях использование этой возможности выглядит достаточно глупо, однако, встречаются ситуации, когда тип переменной определяется очень сложным и длинным выражением, и тогда такая возможность очень выручает. Бывает даже так, что тип переменной непросто определить вручную — больше всего таких ситуаций возникает при так называемом обобщенном программировании.

Наконец, по новому стандарту имеется возможность определить переменную с тем же типом, который является результатом некоторого выражения. Для этого используется конструкция decltype(выражение). Полностью это может выглядеть, например, так:

```
int i = 0;
decltype(i) j;
```

Все переменные, в зависимости от расположения их объявления, делятся на локальные и глобальные. Переменные, объявление которых расположено в какой-либо функции, называются локальными. Ими можно пользоваться только в той функции, в которой они объявлены. Более того, они существуют только пока эта функция выполняется. Когда функция вызывается, эти переменные создаются, и когда она завершает свою работу, они исчезают. При повторном вызове функции они создаются заново, причем скорее всего в другом месте памяти, так что значения локальных переменных не сохраняются от одного вызова функции до другого.

Глобальные переменные объявляются также, как и локальные, но не в теле какой-либо функции, а прямо в файле снаружи от тела любой функции. Их можно использовать в любой функции, и они существуют втечение всего времени работы программы.

В C++ можно объявлять синонимы типов, например, для краткости. Для этого используется ключевое слово typedef. Оно пишется перед декларацией и превращает ее из определения переменной в определение нового синонима типа этой переменной, причем имя декларируемой переменной превращается в имя указанного синонима.

Например, переменную uil типа unsigned long int мы могли бы определить так:

```
unsigned long int uil;
```

Если же мы напишем перед декларацией ключевое слово typedef, декларация превратится в определение нового типа uil, который является синонимом unsigned long int:

```
typedef unsigned long int uil;
```

После этого, типом uil можно пользоваться для определения новых переменных:

```
uil x;
```

Здесь новый тип будет синонимом старого, так что если одна переменная объявлена с новым типом, а другая — со старым, мы вполне можем присвоить одной из этих переменных значение другой.

В C++ можно объявлять константы. Для этого в объявлении перед типом ставится ключевое слово const. Это слово относится ко всем именам, объявляемым в такой декларации, т. е. если в этой декларации объявлялось 3 переменные, добавление const перед типом превратит их всех в константы. Также, константы должны быть инициализированы при объявлении, иначе компилятор выдаст ошибку (что логично, поскольку присвоить константе что-нибудь нельзя — это одно из отличий между инициализацией и присваиванием, хотя и для того, и для другого можно использовать один и тот же символ «=»).

Наряду с классическим модификатором const, превращающим определение переменных в определение констант, новый стандарт предлагает еще один на ту же тему, constexpr. Этот модификатор означает не только const, но что значение данной константы определяется константным выражением, т. е. может быть определено в процессе компиляции. Это существенно для констант, определяемых внутри функций. Увы, это, насколько я понимаю, в Visual Studio 2010 не реализовано, так что здесь мы смотрели вперед с надеждой на Visual Studio 2012 — тоже, увы, не реализовано.

После объявления переменной ей можно присвоить некоторое значение, т. е. записать некоторое значение в переменную. Для этого используется операция «=», левым операндом которой служит имя переменной, значение которой меняется, а правым — выражение, значение которого записывается в переменную. Для использования значения переменной в выражении нужно просто указать ее имя. Более того, та переменная, которой мы присваиваем значение, вполне может входить в выражение. Там она будет означать ее значение до присваивания. Например, следующий фрагмент будет увеличивать значение переменной i на 1:

```
i = i+1;
```

Считать значение переменной с клавиатуры можно при помощи операции «взять из потока». Эта операция имеет то же обозначение, что и операция побитового сдвига вправо (>>). Все вместе будет выглядеть примерно так:

```
cin>>i;
```

Если нужно считать несколько переменных, вполне возможно использовать сцепление. При вводе можно разделять вводимые числа пробелами.

Пример. Написать программу, вводящую вещественное число и печатающую его квадрат.

Решение:

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    double x;
    cout<<"Input a number x="; cin>>x;
    cout<<"x squared="<<x*x<<endl;
    return EXIT_SUCCESS;
}
```

Задачи.

1. Написать программу, вводящую коэффициенты квадратного уравнения и печатающую его дискпиминант.

2. Написать программу, вводящую длину радиуса окружности и печатающую ее длину и площадь ограничиваемого ею круга.

3. Направления кодируются следующим образом: 0 — север, 1 — запад, 2 — юг и 3 — восток. Ввести направление и вывести результат его поворота на 90° а) против часовой стрелки; б) по часовой стрелке.

4*. Грузовик может увезти 32 коробки. Ввести число коробок, которые необходимо увезти и вывести минимальное достаточное для этого число грузовиков.

5. Вычислить 20!. Объяснить результат программы.

Для вещественных чисел также можно использовать элементарные функции, названия некоторых из них отличаются от принятых в математике:

Функция	Запись	
	математика	язык C++
синус	$\sin x$	sin(x)
косинус	$\cos x$	cos(x)
тангенс	$\mathrm{tg}\, x$	tan(x)
арксинус	$\arcsin x$	asin(x)
арккосинус	$\arccos x$	acos(x)
арктангенс	$\mathrm{arctg}\, x$	atan(x)
гиперболический синус	$\mathrm{sh}\, x$	sinh(x)
гиперболический косинус	$\mathrm{ch}\, x$	cosh(x)
гиперболический тангенс	$\mathrm{th}\, x$	tanh(x)
гиперболический арксинус	$\mathrm{arcsh}\, x$	asinh(x)
гиперболический арккосинус	$\mathrm{arcch}\, x$	acosh(x)
гиперболический арктангенс	$\mathrm{arcth}\, x$	atanh(x)
экспонента	e^x	exp(x)
натуральный логарифм	$\ln x$	log(x)
степень	x^y	pow(x,y)
квадратный корень	$\sqrt{x}$	sqrt(x)
корень	$\sqrt[y]{x}$	pow(x,1.0/y)

Для использования этих функций необходимо включить файл cmath при помощи директивы #include (т. е. в начале программы нужно написать #include <cmath>).

Примеры.

1. Написать программу, вводящую длины двух сторон треугольника и величину угла между ними и печатающую длину оставшейся стороны.

Решение:

```
#include <iostream>
#include <cstdlib>
#include <cmath>

using namespace std;

int main()
{
    double a,b,gamma;
    cout<<"Input the lengths: "; cin>>a>>b;
    cout<<"Input the angle: "; cin>>gamma;
    cout<<"The length of the remaining side="
        <<sqrt(a*a+b*b-2*a*b*cos(gamma))<<endl;
    return EXIT_SUCCESS;
}
```

2. Написать программу, которая вычисляет выражение

$$\frac{\sin 5 + 1.75^2}{3e^{\cos 7}}$$

и выводит его результат на экран.

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
int main()
{
    cout<<(sin(5.0)+pow(1.75,2.0))/(3*exp(cos(7.0)))<<endl;
    return EXIT_SUCCESS;
}
```

Обращаем Ваше внимание на то, что вместо sin(5) нужно писать sin(5.0) (попробуйте первый вариант и посмотрите, что получится). Дело в том, что в C++ имеются три разных вещественных типа (float, double и long double), и компилятор не знает, к какому из них нужно преобразовать целое число 5. Что же касается явно указанных вещественных констант вроде 5.0, они по умолчанию относятся к типу double (самому часто используемому), и никаких неоднозначностей здесь не возникает.

Иногда бывает удобно вывести подсказку, описывающую выводимое число. Это можно сделать, например, так:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
int main()
{
    cout<<"Result="<<(sin(5.0)+pow(1.75,2.0))/(3*exp(cos(7.0)))<<endl;
    return EXIT_SUCCESS;
}
```

Иногда также бывает удобно, особенно при выводе таблиц, задавать ширину поля (т. е. места на экране, куда будет выведено число). Если выводимое число занимает меньше места, оно будет дополнено пробелами спереди. Это делается так:

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
using namespace std;
int main()
{
    cout<<"Result="<<setw(15)
        <<(sin(5.0)+pow(1.75,2.0))/(3*exp(cos(7.0)))<<endl;
    return EXIT_SUCCESS;
}
```

В данном случае ширина поля равна 15 символам. Здесь использован манипулятор setw с параметром, задающим ширину поля. Для использования его и других манипуляторов с параметрами нужно включить в программу файл iomanip.

Для вещественных чисел, кроме того, удобно задавать точность, с которой выводится результат. Для этого можно воспользоваться манипулятором setprecision, тоже с параметром, задающим требуемое число значащих цифр. Например, если в предыдущем примере нам требуется 10 значащих цифр, нужно написать

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
using namespace std;
int main()
{
    cout<<"Result="<<setw(15)<<setprecision(10)
        <<(sin(5.0)+pow(1.75,2.0))/(3*exp(cos(7.0)))<<endl;
    return EXIT_SUCCESS;
}
```

При этом важно помнить, что setw действует только на один следующий за ним вывод (т. е. на второе выводимое число его действие уже не распространяется), тогда как setprecision действует до следующего его применения, пока явно не будет установлена другая необходимая точность.

Задачи.

1. Вычислите в вещественных числах значение выражения

$$\frac{\sqrt[3]{\sin^2(1.28)+1}-26+\operatorname{arctg}(1.17+2.95)}{(2.01-\cos^2 3.86)^{5.84}+25.362}.$$

Выведите его значение на экран с подсказкой, шириной поля 20 и 15 значащими цифрами.

2. Ввести длины сторон треугольника и вывести величину угла, противолежащего той стороне, длина которой была введена первой.

3. Ввести длины сторон треугольника и вывести его площадь.

4. Ввести начальную сумму вклада и вывести число, через сколько лет сумма вклада превысит миллион (ежегодно начисляется 3%), без циклов.

5*. Ввести целое положительное число и вывести число цифр в его десятичной записи (без циклов).

6. Ввести сумму и произведение двух вещественных чисел и вывести исходные числа в порядке возрастания.

7. Ввести сумму и сумму квадратов двух вещественных чисел и вывести исходные числа в порядке возрастания.

Кроме того, в C++ имеются дополнительно операции инкремента и декремента, для которых используются обозначения ++ и −− соотвественно. Эти операции всегда одноместны, но могут стоять как до операнда (префиксный вариант), так и после (постфиксный вариант). Раличие этих вариантов в том, какое значение (после или до преобразования) возвращается. Например, если i — целая переменная со значением 0, то после вычисления любого из

выражений ++i и i++ ее значение будет 1. Однако первое из этих выражений вернет значение 1 (увеличение на 1 происходит до извлечения значения из переменной), а второе — 0 (увеличение после извлечения).

Скажем здесь также и про операцию преобразования типов. Синтаксис этой операции таков: (тип)выражение. Эта конструкция вычисляет выражение, после чего пытается преобразовать его тип к тому, который указан в скобках, по возможности сохранив значение, т. е., например, целое число 5 превращается в вещественное число 5.0 (внутреннее их представление в памяти компьютера сильно различается). Эта операция позволяет выполнять и обратное преобразование, превращая вещественное число в целое путем отбрасывания того, что стоит после десятичной точки.

Одной из особенностей C и C++ является то обстоятельство, что тип переменных для хранения одиночных символов (он называется char) является целочисленным (соответствующая переменная на самом деле хранит код символа, т. е. его номер в специальной таблице символов), и потому символьные переменные вполне могут участвовать в арифметических выражениях, а также имеется возможность превращать целые числа в те символы, которые имеют соответствующий номер.

Пример. Ввести целое число от 1 до 26 и вывести большую латинскую букву, имеющую данный номер в алфавите.

Решение:

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    int n;
    cout<<"Input the number: "; cin>>n;
    cout<<"Numbered letter = "<<(char)('A'-1+n)<<endl;
    return EXIT_SUCCESS;
}
```

Задача. Ввести оценку по 99-балльной шкале (от 0 до 99) и вывести разряд, соответствующий этой оценке (оценкам от 90 до 99 соответствует разряд 'A', от 80 до 89 — 'B', от 70 до 79 — 'C' и т. д.).

Операций сравнения в C++, как и в других языках, тоже шесть, и выглядят они так: < (меньше), > (больше), <= (меньше либо равно), >= (больше либо равно), == (равно) и != (не равно). Особенно нужно подчеркнуть, что = в C++ — это присваивание, а проверка на равенство — это == (источник многих ошибок в программах). Результаты этих операций строятся по тем же правилам, что и для логических операций (0 для ложного результата и 1 — для истинного). В отличие от математики, в C++, как и в большинстве языков программирования, неравенства нельзя соединять в цепочки: хотя выражение $10 > 7 > 2$ и является в C++ законным (программа успешно скомпилируется), но результатом его будет неожиданное число 0 (потому что это выражение понимается как (10>7)>2; выражение в скобках истинно, поэтому его значением будет 1, а получившееся таким образом неравенство 1>2 неверно — общий итог всего этого выражения будет ложь, т. е. 0).

Еще в C++ имеются битовые операции и логические. Их значки различаются потому, что в C++ условием может считаться любое выражение, имеющее результат (не имеет результата только выражение, состоящее из вызова функции, не возвращающей результата — в C++ такое тоже называется выражением). Трактовка такого условия следующая: ноль — ложь, все остальное — истина.

Для логических операций используются следующие обозначения: «и» — &&, «или» — ||, «не» — ! (перед операндом). Логические операции трактуют свои операнды, как уже говорилось (ноль — ложь, все остальное — истина) и всегда возвращают 0 для ложного результата и 1 — для истинного. Таблица значений этих операций (вместо 1 операнд может иметь любое ненулевое значение) дана ниже:

x	y	x && y	x || y	!y
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	1	0

Например, первая строка этой таблицы (не заголовок) означает, в частности, что 0 && 0 будет 0 (при x==0 и y==0 имеем x&&y==0). Вообще, результат операции «и» будет истина (true) только в том случае, если оба выражения, которые она соединяет, имеют значение истина. Результат логического «или» будет истина (true), если хотя бы один операнд — истина, а логического «не» — истина, если операнд имеет значение ложь (false).

Логические операции позволяют обойти ограничение, запрещающее соединять неравенства в цепочки: вместо 10>7>2 можно и нужно писать 10>7 && 7>2.

Также, при вычислении логических выражений всегда используется сокращенное вычисление, т. е. сначала вычисляется первый операнд, и если этого достаточно для определения значения всего выражения, второй операнд не вычисляется. Например, если первый операнд логического «и» имеет значение ложь, то результатом всего выражения также будет ложь, независимо от значения второго операнда (который в этом случае даже не будет вычислен). Сокращенное вычисление позволяет писать логические выражения, например, из двух частей, такие, что вторая часть вообще не имеет смысла, если значение первой достаточно для определения значения всего выражения. Самый простой

пример — выражение x != 3 && 1/(x-3)<2. В этом выражении, если первый операнд имеет значение ложь, то второй вообще не может быть вычислен. Забегая вперед, можно сказать, что такого рода условия часто встречаются при работе с массивами (первая часть условия проверяет допустимость значения индекса, а вторая проверяет содержимое элемента массива с указанным индексом).

Следующая удобная операция в C++ называется условной операцией. Синтаксис ее таков:

```
условие? выражение1 : выражение2
```

При вычислении этого выражения сначала вычисляется условие, и если оно истинно, т. е. результат отличен от 0, то вычисляется выражение1 и его результат становится результатом всего выражения. Иначе вычисляется выражение2 и его результат становится результатом всего выражения. Эта операция удобна для анализа случаев прямо внутри выражения, поскольку синтаксис C++ не разрешает использовать условный оператор (как и любой другой оператор тоже) внутри выражения.

Пример. Ввести два целых числа и вывести наименьшее из них.

Решение:

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    int a,b;
    cout<<"Input the numbers: "; cin>>a>>b;
    cout<<"Minimum="<<(a>b? b : a)<<endl;
    return EXIT_SUCCESS;
}
```

Задачи.

1. Ввести длины трех отрезков и вывести слово «да», если из них можно составить треугольник, и «нет», если нельзя.
2. Ввести три вещественных числа и вывести наименьшее из них.
3. Ввести коэффициенты квадратного уравнения — три вещественных числа, и вывести число вещественных корней этого уравнения.
4. Ввести целое число n и вывести $(-1)^n$ (без использования функции pow и циклов).
5. Ввести целое число n и вывести n-ую производную функции cos.

Битовые операции по смыслу похожи на логические, но применяются в каждой позиции двоичного разложения отдельно. Например, результат битового «и» в младшем бите будет иметь результат логического «и» младших битов аргументов, в следующем бите — результат логического «и» вторых битов с конца, и так каждый бит результата побитового «и» будет результатом логического «и» битов в той же позиции аргументов. Битовые операции используются для манипуляций отдельными битами в составе целых чисел, для экономии памяти или работы с регистрами аппаратуры. Битовые операции работают очень быстро по двум причинам: во-первых, целые числа хранятся в памяти компьютера в двоичной системе счисления, и во-вторых, битовые операции могут выполняться параллельно, в отличие от арифметических операций, где нужно в старших разрядах учитывать возможный перенос из младших.

Для побитового «и» в C++ используется двуместная операция & (а есть еще и одноместная операция с таким же значком — о ней позже), для «или» — |, для «не» — ~ (знак этой операции пишется перед операндом), для сдвигов используются обозначения << (сдвиг влево) и >> — вправо (аналоги shl и shr в Pascal). Кроме того, в C++ есть еще операция побитового «исключающего или», для чего используется значок ^ (этот значок в C++ означает совсем не возведение в степень!).

Пример. Написать выражение, дающее i-ю двоичную цифру числа n (0-я цифра — младшая).

Решение:

```
(n>>i) & 1
```

Задачи.

1. Ввести целое число n и вывести 2^n (без циклов и рекурсии).
2. Вывести двоичное представление целого положительного числа, используя битовые операции (в целом числе — 32 бита): а) с ведущими нулями; б) без них.
3. Поменять местами два бита с заданными номерами в двоичном представлении целого положительного числа.
4. Вычеркнуть i-й бит из двоичного представления целого положительного числа (младшие i-го биты остаются на месте, старшие сдвигаются на один разряд вправо).
5. Написать программу, вводящую три целых числа и выводящую новое число, каждый бит которого равен тому значению, которое встречается чаще среди битов с таким же номером у введенных чисел (при помощи побитовых операций; без циклов).

6*. Описать словами результат следующего выражения: $x \ \& \ (x-1)$.

7**. Написать фрагмент программы, по заданному числу n выдающий следующее за ним по порядку число, в двоичной записи которого столько же единиц, сколько в двоичной записи числа n.

В C++ присваивание — не оператор, а операция, и обозначается она просто = ; как и все операции, она возвращает результат (то значение, которое было присвоено). Кроме того, это одна из немногих правоассоциативных операций; это означает, что если у нас есть выражение, в котором имеются две такие операции на одном уровне, первой будет выполнена правая. Все это позволяет использовать сцепление для присваиваний одного и того же значения разным переменным. Например, выражение a = b = 0 является корректным в C++ и в результате его вычисления обе переменные a и b получат значение 0. Также язык C++ имеет набор так называемых совмещенных присваиваний. Например, выражение a += 7 эквивалентно a = a+7 с той только разницей, что адрес переменной a будет вычисляться один раз, а не два. Последнее замечание существенно в том случае, если процесс вычисления адреса переменной имеет побочные эффекты, т. е. при этом некоторые переменные меняют свои значения (такое бывает, например, при заполнении элементов массива с автоматическим переходом к следующему элементу, M[i++] = ...; здесь при вычислении адреса нужного нам элемента массива значение переменной i увеличивается на 1). Совмещенные операции присваивания имеются для всех арифметических и побитовых операций, но не для логических операций или операций сравнения.

Наконец, последняя из операций, которые мы рассмотрим здесь — операция последования; для ее обозначения используется запятая. Смысл этой операции в том, чтобы вычислить последовательно ее операнды. Значение первого операнда пропадает, значение второго становится значением всего выражения в целом. Эта операция, естественно, не распознается в вызовах функций (где, как и следовало ожидать, запятая разделяет параметры функции). Операция последования удобна там, где нужно вычислить несколько выражений в том месте программы, где по правилам должно быть выражение (а не оператор, так что составной оператор туда не поставить). Например, если перед вычислением условия в условном операторе нужно выполнить присваивание, вполне можно воспользоваться операцией последования прямо в условии.

4 Составной оператор

Во всех управляющих структурах, где нужны вложенные операторы, таких, как, например, условный оператор или операторы цикла, разрешается использование в качестве вложенного лишь одного оператора. Однако часто нужно выполнить в случае истинности некоторого условия несколько операторов, или в цикле выполнить более одного оператора. Для этих случаев в C++ предусмотрена конструкция, называющаяся «составной оператор». Она состоит из последовательности операторов, заключенной в фигурные скобки. Операторы в составном операторе не разделяются ничем (еще раз напомним, что в C++ символ «;» не разделяет операторы, как в Pascal, а завершает оператор, сделанный из выражения, и ряд других операторов). В частности, это означает, что обе «;» в следующем фрагменте необходимы:

```
{ i = 0; j = 1; }
```

Правда, используя операцию последования, этот фрагмент можно переписать эквивалентным образом и без составного оператора:

```
i = 0, j = 1;
```

В отличие от Pascal, в C++ можно объявлять переменные в любом составном операторе, а не только в теле какой-либо функции.

5 Условный оператор

Синтаксис условного оператора выглядит так:

```
if(условие) оператор1 else оператор2
```

Обратим внимание на следующие важные моменты:

1) условие всегда заключается в скобки, и условием может быть любое выражение, имеющее результат (как оно трактуется в смысле истинности или ложности, уже говорилось ранее);

2) ключевое слово then отсутствует;

3) прямо в условии допустима декларация одной переменной с инициализацией; в этом случае объявленная в условии оператора if переменная будет иметь то значение, которым она инициализирована, и это же значение будет использовано в качестве условия. Но так объявленная переменная будет видна (т. е. на нее можно сослаться) только в тех операторах, которые стоят под этим if.

Ключевое слово else и отрицательная альтернатива (стоящий после него оператор) могут отсутствовать. Также, если один условный оператор вкладывается в другой, else относится к последнему if, еще не имеющему else.

6 Оператор выбора

В C++ имеется оператор выбора, и выглядит он так:

```
switch(выражение_с_целочисленным_результатом)
{
case вариант1:
    операторы
case вариант2:
    операторы
...
default:
    операторы
}
```

Обратим внимание на следующие важные моменты:

1) в качестве вариантов могут выступать только константы (или константные выражения), значения которых известны на момент компиляции. Ни наборы значений (через «,»), ни диапазоны не допускаются;

2) после обнаружения соответствия значения выражения и какого-то из вариантов case выполняются все операторы, стоящие после соответствующего case, до тех пор, пока не встретится либо оператор «break;», либо конец тела оператора выбора (встречающиеся по дороге другие варианты case просто игнорируются).

3) вариант default может стоять где угодно (в том числе и первым, хотя он все равно будет выбран, только если нет подходящих вариантов case).

7 Операторы цикла

В старом (стандарт 1998 или 2003 годов) C++ имеются 3 разновидности операторов цикла. Первая — цикл while. Синтаксис этого цикла таков:

```
while(условие) оператор
```

Смысл этой конструкции таков: вычисляется условие, и если оно истинно, выполняется оператор, затем снова вычисляется условие, и так до тех пор, пока условие не станет ложным; что касается условия, здесь справедливо все то, что было сказано об условии оператора if. Если условие сразу ложно, оператор не выполняется ни разу. Оператор в этой конструкции называется телом цикла. Однократное выполнение тела цикла называется итерацией цикла. Количество необходимых итераций определяется условием и телом цикла.

Следующий оператор цикла — цикл, в котором условие проверяется после выполнения его тела. Синтаксис его таков:

```
do оператор while(условие);
```

В теле этого цикла (между do и while) допустим только один оператор; если тело этого цикла должно содержать больше одного оператора, нужно использовать составной оператор. Здесь тело цикла всегда выполняется по крайней мере один раз.

Наконец, последний вариант цикла в C++ старого стандарта — это цикл for. Синтаксис его таков:

```
for(начало; условие; приращение) оператор
```

Здесь начало, условие и приращение — выражения. Смысл этой конструкции следующий: сначала вычисляется начало. Затем проверяется условие, и если оно истинно, выполняется оператор (тело цикла), после чего вычисляется приращение. Затем опять проверяется условие и т. д. до тех пор, пока условие не станет ложным. В частности, если условие сразу после вычисления начала ложно, тело цикла не выполняется ни разу.

Опять же, C++ позволяет в качестве начала ставить декларацию с инициализацией. Такая переменная будет видна только в теле данного цикла for.

К этим трем разновидностям цикла новый стандарт добавляет еще одну — так называемый for для диапазонов. Синтаксис его следующий:

```
for(тип элемент: набор) оператор
```

Такая разновидность цикла for выполняет свое тело для каждого элемента указанного набора. Здесь тип — тип элемента, элемент — имя переменной, куда записывается очередной элемент набора перед выполнением тела цикла. В качестве типа элемента можно указать auto. В качестве набора может выступать обычный массив, или один из контейнеров стандартной библиотеки, или даже переменная пользовательского типа, если у нее имеются перегруженные операции для перебора элементов стандартным образом.

К сожалению, эта разновидность цикла также не реализована в Visual Studio 2010 (реализована в Visual Studio 2012). Вообще, стандартный компилятор GCC для linux (особенно последняя версия 4.8.1) реализует новый стандарт

гораздо в большем объеме, чем компилятор C++ из Visual Studio. Возможно, это объясняется тем, что в компиляторе Visual Studio реализовано много собственно Microsoftовских расширений C++, связанных с программированием графических интерфейсов пользователя.

Кроме вышеперечисленных операторов, в C++ имеются еще два оператора для управления выполнением циклов — break; и continue;. Первый из них вызывает немедленное прекращение самого внутреннего цикла, в котором находится (впрочем, если он стоит в теле switch, который вложен в этот цикл — он прерывает switch). Второй вызывает немедленное прекращение текущей итерации такого цикла и переход к следующей. Для циклов while и do ... while это означает переход к проверке условия, для for — переход к вычислению приращения.

К сожалению, эти операторы умеют прерывать только самый внутренний цикл, в котором они находятся. Для прерывания нескольких вложенных циклов одновременно нужно использовать оператор goto.

8 Оператор goto

Последний оператор, который мы рассмотрим здесь (остальные — позже) — это goto. Синтаксис его таков:

```
goto метка;
```

Метка — это идентификатор, т. е. последовательность букв, цифр и знаков подчеркивания, не начинающаяся с цифры. Метка должна быть определена в той же функции, в которой стоит оператор goto. Синтаксис определения метки:

```
метка: оператор;
```

Так пометить можно любой оператор; выполнение goto приведет к выполнению этого оператора, и далее по порядку. Единственное ограничение состоит в том, что оператор goto позволяет выходить из составного оператора, но не позволяет входить в составной оператор или внутрь другой управляющей конструкции, например, цикла.

Вообще говоря, в C++ имеется возможность переходить из некоторой функции внутрь одной из вызывавших ее функций, унаследованная из C (функции setjmp/longjump). Однако в контексте C++ этот способ небезопасен, и поэтому мы не будем его рассматривать здесь, отсылая интересующихся к документации по стандартной библиотеке языка C.

Задачи

1. Написать программу, вводящую натуральное число n и вычисляющую приближенное значение $n!$ по формуле Стирлинга

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

. Вычислить также $n!$ при помощи цикла (хранить $n!$ нужно в вещественной переменной!) и вывести точное значение, приближенное значение, абсолбютную погрешность (модуль разности, модуль вещественного числа вычисляет функция fabs) и относительную погрешность (абсолютная погрешность делится на точное значение).

2. Определить и вывести минимальное целое число n такое, что формула $e \approx \left(1+\frac{1}{n}\right)^n$ имеет относительную погрешность не более 1%.

3. Найти экспериментальным путем корни функции $\sin\sqrt{x}-\frac{1}{2}$ на отрезке [0,20] и сравнить их с точными значениями. Для вычислительного определения корней функции надо рассматривать последовательно точки на указанном отрезке, и, как только функция меняет знак, находить на отрезке между последовательными пробами корень, например, методом половинного деления.

4. Написать программу, вводящую последовательность из 15 целых чисел и выводящую ее, вставляя между каждыми двумя соседними числами, равными единице, 3 (последовательность выводится по мере ввода, без массивов).

5. Напечтать в терминальном окне окружность из символов «*» (при помощи циклов).

6. Ввести 20 символов и вывести максимальное число, содержащееся в этой строке (под числом понимается любая последовательность подряд идущих цифр).