C/C++ и Borland C++ Builder

для начинающих

- Основные элементы языков C/C++
- Визуальная среда программирования
- Создание основных типов приложений
- Работа с базами данных
- Технологии BDE, ADO, MIDAS, DDE
- Создание интернет-приложений

Борис Пахомов

bhv

Борис Пахомов

C/C++ и Borland C++ Builder

для начинающих

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92C,C++
ББК 32.973.26-018.1
П12

Пахомов Б. И.

П12 C/C++ и Borland C++ Builder для начинающих. — СПб.: БХВ-Петербург, 2005. — 640 с.: ил.

ISBN 978-5-94157-507-7

Книга является руководством для начинающих по разработке приложений в среде Borland C++ Builder. Рассмотрены основные элементы языков программирования C/C++ и примеры создания простейших классов и программ. Изложены принципы визуального проектирования и событийного программирования. На конкретных примерах показаны основные возможности визуальной среды разработки C++ Builder, назначение базовых компонентов и процесс разработки различных типов Windows-приложений, в том числе приложений баз данных с использованием технологии BDE, ADO, MIDAS, DDE и интернет-приложений.

Для начинающих программистов

УДК 681.3.068+800.92C,C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Алия Амирова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.10.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 51,6.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-507-7

Содержание

Введение

Мы приступаем к изучению среды Borland C++ Builder. Что это за среда? Какую помощь она оказывает в разработке программного обеспечения и чем отличается от других программных продуктов? Можно ли ее установить на вашем компьютере? Попробуем ответить на эти и ряд других вопросов, касающихся применения Borland C++ Builder (в дальнейшем для краткости Builder).

Builder — это среда, в которой можно осуществлять так называемое *визуальное программирование*, т. е. создавать программы, которые во время исполнения взаимодействуют с пользователем благодаря многооконному графическому интерфейсу. Какими преимуществами обладает многооконный графический интерфейс? Если сказать просто, то в момент выполнения программы на экране могут появляться в цветном изображении элементы управления программой:

- кнопки, на которые можно нажимать мышью, после чего происходят некоторые "привязанные" к этим кнопкам действия;
- поля для ввода/вывода данных;
- списки данных, из которых можно выбирать данные для дальнейших расчетов в программе;
- различные меню, позволяющие выбирать и выполнять определенные действия;
- элементы, контролирующие состояния объектов (типа "включен — выключен, выбран — не выбран" и т. п.);
- элементы, позволяющие следить за ходом некоторых процессов по времени их выполнения;
- элементы, позволяющие выбирать даты из календаря;
- элементы, обеспечивающие стандартный выбор файлов, шрифтов, цвета, настройки принтеров и т. д.;
- элементы, позволяющие вставлять в вашу программу другие программы-объекты (например, программы Word, Excel, анимационные файлы, диаграммы, "устройства" аудио- и видеозаписи, различные клипы мультимедиа и проч.).

Кроме этого, на экране могут появляться различные изображения, иерархические деревья, которые помогают лучше понять ход выполнения программы и управлять им. Среда Builder позволяет работать как с простыми локальными и удаленными базами данных, так и с многозвенными распре-

деленными базами данных. Кроме того, среда Builder позволяет установить соединение и взаимодействие вашей программы с Интернетом.

В среде Builder разработка программ ведется на основе современного метода — объектно-ориентированного программирования.

На рынке программных продуктов есть много сред для автоматизации программирования (например, Visual Basic, Visual C++, Borland Delphi). По мощности и удобству использования со средой Builder может соперничать только Borland Delphi. Но эта последняя, по мнению автора, уступает среде Builder из-за того, что использует алгоритмический язык Object Pascal, в то время как языком среды Builder является алгоритмический язык C++, более мощный и удобный для программирования. Если изучить сначала C++, а потом изучить Pascal и попробовать составлять на нем программы (например, в среде Delphi), то после работы на C++ это не принесет никакой радости. Постоянно что-то раздражает: то надо каждый раз при добавлении новых переменных и меток обращаться к разделу объявлений, то не идет простейшее преобразование данных при присвоении, и надо начинать использовать операцию преобразования типов данных. И потом, в Pascal — тьма типов числовых данных, без которых C++ успешно обходится! На C++ написана масса стандартных программ, которые позволяют применять его во многих областях знания и даже на уровне ассемблера. Мало того, что программа на C++ позволяет включать в свой состав блоки ассемблерных программ, но существуют и стандартные программы на C++, реализующие ассемблерные функции (например, обработку прерываний, работу с секторами дисков и т. п.).

Кроме того, по мнению автора, Pascal более труден для освоения начинающими. Только имея определенный опыт работы на C++, начинаешь глубоко понимать многие моменты в языке Pascal. А школьники, которым его преподают в течение двух лет, а студенты, не имеющие опыта общения с подобными языками?.. Кажется весьма сомнительной необходимость преподавания этим категориям учащихся языка Pascal в качестве их первого алгоритмического языка. Такой метод надолго отбивает охоту у людей к изучению алгоритмических языков вообще. С высоты знаний сегодняшних экономических отношений в обществе приходит в голову крамольная мысль, что кому-то не без выгоды удалось протолкнуть свое детище в широкие массы учащейся молодежи.

В основе языка C++ лежит язык C. Основным, *кардинальным* отличием C от C++ является наличие *классов* в последнем. Поэтому изучение C++ начинается с изучения C.

Предлагаемая книга состоит из двух частей. В первой изучается C и его расширение C++, во второй — собственно среда Builder, использующая C++. Для изучения C(C++) применен такой подход: читателю сначала предлагается программа, написанная на C, затем разъясняется, как и почему

она работает, а затем объясняются все элементы, входящие в программу (объявления переменных, что такое переменные, типы данных, что это такое и т. п.). То есть от практики — к теории, а не наоборот. Автор надеется, что такой подход вызовет больший интерес у начинающих и не отобьет у них охоту к изучению языка. В то время как при старом, традиционном подходе (от теории — к практике) непривычные термины ("типы данных", "переменные" и т. д.) тут же вызывают непонимание, скуку на лицах и неприкрытую зевоту.

Программы, которые мы будем создавать в среде Builder на языке C без применения собственно элементов Builder, — это так называемые *консольные приложения*. То есть программы, которые запускаются без графического интерфейса в консольном окне. Когда запускается консольное приложение, Windows создает консольное (т. е. опорное) окно в текстовом режиме, через которое пользователь взаимодействует с приложением. С этим окном тут же связывается стандартный вывод: все, что будет выводить программа с помощью стандартных программ вывода данных, станет отображаться в этом окне. Консольные приложения обычно не требуют большого пользовательского ввода и выполняют ограниченный набор функций. Мы станем применять консольные приложения для изучения языка C, чтобы потом работать с этим языком в более сложных графических приложениях, имеющих многооконный графический интерфейс, через который пользователь может взаимодействовать с приложением.

Чтобы создать консольное приложение, следует выполнить команды главного меню Builder: **File|New** (в версии 5) или **File|New|Other** (в версии 6) и в открывшемся диалоговом окне выбрать значок **Console Application** на вкладке **New**. После этого в другом открывшемся диалоговом окне надо выбрать тип языка (C или C++) для главной функции будущего приложения-программы.

Примечание

Главная функция — это функция, с которой запускается сама программа. В консольном приложении ее имя — `main()`, для неконсольного же приложения (т. е. приложения с использованием элементов собственно среды Builder) имя главной функции `WinMain()`. Но об этом — по мере изучения материала.

В последнем диалоговом окне есть кнопка для выбора **VCL**. Этой аббревиатурой обозначают визуальные компоненты и другие классы, применяемые в среде Builder. В консольном приложении использовать эту кнопку следует на этапе изучения классов в C++: например, если надо построить приложение со строковым классом `String` (об этом — в свое время).

Итак, для изучения C(C++) строятся консольные приложения. Для изучения же самой среды Builder требуется строить приложения неконсольные,

с многооконными графическими интерфейсами, которые и будут предложены читателю в соответствующих местах книги.

Примечание

Мы пользуемся тем, что среда Builder позволяет в ее пространстве изучать C(C++), хотя это можно было бы делать и вне среды Builder. Но раз уж такая возможность есть, то почему бы ею не воспользоваться?

Для установки Builder требуется, чтобы ваш компьютер имел следующую конфигурацию:

- тактовая частота процессора Intel Pentium — не ниже 90 МГц;
- операционная система — Windows;
- оперативная память — не менее 32 Мбайт;
- свободное пространство на жестком диске — от 120 до 388 Мбайт в зависимости от параметров установки;
- дисковод CD-ROM;
- видеоадаптер не хуже VGA;
- мышь.

Желаю читателям приятного и плодотворного изучения.

Автор

ЧАСТЬ I

Алгоритмический язык C и его расширение C++

Глава 1

Типы данных, простые переменные и основные операторы цикла

Цель этой главы — продемонстрировать начальные элементы программирования на языке С.

Приложения строятся средой Borland C++ Builder в виде специальных конструкций — проектов, которые выглядят для пользователя как совокупность нескольких файлов. Причем в проекте консольного приложения файлов меньше, чем в проектах приложений собственно Builder. Это мы увидим, когда начнем делать приложения для Builder.

Программа на языке С — это совокупность функций, т. е. специальных программ, отвечающих определенным требованиям. Запуск любой программы начинается с запуска *главной функции*. Внутри этой главной функции для реализации заданного алгоритма вызываются все другие необходимые функции. Часть функций создается самим программистом, другая часть — библиотечные функции — поставляется пользователю со средой программирования и используется в процессе разработки программ.

Как перейти к созданию консольного приложения

- Загрузите среду Borland C++ Builder.
- Выполните команды главного меню: **File|New**. Откроется диалоговое окно, показанное на рис. 1.1.

В этом окне дважды щелкните кнопкой мыши на значке **Console Wizard** — Мастера построения заготовок консольных приложений. В результате появится диалоговое окно Мастера (рис. 1.2).

В этом окне активизируйте переключатель **C++**, установите флажок **Console Application** и нажмите **OK**. Мастер сформирует заготовку приложения. Ее вид показан на рис. 1.3.

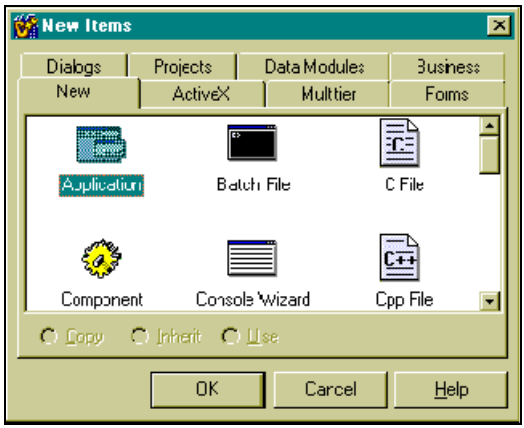

Рис. 1.1. Диалоговое окно **New Items**

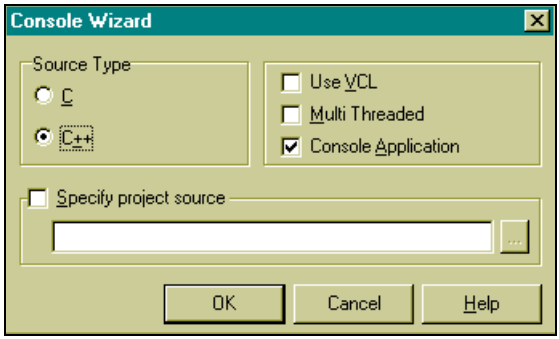

Рис. 1.2. Диалоговое окно Мастера построения заготовок консольных прилоржений

Заготовка состоит из заголовка главной функции `int main(int argc, char* argv[])` и тела, ограниченного фигурными скобками. Преобразуем заголовок функции `main` к виду `main()`, а из тела удалим оператор `return 0`. Все это проделаем с помощью Редактора кода, который открывается одновременно с появлением заготовки консольного приложения на экране: щелкните кнопкой мыши в любом месте поля заготовки и увидите, что курсор установится в месте вашего щелчка. Далее можете набирать любой текст, работать клавишами <Delete>, <Backspace>, клавишами-стрелками и другими необходимыми для ввода и редактирования клавишами.

Мы привели заголовок функции `main` к виду `main()`. Это означает, что наша главная функция не будет иметь аргументов, которые служат для связки консольных приложений. Этим мы заниматься не будем.

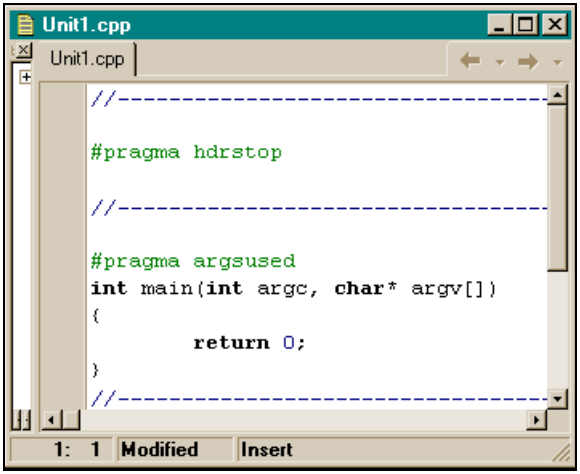

Рис. 1.3. Заготовка консольного приложения

Формирование проекта консольного приложения

Теперь, прежде чем заполнять нашу заготовку какими-то кодами, следует сформировать проект консольного приложения, т. к. приложение в среде Builder существует не само по себе, а в проекте. Для этого снова воспользуемся опцией **File** главного меню. Выполним команду: **File|Save Project As**. Откроется диалоговое окно для сохранения программного модуля заготовки (по умолчанию модулю присваивается имя Unit1, но вы можете дать ему свое имя). Следует выбрать папку, куда вы запишете свой проект, и нажать **OK**. После этого откроется диалоговое окно для сохранения заголовочного модуля проекта (с расширением bpr). Сохраните его, дав ему при необходимости свое имя (по умолчанию заголовочный модуль будет назван Project1). Организационная часть для будущего консольного приложения закончена. Начинаем формировать само приложение, а точнее — его программный модуль Unit1.

Создание простейшего консольного приложения

Запишем в теле функции `main()` следующие две строки:

```
printf("Hello!\n");
getch();
```

Это код нашего первого приложения. Он должен вывести на экран текст "Hello!" и задержать изображение, чтобы оно "не убежало", не исчезло, пока мы рассматриваем, что там появилось на экране. В итоге наше консольное приложение будет иметь вид, представленный на рис. 1.4.

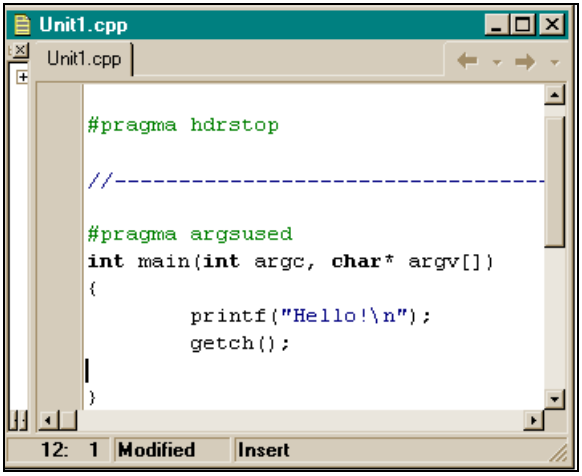

Рис. 1.4. Вид консольного приложения до компиляции

Чтобы приложение заработало, его надо *откомпилировать*, т. е. перевести то, что мы написали на языке С, в машинные коды. Для этого запускается программа-компилятор. Запускается она либо нажатием клавиши <F9>, либо выполнением опции главного меню **Run|Run**. Если мы проделаем подобные действия, то получим картинку, показанную на рис. 1.5.

Картинка показывает, что наша компиляция не удалась: в нижнем поле окна высветилось сообщение о двух ошибках: "Вызов неизвестной функции". Если кнопкой мыши дважды щелкнуть на каждой строке с информацией об ошибке, то в поле функции `main()`, т. е. в нашей программе, подсветится та строка, в которой эта ошибка обнаружена. Разберемся с обнаруженными ошибками.

Откроем опцию **Help|C++ Builder Help** главного меню. Откроется окно помощи. В нем выберем вкладку **Указатель** и в поле **1** наберем имя неизвестной (после компиляции программы) функции `printf`. В поле **2** появится подсвеченная строка с именем набранной в поле **1** функции. Нажмем <Enter>. Откроется окно помощи **Help**, в котором приводятся сведения о функции `printf`, в том числе, в каком файле находится ее описание (**Header file — stdio.h**), и как включать этот файл в текст программного модуля (`#include <stdio.h>`). `#include` — это оператор компилятора. Он включает в текст программного модуля файл, который указан в угловых скобках. Та-

ким же образом с помощью раздела **Help** найдем, что для неизвестной функции `getch()` к программному модулю следует подключить строку `#include <conio.h>`. После этого текст нашей первой программы будет выглядеть, как на рис. 1.6.

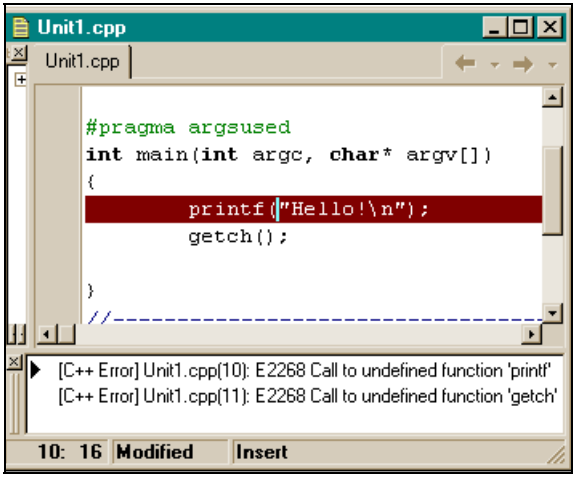

Рис. 1.5. Вид консольного приложения после компиляции

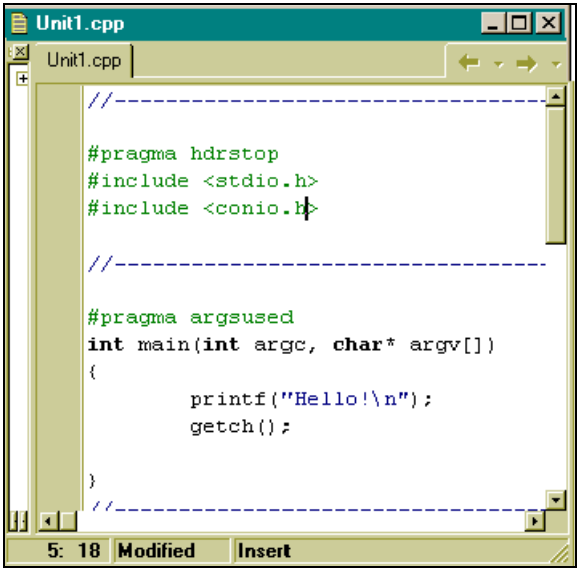

Рис. 1.6. Текст программы
после подключения необходимых библиотек

Запускаем клавишей <F9> компилятор, результат показан на рис. 1.7.

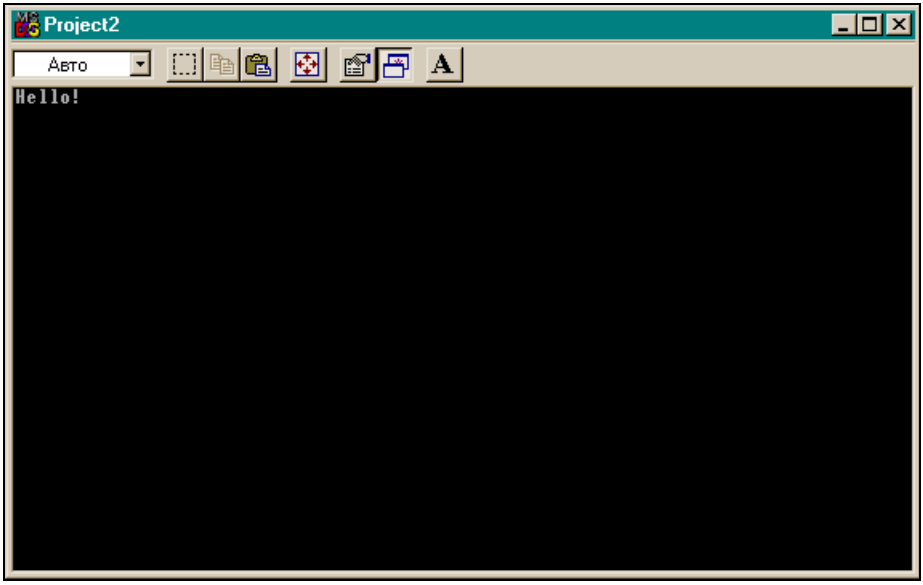

Рис. 1.7. Результат выполнения первой программы

Наша программа успешно откомпилирована и выполнена. В результате ее выполнения в окне черного цвета высветился текст "Hello!". Если теперь нажать любую клавишу, программа завершится, и мы снова увидим ее текст. Сохраним новый проект, выполнив опции **File|Save All**.

Поясним суть программы. Мы уже говорили выше, что любая С-программа строится как множество элементов, называемых функциями, — блоков программных кодов, выполняющих определенные действия. Имена этих блоков кодов, построенных по специальным правилам, задает либо программист, если он сам их конструирует, либо имена уже заданы в поставленной со средой программирования библиотеке стандартных функций. Имя главной функции, с которой собственно и начинается выполнение приложения, задано в среде программирования. Это имя — `main()`. В процессе выполнения программы сама функция `main()` обменивается данными с другими функциями и пользуется их результатами. Обмен данными между функциями происходит через параметры функций, которые указываются в круглых скобках, расположенных вслед за именем функции. Функция может и не иметь параметров, но круглые скобки после имени всегда должны присутствовать: по ним компилятор узнает, что перед ним функция, а не что-либо другое. В нашем примере две функции, использованные в главной функции `main()`: это функция `printf()` и функция `getch()`.

Функция `printf()` в качестве аргумента имеет строку символов (символы, заключенные в двойные кавычки). Среди символов этой строки есть специальный символ, записанный так: `\n`. Это так называемый *управляющий символ* — один из первых 32-х символов таблицы кодировки символов ASCII. Управляющие символы не имеют экранного отображения и используются для управления процессами. В данном случае символ `\n` служит для выбрасывания *буфера функции* `printf()`, в котором находятся остальные символы строки, на экран и установки указателя изображения символов на экране в первую позицию — в начало следующей строки. То есть когда работает функция `printf()`, символы строки по одному записываются в некоторый буфер до тех пор, пока не встретится символ `\n`. Как только символ `\n` прочтен, содержимое буфера тут же передается на устройство вывода (в данном случае — на экран).

Функция `getch()` — это функция ввода одного символа с клавиатуры: она ждет нажатия какой-либо клавиши. Благодаря этой функции результат выполнения программы задерживается на экране до тех пор, пока мы не нажмем любой символ на клавиатуре. Если бы в коде не было функции `getch()`, то после выполнения `printf()` программа дошла бы до конца тела функции `main()`, до закрывающей фигурной скобки, и завершила бы свою работу. В результате этого черное окно, в котором вывелось сообщение Hello!, закрылось бы, и мы не увидели бы результата работы программы. Следовательно, когда мы захотим завершить нашу программу, мы должны нажать любой символ на клавиатуре, программа выполнит функцию `getch()` и перейдет к выполнению следующего оператора. А это будет конец тела `main()`. На этом программа и завершит свою работу. Следует отметить, что основное назначение функции `getch()` — вводить символы с клавиатуры и передавать их символьным переменным, о которых пойдет речь ниже. Но мы воспользовались побочным свойством функции — ждать ввода с клавиатуры и, тем самым, не дать программе завершиться, чтобы мы посмотрели результат ее предыдущей работы.

Программа с оператором *while*

Рассмотрим программу вывода таблицы температур по Фаренгейту и Цельсию.

Формула перевода температур такова: $C = (5 : 9) \times (F - 32)$, где C — это температура по шкале Цельсия, а F — по шкале Фаренгейта. Задается таблица температур по Фаренгейту: 0, 20, 40, ..., 300. Требуется вычислить таблицу по шкале Цельсия и вывести на экран обе таблицы.

Создаем заготовку консольного приложения и сохраняем его описанным выше способом (как в простейшей программе, которую мы разработали выше).

Записываем код новой программы в тело главной функции (листинг 1.1).

Листинг 1.1

```
//---------------------------------------------------------------------------

#pragma hdrstop

#include <stdio.h>
#include <conio.h>

//---------------------------------------------------------------------------



main()
{
  int lower,upper,step;
  float fahr,cels;
  lower=0;
  upper=300;
  step=20;
  fahr=lower;
  while(fahr <= upper)
   {
    cels=(5.0/9.0)*(fahr-32.0);
    printf("%4.0f %6.1f\n",fahr,cels);
    fahr=fahr+step;
   }
  getch();

}
//---------------------------------------------------------------------------
```

Запускаем компилятор клавишей <F9>. Программа откомпилируется и выполнится. Результат высветится в окне (рис. 1.8).

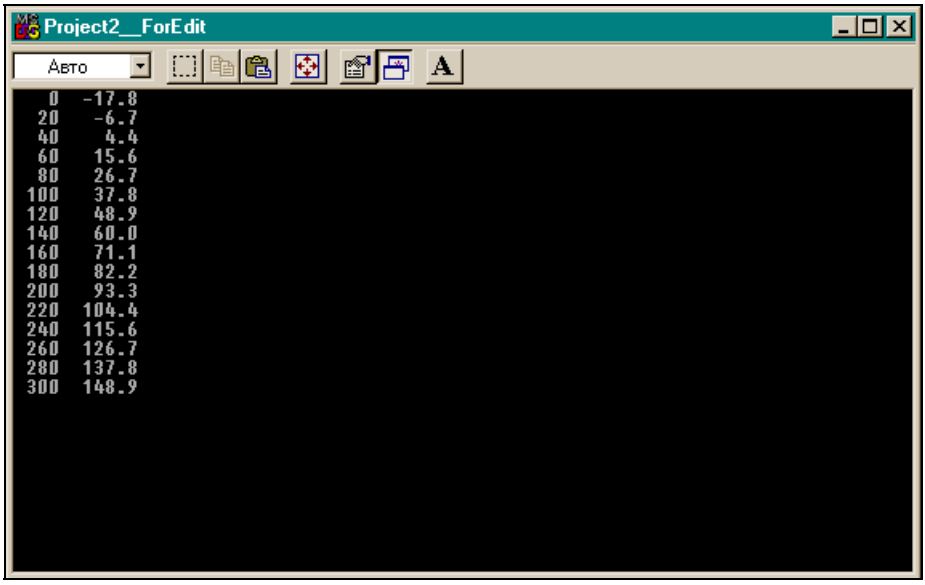

Рис. 1.8. Результат расчета таблицы температур по Цельсию

Имена и типы переменных

Поясним суть программы.

`int lower,upper,step;` — это так называемые "объявления переменных". `lower,upper,step` — имена переменных. Компилятор соотнесет с этими именами определенные адреса в памяти и, начиная с этих адресов, выделит участки памяти (в байтах) в соответствии с тем, какого типа объявлены переменные. В нашем случае тип переменных, заданный при их объявлении, — `int` (от англ. *integer* — целое число). Это означает, что все переменные имеют вид "целое число со знаком" и что под каждое значение числа, которое будет записано на участках `lower`, `upper` или `step`, отведено по 2 байта. Таким образом, имена переменных — это названия тех полочек в памяти компьютера (а каждая полочка имеет свой адрес), где будут находиться данные (числа и не числа), с которыми программа будет работать при реализации алгоритма.

Имена переменным надо давать осмысленно — так, чтобы они отражали характер содержания переменной. В нашем случае `lower`, `upper` и `step` именуют соответственно нижнюю и верхнюю границы таблицы температур по Фаренгейту и шаг этой таблицы. Нижняя граница таблицы (`lower`) равна 0, верхняя (`upper`) — 300, а шаг таблицы (т. е. разность между соседними значениями — `step`) равен 20.

Перечень описываемых переменных одного типа (тип указывается в начале перечня) обязательно должен оканчиваться *точкой с запятой* — сигналом для компилятора, что описание переменных данного типа завершено. В языке С выражение, после которого стоит точка с запятой, считается оператором, т. е. законченным действием. В противном случае компилятор станет при компиляции искать ближайшую точку с запятой и объединять все, что до нее находится, в один оператор (в общем, объединятся разнородные данные) и, в конце концов, выдаст ошибку компиляции.

`float fahr,cels;` — описание переменных с именами `fahr`, `cels`, но тип этих переменных уже иной. Эти переменные — не целые числа, а так называемые числа "с плавающей точкой". "Полочки" в памяти, обозначаемые этими переменными, могут хранить любые вещественные числа, а не только целые. Под этот тип данных компилятор отводит по 4 байта.

Таким образом, перед составлением программы, которая будет оперировать данными (числовыми и нечисловыми), *эти данные следует описать*: им должны быть присвоены типы и имена. Присвоение переменным типов и имен фактически означает, что компилятор определит им место в памяти, куда данные будут помещаться и откуда будут извлекаться при выполнении операций над ними. Следовательно, когда мы пишем $c = a + b$, это означает, что одна часть данных будет извлечена с "полочки" с именем *a,* другая часть данных — с "полочки" с именем *b*, произойдет их суммирование, и результат будет "положен" (записан) на "полочку" с именем *c*. Знак "=" означает "присвоить", это не знак равенства, а операция пересылки. Знак равенства выглядит иначе (о знаке равенства подробно поговорим в *главе 2*). Присваивать некоторой переменной можно не только значение с какой-либо "полочки", т. е. значение другой переменной, но и просто числа. Например, $a = 10$. В этом случае, компилятор просто "положит на полочку" *a* число 10.

Оператор *while*

Чтобы вычислить температуру по Цельсию для каждого значения шкалы по Фаренгейту, не требуется писать программный код для каждой точки шкалы. В этом случае никакой памяти не хватило бы, поскольку шкала может содержать миллиарды точек. В таких случаях выходят из положения так: вычисляют для одной точки, используя некоторый параметр, а потом, изменяя этот параметр, заставляют участок расчета снова выполняться до тех пор, пока параметр не примет определенного значения, после которого повторение расчетов прекращают. Повторение расчетов называют *циклом расчетов*. Для организации циклов существуют специальные операторы цикла, которые "охватывают" участок расчета и "прокручивают" его необходимое количество раз. Одним из таких операторов в языке С является оператор `while` (англ. — до тех пор, пока). Тело этого оператора ограничивается парой фи-

гурных скобок: начинается с открывающей фигурной скобки, а заканчивается закрывающей фигурной скобкой. В это-то тело и помещается прокручиваемый участок. А сколько раз "прокручивать" — определяется *условием окончания цикла*, которое задается в заголовочной части оператора. Вид оператора `while` таков:

```
while(условие окончания цикла)
{
  Тело
}
```

Работает оператор так: в начале проверяется условие окончания цикла. Если оно истинно, то тело оператора выполняется. Если условие окончания цикла ложно, то выполнение оператора прекращается, и начинает выполняться программный код, расположенный непосредственно после закрывающей скобки тела оператора.

Приведем пример истинности условия. Условие может быть записано в общем случае в виде некоторого выражения (переменные, соединенные между собой знаками операций). Например, $a < b$ (*a* меньше *b*). Значение переменной *a* — это то, что лежит на полочке с именем *a*, а значение переменной *b* — то, что лежит на полочке *b*. Если значение переменной *a* действительно меньше значения *b*, то выражение считается истинным, в противном случае — ложным.

Внимательно посмотрев на оператор `while`, можно сделать вывод: для завершения цикла (для этого условие окончания цикла должно стать ложным), надо, чтобы само условие окончания изменялось в теле оператора по мере выполнения цикла и в нужный момент стало бы ложным. Теперь рассмотрим, как это происходит в нашей программе.

Сперва определяются начальные значения переменных `lower`, `upper`, `step`. Параметром, задающим цикл, у нас является переменная `fahr`: ее значение будет меняться от цикла к циклу на величину шага шкалы по Фаренгейту, начиная от минимального, когда `fahr` = `lower` (мы присваиваем ей значение переменной `lower`, которая ранее получила значение нуля — начала шкалы по Фаренгейту), и заканчивая максимальным, когда значение переменной `lower` достигнет значения переменной `upper`, которое мы в начале указали равным 300. Поэтому условие окончания цикла в операторе цикла `while` будет таковым: "пока значение `fahr` не превзойдет значения переменной `upper`". На языке C это записывается в виде

```
while(fahr <= upper)
```

В теле же самого оператора цикла мы записываем на языке C: формулу вычисления значения переменной `cels` (т. е. точки шкалы по Цельсию), функцию `printf()` для вывода значений точек по Фаренгейту и Цельсию,

переменную fahr для изменения параметров цикла: она добавляет значение шага шкалы по Фаренгейту, что подготавливает переход к вычислению переменной cels для нового значения переменной fahr. Это произойдет тогда, когда программа дойдет до выполнения конца тела оператора while (т. е. до закрывающей фигурной скобки) и перейдет к выполнению выражения, стоящего в заголовочной части while и проверке его на истинность/ложность. Если истинность выражения-условия не нарушилась, начнет снова выполняться тело оператора while. Когда же переменная fahr примет значение больше значения upper, цикл завершится: начнет выполняться код, следующий за телом оператора while. А это будет функция getch(), которая потребует ввода символа с клавиатуры, тем самым задерживая закрытие окна, в котором, благодаря функции printf(), появились результаты работы программы. Как только мы нажмем на любую клавишу, функция getch() получит то, что ждала, в результате чего она завершится. Затем начнет выполняться закрывающая скобка тела главной функции main(). После ее обработки наше приложение окончит свою работу.

Поясним операции, примененные при формировании переменной cels. Это арифметические операции деления (/), умножения (*), вычитания (-). Операция деления имеет одну особенность: если ее операнды имеют тип int, то ее результат — всегда целое число, т. к. в этом случае остаток от деления отбрасывается. Поэтому, если бы мы в формуле для вычисления переменной cels записали 5/9, то получили бы 0, а не 0,55. Чтобы этого не случилось нам пришлось "обмануть" операцию деления: мы записали 5.0/9.0, так, будто операнды — в формате плавающей точки. Для таких операндов остаток от деления не отбрасывается.

Функция printf() в общем случае имеет такой формат:

```
printf(Control,arg1,arg2,...,argN);
```

Control — это строка символов, заключенных в двойные кавычки, arg1,arg2,...,argN — имена переменных, значения которых должны быть выведены на устройство вывода. Строка Control содержит в себе данные двух родов: указания на формат переменных arg1,arg2,...,argN (указания на формат расположены в том же порядке, что и переменные arg1,arg2,...,argN), и остальные символы, которые выводятся без всякого форматирования (т. е. без преобразования в другую форму). Обозначение формата всегда начинается с символа %, а заканчивается символом типа форматирования: d — для переменных типа int, f — для float, s — для строк символов и т. д.

Между символом % и символом типа форматирования задается ширина поля вывода, количество знаков после точки (для типа f) и т. д. Полное определение форматов можно посмотреть в разделе **Help** таким же образом, как ранее мы искали описания функций. Так как переменные cels и fahr относятся к

типу `float`, то и в функции `printf()` указан соответствующий формат — `f`. Значение переменной `fahr` выводится целым числом в поле шириной 4 байта, а значение переменной `cels`, имеющее в результате расчетов дробное значение, выводится в поле шириной 6 байт с одним знаком после точки.

Оператор *for*

Кроме оператора `while` цикл позволяет организовать оператор `for`. Перепишем программу расчета температур, рассмотренную выше, в несколько другом виде (листинг 1.2).

Листинг 1.2

```
//---------------------------------------------------------------------------

#pragma hdrstop

#include <stdio.h>
#include <conio.h>

//---------------------------------------------------------------------------



main()
{
  int fahr;
  for(fahr=0; fahr <= 300; fahr= fahr + 20)
   printf("%4d  %6.1f\n",fahr,(5.0/9.0)*(fahr-32.0));
  getch();

}
//---------------------------------------------------------------------
```

Здесь для получения того же результата, что и в предыдущем случае, применен оператор цикла `for`. Тело этого оператора, как и тело оператора `while`, циклически выполняется ("прокручивается"). В нашем случае тело `for` состоит всего из одного оператора — `printf()`, поэтому такое тело не берется в фигурные скобки (если бы тело оператора `while` состояло только из одного оператора, оно тоже не бралось бы в скобки).