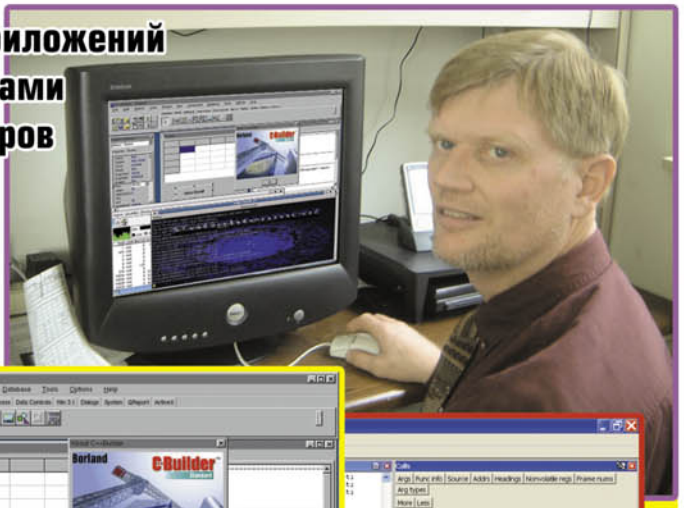
Иванов В. Б.

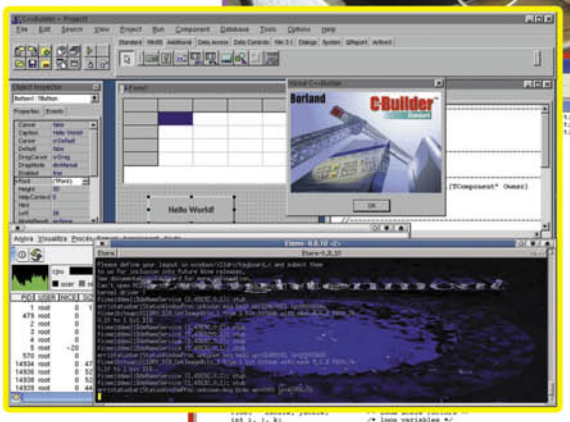
«СОЛОН»

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ НА C/C++

С НУЛЯ ДО МУЛЬТИМЕДИЙНЫХ И СЕТЕВЫХ ПРИЛОЖЕНИЙ

Создание собственных приложений
IP-телефония своими руками
Подборка учебных примеров
Пошаговое изучение





ISBN 5-98003-279-7

ПроПК

УДК 621.396.218
ББК 32.884.1
И20

В. Б. Иванов

И20 Прикладное программирование на C/C++: с нуля до мультимедийных и сетевых приложений. — М.: СОЛОН-ПРЕСС, 2011. — 240 с.: ил. — (Серия «Про ПК»).

ISBN 5-98003-279-7

Книга, которую вы держите в руках, предназначена для изучения средств программирования работы со звуком, изображением и передачей информации в компьютерных сетях с использованием языков программирования C/C++. Никаких специальных знаний для освоения материала, вообще говоря, не понадобится. Книга учит программировать «с нуля», описывает основы объектно-ориентированного программирования, достаточные для решения задач, относящихся к тематике книги. Вы узнаете, как передается информация в локальных и глобальных вычислительных сетях. Рассмотрены различные подходы к работе с виде и аудио: программирование на основе системы Win 32 API, использование готовых библиотек визуальных компонентов для Builder'a, знакомство с проектированием на базе Microsoft DirectX. Подчеркнем, что представляемые здесь «самодельные» программы являются не только учебными, но и могут с успехом использоваться в различных практических целях. В итоге, идя от менее сложному к более сложному, разобраться с проблемами, методами и реализациями задач IP-телефонии. Итогом этого будет разработка действующей системы видеотелефона для локальной компьютерной сети.

Книга сопровождается компакт-диском, содержащим проекты представляемых приложений, включая исполняемые файлы, которые сразу могут быть использованы читателем в своей работе.

ISBN 5-98003-279-7

Глава 1
ЯЗЫК ПРОГРАММИРОВАНИЯ С

Немного о TURBO C++

Эта часть книги посвящена обучению программированию на алгоритмическом языке высокого уровня. Среди многих языков программирования нами выбран язык С. Почему? К моменту написания книги в России язык С по объему использования, вероятно, уступает языку ПАСКАЛЬ, возможно, уступает языку БЭЙСИК. Тем не менее, выбор С не случаен и является нашей рекомендацией, хотя на особо принципиальном значении такого выбора настаивать большого смысла нет. Попытки провести сравнительный анализ языков высокого уровня увели бы нас слишком далеко от темы книги, хотя для интересующихся можно порекомендовать сборник [1], посвященный этим вопросам. Хотелось бы все же сказать несколько слов в пользу выбора С. Во-первых, этот язык при достаточной простоте обладает абсолютно всеми средствами, необходимыми для разработки программ любой степени сложности. Во-вторых, С является базой для освоения своего «потомка» — языка С++, который не только обладает всеми необходимыми возможностями, но и, являясь инструментом объектно-ориентированного программирования, представляет собой систему с широчайшим набором сервисных средств, обеспечивающих, в частности, эффективное программирование для среды **WINDOWS**. В-третьих, С/С++ являются «родными» языками для некоторых базовых систем. В частности, вся операционная система **LINUX** написана на С. Наконец, самым простым, но веским аргументом в пользу С/С++ является то, что в мире в целом на них написано программ существенно больше, чем на любимых в нашей стране ПАСКАЛЕ и БЭЙСИКЕ.

Итак, читателю придется либо согласиться с навязанным ему языком программирования, либо отложить эту книгу в сторону до лучших времен. На этом, к сожалению, насилие над личностью читателя не заканчивается. Ему предлагается согласиться и с рекомендацией по использованию в начале нашей работы системы, среды, в которой будет реализовываться программирование, отладка и выполнение задач. В качестве такой среды в данной части книги мы будем использовать продукт фирмы Borland International, который называется **TURBO C++**, версия 1.0. Хотя отличия между различными системами программирования, с точки зрения пользователя, несомненно, менее существенны по сравнению с различиями между самими языками, различия эти есть, и автор должен предупредить, что некоторые детали про-

грамм, созданных для среды **TURBO C++ 1.0**, могут быть «неправильно понятыми» другими системами. Выбор этой среды обусловлен компактностью пакета (в минимальном наборе можно обойтись примерно 2 мегабайтами пространства на жестком диске), несомненным удобством использования — в пакете интегрированы средства редактирования текстов, компиляции, редактирования связей и отладки. В то же время эта система уже знакома своим интерфейсом очень многим пользователям различных версий пакетов **TURBO PASCAL** и **TURBO BASIC**. Следует отметить, что близкой по характеристикам к **TURBO C++ 1.0** и вряд ли в чем-либо ей уступающей является система фирмы Microsoft, названная **QUICK C** — прародитель пакета **VISUAL C++**. Отметим, что **TURBO C++** разрабатывалась в эпоху **DOS** и ориентирована на эту операционную систему. Что касается последующих версий оболочек для программирования на C/C++ фирмы Borland, таких как **Borland C++**, то, конечно, они являются мощными профессиональными средствами, обеспечивающими широкие возможности, в частности, для программирования в среде **WINDOWS**. Изучение и использование пакета **Borland C++** составляет содержание следующих частей книги.

Как видно из самого названия пакета **TURBO C++**, в нем реализованы возможности программирования на языке C++. Язык C++ совместим снизу вверх с языком C. Тем не менее, здесь мы будем использовать пакет в режиме обработки C-программ. Связано это с тем, что обработка выполняется несколько быстрее, чем для C++-программ. Обеспечивается такой режим обработки тем, что файлы, содержащие тексты программ, снабжаются расширением .c (в отличие от расширения .cpp для C++-программ) и установкой соответствующей настройки системы.

Мы не будем подробно знакомиться с приемами работы с пакетом **TURBO C++**, поскольку детальное знакомство заняло бы слишком много времени. Для такого знакомства отошлем читателя к многочисленным пособиям, имеющимся в продаже и в библиотеках. Кроме того, пакет снабжен хорошо развитой системой помощи, которую можно вызвать на любом этапе обработки программы. Укажем только, что информацию о текущем режиме или текущей аварии, случившейся с вашей программой, можно получить нажатием функциональной клавиши **F1**. Информация о какой-либо конструкции языка может быть вызвана следующим образом. В режиме редактирования текста программы курсор ввода помещается в любое место интересующей вас конструкции (разумеется, она должна быть отображена на экране), нажимается комбинация клавиш **Ctrl** и **F1**. На экране появятся нужная информация и пример использования конструкции.

Содержание этого небольшого раздела не может считаться в полной мере достаточным для углубленного изучения языка C. В настоящее время с литературой по языку C особых проблем нет. Есть проблемы с хорошей литературой. Дело в том, что в продаже имеется множество книг, претендующих на полное описание языка, но на самом деле далеко не соответствующих этим претензиям. В качестве лучшей, на наш взгляд, книги для самостоятельного изучения C можно рекомендовать книгу [2], авторами которой являются сами разработчики языка. Попутно хотелось бы предостеречь читателей от попыток

сразу начинать с изучения C++. Личный и преподавательский опыт автора и здесь позволяет утверждать, что идти нужно от простого к сложному и не забывать вовремя остановиться и оглядеться. В то же время мы гарантируем, что все важнейшие моменты, необходимые для разработки мультмедийных и коммуникационных приложений, будут освещены в достаточной степени.

Имеет смысл отметить, что в данной главе использован далекий от традиционного подход к изложению материалов по программированию. Освоение языка здесь основывается, прежде всего, на демонстрации примеров программ, а не на формальном описании различных конструкций C. В этой связи в очередном примере программы могут встретиться новые элементы, назначение которых может быть описано где-то позднее. Читателю предлагается набраться терпения и в данный момент рассматривать такие элементы просто как «данную свыше» необходимость. Нам остается только торжественно обещать, что рано или поздно все используемые элементы программы будут пояснены.

И последняя рекомендация перед путешествием по этой главе. Прочитайте ее сначала достаточно бегло, но последовательно, не пропуская тех мест, которые вам кажутся не стоящими вашего внимания, и только после этого приступайте к детальному изучению материала.

Этапы обработки программ

Работа с программой начинается с написания ее текста, то есть создания текстового файла, имеющего произвольное имя и расширение «.c». Например, ваша первая программа вполне обоснованно может быть названа **one.c**. Программа может быть набрана в любом редакторе, генерирующем текст **DOS**, однако странно было бы отказываться от услуг встроенного в **TURBO C** редактора, который имеет все необходимые средства работы с текстом.

Когда текст программы написан, начинается важнейший этап обработки — компиляция программы. На этом этапе производится перевод текста с языка, понятного программисту, на язык, понятный компьютеру. Процесс компиляции идет в диалоговом режиме. Программист получает информацию об обнаруженных компилятором ошибках, исправляет их и снова компилирует программу. Так продолжается до тех пор, пока система не информирует об успешном завершении компиляции. Один из способов запуска компилятора на обработку программы, находящейся в текущем окне системы, осуществляется нажатием комбинации клавиш **Alt** и **F9**. Следует заметить, что большинство операций в процессе обработки программы могут быть активизированы и с помощью манипулятора мышь, но разобраться с этим читателю предлагается самостоятельно. В результате успешной компиляции генерируется так называемый объектный код программы — некий полуфабрикат, к которому в дальнейшем необходимо кое-что присоединить для получения готового продукта. Объектный код записывается в специальный файл, имеющий (если не предпринять специальных действий) то же имя, что и текстовый файл, и расширение **obj**, например **one.obj**. Объектный файл записывается на диск. Место его размещения зависит от настройки системы. Рекомендуется конфигуриро-

вать систему таким образом, чтобы выходные файлы сохранялись в рабочем каталоге — каталоге, содержащем текстовый файл. Крайне нежелательно, если окажется, что выходные файлы записываются куда-либо в системные каталоги **TURBO C**, что, некстати, предусмотрено по умолчанию. Дело в том, что объектный файл становится ненужным после окончательной отладки программы и его следует удалить с диска. Чтобы не забывать это делать, желательно иметь файл «на виду» в рабочем каталоге.

Следующий этап называется редактированием связей — линковкой. На этой стадии к объектному коду присоединяются некоторые стандартные (системные) компоненты, необходимые для данной программы. Линкер вызывается нажатием клавиши **F9**. Ошибки редактирования связей обычно обусловлены неправильной конфигурацией системы или порчей отдельных средств оболочки **TURBO C**. Устранение таких ошибок требует определенной квалификации, которая, как известно, приходит с опытом. В результате успешной линковки в файл с расширением **.exe** и тем же, что и у исходного, именем записывается исполняемый код программы. Поскольку **exe**-файл также сохраняется на диске, исполняемый файл может быть запущен на выполнение уже вне оболочки — из операционной системы **DOS** или **WINDOWS**. Чтобы выйти из оболочки, достаточно нажать комбинацию клавиш **Alt+X**. Теперь, находясь в рабочем каталоге, можно обычным образом запустить программу, набрав в командной строке **опе.exe**. Можно, однако, запустить программу на выполнение и не покидая **TURBO**-систему. Для этого нажимается комбинация **Ctrl+F9**. Для просмотра результатов выполнения программы необходимо открыть рабочий экран — нажать клавиши **Alt+F5**. Возврат в экран оболочки осуществляется повтором этой комбинации. Можно автоматизировать выполнение последовательности компиляция — линковка — исполнение программы. Для этого сразу после подготовки текста программы нажимается комбинация **Ctrl+F9**. При обнаружении ошибок на любом из этапов обработки программы система остановится в соответствующем месте и выдаст необходимую информацию. Заметим, что ошибки могут возникать и на этапе запуска готовой программы — так называемые ошибки времени исполнения (runtime errors). Это, например, результат попытки деления на ноль или извлечения квадратного корня из отрицательного числа.

Традиционно изучение системы программирования начинается с простейшей программы, которая должна выполнить единственную задачу — вывести на экран некоторое приветствие. Отдавая дань этой традиции, предложим читателю проделать указанные выше действия с программой, текст которой выглядит следующим образом:

```
#include<stdio.h>
#include<conio.h>
          void main()
          {
              clrscr();
              printf("Hello World!");
              getch();
          }
```

Не будем пока рассматривать саму программу, которую будем считать «нулевой», а постараемся только без ошибок набрать ее текст, выполнить компиляцию и линковку. Если все будет сделано правильно, то после запуска программы на пустом экране вы увидите это самое приветствие

```
Hello World!
```

которое будет там находиться до тех пор, пока вы не нажмете любую клавишу на клавиатуре.

Наша первая программа

Рассмотрим текст нашей первой программы:

```
#include<stdio.h>
   void main(void)
   { int a,b,c;
     a=4; b=3;
     c==a+b;
     printf("%d+%d=%d",a,b,c); }
```

Прежде всего, заметим, что C-программа различает строчные (малые) и прописные (большие) буквы. Необходимо помнить, что служебные слова языка, например первое в приведенной программе слово **#include**, пишутся, как правило, именно строчными буквами и «A» это совсем не то же самое, что «a».

Теперь несколько слов о размещении отдельных частей программы друг относительно друга. Здесь вам предоставлена большая свобода, которой нужно воспользоваться для создания текста, удобного для чтения. В программе можно оставлять пустые строки, вставлять последовательности пробелов (за исключением неразрываемых конструкций, таких как служебные слова). Не рекомендуется делать длинные строки, не вписывающиеся в ширину экрана — строку можно разорвать, перенеся оставшуюся часть на новую строчку. Нельзя разрывать только текст внутри кавычек. Вернее, в этом случае следует принимать специальные меры.

Приведенная выше программа начинается с так называемых директив препроцессора — строк, следующих за символом #, называемым решеткой, и одной функции, имеющей имя **main**. О директивах препроцессора речь пойдет позже, а сейчас — ВАЖНЕЙШИЙ МОМЕНТ: любая программа обязана содержать функцию с именем **main** — главная. Эта функция, вне зависимости от того, где она находится в тексте, всегда будет выполняться первой. В данном же случае функция **main** вообще является единственной функцией программы. Общая структура функции выглядит следующим образом:

```
ТипФункции ИмяФуякции(Параметры){ ТелоФункции};
```

Исходя из этого, мы видим, что наша функция с именем **main** имеет загадочный тип **void** и в качестве параметра тоже фигурирует это замечательное

слово. Не вдаваясь пока в подробности, укажем, что слово это может иметь смысл «никакой» или, как это ни странно, — «любой». Тогда вторую строку программы следует понимать в том смысле, что функция **main** не имеет никакого типа и никаких параметров. Что же означают эти премудрости, будет ясно из дальнейшего, а пока читателю предлагается оформлять главную функцию именно в таком виде.

Все действия, которыми предписывается выполнять функции, находятся в теле функции, внутри фигурных скобок. Важнейшими элементами программы являются переменные. Понятие переменной близко к понятию переменной величины в математике. Переменная может принимать те или иные значения, изменять свое значение в процессе выполнения программы, входить в различные выражения. Каждая переменная, используемая в программе, наделяется именем. Имя должно начинаться с буквы и не должно содержать специальных символов вроде @ или *. Напомним, что переменные **f** и **F** — это две разные переменные. Для того чтобы оперировать переменной, ее необходимо описать (объявить, декларировать — все три термина в ходу у программистов и имеют близкие смыслы) внутри или вне функции, причем все описания внутри функции в языке C (но не в C++) делаются в начале тела функции, сразу за открывающей фигурной скобкой. Структура описания такова:

```
ТипПеременных Имя1,Имя2,Имя3,...;
```

Здесь перечисляются через запятую имена всех переменных, имеющих данный тип. Взглянув на нашу программу, мы увидим, что в ней описаны три переменные, имеющие тип **int**.

В C существуют следующие основные типы данных:

- **int** — тип целых чисел, двоичное представление которых может занимать 2 байта;
- **short** — короткие целые, которые занимают 1 байт;
- **long** — длинные целые (для переменных, принимающих большие целочисленные значения);
- **float** — тип вещественных чисел (чисел с плавающей точкой);
- **double** — вещественные двойной точности (для вещественных чисел, в диапазоне от очень больших до очень малых по модулю значения);
- **char** — символьный тип (для переменных, принимающих значение символа).

Последний тип, наверное, несколько непривычен. То, что целая переменная **x** имеет значение, равное, допустим, семи, выглядит вполне естественным, но нужно принять как должное то, что символьная переменная s может иметь значение **R**, или значение #, или, к примеру, значение <. Более того, символьные переменные могут иметь даже «невидимые» значения, которые не отображаются на мониторе и не присутствуют в явном виде на клавиатуре компьютера. Но об этом чуть позже...

Название типа может быть снабжено модификатором **unsigned**, что означает беззнаковый. Переменные, описанные таким образом, могут принимать

только неотрицательные значения. Так, переменная, описанная как **short a**, может изменяться в диапазоне целых чисел от −128 до +127. Если же она описана как **unsigned short a**, то пределы ее допустимых значений — это числа от 0 до 255.

Итак, в нашей программе сразу в начале тела функции имеется описание переменных. Это описание является частным случаем того, что в C называется выражением. Если в естественных языках текст состоит из отдельных предложений, заканчивающихся точкой, то текст на языке C состоит из отдельных выражений, как правило, заканчивающихся точкой с запятой — символом «**;**». Понятие выражения близко к понятию оператора в других языках программирования, однако является более широким.

В двух следующих строках программы записаны так называемые выражения присваивания. Смысл их очевиден — переменные получают конкретные значения. Рассмотрим, каким образом записываются эти конкретные значения в языке. Проще всего дело обстоит с записью целочисленных констант. Они представляются точно так, как в арифметике, — совокупностью арабских цифр со знаком или без знака перед числом: 6, 14, −21, +457, 0012. Есть возможность использования шестнадцатеричного, восьмеричного, а при необходимости и двоичного представления чисел. Запись вещественных чисел в форме с фиксированной запятой отличается от арифметической записи только тем, что вместо десятичной запятой, отделяющей целую часть числа от дробной, используется десятичная точка: 3.1415927, −12.7. Число .872 есть не что иное, как 0,872 в арифметической записи. То есть вы можете не писать незначащий 0 как в целой, так и в дробной части. Число 23. есть число 23,0. Однако последнее число является вещественным — это не совсем то, что 23. Важность этого различия будет понятна из последующего материала.

Другая форма представления вещественных чисел называется экспоненциальной, или формой с плавающей точкой. Речь идет о записи чисел с мантиссой и порядком, например, $2{,}72 \cdot 10^7$. При записи таких чисел сначала пишется мантисса, затем символ **e** или **E** и после него — порядок. Так, для записи физической постоянной Больцмана $1{,}38 \cdot 10^{-16}$ эрг/град следует использовать форму **1.38e-16**.

Символьная константа — это символ, заключенный в апострофы: **'P'**, **'c'**. Не следует путать апостроф **'** и кавычки «. Наряду с обычными символами в C, как уже говорилось, имеется набор специальных символов, которые не имеют изображения, однако имеют вполне определенное значение, а, следовательно, и назначение в программе. В тексте программы такие символы изображаются последовательностью двух знаков, первый из которых есть «обратная черта» \. Перечислим некоторые из таких символов с указанием действий, вызываемых ими:

- **\n** — переход на новую строку;
- **\a** — выдача звукового сигнала;
- **\b** — возврат на 1 шаг влево;
- **\f** — переход на новую страницу;

- **r** — возврат в начало строки;
- \\ — сам символ \;
- **?** — символ **?**;
- \' — символ ';
- \" — символ ".

Все эти символы играют важную роль при отображении результатов работы программы на экране компьютера и часто используются в функции печати **printf**, к рассмотрению которой мы сейчас и переходим.

Вывод информации на экран, функция printf

Читатель не ошибается, полагая, что завершающее первую программу выражение осуществляет вывод информации на экран. Это выражение представляет собой функцию, имеющую имя **printf**. Но где же описание этой функции? Его ведь в программе нет. Действительно, в явном виде мы такого описания в программе не имеем. Дело в том, что эта функция относится к разряду так называемых стандартных функций. Ее изготовили без нас разработчики системы. Спасибо им за это! Описание, или прототип, этой функции находится там, где ему положено быть, а наша забота в том, чтобы указать системе, где его искать. Именно это мы и делаем в первой строке препроцессорной директивой **include**. Эта директива подключает к нашей программе так называемый заголовочный файл **stdio.h**, который в каком-то смысле и содержит описание ряда стандартных функций, в том числе и **printf**. Расширение **.h** как раз и происходит от термина «заголовок» — **head**.

Структура функции с именем **printf** в общем виде такова:

```
printf("строка",Выражение1,Выражение2,Выражение3,...);
```

Заметим, что в зависимости от того, что находится внутри кавычек, эти самые Выражение1, Выражение2 и т. д. могут присутствовать или отсутствовать, а их количество практически не ограничено.

Функция работает следующим образом. Начинается просмотр содержимого строки в кавычках. Все символы после открывающих кавычек дословно и последовательно отображаются на экране, начиная с той позиции, где курсор находился перед началом работы функции. Если встречается один из специальных символов, указанных выше, выполняется действие, предписываемое этим символом. В частности, если встретился символ **\n**, то курсор переместится в начало следующей строки, и печать на экран будет продолжена с этой позиции. Содержимое строки будет печататься таким образом, пока не встретятся закрывающие кавычки (и на этом работа **printf** завершится и программа будет выполняться дальше) или очередным символом в строке не окажется знак **%**. В последнем случае система анализирует то, что находится после этого знака. Если далее снова следует **%**, то просто выводится сам символ **%**,

иначе будет напечатано значение очередного выражения из списка, следующего после закрывающих кавычек и запятой. Таким образом, программист должен следить за соответствием количества знаков % в строке внутри кавычек и количеством выражений в списке.

За знаком %, предписывающим печать значения выражения, должна следовать форматная конструкция, определяющая, в какой форме будет выведено это значение. В простейшем случае форматная конструкция представляет собой букву. Наиболее часто используются следующие форматные конструкции:

- **%d** — для вывода целых десятичных чисел (**int**);
- **%x** — для вывода целого числа в шестнадцатеричной системе счисления;
- **%f** — для вывода вещественных с фиксированной запятой (**float, double**);
- **%e** — для вывода вещественных в экспоненциальной форме (**float, double**);
- **%g** — для вывода вещественных с автоматическим (выбираемым самой системой) определением формы с плавающей или фиксированной запятой;
- %**c** — для вывода символа (**char**);
- **%s** — для вывода строки символов (этот вариант будет прокомментирован позже);
- **%p** — для вывода адреса объекта (об этом мы также будем говорить дальше).

Перед форматной буквой может стоять символ модификатора **l** (**long**) или **u** (**unsigned**). Так формат **%lf** применяется к длинному вещественному, то есть к типу **double**, а формат **%ud** — к беззнаковому целому.

Теперь мы можем вернуться к первой программе и обнаружить, что в результате ее работы на экран будет выведено следующее:

```
4+3=7
```

Между символом % и форматной буквой может стоять конструкция, управляющая количеством позиций, отводимых под печатаемую величину. Для целого и символьного форматов эта конструкция — просто целое число. Формат **%4d** означает, что под целое число будет отведено на экране поле из четырех позиций. Печатаемое число будет прижато к правому краю этого поля так, что, например, перед двухзначным положительным числом слева в таком формате будут две пустые позиции, число —123 займет все выделенное ему поле, а при печати числа 32 767 будет автоматически добавлена одна недостающая позиция. Аналогично этому при выводе символа по формату **%5c** сначала будет напечатано четыре пустых позиции, а затем символ. При выводе вещественного числа управляющая полем конструкция представляет собой два беззнаковых целых числа, разделенных точкой, например 6.2. Первое число есть полная ширина поля, предоставляемого выводимому числу, второе — количество позиций, выделяемых под дробную часть числа. Поскольку десятичная точка сама занимает одну позицию, понятно, что первое число должно быть, как минимум, на единицу больше второго. Впрочем, если система обна-

ружит какое-либо несоответствие в форматном выражении, она сама добавит необходимое число позиций. Так, в результате выполнения фрагмента программы:

```
a=-0.016722272;
printf("a=%7.31f",a);
```

будет напечатано **a= -0.017** (после знака равенства вставлена одна пустая позиция). Как видите, умная система оставила три знака в дробной части, выполнив округление.

В данный момент уместно сделать одно важное общее замечание о языке С. Программируя на С, вы можете допускать весьма большие вольности, которые не допускаются в других языках. С другой стороны, этот факт подразумевает и особую ответственность, накладываемую на программиста. Проиллюстрируем это одним ярким примером. Оказывается, что тип, задаваемый форматным выражением, вовсе не обязан совпадать с типом печатаемого выражения. Так, печатая вещественное число по целому формату, вы получите разумный результат — целое число, полученное из вещественного округлением. Печатая целое число по вещественному формату, вы получите на экране вещественное число, у которого в дробной части напечатаются нули. Эти преобразования недоразумений не вызывают. Несколько неожиданным покажется то, что можно совершенно спокойно печатать символ по целому и даже по вещественному формату. Не будет ошибкой и попытка напечатать вещественное число по символьному формату.

Попробуем разобраться, что будет напечатано в результате работы следующего фрагмента программы:

```
t='u';
printf("Код символа %c=%d" ,t,t) ;
```

Символьная переменная **t** принимает значение символа **u**. Первое форматное выражение **%c** обычным образом выведет значение переменной — букву **u**. Второе форматное выражение **%d** обратится к своему аргументу, а это снова переменная **t**, и попытается вывести его значение в целом числовом формате. Здесь уместно напомнить, что вся информация в компьютере хранится в цифровом виде, и в ячейках памяти, отведенных для переменной **t**, находится вовсе не сам символ, а некий набор нулей и единиц — цифровой код символа. Поэтому ясно, что нет никакого «криминала» в попытке вывести символьное выражение в целом формате. Все символы имеют свой код, и именно десятичное представление этого кода будет напечатано:

Код символа u=75

Наоборот, если попытаться напечатать значение числового выражения по символьному формату, система сделает необходимое преобразование и выведет соответствующий символ. Разница в том, что в первом случае мы получим вполне разумный и полезный результат (можно напечатать таблицу кодов символов), а во втором случае результат совершенно бессмыслен.

Арифметические выражения, математические и некоторые другие полезные функции

Вооружившись изрядным запасом знаний, переходим к новой программе:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
double a;
const double Pi=3.1415927;
void main()
{ clrscr();
   a=30.0;
   printf("Sin(%.4.1lf)=%6.3lf",a,sin(a*Pi/180.0));}
```

Что здесь нового? Прежде всего, мы подключили к программе два новых заголовочных файла: **conio.h** и **math.h**. Первый из них позволяет использовать функцию **clrscr**, а второй — функцию **sin**. Далее мы видим, что описание переменной **a** оказалось вне главной функции **main**. Тем самым, переменная **a** стала глобальной и имела бы право на существование не только в функции **main**, но и в любых других функциях программы, если бы таковые имелись. Поскольку у нас в программе лишь одна функция, не имеет никакого значения, является ли переменная глобальной или локальной. Дальше мы могли бы сказать, что делается описание глобальной вещественной переменной **Pi**, если бы не служебное слово **const**. Это слово указывает на то, что мы описали не переменную, а, напротив, постоянную величину — глобальную константу **Pi**, которая получила значение числа Пифагора. Слово **const** означает, что величина **Pi** не должна и не может быть изменена нигде в программе. Указанным способом константы, как и переменные, могут получать начальные значения непосредственно при их описании.

Следующая новость — внутри круглых скобок после имени головной функции теперь нет слова **void**. Но мы уже знаем, что это слово означает «никакой». Ну, а коли так, значит, можно обойтись и пустыми скобками. Да будет так! Так будет и с новой функцией **clrscr()**, которая не требует никаких параметров или аргументов в силу своей скромной, но крайне полезной миссии.

Чтобы получить полную информацию о новой, пока вам незнакомой, конструкции языка, можно воспользоваться справочной системой **TURBO C**. Для этого, как уже упоминалось, курсор наводится на любую букву конструкции и нажимается комбинация клавиш **Ctrl+F1**. Открывается окно, содержащее описание конструкции и пример ее использования. Просмотрев справку, вы возвращаетесь в рабочий режим, нажав клавишу **Esc**. Проделав такую операцию для **clrscr**, вы узнаете, что эта функция очищает рабочий экран от всего того «мусора», который достался вам в наследство от работы предыдущей программы или предыдущего запуска этой же программы. Кроме того, вы узнаете, что функция требует подключения заголовочного файла **conio.h** и используется так, как это сделано в нашей программе.

В нашей программе производится вычисление значения тригонометрической функции **sin** от угла 30.0 градусов. Первое форматное выражение в **printf**

печатает сам угол, значение которого получила переменная **a**, а второе форматное выражение печатает значение синуса этого угла. Читателю уже понятно оформление печати текстом, обеспечивающим красивый вывод результата работы программы:

Sin(30.0)= 0.500

В аргументе синуса стоит не просто величина угла, а арифметическое выражение, выполняющее преобразование угла из градусов в радианы, как этого требует данная функция. Хотя мы еще не знакомились с правилами записи арифметических выражений, данный пример тривиален и, без сомнения, не вызовет вопросов.

Основных арифметических действий в С, в отличие от школьной арифметики, шесть. Сложение и вычитание не имеют каких-либо особенностей и выполняются с помощью традиционных знаков + и −. С умножением тоже нет никаких хлопот. Надо только помнить, что, в силу отсутствия на клавиатуре подходящих символов, знаком операции умножения является звездочка *. У операции деления, использующей символ /, есть особенность. Во-первых, деление может быть двух видов. Значение выражения **3/4.0** равно 0.75, в чем у читателя нет оснований сомневаться. А зря, поскольку значение выражения **3/4** равно нулю! Секрет прост. Если и делимое, и делитель есть выражения целого типа, как в последнем случае, то деление выполняется целочисленно (еще пример: **6/4** дает 1), если же любой из операндов — или делимое, или делитель, или и тот и другое вместе вещественного типа, то выполняется привычное вещественное деление. В первом случае делитель является вещественным числом, благодаря наличию десятичной точки, хотя значение его равно точно четырем.

Пятым арифметическим действием является целочисленная операция получения остатка от деления. Знак операции есть %. Так, результат действия **16%3** равен 1; **15%3** дает 0 (15 на 3 делится нацело), а **14%3** = 2.

Последняя рассматриваемая нами операция, а точнее, две операции, называется сдвигом и применяется к целочисленным величинам. Записываются эти операции в виде **m<<n** и **m>>n**. В результате их выполнения двоичный код числа **m** сдвигается на **n** разрядов влево или вправо соответственно. Не очень понятно? Не вдаваясь в некоторые детали, требующие представлений о двоичной системе счисления, самостоятельное рассмотрение которых предлагаем любопытным читателям, поясним, что первая из операций дает результат $m2^n$, а вторая — $m/2^n$. В частности, действие **m<<1** просто удваивает значение переменной **m**.

Наряду с основными, так называемыми бинарными, операциями в С существуют унарные операции инкремента и декремента. Эти операции по канонам классического С также применяются к целым. Однако в **TURBO C++ 1.0** допущена «не рекламируемая» вольность, по которой операции применимы и к вещественным. Все же автор не рекомендует такой вольностью пользоваться. Операция инкремента записывается в виде **++Имя** (префиксная форма) или **Имя++** (постфиксная форма). В обоих случаях в результате инкрементирова-

ния значение переменной увеличивается на единицу. Разница в записи заключается в том, что в первом случае переменная сначала увеличивается, а затем ее увеличенное значение используется в выражении, а во втором случае переменная сначала используется в выражении, а потом увеличивается. Точно так обстоит дело и с унарной операцией декремента **-Имя** и **Имя-**, однако здесь происходит уменьшение значения на единицу.

Вместо операций сдвига и инкремента/декремента можно, конечно, получить желаемый результат, используя только сложение, вычитание, умножение и деление, но нужно знать, что эти «экзотические» операции выполняются процессором значительно быстрее. Кроме того, использование в программах сдвига и унарных операций говорит о квалификации программиста и качестве программ.

В сложных арифметических выражениях приоритеты выполнения основных действий обычные: умножение и деление имеют преимущество по отношению к сложению и вычитанию. Действия выполняются слева направо. При необходимости можно использовать круглые скобки как в обычной математике.

Вот типичный пример ошибки, свойственной новичкам. Допустим, необходимо вычислить частное от деления 1 на 2a. Начинающий программист может записать выражение **1/2*a**. В результате вообще получится 0. Догадываетесь почему? Нет? Вернитесь на несколько строк назад. Исправим допущенную оплошность, записав **1.0/2*a**. Теперь при вычислении значения выражения единица разделится вещественным делением на двойку и результат 0.5 умножится на **a**. Оказывается, мы подсчитали величину a/2. Верной будет такая запись: **1.0/(2*a)**. Этот вариант действительно правильный, но не оптимальный. Поскольку операция умножения выполняется процессором значительно медленнее, чем операция сложения, оптимальной по быстродействию будет запись **1.0/ (a+a)**.

Закрепим полученные знания, рассмотрев следующую программу, в которой, кстати, используем еще одну новинку:

```
#include<stdio.h>
#include<conio.h>
void main()
{ int i==7,j=4; double id=7.0,jd=4.;
cIrscr();
printf("i/j=%d\n",i/j);
printf("id/jd=%lf\n",id/jd);
printf ( "I%%j =%d\n", I%j ) ;
printf("id*jd=%lf",id*jd);
i=getch(); }
```

В результате работы этой программы на экран будет выведена следующая информация:

```
i/j=l
id/jd=1.750000
i%j==3
id*jd=28.000000
```

Печать будет выполнена в четырех строках, поскольку в первых трех функциях печати предусмотрен переход на новую строку — символ **\n**. Автор надеется, что числовые значения выполненных операций теперь читателю понятны.

Обещанная новинка находится в самом конце программы — это функция **getch()**, прототип которой задается через уже знакомый заголовочный файл **conio.h**. Функция производит передачу в программу символа, который вводится с клавиатуры (а если вы хотите щегольнуть компьютерным жаргоном, то сообщите своему знакомому, что здесь производится ввод символа с консоли).

В использованном варианте программы целая переменная **i** получит числовое значение, равное коду символа, клавишу которого вы нажмете на клавиатуре. Однако в данной программе передача символа в программу это только средство, а не цель. Цель состоит в обеспечении удобства работы с программой. Когда программа запускается на исполнение, в частности, из оболочки **TURBO C**, перед вами на секунду открывается рабочий экран с результатами работы, после чего система снова возвращается в экран текстового редактора оболочки, и, чтобы снова взглянуть на результаты, вам приходится нажимать клавиши **Alt+F5**. Если же в конце программы стоит обращение к функции **getch**, то рабочий экран будет открыт до тех пор, пока вы не введете символ с консоли, то есть пока не нажмете какую-либо клавишу. Вот и вся маленькая хитрость. Заметим, что само присвоение переменной **i** значения здесь никакой роли не играет. По этой причине вместо **i=getch();** можно просто ограничиться вызовом функции без присвоения **getch();** — это тоже удобная «вольность» языка С.

Теперь мы подошли к ответственному моменту — использованию в С-программах математических функций. Таких функций в **TURBO C** весьма большое количество. Мы познакомимся только с частью из них с помощью следующей таблицы:

sin(x) — x задается в радианах;

cos(x) — x задается в радианах;

tan(x) — тангенс, x в радианах;

asin(x) — арксинус, $-\pi/2 < x < \pi/2$;

acos(x) — арккосинус, 0 <: x < π;

atan(x) — арктангенс, $-\pi/2 < x < \pi/2$;

atan2(y,x) — арктангенс величины y/x, $-\pi < y/x < \pi$;

sinh(x) — гиперболический синус;

cosh(x) — гиперболический косинус;

tanh(x) — гиперболический тангенс;

exp(x) — e^x;

log(x) — натуральный логарифм, $x > 0$;

logl0(x) — десятичный логарифм;

pow(x,y) — x^y. Недопустимо: x = 0, y < 0 или x < 0, y — нецелое;

sqrt(x) — $\sqrt{x}$, x >= 0;

abs(x) — модуль x, x — целое;

fabs(x) — модуль x, x — нецелое;

floor(x) — округление x «вниз» (тип double);

ceil(x) — округление x «вверх» (тип double);

ldexp(x,n) — $x2^n$;

fmod(x,y) — остаток от вещественного деления x на y.

Обратим внимание на то, что функции **ceil** и **floor**, хотя и вычисляют (или, как принято говорить у программистов, «возвращают») целочисленные значения, имеют вещественный тип. Вообще для определения типа любой функции и типов ее аргументов рекомендуется просмотреть информацию об этой функции через справочную систему способом, описанным выше. Напомним еще раз, что использование математических функций требует подключения заголовочного файла **math.h**.

Выражения присваивания и ввод данных в программу

Теперь перед нами стоит важная задача — научиться вводить в программу извне исходную информацию. Разумеется, начальные данные для решения конкретной задачи могут быть заданы в выражениях описания переменных или с помощью выражений присваивания, как мы это делали до сих пор. Понятно, однако, что на этом пути мы далеко не продвинемся, поскольку при этом изменения вводимых в программу величин потребуют изменений в тексте, перекомпиляции и т. д. Конечно, в C есть средства более удобного общения пользователя с программой, но сначала мы все же более подробно поговорим о выражениях присваивания. В простейшем виде выражение присваивания имеет структуру:

```
ИмяПеременной=Выражение;
```

Слева от знака равенства может стоять именно имя переменной и ничто другое. Справа стоит любое выражение, в том числе, возможно, снова выражение присваивания. Работает выражение присваивания так. Сначала вычисляется значение выражения в правой части, затем это значение приводится к типу, с которым была описана переменная в левой части, и после этого переменная получает соответствующее значение. На основании такого определения ясно, что странная, с точки зрения алгебры, запись **P=P+1** является совершенно нормальной конструкцией языка C. Означает такое выражение присваивания то, что после его выполнения переменная **P** будет иметь значение, увеличенное на единицу по сравнению с тем значением, которое было у нее до присваивания. Также на основании определения можно легко понять,

что в результате «каскадного» присваивания типа **a=b=c=3.1415927** все три переменные будут иметь значение пифагорова числа. Подчеркнем, что присваивание здесь будет выполняться справа налево. Сначала **c** получит значение этой константы, затем **b** примет значение, равное величине переменной **c**, а затем и **a** не останется обойденной вниманием. Для любопытных дадим повод к размышлению. А что будет, если переменные в «каскадном» присваивании имеют разные типы? Тут, как говорится, возможны варианты.

Кроме обычного выражения присваивания в языке С реализованы сложные присваивания, имеющие вид:

Имя< операция> =Выражение;

Имя — имя переменной. В качестве конструкции **<операция>** может стоять любой знак арифметического действия или операции сдвига +, −, *, /, %, >>, <<. Такое сложное присваивание дает тот же результат, что и простое присваивание в виде **Имя=Имя<операция> Выражение**, однако сложное присваивание, использованное там, где оно может заменить простое, выполняется быстрее. В этой связи в оптимально написанной программе обычно широко используются сложные присваивания. Типы данных, допустимые в этих выражениях, зависят от типов операций. Выражение **a*=r** эквивалентно выражению **a=a*r** и допустимо для любых типов данных. Выражение **a%=b** присваивает переменной **a** новое значение, равное остатку от целочисленного деления старого значения **a** на **b**, и применимо, очевидно, к целым. Целочисленным является и выражение **a>>=2**, результат которого — новое значение **a** есть старое значение **a**, целочисленно деленное на 4. Разумеется, такое присвоение выполняется существенно быстрее, нежели выражения **a=a/4**.

Как уже говорилось, придание переменным значений с помощью присваивания не решает проблему ввода данных в программу в целом. Если вы хотите многократно использовать одну программу с различными входными данными, то вам поможет функция **scanf**. Продемонстрируем ее работу в следующей программе:

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
viod main()
{ double a.b,c,xl,x2;
clrscr();
printf ("a="); scanf ("%lf ",&a);
printf ("b="); scanf ("%lf ",&b);
printf ("c=") ; scanf("%lf" ,&c) ;
xl=(-b+(c=sqrt(b*b-4.0*a*c)))/(a+a);
x2=(-b-c)/(a+a);
printf ("xl=%f\n",xl);
printf ("x2=%f\n",x2);
getch(); }
```

Оглавление