

Б. И. Пахомов

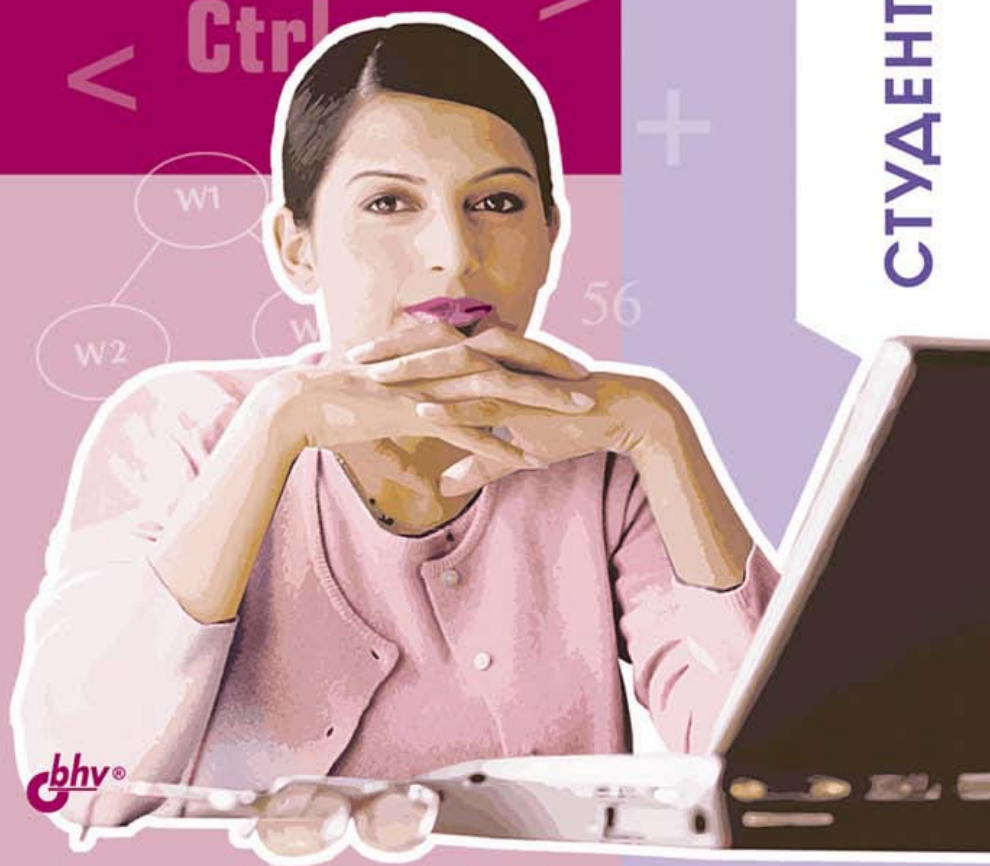
C/C++ и Borland C++ Builder

#include <vcl.h>

< Ctrl

>

ДЛЯ
СТУДЕНТА



УДК 681.3.06+800.92(075.8)C++
ББК 32.973.26-018.1я73
П12

Пахомов Б. И.

П12 C/C++ и Borland C++ Builder для студента. — СПб.:
БХВ-Петербург, 2006. — 448 с.: ил.

ISBN 5-94157-816-4

Книга является руководством для студентов, начинающих изучать среду Borland C++ Builder и языки программирования C/C++. Рассмотрены основные элементы языков 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- и интернет-приложений, в том числе приложений с использованием технологий BDE и MIDAS.

Для студентов, изучающих курс информационных технологий

УДК 681.3.06+800.92(075.8)C++
ББК 32.973.26-018.1я73

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Алия Шаулис</i>
Компьютерная верстка	<i>Натали Смирновой</i>
Корректор	<i>Наталия Першакова</i>
Дизайн серии	<i>Игоря Цырульникова</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 17.01.06.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 28.

Тираж 3000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 5-94157-816-4

© Пахомов Б. И., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Введение	1
----------------	---

ЧАСТЬ I. ОСНОВЫ АЛГОРИТМИЧЕСКИХ ЯЗЫКОВ

С И C++	7
---------------	---

Глава 1. Типы данных, простые переменные и основные операторы цикла	9
--	---

Как перейти к созданию консольного приложения	9
Формирование проекта консольного приложения	11
Создание простейшего консольного приложения	12
Программа с оператором <i>while</i>	16
Имена и типы переменных	18
Оператор <i>while</i>	20
Оператор <i>for</i>	23
Символические константы	25

Глава 2. Программы для работы с символьными данными	27
---	----

Программа копирования символьного файла. Вариант 1	29
Программа копирования символьного файла. Вариант 2	32
Подсчет символов в файле. Вариант 1	33
Подсчет символов в файле. Вариант 2	35
Подсчет количества строк в файле	37
Подсчет количества слов в файле	39

Глава 3. Работа с массивами данных	43
--	----

Одномерные массивы	43
Многомерные массивы	47

Глава 4. Создание и использование функций49

Создание некоторых функций.....	52
Ввод строки с клавиатуры	52
Функция выделения подстроки из строки	56
Функция копирования строки в строку.....	58
Головная программа для проверки функций <i>getline()</i> , <i>substr()</i> , <i>copy()</i>	59
Внешние и внутренние переменные.....	62
Область действия переменных.....	65
Как создать свой внешний файл	66
Атрибут <i>static</i>	67
Рекурсивные функции.....	69
Быстрый вызов функций.....	70

Глава 5. Основные стандартные функции для работы с символьными строками.....71

Функция <i>sprintf (s, Control,arg1,arg2,,argN)</i>	71
Функция <i>strcpy(s1,s2)</i>	71
Функция <i>strcmp(s1,s2)</i>	72
Функция <i>strcmpi(s1,s2)</i>	73
Функция <i>strcat(s1,s2)</i>	73
Функция <i>strlen(s)</i>	73
Пример программы проверки функций	73

Глава 6. Дополнительные сведения о типах данных, операциях, выражениях и элементах управления.....79

Новые типы переменных	79
Константы.....	83
Новые операции	84
Операции отношения.....	84
Логические операции.....	85
Унарная операция отрицания	85
Преобразование типов данных.....	86
Побитовые логические операции.....	88
Операции и выражения присваивания	89
Условное выражение.....	91
Операторы и блоки	91

Конструкция <i>if-else</i>	92
Конструкция <i>else-if</i>	92
Переключатель <i>switch</i>	97
Уточнение по работе оператора <i>for</i>	102
Оператор <i>continue</i>	102
Оператор <i>goto</i> и метки	103

Глава 7. Работа с указателями и структурами данных..... 104

Указатель.....	104
Указатели и массивы.....	106
Операции над указателями.....	108
Указатели и аргументы функций.....	108
Указатели символов и функций.....	110
Передача в качестве аргумента функции массивов размерности больше единицы.....	115
Массивы указателей.....	116
Указатели на функции.....	117
Структуры.....	120
Объявление структур.....	120
Обращение к элементам структур.....	122
Структуры и функции.....	125
Программы со структурами.....	126
Функция возвращает структуру.....	126
Функция возвращает указатель на структуру.....	130
Программы упрощенного расчета заработной платы одного работника.....	133
Рекурсия в структурах.....	135
Битовые поля в структурах.....	143

Глава 8. Классы в C++ 145

Объектно-ориентированное программирование	145
Классы.....	145
Принципы построения классов.....	147
Инкапсуляция.....	147
Наследование.....	148
Полиморфизм.....	151

Пример создания классов.....	152
Создание и использование класса "не компонента"	152
Создание и использование класса "компонента"	154
Глава 9. Ввод и вывод в С и С++	159
Ввод и вывод в С.....	159
Ввод-вывод файлов	159
Функции для работы с файлами	160
Стандартный ввод-вывод.....	161
Функции стандартного ввода-вывода	163
Ввод-вывод в С++	163
Общие положения	163
Ввод-вывод с использованием разных классов.....	165
Пространства имен	165
Работа с классом <i>fstream</i>	167
Работа с классом <i>ofstream</i>	171
Работа с классом <i>ifstream</i>	172
Работа с бинарным файлом	175
Стандартный ввод-вывод в С++	177
Общие положения.....	177
Стандартный вывод <i>cout</i>	178
Стандартный ввод <i>cin</i>	179
 ЧАСТЬ II. ПРОГРАММИРОВАНИЕ	
В СРЕДЕ BORLAND C++ BUILDER.....	181
 Глава 10. Начало изучения среды Borland C++ Builder	183
Как приступить к разработке нового приложения.	
Создание проекта	183
Файлы проекта	185
Инспектор объекта.....	188
Вкладка <i>Properties</i>	189
Вкладка <i>Events</i>	190
Работа с Инспектором объекта	193
Редактор кода, сpp-модуль и h-файл	193
Как начать редактирование текста программного модуля	200

Контекстное меню Редактора кода	201
Суфлер кода (подсказчик)	203
Класс <i>TForm</i>	206
Дизайнер форм	206
Помещение компонента в форму	206
Другие действия с Дизайнером форм	207
Контекстное меню формы	208
Добавление новых форм к проекту	212
Организация работы с множеством форм	214
Вызов формы на выполнение	216
Свойства формы	216
События формы	228
Методы формы	230

Глава 11. Компоненты, создающие интерфейс

между пользователем и приложением 232

Компонент <i>TButton</i>	232
Некоторые свойства <i>TButton</i>	236
Некоторые события <i>TButton</i>	237
Некоторые методы <i>TButton</i>	237
Как сделать вывод текста в поле кнопки многострочным	237
Компонент <i>TPanel</i>	239
Некоторые свойства <i>TPanel</i>	239
Некоторые события <i>TPanel</i>	240
Методы <i>TPanel</i>	241
Компонент <i>TLabel</i>	241
Некоторые свойства <i>TLabel</i>	241
События <i>TLabel</i>	242
Компонент <i>TEdit</i>	242
Некоторые свойства <i>TEdit</i>	242
События <i>TEdit</i>	243
Некоторые методы <i>TEdit</i>	244
Компонент <i>TMainMenu</i>	244
Некоторые свойства <i>TMainMenu</i>	247
Некоторые свойства опций <i>TMainMenu</i>	248
События <i>TMainMenu</i>	249

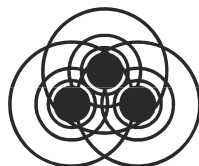
Компонент <i>TPopupMenu</i>	249
Свойства <i>TPopupMenu</i>	250
События и методы <i>TPopupMenu</i>	251
Компонент <i>TMemo</i>	251
Некоторые свойства <i>TMemo</i>	251
События и методы <i>TMemo</i>	253
Задача регистрации пользователя в приложении.....	253
Регистрация пользователя.....	254
Приложение.....	259
Некоторые функции выдачи сообщений и перевода данных из одного типа в другой.....	272
Компонент <i>TListBox</i>	275
Как использовать <i>TListBox</i>	276
Как формировать список строк.....	276
Некоторые свойства <i>TListBox</i>	277
События <i>TListBox</i>	278
Некоторые методы <i>TListBox</i>	278
Включение горизонтальной полосы прокрутки списка.....	278
Компонент <i>TComboBox</i>	279
Компонент <i>TMaskEdit</i>	280
Задание маски.....	283
Поля и кнопки в окне Редактора маски.....	284
Куда помещается текст, введенный по маске.....	284
Проверочная программа.....	284
Компонент <i>TCheckBox</i>	286
Компонент <i>TRadioButton</i>	288
Компонент <i>TRadioGroup</i>	290
Компонент <i>TCheckListBox</i>	291
Компонент <i>TImage</i>	292
Некоторые свойства <i>TImage</i>	292
Компонент <i>TShape</i>	296
Компонент <i>TBevel</i>	296
Компонент <i>TPageControl</i>	297
Как задавать страницы.....	297
Некоторые свойства страницы <i>TTabSheet</i>	298
Некоторые свойства <i>TPageControl</i>	298
Некоторые события <i>TPageControl</i>	300

Компонент <i>TOpenDialog</i>	300
Некоторые свойства <i>TOpenDialog</i>	302
Некоторые события <i>TOpenDialog</i>	304
Компонент <i>TSaveDialog</i>	304
Компонент <i>TOpenPictureDialog</i>	306
Компонент <i>TSavePictureDialog</i>	306
Компонент <i>TFontDialog</i>	306
Некоторые свойства <i>TFontDialog</i>	307
Некоторые события <i>TFontDialog</i>	308
Компонент <i>TColorDialog</i>	308
Некоторые свойства <i>TColorDialog</i>	309
События <i>TColorDialog</i>	310
Компонент <i>TPrintDialog</i>	310
Свойства <i>TPrintDialog</i>	310
Компонент <i>TPrinterSetupDialog</i>	311
OLE-объекты.....	311
Некоторые свойства OLE-контейнера.....	312
Выбор объекта для вставки в контейнер.....	316
Компонент <i>TUpDown</i>	316
Некоторые свойства <i>TUpDown</i>	317
Компонент <i>TTimer</i>	318
Компонент <i>TProgressBar</i>	319
Компонент <i>TDateTimePicker</i>	321
Некоторые свойства <i>TDateTimePicker</i>	322
О работе с датами.....	323
Примеры использования дат.....	324
Пример 1.....	324
Пример 2.....	325
Пример 3.....	326
Пример 4.....	326
Пример 5.....	326
Пример 6.....	327
Глава 12. Базы данных.....	328
Что такое база данных.....	328
Создание базы данных.....	329
Создание таблицы базы данных.....	331
Задание полей таблицы.....	331

Поле <i>Field Name</i>	331
Поле <i>Type</i>	331
Поле <i>Key</i>	336
Другие элементы диалогового окна для создания таблицы	336
Элементы <i>Table properties</i>	336
Кнопка <i>Borrow</i>	340
Компоненты работы с базой данных	340
Компонент <i>TTable</i>	340
Некоторые свойства <i>TTable</i>	341
Как настраивать компонент <i>TTable</i> на конкретную таблицу базы данных	347
Некоторые методы <i>TTable</i>	348
Компонент <i>TDataSource</i>	353
Компонент <i>TDBGrid</i>	354
Некоторые свойства <i>TDBGrid</i>	354
Компонент <i>TDBNavigator</i>	355
Как используется <i>TDBNavigator</i>	355
О компонентах работы с полями набора данных.....	355
Пример ввода данных в таблицу.....	356
Компонент <i>TQuery</i>	360
Некоторые свойства <i>TQuery</i>	360
Методы <i>TQuery</i>	366
Общие сведения о хранимых процедурах.....	367
Глава 13. Вывод отчетов.....	369
Получение простейшего отчета	369
Свойства <i>TQuickRep</i>	372
Формирование отчета	374
Свойства <i>TQRDBText</i>	374
Пример отчета, печатающего изображения	378
Глава 14. Некоторые компоненты вкладки <i>Internet</i>	380
Компонент <i>TServerSocket</i>	380
Свойства <i>TServerSocket</i>	381
Компонент <i>TClientSocket</i>	384
Свойства <i>TClientSocket</i>	384
События <i>TClientSocket</i>	385

Пример соединения по протоколу TCP/IP	386
Компонент <i>TCppWebBrowser</i>	396
Пример приложения, запускающего Internet Explorer для вывода локального документа	397
Глава 15. Примеры из технологии MIDAS	401
Компонент <i>TDataSetProvider</i>	401
Свойства <i>TDataSetProvider</i>	401
Компонент <i>TClientDataSet</i>	404
Свойства <i>TClientDataSet</i>	404
Компонент <i>TDCOMConnection</i>	407
Свойства <i>TDCOMConnection</i>	407
Компонент <i>TSocketConnection</i>	410
Свойства <i>TSocketConnection</i>	410
Компонент <i>TWebConnection</i>	412
Свойства <i>TWebConnection</i>	412
Использование компонента <i>TClientDataSet</i> . Пример 1	413
Использование компонента <i>TClientDataSet</i> . Пример 2	415
Предметный указатель	427

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 применяется в процессе разработки программ.

Как перейти к созданию консольного приложения

1. Загрузите среду Borland C++Builder.
2. Выполните команды главного меню: **File/New**. Откроется диалоговое окно **New Items**, показанное на рис. 1.1.

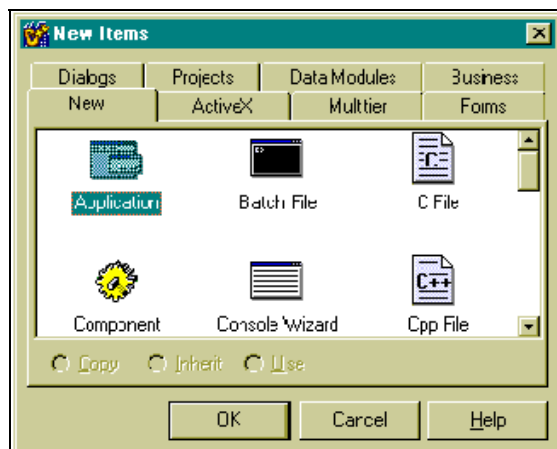


Рис. 1.1. Диалоговое окно **New Items**

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).

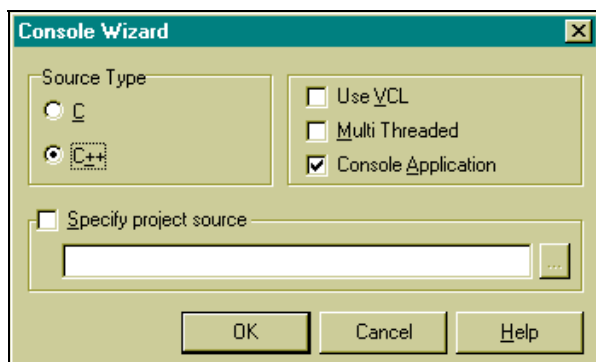


Рис. 1.2. Диалоговое окно Мастера построения заготовок консольных приложений

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

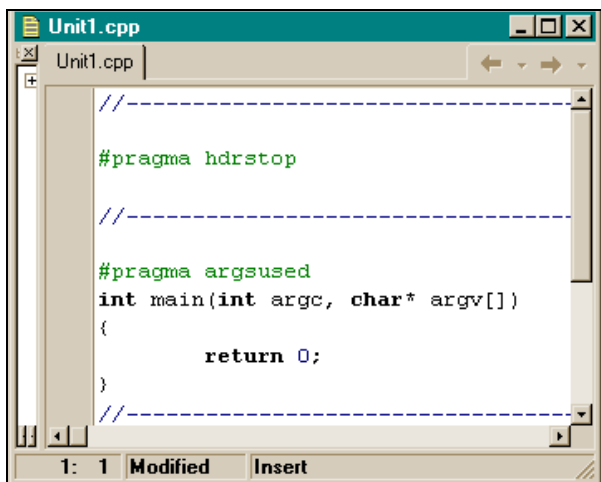


Рис. 1.3. Заготовка консольного приложения

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 сделаем с помощью Редактора кода, который открывается одновременно с появлением заготовки консольного приложения на экране: щелкните кнопкой мыши в любом месте поля заготовки, и вы 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.

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 программно 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 **ОК**. 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.

Чтобы приложение заработало, его надо *откомпилировать*, т. е. перевести то, что мы написали текстом на языке C 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 картину, показанную на рис. 1.5.

На рисунке видно, 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 кото-

рой эта ошибка обнаружена. Разберемся с обнаруженными ошибками.

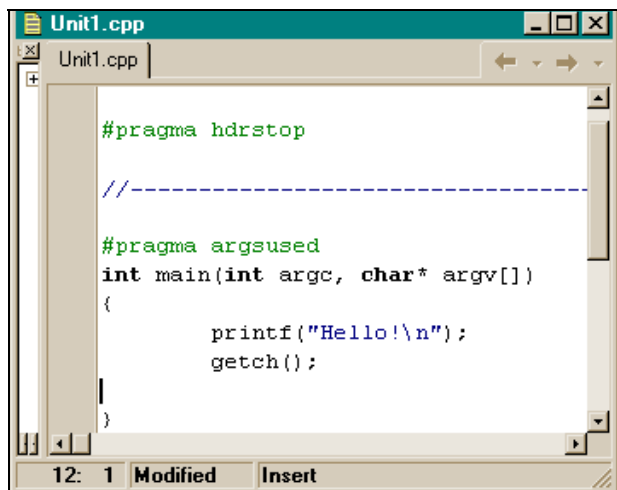


Рис. 1.4. Код консольного приложения до компиляции

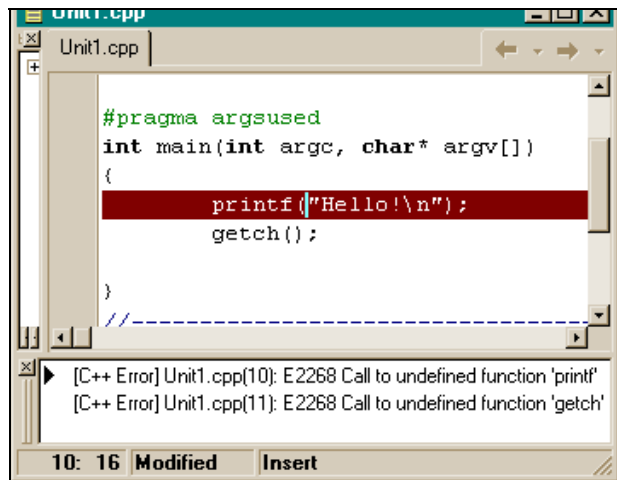


Рис. 1.5. Консольное приложение после неудачной компиляции

Выберем 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. Таким 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.

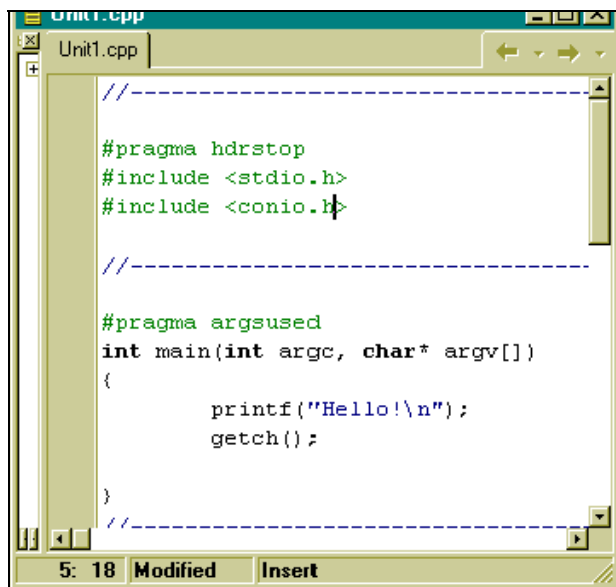


Рис. 1.6. Текст программы
после подключения необходимых библиотек

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



Рис. 1.7. Результат выполнения первой программы

Наша программа успешно откомпилировалась и сразу же выполнялась. В результате ее выполнения в окне черного цвета высветилось слово **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 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: $C = (5 : 9) * (F - 32)$, 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).

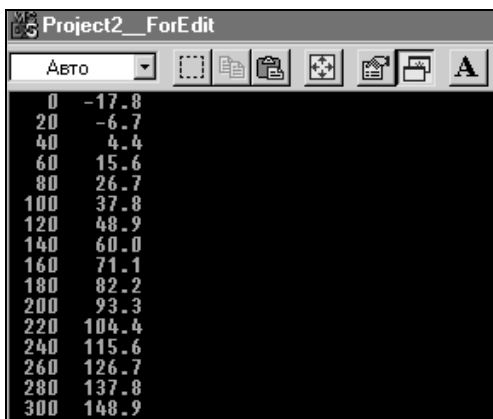
Записываем код новой программы в тело главной функции (листинг 1.1).

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).



0	-17.8
20	-6.7
40	4.4
60	15.6
80	26.7
100	37.8
120	48.9
140	60.0
160	71.1
180	82.2
200	93.3
220	104.4
240	115.6
260	126.7
280	137.8
300	148.9

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 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 (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 (`float`). 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 см. 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 C является оператор *while* (англ. — до тех пор, пока). Тело этого оператора ограничивается парой фигурных 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, где 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 (`-`). У операции деления есть одна особенность: если ее операнды имеют тип `int`, то результат деления — 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`, как будто операнды — числа с плавающей точкой. 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 тип переменных `arg1, arg2, ..., argN` (указания на тип 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).