

Содержание

Об авторе	24
Введение	25
Изменения, внесенные в четвертое издание	25
Содержание книги	25
Рекомендуемая литература	26
ЧАСТЬ I. ОСНОВЫ ЯЗЫКА C++: ПОДМНОЖЕСТВО C	27
Глава 1. Обзор языка C	29
Происхождение и история языка C	30
Сравнение стандартов C89 и C99	30
C — язык среднего уровня	31
C — структурированный язык	32
C — язык для программистов	34
Структура программы на языке C	35
Библиотека и связывание	36
Раздельная компиляция	37
Расширения файлов .C и .CPP	37
Глава 2. Выражения	39
Пять основных типов данных	40
Модификация основных типов	41
Идентификаторы	42
Переменные	43
Где объявляются переменные	43
Локальные переменные	43
Формальные параметры	45
Глобальные переменные	46
Квалификаторы const и volatile	47
Квалификатор const	47
Квалификатор volatile	48
Спецификаторы хранения	49
Спецификатор extern	49
Статические переменные	51
Локальные статические переменные	51
Глобальные статические переменные	52
Спецификатор register	53
Инициализация переменных	54
Константы	54
Шестнадцатеричные и восьмеричные константы	55
Строковые константы	55
Управляющие символьные константы	56
Операторы	56
Оператор присваивания	57
Преобразования типов в операторе присваивания	57
Множественные присваивания	58
Арифметические операторы	58

Инкрементация и декрементация	59
Операторы сравнения и логические операторы	60
Побитовые операторы	62
Тернарный оператор	65
Операторы взятия адреса и разыменования	66
Статический оператор sizeof	67
Оператор последовательного вычисления	68
Оператор доступа к члену структуры и ссылки на член структуры	68
Операторы “[]” и “()”	69
Приоритеты операторов	69
Выражения	70
Порядок вычислений	70
Преобразование типов в выражениях	70
Приведение типов	71
Пробелы и круглые скобки	72
Составные операторы присваивания	72
Глава 3. Операторы	73
Истинные и ложные значения в языках C и C++	74
Условные операторы	75
Оператор if	75
Вложенные операторы if	76
Цепочка операторов if-then-else	77
Тернарная альтернатива	78
Условное выражение	80
Оператор switch	81
Вложенные операторы switch	83
Операторы цикла	84
Цикл for	84
Варианты цикла for	85
Бесконечный цикл	88
Пустой цикл for	89
Цикл while	89
Цикл do-while	91
Объявление переменных в условных операторах и циклах	92
Операторы перехода	93
Оператор return	93
Оператор goto	93
Оператор break	94
Функция exit	95
Оператор continue	96
Операторы-выражения	97
Блок	97
Глава 4. Массивы и строки	99
Одномерные массивы	100
Создание указателя на массив	101
Передача одномерного массива в функцию	101
Строки, завершающиеся нулевым байтом	103
Двухмерные массивы	104
Массивы строк	107
Многомерные массивы	108
Индексация указателей	109

Инициализация массива	111
Инициализация безразмерного массива	112
Игра в крестики-нолики	113
Глава 5. Указатели	117
Что такое указатель	118
Указатели	118
Операторы для работы с указателями	119
Выражения, содержащие указатели	120
Присваивание указателей	120
Адресная арифметика	120
Сравнение указателей	121
Указатели и массивы	123
Массивы указателей	124
Косвенная адресация	125
Инициализация указателей	126
Указатели на функции	127
Функции динамического распределения памяти	130
Проблемы, возникающие при работе с указателями	131
Глава 6. Функции	135
Общий вид функции	136
Область видимости функции	136
Аргументы функции	137
Передача параметров по значению и по ссылке	137
Передача параметров по ссылке	138
Передача массивов в качестве параметров	139
Аргументы функции main(): argc и argv	141
Оператор return	143
Возврат управления из функции	143
Возвращаемые значения	145
Возврат указателей	146
Функции типа void	147
Зачем нужен оператор return в функции main()	148
Рекурсия	148
Прототипы функций	149
Прототипы стандартных библиотечных функций	151
Определение списка параметров переменной длины	151
Объявление параметров функции в классическом и современном стиле	152
Глава 7. Структуры, объединения, перечисления и оператор typedef	153
Структуры	154
Доступ к членам структуры	156
Присваивание структур	157
Массивы структур	157
Передача структур функциям	158
Передача членов структур	158
Передача целых структур	158
Указатели на структуры	160
Объявление указателей на структуры	160
Использование указателей на структуры	160
Массивы и структуры внутри структур	162
Битовые поля	163
Объединения	165

Перечисления	167
Применение оператора sizeof для обеспечения машиннезависимости	169
Оператор typedef	170
Глава 8. Ввод-вывод на консоль	173
Важное замечание прикладного характера	174
Чтение и запись символов	175
Проблемы, связанные с функцией getchar()	176
Альтернативы функции getchar()	176
Чтение и запись строк	177
Форматированный ввод-вывод на консоль	179
Функция printf()	179
Вывод символов	180
Вывод чисел	180
Вывод адресов	181
Спецификатор %p	181
Модификаторы формата	182
Модификатор минимальной ширины поля	182
Модификатор точности	183
Выравнивание вывода	184
Обработка данных других типов	184
Модификаторы * и #	185
Функция scanf()	185
Спецификаторы формата	186
Ввод чисел	186
Ввод целых чисел без знака	187
Ввод отдельных символов	187
Ввод строк	187
Ввод адреса	188
Спецификатор %p	188
Использование набора сканируемых символов	188
Пропуск нежелательных разделителей	189
Символы, не являющиеся разделителями	189
Функции scanf() следует передавать адреса	189
Модификаторы формата	189
Подавление ввода	190
Глава 9. Файловый ввод-вывод	191
Файловые системы языков C и C++	192
Потоки и файлы	192
Потоки	192
Текстовые потоки	193
Бинарные потоки	193
Файлы	193
Основы файловой системы	194
Указатель файла	194
Открытие файла	195
Закрытие файла	196
Запись символа	196
Чтение символа	197
Применение функций fopen(), getc(), putc() и fclose()	197
Применение функции feof()	198
Работа со строками: функции fputs() и fgets()	200

Функция <code>rewind()</code>	200
Функция <code>feof()</code>	201
Удаление файла	203
Очистка потока	203
Функции <code>fread()</code> и <code>fwrite()</code>	204
Применение функций <code>fread()</code> и <code>fwrite()</code>	204
Функция <code>fseek()</code> и файлы с произвольным доступом	205
Функции <code>fprintf()</code> и <code>fscanf()</code>	207
Стандартные потоки	208
Связь с консольным вводом-выводом	209
Применение функции <code>freopen()</code> для перенаправления стандартных потоков	209
Глава 10. Препроцессор и комментарии	211
Препроцессор	212
Директива <code>#define</code>	212
Определение функций в виде макросов	214
Директива <code>#error</code>	214
Директива <code>#include</code>	215
Директивы условной компиляции	215
Директивы <code>#if</code> , <code>#else</code> , <code>#elif</code> и <code>#endif</code>	215
Директивы <code>#ifdef</code> и <code>#ifndef</code>	217
Директива <code>#undef</code>	218
Оператор <code>defined</code>	219
Директива <code>#line</code>	219
Директива <code>#pragma</code>	220
Операторы препроцессора <code>#</code> и <code>##</code>	220
Имена предопределенных макросов	221
Комментарии	221
Однострочные комментарии	222
ЧАСТЬ II. ЯЗЫК C++	223
Глава 11. Обзор языка C++	225
Истоки языка C++	226
Что такое объектно-ориентированное программирование	227
Инкапсуляция	228
Полиморфизм	228
Наследование	229
Некоторые основные принципы языка C++	229
Пример программы на языке C++	229
Операторы ввода-вывода	232
Объявление локальных переменных	232
Правило “ <code>int</code> по умолчанию”	233
Тип данных <code>bool</code>	234
Старый и новый стиль языка C++	234
Новый стиль заголовков	235
Пространства имен	236
Работа со старым компилятором	237
Введение в классы	237
Перегрузка функций	240
Перегрузка операторов	243
Наследование	243
Конструкторы и деструкторы	247

Ключевые слова языка C++	250
Структура программы на языке C++	250
Глава 12. Классы и объекты	251
Классы	252
Связь между структурами и классами	254
Связь между объединениями и классами	256
Безымянные объединения	257
Дружественные функции	258
Дружественные классы	261
Подставляемые функции	262
Определение подставляемых функций внутри класса	264
Конструкторы с параметрами	265
Конструкторы с одним параметром: особый случай	267
Статические члены класса	267
Статические переменные-члены	267
Статические функции-члены	271
Вызов конструкторов и деструкторов	272
Оператор разрешения области видимости	274
Вложенные классы	274
Локальные классы	274
Передача объектов функциям	275
Возвращение объектов	277
Присваивание объектов	278
Глава 13. Массивы, указатели, ссылки и операторы динамического распределения памяти	279
Массивы объектов	280
Инициализированные и неинициализированные массивы	282
Указатели на объекты	282
Проверка типа указателей	284
Указатель this	284
Указатели на производные типы	285
Указатели на члены класса	287
Ссылки	289
Передача параметров с помощью ссылок	290
Передача ссылок на объекты	292
Возврат ссылок	293
Независимые ссылки	294
Ссылки на производные типы	294
Ограничения на ссылки	295
Стиль	295
Операторы динамического распределения памяти	295
Инициализация выделяемой памяти	297
Выделение памяти для массивов	297
Выделение памяти для объектов	298
Альтернатива nothrow	302
Буферизованный оператор new	302
Глава 14. Перегрузка функций, конструкторы копирования и аргументы по умолчанию	303
Перегрузка функций	304
Перегрузка конструкторов	305

Перегрузка конструктора для достижения гибкости	305
Создание инициализированных и неинициализированных объектов	307
Конструктор копирования	308
Определение адреса перегруженной функции	311
Анахронизм overload	312
Аргументы функции по умолчанию	312
Аргументы по умолчанию и перегрузка	316
Правильное применение аргументов по умолчанию	317
Перегрузка функций и неоднозначность	317
Глава 15. Перегрузка операторов	321
Создание операторной функции-члена	322
Создание префиксной и постфиксной форм операторов инкрементации и декрементации	326
Перегрузка сокращенных операторов присваивания	327
Ограничения на перегруженные операторы	327
Перегрузка операторов с помощью дружественных функций	327
Применение дружественных функций для перегрузки операторов “++” и “—”	329
Дружественные операторные функции повышают гибкость	331
Перегрузка операторов new и delete	332
Перегрузка операторов new и delete для массивов	336
Перегрузка операторов new и delete, не генерирующих исключительной ситуации	338
Перегрузка некоторых специальных операторов	339
Перегрузка оператора “[]”	339
Перегрузка оператора “()”	342
Перегрузка оператора “->”	343
Перегрузка оператора “ , ”	344
Глава 16. Наследование	347
Управление доступом к членам базового класса	348
Наследование и защищенные члены	349
Защищенное наследование	352
Множественное наследование	353
Конструкторы, деструкторы и наследование	354
Когда вызываются конструкторы и деструкторы	354
Передача параметров конструктору базового класса	357
Предоставление доступа	360
Виртуальные базовые классы	362
Глава 17. Виртуальные функции и полиморфизм	367
Виртуальные функции	368
Вызов виртуальной функции с помощью ссылки на объект базового класса	370
Атрибут virtual наследуется	371
Виртуальные функции являются иерархическими	372
Чисто виртуальные функции	374
Абстрактные классы	376
Применение виртуальных функций	376
Сравнение раннего и позднего связывания	378
Глава 18. Шаблоны	379
Обобщенные функции	380
Функция с двумя обобщенными типами	382
Явная перегрузка обобщенной функции	382

Перегрузка шаблонной функции	384
Использование стандартных параметров шаблонных функций	385
Ограничения на обобщенные функции	385
Применение обобщенных функций	386
Обобщенная сортировка	386
Уплотнение массива	388
Обобщенные классы	389
Пример использования двух обобщенных типов данных	391
Применение шаблонных классов: обобщенный массив	392
Применение стандартных типов в обобщенных классах	393
Применение аргументов по умолчанию в шаблонных классах	395
Явные специализации классов	396
Ключевые слова <code>typename</code> и <code>export</code>	397
Мощь шаблонов	398
Глава 19. Обработка исключительных ситуаций	399
Основы обработки исключительных ситуаций	400
Перехват классов исключительных ситуаций	404
Применение нескольких операторов <code>catch</code>	405
Обработка производных исключительных ситуаций	406
Тонкости обработки исключительных ситуаций	407
Перехват всех исключительных ситуаций	407
Ограничения исключительных ситуаций	409
Повторное генерирование исключительной ситуации	410
Функции <code>terminate()</code> и <code>unexpected()</code>	411
Обработчики, связанные с функциями <code>terminate()</code> и <code>unexpected()</code>	412
Функция <code>uncaught_exception()</code>	413
Классы <code>exception</code> и <code>bad_exception</code>	413
Применение обработки исключительных ситуаций	413
Глава 20. Основы системы ввода-вывода	415
Сравнение старой и новой систем ввода-вывода	416
Потоки	416
Классы потоков в языке C++	417
Встроенные потоки в языке C++	418
Форматированный ввод-вывод	418
Форматирование с помощью членов класса <code>ios</code>	418
Установка флагов формата	419
Сброс флагов формата	420
Перегруженная форма функции <code>setf()</code>	421
Проверка флагов форматирования	422
Установка всех флагов	423
Применение функций <code>width()</code> , <code>precision()</code> и <code>fill()</code>	423
Применение манипуляторов формата	425
Перегрузка операторов “<<” и “>>”	427
Создание собственных функций вставки	427
Создание собственных функций извлечения	432
Создание собственных манипуляторов	434
Глава 21. Файловая система	437
Заголовок <code><fstream></code> и классы файлов	438
Открытие и закрытие файла	438
Чтение и запись текстовых файлов	440
Бесформатный и бинарный ввод-вывод	442

Сравнение символов и байтов	442
Функции put() и get()	442
Функции read() и write()	444
Дополнительные функции get()	446
Функция getline()	446
Распознавание конца файла	447
Функция ignore()	449
Функции peek() и putback()	450
Функция flush()	450
Произвольный доступ	450
Определение текущей позиции	453
Статус ввода-вывода	453
Настройка ввода-вывода в файлы	455
Глава 22. Динамическая идентификация типа и операторы приведения	457
Динамическая идентификация типа (RTTI)	458
Применение динамической идентификации типа	462
Применение оператора typeid к шаблонным классам	464
Операторы приведения типов	465
Оператор dynamic_cast	465
Замена оператора typeid оператором dynamic_cast	468
Применение оператора dynamic_cast к шаблонным классам	470
Оператор const_cast	471
Оператор static_cast	473
Оператор reinterpret_cast	473
Глава 23. Пространства имен, преобразования функций и другие новшества	475
Пространства имен	476
Основы пространств имен	476
Директива using	479
Неименованные пространства имен	480
Некоторые особенности пространств имен	481
Пространство имен std	483
Создание функций преобразования	484
Функции-члены с атрибутами const и mutable	487
Функции-члены с атрибутом volatile	489
Явные конструкторы	489
Инициализация членов класса	490
Применение ключевого слова asm	494
Спецификации связей	495
Буферизованный ввод-вывод	495
Классы буферизованного вывода	496
Создание буферизованного потока вывода	496
Применение буферизованного ввода	497
Буферизованный ввод-вывод	499
Применение динамических массивов	499
Применение бинарных операций ввода-вывода к буферизованным потокам	500
Отличия между языками C и C++	501
Глава 24. Введение в стандартную библиотеку шаблонов	503
Обзор библиотеки STL	504
Контейнеры	504
Алгоритмы	505

Итераторы	505
Другие элементы библиотеки STL	505
Контейнерные классы	506
Общие принципы функционирования	507
Векторы	508
Доступ к элементам вектора с помощью итератора	511
Вставка и удаление элементов вектора	513
Вектор, содержащий объекты класса	514
Списки	516
Функция end()	519
Сравнение функций push_front() и push_back()	520
Сортировка списка	521
Вставка одного списка в другой	521
Список, содержащий объекты класса	523
Ассоциативные контейнеры	524
Ассоциативный массив, содержащий объекты	528
Алгоритмы	529
Подсчет	531
Удаление и замена элементов	533
Изменение порядка следования элементов последовательности	535
Преобразование последовательности	535
Применение функторов	537
Унарные и бинарные функторы	537
Применение встроенных функторов	537
Создание функтора	539
Применение редакторов связей	541
Класс string	542
Некоторые функции — члены класса string	546
Основные манипуляторы	546
Поиск символа в строке	548
Сравнение строк	550
Создание C-строки	550
Строки как контейнеры	550
Запись строки в другой контейнер	551
Заключительные замечания о библиотеке STL	552
ЧАСТЬ III. БИБЛИОТЕКА СТАНДАРТНЫХ ФУНКЦИЙ	553
Глава 25. Функции ввода-вывода языка C	555
Функция clearerr	556
Функция fclose	556
Функция feof	557
Функция ferrror	557
Функция fflush	557
Функция fgetc	557
Функция fgetpos	558
Функция fgets	558
Функция fopen	558
Функция fprintf	559
Функция fputc	560
Функция fputs	560
Функция fread	560
Функция freopen	560

Функция fscanf	561
Функция fseek	561
Функция fsetpos	562
Функция ftell	562
Функция fwrite	562
Функцияgetc	562
Функция getchar	563
Функция gets	563
Функция perror	563
Функция printf	564
Функция putchar	566
Функция puts	566
Функция remove	567
Функция rename	567
Функция rewind	567
Функция scanf	567
Функция setbuf	570
Функция setvbuf	570
Функция sprintf	570
Функция sscanf	571
Функция tmpfile	571
Функция tmpnam	571
Функция ungetc	571
Функции vprintf, vfprintf и vsprintf	572
Глава 26. Строковые и символьные функции	573
Функция isalnum	574
Функция isalpha	574
Функция iscntrl	574
Функция isdigit	575
Функция isgraph	575
Функция islower	575
Функция isprint	575
Функция ispunct	575
Функция isspace	575
Функция isupper	576
Функция isxdigit	576
Функция memchr	576
Функция memcmp	576
Функция memcpy	576
Функция memmove	577
Функция memset	577
Функция strcat	577
Функция strchr	577
Функция strcmp	577
Функция strcoll	578
Функция strcpy	578
Функция strcspn	578
Функция strerror	578
Функция strlen	579
Функция strncat	579
Функция strncmp	579
Функция strncpy	579

Функция strpbrk	580
Функция strrchr	580
Функция strspn	580
Функция strstr	580
Функция strtok	580
Функция strxfrm	581
Функция tolower	581
Функция toupper	581
Глава 27. Математические функции	583
Функция acos	584
Функция asin	584
Функция atan	585
Функция atan2	585
Функция ceil	585
Функция cos	585
Функция cosh	585
Функция exp	586
Функция fabs	586
Функция floor	586
Функция fmod	586
Функция frexp	586
Функция ldexp	587
Функция log	587
Функция log10	587
Функция modf	587
Функция pow	587
Функция sin	588
Функция sinh	588
Функция sqrt	588
Функция tan	588
Функция tanh	589
Глава 28. Функции времени, даты и локализации	591
Функция asctime	592
Функция clock	592
Функция ctime	593
Функция difftime	593
Функция gmtime	593
Функция localeconv	593
Функция localtime	594
Функция mktime	595
Функция setlocale	595
Функция strftime	595
Функция time	596
Глава 29. Функции динамического распределения памяти	597
Функция calloc	598
Функция free	598
Функция malloc	598
Функция realloc	599
Глава 30. Служебные функции	601
Функция abort	602

Функция abs	602
Макрос assert	602
Функция atexit	603
Функция atof	603
Функция atoi	603
Функция atol	604
Функция bsearch	604
Функция div	604
Функция exit	605
Функция getenv	605
Функция labs	605
Функция ldiv	605
Функция longjmp	606
Функция mblen	606
Функция mbstowcs	606
Функция mbtowc	607
Функция qsort	607
Функция raise	607
Функция rand	608
Функция setjmp	608
Функция signal	608
Функция srand	609
Функция strtod	609
Функция strtol	609
Функция strtoul	610
Функция system	610
Функции va_arg, va_start и va_end	611
Функция wctombs	611
Функция wctomb	611
Глава 31. Функции обработки расширенных символов	613
Функции классификации расширенных символов	614
Функции ввода-вывода расширенных символов	616
Функции обработки строк, состоящих из расширенных символов	617
Функции преобразования строк, состоящих из расширенных символов	618
Функции обработки массивов расширенных символов	618
Функции преобразования многобайтовых и расширенных символов	619
ЧАСТЬ IV. БИБЛИОТЕКА СТАНДАРТНЫХ КЛАССОВ	621
Глава 32. Стандартные классы ввода-вывода	623
Классы ввода-вывода	624
Заголовки ввода-вывода	625
Флаги форматирования и манипуляторы ввода-вывода	626
Некоторые типы данных	627
Типы streamsize и streamoff	627
Типы streampos и wstreampos	627
Типы pos_type и off_type	628
Тип openmode	628
Тип iostate	628
Тип seekdir	628
Класс failure	628
Перегрузка операторов “<<” и “>>”	629

Универсальные функции ввода-вывода	629
Функция bad	629
Функция clear	629
Функция eof	629
Функция exceptions	630
Функция fail	630
Функция fill	630
Функция flags	630
Функция flush	630
Функции fstream, ifstream и ofstream	631
Функция gcount	631
Функция get	631
Функция getline	632
Функция good	633
Функция ignore	633
Функция open	633
Функция peek	634
Функция precision	634
Функция put	634
Функция putback	634
Функция rdbuf	635
Функция read	635
Функция readsome	635
Функции seekg и seekp	635
Функция setf	636
Функция setstate	637
Функция str	637
Функции stringstream, istringstream и ostringstream	637
Функция sync_with_stdio	638
Функции tellg и tellp	638
Функция unsetf	639
Функция width	639
Функция write	639
Глава 33. Стандартные контейнерные классы	641
Контейнерные классы	642
Класс bitset	643
Класс deque	645
Класс list	646
Класс map	648
Класс multimap	650
Класс multiset	652
Класс queue	654
Класс priority_queue	655
Класс set	655
Класс stack	657
Класс vector	658
Глава 34. Стандартные алгоритмы	661
Алгоритм adjacent_find	662
Алгоритм binary_search	662
Алгоритм copy	662
Алгоритм copy_backward	663
Алгоритм count	663

Алгоритм count_if	663
Алгоритм equal	663
Алгоритм equal_range	663
Алгоритмы fill и fill_n	664
Алгоритм find	664
Алгоритм find_end	664
Алгоритм find_first_of	664
Алгоритм find_if	665
Алгоритм for_each	665
Алгоритмы generate и generate_n	665
Алгоритм includes	665
Алгоритм inplace_merge	665
Алгоритм iter_swap	666
Алгоритм lexicographical_compare	666
Алгоритм lower_bound	666
Алгоритм make_heap	666
Алгоритм max	667
Алгоритм max_element	667
Алгоритм merge	667
Алгоритм min	667
Алгоритм min_element	668
Алгоритм mismatch	668
Алгоритм next_permutation	668
Алгоритм nth_element	668
Алгоритм partial_sort	669
Алгоритм partial_sort_copy	669
Алгоритм partition	669
Алгоритм pop_heap	669
Алгоритм prev_permutation	670
Алгоритм push_heap	670
Алгоритм random_shuffle	670
Алгоритмы remove, remove_if, remove_copy и remove_copy_if	670
Алгоритмы replace, replace_copy, replace_if и replace_copy_if	671
Алгоритмы reverse и reverse_copy	672
Алгоритмы rotate и rotate_copy	672
Алгоритм search	672
Алгоритм search_n	672
Алгоритм set_difference	673
Алгоритм set_intersection	673
Алгоритм set_symmetric_difference	673
Алгоритм set_union	674
Алгоритм sort	674
Алгоритм sort_heap	674
Алгоритм stable_partition	675
Алгоритм stable_sort	675
Алгоритм swap	675
Алгоритм swap_ranges	675
Алгоритм transform	675
Алгоритм unique и unique_copy	676
Алгоритм upper_bound	676
Глава 35. Стандартные итераторы, распределители памяти и функторы	677
Итераторы	678
Основные типы итераторов	678

Классы низкоуровневых итераторов	679
Класс iterator	679
Класс iterator_traits	679
Встроенные итераторы	679
Класс insert_iterator	680
Класс back_insert_iterator	681
Класс front_insert_iterator	682
Класс reverse_iterator	682
Класс istream_iterator	683
Класс istreambuf_iterator	684
Класс ostream_iterator	684
Класс ostreambuf_iterator	685
Две функции для работы с итераторами	686
Функторы	686
Функторы	686
Редакторы связей	687
Инверторы	688
Адаптеры	689
Адаптеры указателей на функции	689
Адаптеры указателей на функции — члены класса	690
Распределители памяти	691
Глава 36. Класс string	693
Класс basic_string	694
Класс char_traits	701
Глава 37. Числовые классы	703
Класс complex	704
Класс vallaray	706
Классы slice и gslice	716
Вспомогательные классы	717
Числовые алгоритмы	718
Класс accumulate	718
Алгоритм adjacent_difference	718
Алгоритм inner_product	719
Алгоритм partial_sum	720
Глава 38. Обработка исключительных ситуаций и прочие классы	723
Исключительные ситуации	724
Заголовок <exception>	724
Заголовок <stdexcept>	725
Класс auto_ptr	726
Класс pair	727
Локализация	728
Прочие классы	728
ЧАСТЬ V. ПРИЛОЖЕНИЯ НА ЯЗЫКЕ C++	729
Глава 39. Интеграция новых классов: пользовательский класс для работы со строками	731
Класс StrType	732
Ввод и вывод строк	735
Функции присваивания	735
Конкатенация	737

Вычитание подстроки	738
Операторы сравнения	740
Прочие строковые функции	741
Полное определение класса StrType	741
Применение класса StrType	748
Принципы создания и интеграции новых типов	750
Проблема	750
Глава 40. Синтаксический анализ выражений	751
Выражения	752
Синтаксический анализ выражений: постановка задачи	753
Синтаксический анализ выражения	753
Класс parser	755
Разбор выражения на составные части	755
Простая программа синтаксического анализа выражений	758
Принципы работы синтаксического анализатора	762
Синтаксический анализатор, работающий с переменными	763
Проверка синтаксических ошибок при рекурсивном нисходящем анализе	770
Создание обобщенного синтаксического анализатора	771
Некоторые задачи	777
Приложение А. Расширение языка C++ для платформы .NET	779
Дополнительные ключевые слова	780
Ключевое слово <code>__abstract</code>	780
Ключевое слово <code>__box</code>	780
Ключевое слово <code>__delegate</code>	781
Ключевое слово <code>__event</code>	781
Ключевое слово <code>__finally</code>	781
Ключевое слово <code>__gc</code>	781
Ключевое слово <code>__identifier</code>	781
Ключевое слово <code>__interface</code>	781
Ключевое слово <code>__nogs</code>	781
Ключевое слово <code>__pin</code>	781
Ключевое слово <code>__property</code>	781
Ключевое слово <code>__sealed</code>	782
Ключевое слово <code>__try_cast</code>	782
Ключевое слово <code>__typeof</code>	782
Ключевое слово <code>__value</code>	782
Расширение директив препроцессора	782
Атрибут <code>attribute</code>	782
Компилирование управляемых программ на языке C++	782
Приложение Б. Язык C++ и робототехника	783
Предметный указатель	787

Часть I. Основы языка C++: подмножество C

Вероятно, читателям уже известно, что язык C++ создан на основе языка C. Фактически язык C++ включает в себя весь язык C, и все программы (за некоторым исключением), написанные на языке C, можно считать программами на языке C++. В процессе разработки языка C++ в качестве отправной точки был выбран язык C. Затем к нему были добавлены новые свойства и возможности, разработанные для поддержки объектно-ориентированного программирования. Однако от языка C при этом не отказались, и стандарт 1989 года ANSI/ISO C Standard стал *базовым документом* при создании международного стандарта языка C++ (International Standard). Таким образом, осваивая язык C++, программисты одновременно овладевают языком C.

Разделение свойств языка C и специфических особенностей языка C++ позволяет достичь трех основных целей.

- Четко провести разделительную линию между языками C и C++.
- Предоставить читателям, владеющим языком C, возможность легко усвоить информацию об особенностях языка C++.
- Выделить и подробно описать свойства языка C++, унаследованные от языка C.

Очень важно точно провести линию, отделяющую язык C от языка C++, поскольку они очень широко распространены, и от программиста иногда требуется поддержка программ, написанных на обоих языках. Если вы пишете программу на языке C, нужно четко понимать, где заканчивается язык C и начинается язык C++. Многие программисты, работающие на языке C++, иногда пишут программы на языке C и не используют специфические свойства языка C++. В особенности это относится к программированию встроенных систем и поддержке существующих приложений. Таким образом, понимание различий между этими языками является необходимым условием профессионального программирования на C++.

Полное и свободное владение языком C абсолютно необходимо для перевода программ на язык C++. Чтобы сделать это на высоком уровне, необходимо хорошо знать язык C. Например, без ясного понимания механизмов ввода-вывода, предусмотренных в языке C, невозможно трансформировать программу, осуществляющую интенсивный обмен данными с внешними устройствами, в эффективную программу на языке C++.

Многие читатели уже владеют языком C. Вследствие этого выделение тем, связанных с языком C, позволяет опытным программистам избежать повторения пройденного и перейти непосредственно к изучению особенностей языка C++. Разумеется, в части I подчеркиваются малейшие отличия языка C++ от языка C. Кроме того, отделение подмножества C от остальных свойств языка C++ позволяет в дальнейшем сосредоточиться на его объектно-ориентированных особенностях.

Несмотря на то что язык C++ полностью содержит язык C, как правило, не все свойства языка C используются в программах, написанных “в стиле C++”. Например, система ввода-вывода, предусмотренная в языке C, по-прежнему доступна в языке C++, хотя в C++ существуют свои объектно-ориентированные механизмы ввода-вывода данных. Еще одним примером такого анахронизма является препроцессор. Он играет чрезвычайно важную роль в языке C и очень скромную — в языке C++. Обсуждение свойств, присущих “стилю языка C”, в первой части книги позволяет избежать путаницы в остальных главах.

Запомните: подмножество C, описанное в части I, является ядром языка C++ и фундаментом, на котором воздвигнуты его объектно-ориентированные конструкции. Все свойства, описанные здесь, являются неотъемлемой частью языка C++ и могут быть использованы в ваших программах.

На заметку

Часть I содержит материалы из моей книги C: The Complete Reference (McGraw-Hill-Osborne) (Русский перевод: Г. Шилдт. Полный справочник по языку C. — М.: Изд. дом “Вильямс”, 2001. — Прим. ред.). Если язык C интересует вас как самостоятельный язык программирования, эта книга окажется для вас ценным помощником.

Глава 1

Обзор языка С

Чтобы понять язык C++, необходимо понять мотивы его создания, идеи, положенные в его основу, и свойства, которые он унаследовал от своих предшественников. Таким образом, история языка C++ начинается с языка C. В главе представлен обзор языка C, а также описана история его возникновения, способы применения и основные принципы. Поскольку язык C++ создан на основе языка C, эту главу можно считать описанием предыстории языка C++. Многие из того, что сделало язык C++ таким популярным, уходит корнями в язык C.

Происхождение и история языка C

Язык C был изобретен и впервые реализован Деннисом Ритчи (Dennis Ritchie) на компьютере DEC PDP-11 под управлением операционной системы UNIX. Язык C появился в результате развития языка под названием BCPL. В свою очередь, этот язык был разработан Мартином Ричардсом (Martin Richards) под влиянием другого языка, имевшего название B, автором которого был Кен Томпсон (Ken Thompson). Итак, в 1970-х годах развитие языка B привело к появлению языка C.

Многие годы фактическим стандартом языка C была версия для операционной системы UNIX. Впервые она была описана в книге Брайана Кернигана (Brian Kernighan) и Денниса Ритчи *The C Programming Language* в 1978 году. (Русский перевод: Керниган Б., Ритчи Д. Язык программирования C. — СПб: Невский диалект, 2001. — Прим. ред.). Летом 1983 года был создан комитет Американского института национальных стандартов (American National Standards Institute — ANSI), целью которого была разработка стандарта языка C. Работа комитета неожиданно растянулась на шесть лет.

В итоге стандарт ANSI C был одобрен в декабре 1989 года и стал распространяться в начале 1990-го. Этот стандарт был также одобрен Организацией международных стандартов (International Standards Organization — ISO), получив название *ANSI/ISO Standard C*. В 1995 году была одобрена Первая поправка, которая помимо всего прочего добавила несколько новых библиотечных функций. В 1989 году стандарт языка C вместе с Первой поправкой стали *базовым документом* для стандарта языка C++, в котором было выделено *подмножество C*. Версию языка C, определенную стандартом 1989 года, обычно называют *C89*.

После 1989 года в центре внимания программистов оказался язык C++. Развитие этого языка на протяжении 1990-х годов завершилось одобрением стандарта в конце 1998 года. Между тем работа над языком C продолжалась, не вызывая излишнего шума. В итоге в 1999 году появился новый стандарт языка C, который обычно называют *C99*. В целом стандарт C99 сохранил практически все свойства стандарта C89, не изменив основных аспектов языка. Таким образом, язык C, описанный стандартом C99, практически совпадает с языком, соответствующим стандарту C89. Комитет по разработке стандарта C99 сосредоточился на двух вопросах: включении в язык нескольких математических библиотек и развитии некоторых специфических и весьма сложных свойств, например, массивов переменной длины и квалификатора указателей **restrict**. В стандарт C99 вошли также некоторые свойства, позаимствованные из языка C++, например, однострочные комментарии. Поскольку разработка стандарта языка C++ завершилась до создания стандарта C99, ни одно из новшеств языка C не вошло в стандарт C++.

Сравнение стандартов C89 и C99

Несмотря на то что все новшества, внесенные в стандарт C99, весьма важны с теоретической точки зрения, они имели мало практических последствий, так как до сих пор нет ни одного широко распространенного компилятора, который

поддерживал бы стандарт C99. Большинство программистов до сих пор считают языком C его вариант, определенный стандартом C89. Именно его реализуют все основные компиляторы. Более того, подмножество C языка C++ описывается именно стандартом C89. Хотя некоторые новшества, включенные в стандарт C99, в конце концов обязательно будут учтены следующим стандартом языка C++, пока они несовместимы с языком C++.

Поскольку подмножество C языка C++ соответствует стандарту C89, и эту версию изучают большинство программистов, именно ее мы рассмотрим в части I. Итак, используя название C, мы будем иметь в виду версию языка, определенную стандартом C89. Однако мы будем отмечать важные различия между версиями C89 и C99, поскольку это улучшит совместимость языков C и C++.

C — язык среднего уровня

Язык C часто называют *языком среднего уровня*. Это не означает, что он менее эффективен, более неудобен в использовании или менее продуман, чем языки высокого уровня, такие как Basic или Pascal. Отсюда также не следует, что он запутан, как язык ассемблера (и порождает связанные с этим проблемы). Это выражение означает лишь, что язык C объединяет лучшие свойства языков высокого уровня, возможности управления и гибкость языка ассемблера. В табл. 1.1 показано место языка C среди других языков программирования.

Таблица 1.1. Место языка C среди остальных языков программирования

Высший уровень	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
Средний уровень	Java
	C#
	C++
	C
	Forth
Низший уровень	Macro-assembler
	Assembler

Будучи языком среднего уровня, язык C позволяет осуществлять манипуляции с битами, байтами и адресами — основными элементами, с которыми работают функции операционной системы. Несмотря на это, программы, написанные на языке C, можно выполнять на разных компьютерах. Это свойство программ называется *машинонезависимостью* (portability). Например, если программу, написанную для операционной системы UNIX, можно легко преобразовать, чтобы она работала на платформе Windows, то говорят, что такая программа является *машинонезависимой* (portable).

Все языки высокого уровня используют концепцию типов данных. *Тип данных* (data type) определяет множество значений, которые может принимать переменная, а также множество операций, которые над ней можно выполнять. К основным типам данных относятся целое число, символ и действительное число. Несмотря на то что в языке C существует пять встроенных типов данных, он не является строго типизи-

рованным языком, как языки Pascal и Ada. В языке C разрешены практически все преобразования типов. Например, в одном и том же выражении можно свободно использовать переменные символьного и целочисленного типов.

В отличие от языков высокого уровня, язык C практически не проверяет ошибки, возникающие на этапе выполнения программ. Например, не осуществляется проверка возможного выхода индекса массива за пределы допустимого диапазона. Предотвращение ошибок такого рода возлагается на программиста.

Кроме того, язык C не требует строгой совместимости типов параметров и аргументов функций. Как известно, языки высокого уровня обычно требуют, чтобы тип аргумента точно совпадал с типом соответствующего параметра. Однако в языке C такого условия нет. Он позволяет использовать аргумент любого типа, если его можно разумным образом преобразовать в тип параметра. Кроме того, язык C предусматривает средства для автоматического преобразования типов.

Особенность языка C заключается в том, что он позволяет непосредственно манипулировать битами, байтами, машинными словами и указателями. Это делает его очень удобным для системного программирования, в котором эти операции широко распространены.

Другой важный аспект языка состоит в том, что в нем предусмотрено очень небольшое количество ключевых слов, которые можно использовать для конструирования выражений. Например, стандарт C89 содержит лишь 32 ключевых слова, а стандарт C99 добавил к ним всего 5 слов. Некоторые языки программирования содержат в несколько раз больше ключевых слов. Скажем, самые распространенные версии языка BASIC предусматривают более 100 таких слов!

C — структурированный язык

Возможно, вы уже слышали словосочетание *блочно-структурированный* (block-structured) по отношению к языку программирования. Хотя этот термин нельзя напрямую применять к языку C, его обычно тоже называют *структурированным*. Он имеет много общего с другими структурированными языками, такими как ALGOL, Pascal и Modula-2.

На заметку

Язык C (как и C++) не считается блочно-структурированным, поскольку не позволяет объявлять одну функции внутри других.

Отличительной особенностью структурированных языков является *обособление* кода и данных (compartmentalization). Оно позволяет выделять и скрывать от остальной части программы данные и инструкции, необходимые для решения конкретной задачи. Этого можно достичь с помощью подпрограмм (subroutines), в которых используются локальные (временные) переменные. Используя локальные переменные, можно создавать подпрограммы, не порождающие побочных эффектов в других модулях. Это облегчает координацию модулей между собой. Если программа разделена на обособленные функции, нужно лишь знать, что делает та или иная функция, не интересуясь, как именно она выполняет свою задачу. Помните, что чрезмерное использование глобальных переменных (которые доступны в любом месте программы) повышает вероятность ошибок и нежелательных побочных эффектов. (Каждый программист, работавший на языке BASIC, хорошо знает эту проблему.)

На заметку

Концепция обособления широко применяется в языке C++. В частности, каждая часть программы, написанной на этом языке, может четко управлять доступом к ней из других модулей.

Структурные языки предоставляют широкий спектр возможностей. Они допускают использование вложенных циклов, например **while**, **do-while** и **for**.

В структурированных языках использование оператора `goto` либо запрещено, либо нежелательно и не включается в набор основных средств управления потоком выполнения программы (как это принято в стандарте языка BASIC и в традиционном языке FORTRAN). Структурированные языки позволяют размещать несколько инструкций программы в одной строке и не ограничивают программиста жесткими полями для ввода команд (как это делалось в старых версиях языка FORTRAN).

Рассмотрим несколько примеров структурированных и неструктурированных языков (табл. 1.2).

Таблица 1.2. Структурированные и неструктурированные языки программирования

Неструктурированные	Структурированные
FORTRAN	Pascal
BASIC	Ada
COBOL	Java
	C#
	C++
	C
	Modula-2

Структурированные языки считаются более современными. В настоящее время неструктурированность является признаком устаревших языков программирования, и лишь немногие программисты выбирают их для создания серьезных приложений.

На заметку

Новые версии старых языков программирования (например Visual Basic) включают элементы структурированности. И все же врожденные недостатки этих языков вряд ли будут до конца исправлены, поскольку структурированность не закладывалась в их основу с самого начала.

Основным структурным элементом языка C является *функция*. Именно функции служат строительными блоками, из которых создается программа. Они позволяют разбивать программу на модули, решающие отдельные задачи. Создав функцию, можно не беспокоиться о побочных эффектах, которые она вызовет в других частях программы. Способность создавать отдельные функции чрезвычайно важна при реализации больших проектов, в которых один фрагмент кода не должен взаимодействовать с другими частями программы непредсказуемым образом.

Другой способ структурирования и обособления программы, написанной на языке C, — блоки. *Блок* (code block) — это группа операторов, логически связанных между собой, и рассматриваемых как единое целое. В языке C блок можно создать с помощью фигурных скобок, ограничивающих последовательность операторов. Вот типичный пример блока.

```
if (x < 10) {
    printf("Слишком мало, попробуйте снова.\n");
    scanf("%d", &x);
}
```

Два оператора, расположенных внутри фигурных скобок, выполняются, если значение переменной `x` меньше 10. Эти два оператора вместе с фигурными скобками образуют блок. Блок — это логическая единица, поскольку оба оператора обязательно должны быть выполнены. Блоки позволяют ясно, элегантно и эффективно реализовывать различные алгоритмы. Более того, они помогают программисту лучше выразить природу алгоритма.

C — язык для программистов

Как ни странно, не все языки программирования предназначены для программистов. Рассмотрим классические примеры языков, ориентированных не на программистов, — COBOL и BASIC. Язык COBOL был разработан вовсе не для того, чтобы облегчить участь программистов, улучшить ясность кода и повысить его надежность, и даже не для того, чтобы ускорить выполнение программ. Он был создан, в частности, для того, чтобы люди, не являющиеся программистами, могли читать и (по возможности) понимать (хотя это вряд ли) написанные на нем программы. В свою очередь, язык BASIC был разработан для пользователей, которые решают на компьютере простые задачи.

В противоположность этому, язык C был создан для программистов, учитывал их интересы и многократно проверялся на практике, прежде чем был окончательно реализован. В итоге этот язык дает программистам именно то, чего они желали: сравнительно небольшое количество ограничений, минимум претензий, блочные структуры, изолированные функции и компактный набор ключевых слов. Язык C обладает эффективностью ассемблера и структурированностью языков ALGOL или Modula-2. Поэтому неудивительно, что именно языки C и C++ стали наиболее популярными среди профессиональных программистов высокого уровня.

Тот факт, что язык C часто используют вместо ассемблера, является одной из основных причин его популярности. Язык ассемблера использует символическое представление фактического двоичного кода, который непосредственно выполняется компьютером. Каждая операция, выраженная на языке ассемблера, представляет собой отдельную задачу, выполняемую компьютером. Хотя язык ассемблера предоставляет программисту наибольшую гибкость, разрабатывать и отлаживать программы на нем довольно сложно. Кроме того, поскольку язык ассемблера является неструктурированным, код напоминает спагетти — запутанную смесь переходов, вызовов и индексов. Вследствие этого программы, написанные на языке ассемблера, трудно читать, модифицировать и эксплуатировать. Вероятно, основным недостатком программ на языке ассемблера является их машинозависимость. Программа, предназначенная для конкретного центрального процессора, не может выполняться на компьютерах другого типа.

Изначально язык C предназначался для системного программирования. *Системная программа* (system program) представляет собой часть операционной системы или является одной из ее утилит. Рассмотрим некоторые из них.

- Операционные системы
- Интерпретаторы
- Редакторы
- Компиляторы
- Файловые утилиты
- Оптимизаторы
- Диспетчеры реального времени
- Драйверы

По мере роста популярности языка C многие программисты стали применять его для программирования всех задач, используя его машинезависимость и эффективность, а кроме того, он им просто нравился! К моменту появления языка C языки

программирования прошли сложный и трудный путь совершенствования. Разумеется, вновь созданный язык вобрал в себя все лучшее.

С появлением языка C++ некоторые программисты посчитали, что язык C потеряет самостоятельность и сойдет со сцены. Однако этого не произошло. Во-первых, не все программы должны быть объектно-ориентированными. Например, программное обеспечение встроенных систем по-прежнему создается на языке C. Во-вторых, существует огромное множество программ на языке C, которые активно эксплуатируются и нуждаются в модификации. Поскольку язык C является основой языка C++, он продолжает широко использоваться, имея блестящие перспективы.

Структура программы на языке C

В табл. 1.3 перечислены 32 ключевых слова, которые используются при формировании синтаксиса языка C, стандарта C89 и подмножества C языка C++. Все они, конечно, являются и ключевыми словами языка C++.

Таблица 1.3. Ключевые слова подмножества C языка C++

<code>auto</code>	<code>double</code>	<code>int</code>	<code>struct</code>
<code>break</code>	<code>else</code>	<code>long</code>	<code>switch</code>
<code>case</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>
<code>char</code>	<code>extern</code>	<code>return</code>	<code>union</code>
<code>const</code>	<code>float</code>	<code>short</code>	<code>unsigned</code>
<code>continue</code>	<code>for</code>	<code>signed</code>	<code>void</code>
<code>default</code>	<code>goto</code>	<code>sizeof</code>	<code>volatile</code>
<code>do</code>	<code>if</code>	<code>static</code>	<code>while</code>

Кроме того, многие компиляторы для более эффективного использования среды программирования вносят в язык C дополнительные ключевые слова. Например, некоторые компиляторы предусматривают ключевые слова для управления памятью процессоров семейства 8086, поддержки многоязычного программирования и доступа к системным прерываниям. Перечислим некоторые из этих расширенных ключевых слов.

<code>asm</code>	<code>_cs</code>	<code>_ds</code>	<code>_es</code>
<code>_ss</code>	<code>cdecl</code>	<code>far</code>	<code>huge</code>
<code>interrupt</code>	<code>near</code>	<code>pascal</code>	

Ваш компилятор может изменить этот список, стремясь наиболее эффективно использовать конкретную среду программирования.

Обратите внимание на то, что все ключевые слова набраны строчными буквами. Язык C/C++ чувствителен к регистру (case sensitive), т.е. прописные и строчные буквы в нем различаются. Это значит, что слово `else` является ключевым, а слово `ELSE` — нет. Ключевые слова нельзя использовать в программе для иных целей, например, в качестве имени переменной или функции.

Все программы на языке C состоят из одной или нескольких функций. В любом случае программа должна содержать функцию `main()`, которая при выполнении программы вызывается первой. В хорошо написанном коде функция `main()` должна содержать, по существу, схему работы всей программы. Несмотря на то что имя `main()` не включено в список ключевых слов, по своей природе оно является именно таковым. Скажем, назвать переменную именем `main` нельзя, так как компилятор сразу выдаст сообщение об ошибке.

Общий вид программы на языке C показан в листинге 1.1. Функции с именами `f1()`, ..., `fN()` определяются пользователем.

Листинг 1.1. Общий вид программы на языке C

```
объявления глобальных переменных
тип_возвращаемого_значения main (список параметров)
{
    последовательность операторов
}

тип_возвращаемого_значения f1 (список параметров)
{
    последовательность операторов
}

тип_возвращаемого_значения f2 (список параметров)
{
    последовательность операторов
}

:
.

тип_возвращаемого_значения fN (список параметров)
{
    последовательность операторов
}
```

Библиотека и связывание

С формальной точки зрения можно написать законченную и вполне осмысленную программу на языке C, не используя ни одной стандартной функции. Однако это довольно затруднительно, так как в языке C нет ключевых слов, имеющих отношение к вводу-выводу, математическим операциям высокого уровня или обработке символов. В результате большинство программ вынуждены вызывать различные функции, содержащиеся в *стандартной библиотеке* (standard library).

Все компиляторы языка C++ сопровождаются стандартной библиотекой функций, выполняющих наиболее распространенные задачи. В этой библиотеке определен минимальный набор функций, которые поддерживаются большинством компиляторов. Однако конкретный компилятор может содержать много других функций. Например, стандартная библиотека не содержит графических функций, но в каждом компиляторе всегда есть определенный набор таких функций.

Стандартная библиотека языка C++ разделена на две части: функции и классы. Стандартная библиотека функций представляет собой наследие языка C. Язык C++ поддерживает все функции, предусмотренные стандартом C89. Таким образом, все стандартные функции языка C можно свободно использовать в программах на языке C++.

Кроме стандартной библиотеки функций, язык C++ имеет собственную библиотеку классов. Эта библиотека состоит из объектно-ориентированных модулей, которые можно использовать в собственных программах. Кроме того, существует стандартная

библиотека шаблонов STL, содержащая широкий набор готовых решений многих задач. В части I используется только стандартная библиотека функций, поскольку именно она относится к языку C.

Стандартная библиотека состоит из многих универсальных функций. При вызове библиотечной функции компилятор “запоминает” ее имя. Позднее редактор связей объединит ваш код с объектным кодом этой библиотечной функции. Этот процесс называется *редактированием связей* (linking). Некоторые компиляторы имеют свои собственные редакторы связей, остальные используют редактор связей, предусмотренный операционной системой.

Функции в библиотеке имеют *машинезависимый* формат (relocatable format). Это значит, что адреса памяти для разных машинных инструкций не являются абсолютными — сохраняется лишь информация о смещении их адреса. Фактические адреса ячеек, используемых стандартными функциями, определяются во время редактирования связей. Подробное описание этих процессов можно найти в соответствующих технических руководствах. Более детальные объяснения были бы излишни.

В стандартной библиотеке содержится много функций, которые могли бы оказаться полезными. Они представляют собой крупные строительные блоки, из которых можно сконструировать свою программу. В частности, если какую-то функцию вы используете в своих программах очень часто, ее следует поместить в библиотеку.

Раздельная компиляция

Большинство коротких программ обычно можно уместить в одном файле. Однако по мере возрастания объема программы увеличивается время ее компиляции. Для решения этой проблемы в языке C/C++ предусмотрена возможность делить программу на файлы и компилировать каждый из них отдельно. Скомпилировав все файлы, отредактировав связи между ними и библиотечными функциями, вы получите заверченный объектный код. Преимущество раздельной компиляции заключается в том, что при изменении кода, записанного в одном из файлов, нет необходимости компилировать заново всю программу. Это существенно экономит время на этапе компиляции. Документация, сопровождающая компиляторы языка C/C++, содержит инструкции, которые позволят вам скомпилировать программу, записанную в нескольких файлах.

Расширения файлов .C и .CPP

Разумеется, программы, приведенные в части I, являются вполне корректными программами на языке C++. Их можно компилировать с помощью любого современного компилятора языка C++. Одновременно они являются корректными программами на языке C и могут компилироваться с помощью компиляторов этого языка. Итак, если вы собираетесь писать программы на языке C, можете рассматривать программы из первой части в качестве примера. Традиционно программы на языке C используют расширения файла **.C**, а программы на языке C++ — **.CPP**. Компилятор языка C++ использует расширение файла для определения типа программы. Это имеет большое значение, поскольку компилятор рассматривает любую программу, использующую расширение **.C**, как программу на языке C, а программу, записанную в файл с расширением **.CPP**, — как программу на языке C++. Если обратное не указано явно, вы можете выбрать любое расширение для файла с программой из первой

части книги. Однако программы, приведенные в остальной части книги, должны быть записаны в файлы с расширением **.CPP**.

И последнее: хотя язык C является подмножеством языка C++, между ними существует несколько отличий. В некоторых случаях программу на языке C нужно компилировать *именно как программу на языке C* (используя расширение **.C**). Все такие случаи мы оговариваем отдельно.