# 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

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

**Пахомов Б. И.**

П12 С/С++ и Borland С++ 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 С++ Builder. Рассмотрены основные элементы языков программирования С/С++ и примеры создания простейших классов и программ. Изложены принципы визуального проектирования и событийного программирования. На конкретных примерах показаны основные возможности визуальной среды разработки С++ 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+800.92С,С++
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

© Пахомов Б. И., 2005
© Оформление, издательство "БХВ-Петербург", 2005

# Содержание

**Введение** ... **1**

**Часть I. Алгоритмический язык С и его расширение С++** ... **5**

**Глава 1. Типы данных, простые переменные и основные операторы цикла** ... **7**

Как перейти к созданию консольного приложения ... 7
Формирование проекта консольного приложения ... 9
Создание простейшего консольного приложения ... 9
Программа с оператором *while* ... 13
Имена и типы переменных ... 15
Оператор *while* ... 16
Оператор *for* ... 19
Символические константы ... 20

**Глава 2. Программы для работы с символьными данными** ... **22**

Программа копирования символьного файла. Вариант 1 ... 24
Программа копирования символьного файла. Вариант 2 ... 26
Подсчет символов в файле. Вариант 1 ... 27
Подсчет символов в файле. Вариант 2 ... 29
Подсчет количества строк в файле ... 31
Подсчет количества слов в файле ... 33

**Глава 3. Работа с массивами данных** ... **37**

Одномерные массивы ... 37
Многомерные массивы ... 40

**Глава 4. Создание и использование функций** ... **42**

Создание некоторых функций ... 44
Ввод строки с клавиатуры ... 44
Функция выделения подстроки из строки ... 47
Функция копирования строки в строку ... 48
Головная программа для проверки функций *getline()*, *substr()*, *copy()* ... 49
Внешние и внутренние переменные ... 52
Область действия переменных ... 55
Как создать свой внешний файл ... 56
Атрибут *static* ... 57
Рекурсивные функции ... 58
Быстрый вызов функций ... 59

**Глава 5. Основные стандартные функции для работы с символьными строками ....60**

Функция *sprintf (s, Control, arg1, arg2,, argN)* ....60
Функция *strcpy(s1,s2)* ....60
Функция *strcmp(s1,s2)* ....61
Функция *strcmpi(s1,s2)* ....61
Функция *strcat(s1,s2)* ....61
Функция *strlen(s)* ....61
Пример программы проверки функций ....62

**Глава 6. Дополнительные сведения о типах данных, операциях, выражениях и элементах управления ....67**

Новые типы переменных ....67
Константы ....70
Новые операции ....71
Преобразование типов данных ....72
Побитовые логические операции ....74
Операции и выражения присваивания ....74
Условное выражение ....76
Операторы и блоки ....77
Конструкция *if...else* ....77
Конструкция *else...if* ....77
Переключатель *switch* ....82
Уточнение по работе оператора *for* ....85
Оператор *continue* ....85
Оператор *goto* и метки ....86

**Глава 7. Работа с указателями и структурами данных ....87**

Указатель ....87
Указатели и массивы ....88
Операции над указателями ....90
Указатели и аргументы функций ....90
Указатели символов и функций ....92
Передача в качестве аргумента функции массивов размерности больше единицы ....97
Массивы указателей ....97
Указатели на функции ....98
Структуры ....101
Объявление структур ....101
Обращение к элементам структур ....103
Структуры и функции ....105
Программы со структурами ....106
Рекурсия в структурах ....114
Битовые поля в структурах ....121

**Глава 8. Классы в C++ ..... 123**
Объектно-ориентированное программирование ..... 123
Классы ..... 123
Принципы построения классов ..... 124
Пример создания классов ..... 128

**Глава 9. Ввод и вывод в C и C++ ..... 134**
Ввод и вывод в C ..... 134
Ввод/вывод файлов ..... 134
Стандартный ввод/вывод ..... 140
Ввод/вывод в C++ ..... 154
Общие положения ..... 154
Ввод/вывод с использованием разных классов ..... 155
Стандартный ввод/вывод в C++ ..... 166

**ЧАСТЬ II. СРЕДА BORLAND C++ BUILDER ..... 175**

**Глава 10. Начало изучения среды Borland C++ Builder ..... 177**
Как приступить к разработке нового приложения. Создание проекта ..... 177
Файлы проекта ..... 179
Инспектор объекта ..... 181
Вкладка *Properties* ..... 182
Вкладка *Events* ..... 183
Работа с Инспектором объекта ..... 185
Редактор кода, cpp-модуль и h-файл ..... 186
Как начать редактирование текста программного модуля ..... 191
Контекстное меню Редактора кода ..... 192
Суфлер кода (подсказчик) ..... 194
Класс *TForm* ..... 196
Дизайнер форм ..... 196
Помещение компонента в форму ..... 197
Другие действия с Дизайнером форм ..... 197
Контекстное меню формы ..... 198
Добавление новых форм к проекту ..... 202
Организация работы с множеством форм ..... 203
Вызов формы на выполнение ..... 204
Свойства формы ..... 205
События формы ..... 215
Методы формы ..... 216
Компонент *TButton* ..... 217
Свойства *TButton* ..... 217
События *TButton* ..... 218
Методы *TButton* ..... 218
Как сделать вывод текста в поле кнопки многострочным ..... 219

**Глава 11. Компоненты *TPanel*, *TLabel*, *TEdit*, *TMainMenu*, *TPopupMenu*, *TMemo*..... 221**

Компонент *TPanel*..... 221
Свойства *TPanel*..... 221
События *TPanel*..... 223
Методы *TPanel*..... 224
Компонент *TLabel*..... 225
Свойства *TLabel*..... 226
События *TLabel*..... 227
Компонент *TEdit*..... 227
Свойства *TEdit*..... 228
События *TEdit*..... 229
Методы *TEdit*..... 229
Компонент *TMainMenu*..... 230
Свойства *TMainMenu*..... 233
События *TMainMenu*..... 236
Компонент *TPopupMenu*..... 236
Свойства *TPopupMenu*..... 239
События и методы *TPopupMenu*..... 239
Компонент *TMemo*..... 239
Свойства *TMemo*..... 240
События и методы *TMemo*..... 242

**Глава 12. Задача регистрации пользователя в приложении..... 243**

Регистрация пользователя..... 243
Приложение..... 248

**Глава 13. Некоторые функции вывода сообщений и перевода данных из одного типа в другой..... 260**

**Глава 14. Компоненты *TListBox, TComboBox, TMaskEdit*..... 264**

Компонент *TListBox*..... 264
Как использовать *TListBox*..... 264
Как формировать список строк..... 265
Свойства *TListBox*..... 265
События *TListBox*..... 269
Методы *TListBox*..... 269
Включение горизонтальной полосы прокрутки списка..... 269
Компонент *TComboBox*..... 270
Компонент *TMaskEdit*..... 271
Задание маски..... 274

**Глава 15. Компоненты *TCheckBox*, *TRadioButton*, *TRadioGroup*, *TCheckListBox*..... 278**

Компонент *TCheckBox*..... 278
Компонент *TRadioButton*..... 282

Компонент *TRadioGroup* ... 283
Компонент *TCheckListBox* ... 287

**Глава 16. Компоненты *TIMage*, *TShape*, *TBevel* ... 298**

Компонент *TIMage* ... 298
Свойства *TImage* ... 300
Компонент *TShape* ... 303
События *TShape* ... 304
Компонент *TBevel* ... 305
Свойства *TBevel* ... 305

**Глава 17. Компоненты *TPageControl*, *TScrollBar*, *TScrollBox* ... 306**

Компонент *TPageControl* ... 306
Как задавать страницы ... 306
Свойства страницы *TTabSheet* ... 307
Свойства *TPageControl* ... 308
События *TPageControl* ... 310
Компонент *TScrollBar* ... 310
Свойства *TScrollBar* ... 311
События *TScrollBar* ... 312
Компонент *TScrollBox* ... 316
События *TScrollBox* ... 317
Пример приложения ... 317

**Глава 18. Компоненты вкладки *Dialogs* ... 321**

Компонент *TOpenDialog* ... 321
Свойства *TOpenDialog* ... 323
События *TOpenDialog* ... 325
Компонент *TSaveDialog* ... 326
Компонент *TOpenPictureDialog* ... 327
Компонент *TSavePictureDialog* ... 328
Компонент *TFontDialog* ... 328
Свойства *TFontDialog* ... 329
События *TFontDialog* ... 331
Компонент *TColorDialog* ... 331
Свойства *TColorDialog* ... 331
События *TColorDialog* ... 334
Компонент *TPrintDialog* ... 334
Свойства *TPrintDialog* ... 334
События *TPrintDialog* ... 336
Компонент *TPrinterSetupDialog* ... 337

**Глава 19. OLE-объекты ... 338**

Свойства OLE-контейнера ... 339
Выбор объекта для вставки в контейнер ... 345

**Глава 20. Компоненты *TUpDown*, *TTimer*, *TProgressBar*, *TDateTimePicker* ......... 349**
Компонент *TUpDown* ........ 349
Свойства *TUpDown* ........ 349
Компонент *TTimer* ........ 351
Компонент *TProgressBar* ........ 353
Компонент *TDateTimePicker* ........ 354
Свойства *TDateTimePicker* ........ 355

**Глава 21. Примеры работы с датами ........ 358**
Методы класса *TDateTime* ........ 359
Пример 1 ........ 359
Пример 2 ........ 361
Пример 3 ........ 363
Пример 4 ........ 364
Пример 5 ........ 365
Пример 6 ........ 365
Пример 7 ........ 366

**Глава 22. Компоненты *TPaintBox*, *TTreeView* ........ 371**
Компонент *TPaintBox* ........ 371
Свойства *TPaintBox* ........ 371
Методы *TPaintBox* ........ 376
Компонент *TTreeView* ........ 377
Свойства *TTreeView* ........ 379
Работа с узлами. Свойства *TTreeNode* ........ 382

**Глава 23. Базы данных ........ 388**
Что такое база данных ........ 388
Создание базы данных ........ 389
Создание таблицы базы данных ........ 394
Задание полей таблицы ........ 395
Другие элементы диалогового окна для создания таблицы ........ 399
Кнопка *Borrow* ........ 410
Пример создания таблицы БД ........ 411

**Глава 24. Компоненты работы с базой данных ........ 415**
Компонент *TTable* ........ 415
Свойства *TTable* ........ 415
Как настраивать компонент *TTable* на конкретную таблицу базы данных ........ 430
Методы *TTable* ........ 430
Пример работы с *TTable* при расчете заработной платы ........ 437
Компонент *TDataSource* ........ 444
Свойства *TDataSource* ........ 445

Компонент *TDBGrid* ... 446
Свойства *TDBGrid* ... 446
События *TDBGrid* ... 450
Компонент *TDBNavigator* ... 451
Как используется *TDBNavigator* ... 451
Свойства *TDBNavigator* ... 452
О компонентах работы с полями набора данных ... 452
Примеры работы с данными БД ... 453
Пример ввода данных в таблицу ... 453
Пример использования фильтра в таблице ... 458
Пример использования данных Редактора полей таблицы для работы с БД ... 463
Компонент *TQuery* ... 464
Свойства *TQuery* ... 464
Пример запроса с использованием свойства *DataSource* ... 472
Методы *TQuery* ... 473
Запрос на выборку из двух таблиц с применением метода задания диапазона записей в одной таблице ... 474
Общие сведения о хранимых процедурах ... 485

**Глава 25. Компоненты *TDBLookupListBox*, *TDBChart* ... 487**

Компонент *TDBLookupListBox* ... 487
Свойства *TDBLookupListBox* ... 487
Пример применения *TDBLookupListBox* ... 488
Компонент *TDBChart* ... 492
Вкладка *Chart* ... 493
Вкладка *Series* ... 496
Возврат к вкладке *Chart* ... 503
Пример применения диаграммы ... 508

**Глава 26. Вывод отчетов ... 513**

Получение простейшего отчета ... 513
Свойства *TQRBand* ... 514
Свойства *TQuickRep* ... 516
Формирование отчета ... 518
Свойства *TQRDBText* ... 519
Пример отчета, печатающего изображения ... 523

**Глава 27. Переход от BDE к ADO ... 526**

Как перейти на ADO с BDE ... 526
Компонент *TADOConnection* ... 527
Компонент *TADOTable* ... 538
Компонент *TADOQuery* ... 542
Пример работы с БД ... 543

**Глава 28. Некоторые компоненты вкладки *Internet*..... 547**
Компонент *TServerSocket*..... 547
Свойства *TServerSocket*..... 548
Компонент *TClientSocket*..... 550
Свойства *TClientSocket*..... 550
События *TClientSocket*..... 552
Пример соединения по протоколу TCP/IP..... 553
Компонент *TWebDispatcher*..... 562
Свойства *TWebDispatcher*..... 562
Компонент *TPageProducer*..... 565
Компонент *TQueryTableProducer*..... 566
Свойства *TQueryTableProducer*..... 566
Методы *TQueryTableProducer*..... 569
Компонент *TDataSetTableProducer*..... 569
Компонент *TCppWebBrowser*..... 569
Пример приложения, запускающего Internet Explorer для вывода локального документа..... 570

**Глава 29. Примеры из технологии MIDAS..... 574**
Компонент *TDataSetProvider*..... 574
Свойства *TDataSetProvider*..... 574
Компонент *TClientDataSet*..... 577
Свойства *TClientDataSet*..... 577
Компонент *TDCOMConnection*..... 581
Свойства *TDCOMConnection*..... 581
Компонент *TSocketConnection*..... 583
Свойства *TSocketConnection*..... 583
Компонент *TWebConnection*..... 585
Свойства *TWebConnection*..... 585
Использование компонента *TClientDataSet*. Пример 1..... 586
Использование компонента *TClientDataSet*. Пример 2..... 589

**Глава 30. Технология DDE..... 599**
Основы DDE..... 599
Использование DDE..... 599
DDE-серверы..... 604
DDE-клиенты..... 608
Пример установления связи с программой Database Desktop..... 618

**Предметный указатель..... 625**

# Глава 1

# Типы данных, простые переменные и основные операторы цикла

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

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 C — 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.

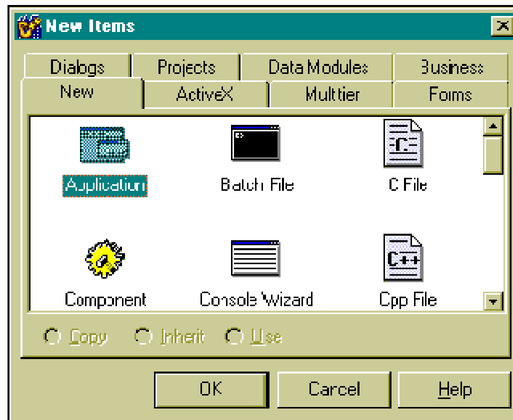


**Рис. 1.1.** Диалоговое окно **New Items**

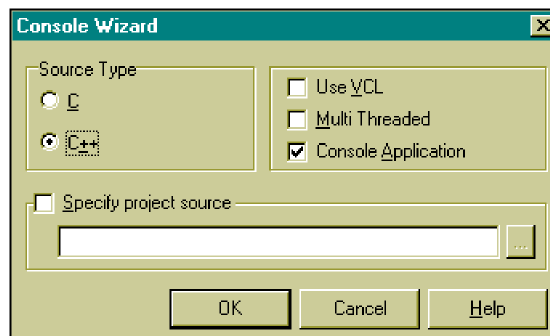


**Рис. 1.2.** Диалоговое окно Мастера построения заготовок консольных прилоржений

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.

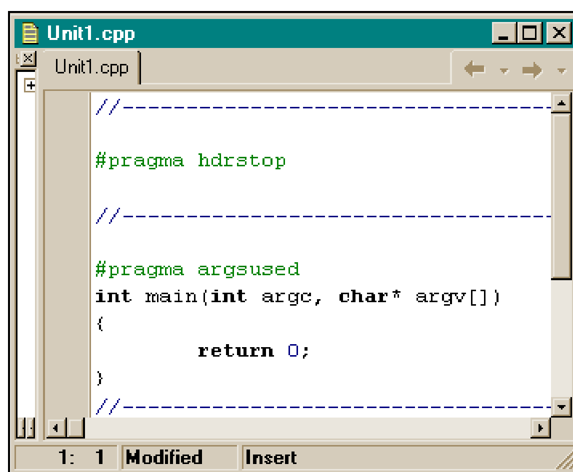


**Рис. 1.3.** Заготовка консольного приложения

## 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.

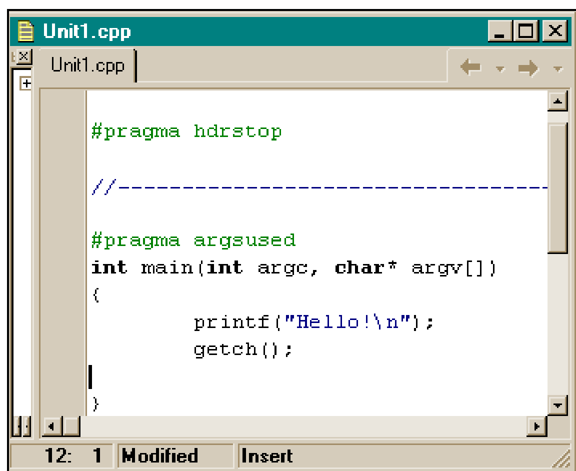


**Рис. 1.4.** Вид консольного приложения до компиляции

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.

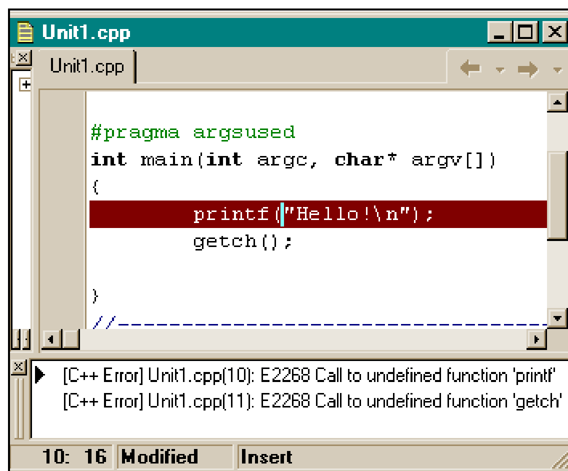


**Рис. 1.5.** Вид консольного приложения после компиляции

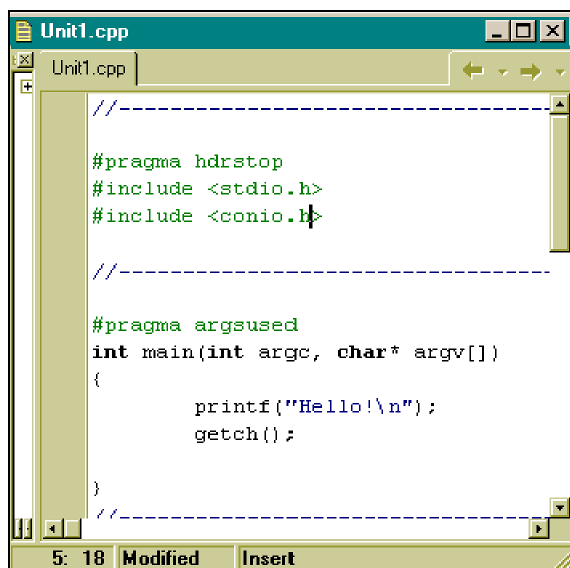


**Рис. 1.6.** Текст программы
после подключения необходимых библиотек

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

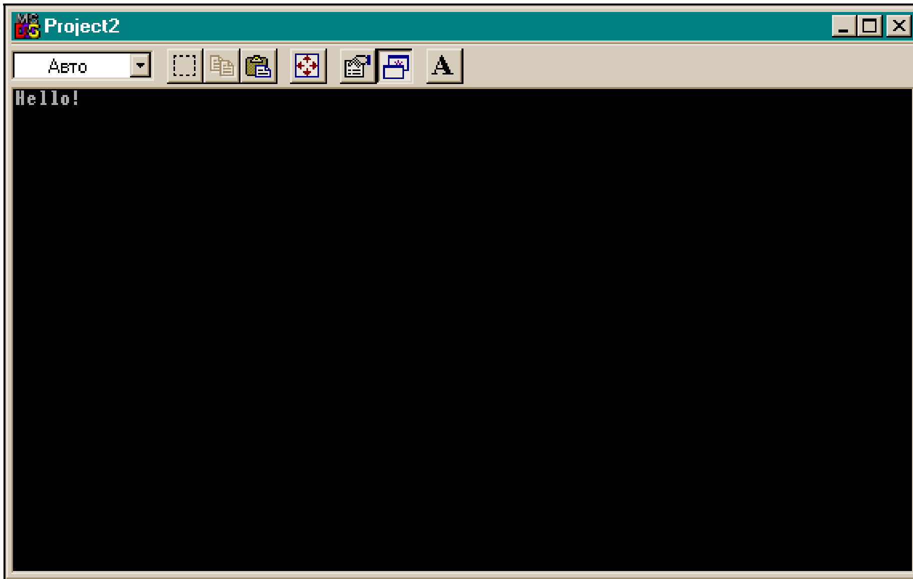


**Рис. 1.7.** Результат выполнения первой программы

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).

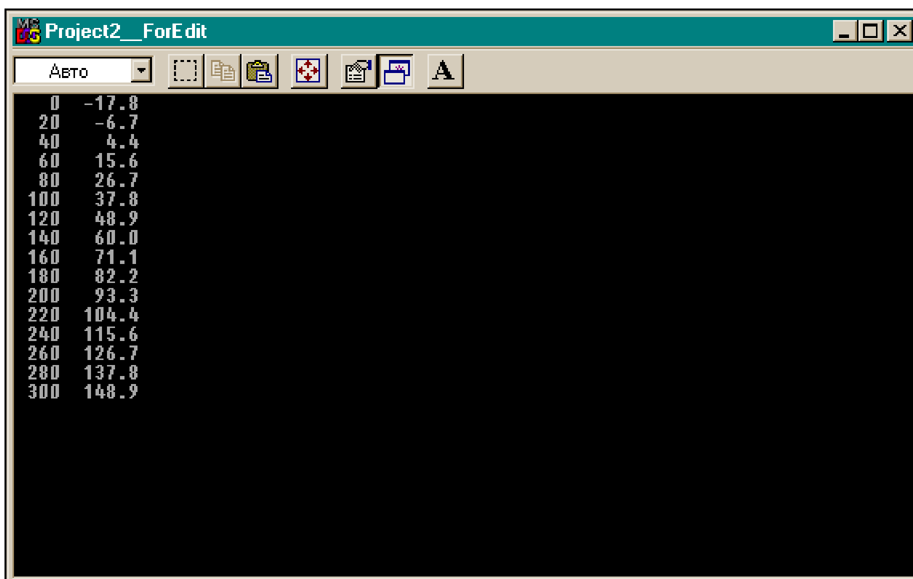


**Рис. 1.8.** Результат расчета таблицы температур по Цельсию

## 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 C 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 C 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).

Мы видим, что запись программы приобрела более компактный вид. В заголовочной части оператора `for` расположены три выражения, первые два из которых оканчиваются точкой с запятой, третье — круглой скобкой, обозначающей границу заголовочной части `for` (для компилятора этого достаточно, чтобы понять, что третье выражение завершилось). Как говорят, в данном случае, "цикл идет по переменной `fahr`": в первом выражении она получает начальное значение, второе выражение — это условие окончания цикла (цикл закончится тогда, когда `fahr` примет значение большее 300), а третье выражение изменяет параметр цикла на величину шага цикла.

Работа идет так: инициализируется переменная цикла (т. е. получает начальное значение), затем проверяется условие продолжения цикла. Если оно истинно, то выполняется сначала тело оператора (в данном случае, функция `printf()`), затем управление передается в заголовочную часть оператора `for`, после чего вычисляется третье выражение (изменяется параметр цикла) и проверяется значение второго выражения: если оно истинно, то выполняется тело, затем управление снова передается на вычисление третьего выражения и т. д. Если же второе выражение становится ложным, то выполнение оператора `for` завершается и начинает выполняться оператор, следующий непосредственно за ним. А это — завершающая фигурная скобка `main()`, после чего функция `main()` завершается.

В данном примере следует обратить внимание на аргумент функции `printf()`: вместо обычной переменной там стоит целое выражение, которое сначала будет вычислено, а потом его значение функцией выведется на устройство вывода. Выражение можно указывать в качестве аргумента функции, исходя из правила языка C: *"В любом контексте, в котором допускается использование переменной некоторого типа, можно использовать и выражение этого же типа"*.

# Символические константы

Задание конкретных чисел в теле программы — не очень хороший стиль программирования, т. к. такой подход затрудняет дальнейшую модификацию программы и ее понимание. При создании программы надо стремиться задавать все конкретные данные в начале программы, используя специальный оператор компилятора `#define`, который позволяет соотнести с каждым конкретным числом или выражением набор символов — *символических* (не символьных! символьные — это другое) *констант*. В этом случае на местах конкретных чисел в программе будут находиться символические константы, которые в момент компиляции программы будут заменены на соответствующие им числа, — но это уже невидимо для программиста. Отсюда и название "символические константы": это не переменные, которые имеют свой адрес и меняют свое значение по мере работы программы, а постоянные,

которые один раз получают свое значение и не меняют его. С учетом сказанного наша программа из листинга 1.2 примет следующий вид (листинг 1.3).

**Листинг 1.3**

```
#pragma hdrstop

#include <stdio.h>
#include <conio.h>
#define lower 0
#define upper 300
#define step   20

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

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

Теперь, когда начнется компиляция, компилятор просмотрит текст программы и заменит в нем все символические константы (в данном случае это: `lower`, `upper`, `step`) на их значения, заданные оператором `#define`. Заметим, что после этого оператора никаких точек с запятой ставить не требуется, т. к. это оператор не языка C, а компилятора. И если нам понадобится изменить значения переменных `lower`, `upper`, `step`, нам не придется разбираться в тексте программы, а достаточно будет посмотреть в ее начало, быстро найти изменяемые величины и выполнить их модификацию.