

# Содержание

<b>Предисловие</b>	<b>25</b>
Обращение к студентам	28
Обращение к преподавателям	29
Стандарт ISO	30
Поддержка	31
Благодарности	31
<b>Глава 0. Обращение к читателям</b>	<b>33</b>
0.1. Структура книги	34
0.1.1. Общие принципы	35
0.1.2. Упражнения, задачи и т.п.	37
0.1.3. Что дальше	38
0.2. Принципы обучения и изучения	39
0.2.1. Порядок изложения	42
0.2.2. Программирование и языки программирования	44
0.2.3. Переносимость	45
0.3. Программирование и компьютерные науки	46
0.4. Творческое начало и решение задач	46
0.5. Обратная связь	47
0.6. Библиографические ссылки	47
0.7. Биографии	48
Бьярне Страуструп	48
Лоуренс “Пит” Петерсен	49
Ждем ваших отзывов!	51
<b>Глава 1. Компьютеры, люди и программирование</b>	<b>53</b>
1.1. Введение	54
1.2. Программное обеспечение	55
1.3. Люди	58
1.4. Компьютерные науки	62
1.5. Компьютеры повсюду	62
1.5.1. С экранами и без них	63
1.5.2. Кораблестроение	64
1.5.3. Телекоммуникации	65
1.5.4. Медицина	68
1.5.5. Информация	69
1.5.6. Вид сверху	71
1.5.7. И что?	73
1.6. Идеалы программистов	73

<b>Часть I. Основы</b>	81
<b>Глава 2. Hello, World!</b>	83
2.1. Программы	84
2.2. Классическая первая программа	85
2.3. Компиляция	88
2.4. Редактирование связей	91
2.5. Среды программирования	92
<b>Глава 3. Объекты, типы и значения</b>	99
3.1. Ввод	100
3.2. Переменные	102
3.3. Ввод и тип	104
3.4. Операции и операторы	106
3.5. Присваивание и инициализация	109
3.5.1. Пример: выявление повторяющихся слов	111
3.6. Составные операторы присваивания	113
3.6.1. Пример: выявление повторяющихся слов	114
3.7. Имена	115
3.8. Типы и объекты	117
3.9. Безопасность типов	119
3.9.1. Безопасные преобразования	120
3.9.2. Небезопасные преобразования	121
<b>Глава 4. Вычисления</b>	129
4.1. Вычисления	130
4.2. Цели и средства	132
4.3. Выражения	135
4.3.1. Константные выражения	136
4.3.2. Операторы	138
4.3.3. Преобразования	140
4.4. Инструкции	141
4.4.1. Инструкции выбора	143
4.4.2. Итерация	149
4.5. Функции	154
4.5.1. Зачем нужны функции	156
4.5.2. Объявления функций	157
4.6. Вектор	158
4.6.1. Обход вектора	160
4.6.2. Увеличение вектора	160
4.6.3. Числовой пример	161
4.6.4. Текстовый пример	164
4.7. Языковые возможности	166
<b>Глава 5. Ошибки</b>	173
5.1. Введение	174
5.2. Источники ошибок	176

5.3. Ошибки времени компиляции	177
5.3.1. Синтаксические ошибки	178
5.3.2. Ошибки, связанные с типами	179
5.3.3. Не ошибки	180
5.4. Ошибки времени редактирования связей	181
5.5. Ошибки времени выполнения программы	182
5.5.1. Обработка ошибок в вызывающем коде	183
5.5.2. Обработка ошибок в вызываемом коде	185
5.5.3. Сообщения об ошибках	187
5.6. Исключения	188
5.6.1. Неправильные аргументы	189
5.6.2. Ошибки, связанные с диапазоном	190
5.6.3. Неправильный ввод	192
5.6.4. Сужающие преобразования	196
5.7. Логические ошибки	197
5.8. Оценка	200
5.9. Отладка	201
5.9.1. Практические советы по отладке	203
5.10. Пред- и постусловия	207
5.10.1. Постусловия	209
5.11. Тестирование	210
<b>Глава 6. Написание программ</b>	<b>217</b>
6.1. Задача	218
6.2. Размышления над задачей	219
6.2.1. Стадии разработки программы	220
6.2.2. Стратегия	220
6.3. Вернемся к калькулятору	223
6.3.1. Первая попытка	224
6.3.2. Лексемы	226
6.3.3. Реализация лексем	228
6.3.4. Использование лексем	230
6.3.5. Назад к школьной доске!	232
6.4. Грамматики	233
6.4.1. Отступление: грамматика английского языка	238
6.4.2. Написание грамматики	239
6.5. Превращение грамматики в программу	241
6.5.1. Реализация грамматических правил	241
6.5.2. Выражения	242
6.5.3. Термы	246
6.5.4. Первичные выражения	248
6.6. Испытание первой версии	249
6.7. Испытание второй версии	254
6.8. Поток лексем	255
6.8.1. Реализация класса <code>Token_stream</code>	257

6.8.2. Чтение лексем	259
6.8.3. Считывание чисел	260
6.9. Структура программы	261
<b>Глава 7. Завершение программы</b>	<b>267</b>
7.1. Введение	268
7.2. Ввод и вывод	268
7.3. Обработка ошибок	270
7.4. Отрицательные числа	275
7.5. Остаток от деления: %	276
7.6. Приведение кода в порядок	278
7.6.1. Символические константы	278
7.6.2. Использование функций	280
7.6.3. Размещение кода	281
7.6.4. Комментарии	283
7.7. Восстановление после ошибок	285
7.8. Переменные	288
7.8.1. Переменные и определения	288
7.8.2. Использование имен	293
7.8.3. Предопределенные имена	296
7.8.4. Это все?	296
<b>Глава 8. Технические детали: функции и прочее</b>	<b>301</b>
8.1. Технические детали	302
8.2. Объявления и определения	303
8.2.1. Виды объявлений	308
8.2.2. Объявления переменных и констант	308
8.2.3. Инициализация по умолчанию	310
8.3. Заголовочные файлы	310
8.4. Область видимости	313
8.5. Вызов функции и возврат значения	319
8.5.1. Объявление аргументов и тип возвращаемого значения	319
8.5.2. Возврат значения	321
8.5.3. Передача параметров по значению	322
8.5.4. Передача параметров по константной ссылке	323
8.5.5. Передача параметров по ссылке	325
8.5.6. Сравнение механизмов передачи параметров по значению и по ссылке	328
8.5.7. Проверка аргументов и преобразование типов	331
8.5.8. Реализация вызова функции	332
8.5.9. constexpr-функции	337
8.6. Порядок вычислений	338
8.6.1. Вычисление выражения	340
8.6.2. Глобальная инициализация	340
8.7. Пространства имен	342
8.7.1. Объявления using и директивы using	343

<b>Глава 9. Технические детали: классы и прочее</b>	<b>351</b>
9.1. Типы, определенные пользователем	352
9.2. Классы и члены класса	354
9.3. Интерфейс и реализация	354
9.4. Разработка класса	356
9.4.1. Структура и функции	356
9.4.2. Функции-члены и конструкторы	358
9.4.3. Соккрытие деталей	360
9.4.4. Определение функций-членов	362
9.4.5. Ссылка на текущий объект	365
9.4.6. Сообщения об ошибках	365
9.5. Перечисления	367
9.5.1. “Простые” перечисления	369
9.6. Перегрузка операторов	370
9.7. Интерфейсы классов	371
9.7.1. Типы аргументов	372
9.7.2. Копирование	375
9.7.3. Конструкторы по умолчанию	376
9.7.4. Константные функции-члены	379
9.7.5. Члены и вспомогательные функции	381
9.8. Класс Date	383
<b>Часть II. Ввод и вывод</b>	<b>391</b>
<b>Глава 10. Потоки ввода и вывода</b>	<b>393</b>
10.1. Ввод и вывод	394
10.2. Модель потока ввода-вывода	396
10.3. Файлы	398
10.4. Открытие файла	399
10.5. Чтение и запись файла	401
10.6. Обработка ошибок ввода-вывода	403
10.7. Считывание отдельного значения	407
10.7.1. Разделение задачи на управляемые части	409
10.7.2. Отделение диалога от функции	412
10.8. Операторы вывода, определенные пользователем	413
10.9. Операторы ввода, определенные пользователем	414
10.10. Стандартный цикл ввода	415
10.11. Чтение структурированного файла	417
10.11.1. Представление в памяти	418
10.11.2. Чтение структурированных значений	420
10.11.3. Изменение представлений	424
<b>Глава 11. Настройка ввода и вывода</b>	<b>429</b>
11.1. Регулярность и нерегулярность	430
11.2. Форматирование вывода	431
11.2.1. Вывод целых чисел	431

11.2.2. Ввод целых чисел	434
11.2.3. Вывод чисел с плавающей точкой	435
11.2.4. Точность	436
11.2.5. Поля	437
11.3. Открытие файла и позиционирование	438
11.3.1. Режимы открытия файлов	439
11.3.2. Бинарные файлы	440
11.3.3. Позиционирование в файлах	443
11.4. Строковые потоки	444
11.5. Ввод, ориентированный на строки	446
11.6. Классификация символов	447
11.7. Использование нестандартных разделителей	449
11.8. И еще много чего	456
<b>Глава 12. Модель вывода на экран</b>	<b>463</b>
12.1. Почему графика?	464
12.2. Модель вывода на дисплей	465
12.3. Первый пример	467
12.4. Использование библиотеки графического пользовательского интерфейса	471
12.5. Координаты	472
12.6. Класс Shape	473
12.7. Использование примитивов Shape	474
12.7.1. Заголовочные файлы и функция main	474
12.7.2. Почти пустое окно	475
12.7.3. Оси координат	477
12.7.4. График функции	479
12.7.5. Многоугольники	480
12.7.6. Прямоугольник	481
12.7.7. Заполнение	484
12.7.8. Текст	484
12.7.9. Изображения	486
12.7.10. И многое другое	487
12.8. Запуск программы	488
12.8.1. Исходные файлы	490
<b>Глава 13. Графические классы</b>	<b>495</b>
13.1. Обзор графических классов	496
13.2. Классы Point и Line	498
13.3. Класс Lines	501
13.4. Класс Color	504
13.5. Класс Line_style	507
13.6. Класс Open_polyline	509
13.7. Класс Closed_polyline	510
13.8. Класс Polygon	512
13.9. Класс Rectangle	514

13.10. Управление неименованными объектами	519
13.11. Класс Text	521
13.12. Класс Circle	523
13.13. Класс Ellipse	525
13.14. Класс Marked_polyline	527
13.15. Класс Marks	529
13.16. Класс Mark	530
13.17. Класс Image	532
<b>Глава 14. Проектирование графических классов</b>	<b>539</b>
14.1. Принципы проектирования	540
14.1.1. Типы	540
14.1.2. Операции	542
14.1.3. Именованное	543
14.1.4. Изменяемость	545
14.2. Класс Shape	546
14.2.1. Абстрактный класс	547
14.2.2. Управление доступом	549
14.2.3. Рисование фигур	552
14.2.4. Копирование и изменчивость	556
14.3. Базовые и производные классы	557
14.3.1. Схема объекта	559
14.3.2. Порождение классов и определение виртуальных функций	561
14.3.3. Перекрытие	562
14.3.4. Доступ	564
14.3.5. Чисто виртуальные функции	565
14.4. Преимущества объектно-ориентированного программирования	567
<b>Глава 15. Графическое представление функций и данных</b>	<b>575</b>
15.1. Введение	576
15.2. Графики простых функций	576
15.3. Класс Function	580
15.3.1. Аргументы по умолчанию	582
15.3.2. Другие примеры	583
15.3.3. Лямбда-выражения	585
15.4. Оси координат	586
15.5. Аппроксимация	588
15.6. Графическое представление данных	594
15.6.1. Чтение файла	596
15.6.2. Общая схема	598
15.6.3. Масштабирование данных	598
15.6.4. Построение графика	600

<b>Глава 16. Графические пользовательские интерфейсы</b>	<b>607</b>
16.1. Альтернативы пользовательского интерфейса	608
16.2. Кнопка Next	609
16.3. Простое окно	611
16.3.1. Функции обратного вызова	613
16.3.2. Цикл ожидания	616
16.3.3. Лямбда-выражения в качестве функций обратного вызова	617
16.4. Класс Button и другие разновидности Widget	618
16.4.1. Класс Widget	618
16.4.2. Класс Button	620
16.4.3. Классы In_box и Out_box	620
16.4.4. Класс Menu	621
16.5. Пример	622
16.6. Инверсия управления	626
16.7. Добавление меню	627
16.8. Отладка GUI-программы	632
<b>Часть III. Данные и алгоритмы</b>	<b>639</b>
<b>Глава 17. Векторы и динамически выделяемая память</b>	<b>641</b>
17.1. Введение	642
17.2. Основы	644
17.3. Память, адреса и указатели	646
17.3.1. Оператор sizeof	649
17.4. Динамически распределяемая память и указатели	650
17.4.1. Размещение в динамической памяти	651
17.4.2. Доступ с помощью указателей	653
17.4.3. Диапазоны	654
17.4.4. Инициализация	656
17.4.5. Нулевой указатель	657
17.4.6. Освобождение памяти	658
17.5. Деструкторы	661
17.5.1. Генерируемые деструкторы	663
17.5.2. Деструкторы и динамическая память	664
17.6. Доступ к элементам	665
17.7. Указатели на объекты класса	666
17.8. Путаница с типами: void* и операторы приведения типов	668
17.9. Указатели и ссылки	671
17.9.1. Указатели и ссылки как параметры функций	672
17.9.2. Указатели, ссылки и наследование	673
17.9.3. Пример: списки	674
17.9.4. Операции над списками	676
17.9.5. Использование списков	677
17.10. Указатель this	679
17.10.1. Еще об использовании списков	681



<b>Глава 18. Векторы и массивы</b>	687
18.1. Введение	688
18.2. Инициализация	689
18.3. Копирование	691
18.3.1. Копирующие конструкторы	693
18.3.2. Копирующее присваивание	694
18.3.3. Терминология, связанная с копированием	696
18.3.4. Перемещение	697
18.4. Основные операции	700
18.4.1. Явные конструкторы	702
18.4.2. Отладка конструкторов и деструкторов	703
18.5. Доступ к элементам вектора	706
18.5.1. Константная перегрузка	707
18.6. Массивы	708
18.6.1. Указатели на элементы массива	710
18.6.2. Указатели и массивы	712
18.6.3. Инициализация массива	714
18.6.4. Проблемы с указателями	716
18.7. Примеры: палиндром	719
18.7.1. Палиндромы с использованием <code>string</code>	719
18.7.2. Палиндромы с использованием массивов	720
18.7.3. Палиндромы с использованием указателей	721
<b>Глава 19. Векторы, шаблоны и исключения</b>	729
19.1. Проблемы	730
19.2. Изменение размера	734
19.2.1. Представление	734
19.2.2. Функции <code>reserve</code> и <code>capacity</code>	736
19.2.3. Функция <code>resize</code>	736
19.2.4. Функция <code>push_back</code>	737
19.2.5. Присваивание	738
19.2.6. Текущее состояние дел	740
19.3. Шаблоны	741
19.3.1. Типы как шаблонные параметры	741
19.3.2. Обобщенное программирование	744
19.3.3. Концепции	747
19.3.4. Контейнеры и наследование	750
19.3.5. Целые числа как параметры шаблонов	751
19.3.6. Вывод аргументов шаблона	753
19.3.7. Обобщение класса <code>vector</code>	753
19.4. Проверка диапазона и исключения	757
19.4.1. Отступление от темы: вопросы проектирования	758
19.4.2. Признание в использовании макроса	760
19.5. Ресурсы и исключения	762
19.5.1. Потенциальные проблемы управления ресурсами	763
19.5.2. Захват ресурса — это инициализация	765

19.5.3. Гарантии	766
19.5.4. Класс <code>unique_ptr</code>	767
19.5.5. Возврат посредством перемещения	769
19.5.6. RAII для класса <code>vector</code>	770
<b>Глава 20. Контейнеры и итераторы</b>	<b>777</b>
20.1. Хранение и обработка данных	778
20.1.1. Работа с данными	779
20.1.2. Обобщение кода	780
20.2. Принципы библиотеки STL	783
20.3. Последовательности и итераторы	787
20.3.1. Вернемся к примерам	790
20.4. Связанные списки	791
20.4.1. Операции над списками	793
20.4.2. Итерация	794
20.5. Очередное обобщение класса <code>vector</code>	797
20.5.1. Обход контейнера	799
20.5.2 <code>auto</code>	799
20.6. Пример: простой текстовый редактор	801
20.6.1. Строки	803
20.6.2. Итерация	804
20.7. Классы <code>vector</code> , <code>list</code> и <code>string</code>	808
20.7.1. Операции <code>insert</code> и <code>erase</code>	810
20.8. Адаптация нашего класса <code>vector</code> к STL	813
20.9. Адаптация встроенных массивов к библиотеке STL	815
20.10. Обзор контейнеров	817
20.10.1. Категории итераторов	820
<b>Глава 21. Алгоритмы и ассоциативные массивы</b>	<b>827</b>
21.1. Алгоритмы стандартной библиотеки	828
21.2. Простейший алгоритм: <code>find()</code>	829
21.2.1. Примеры использования обобщенных алгоритмов	832
21.3. Универсальный поиск: <code>find_if()</code>	833
21.4. Функциональные объекты	835
21.4.1. Абстрактная точка зрения на функциональные объекты	836
21.4.2. Предикаты на членах класса	838
21.4.3. Лямбда-выражения	839
21.5. Численные алгоритмы	840
21.5.1. Алгоритм <code>accumulate()</code>	841
21.5.2. Обобщение алгоритма <code>accumulate()</code>	842
21.5.3. Алгоритм <code>inner_product()</code>	844
21.5.4. Обобщение алгоритма <code>inner_product()</code>	845
21.6. Ассоциативные контейнеры	846
21.6.1. Контейнер <code>map</code>	847
21.6.2. Обзор контейнера <code>map</code>	849
21.6.3. Еще один пример использования <code>map</code>	852

21.6.4. Контейнер <code>unordered_map</code>	854
21.6.5. Контейнер <code>set</code>	857
21.7. Копирование	859
21.7.1. Алгоритм <code>copy</code>	859
21.7.2. Итераторы потоков	860
21.7.3. Использование <code>set</code> для поддержки упорядоченности	863
21.7.4. Алгоритм <code>copy_if</code>	863
21.8. Сортировка и поиск	864
21.9. Алгоритмы контейнеров	866

## Часть IV. Дополнительные темы 873

### Глава 22. Идеалы и история 875

22.1. История, идеалы и профессионализм	876
22.1.1. Цели и философия языка программирования	877
22.1.2. Идеалы программирования	878
22.1.3. Стили и парадигмы	887
22.2. Обзор истории языков программирования	891
22.2.1. Первые языки программирования	892
22.2.2. Корни современных языков программирования	894
22.2.3. Семейство языков Algol	901
22.2.4. Язык программирования Simula	909
22.2.5. Язык программирования C	911
22.2.6. Язык программирования C++	915
22.2.7. Современное состояние дел	919
22.2.8. Источники информации	920

### Глава 23. Работа с текстом 925

23.1. Текст	926
23.2. Строки	926
23.3. Потоки ввода-вывода	930
23.4. Ассоциативные контейнеры	931
23.4.1. Детали реализации	937
23.5. Проблема	939
23.6. Идея регулярных выражений	941
23.6.1. Необработанные строковые литералы	944
23.7. Поиск с помощью регулярных выражений	945
23.8. Синтаксис регулярных выражений	947
23.8.1. Символы и специальные символы	948
23.8.2. Классы символов	949
23.8.3. Повторения	950
23.8.4. Группировка	951
23.8.5. Альтернативы	951
23.8.6. Наборы символов и диапазоны	952
23.8.7. Ошибки в регулярных выражениях	954

23.9. Сопоставление регулярных выражений	955
23.10. Ссылки	960
<b>Глава 24. Числа</b>	<b>965</b>
24.1. Введение	966
24.2. Размер, точность и переполнение	967
24.2.1. Пределы числовых диапазонов	970
24.3. Массивы	971
24.4. Многомерные массивы в стиле языка C	972
24.5. Библиотека <code>Matrix</code>	974
24.5.1. Размерности и доступ	975
24.5.2. Одномерная матрица	978
24.5.3. Двумерные матрицы	981
24.5.4. Ввод-вывод матриц	984
24.5.5. Трехмерные матрицы	984
24.6. Пример: решение систем линейных уравнений	985
24.6.1. Классическое исключение Гаусса	987
24.6.2. Выбор опорного элемента	988
24.6.3. Тестирование	989
24.7. Случайные числа	991
24.8. Стандартные математические функции	994
24.9. Комплексные числа	995
24.10. Ссылки	997
<b>Глава 25. Программирование встроенных систем</b>	<b>1003</b>
25.1. Встроенные системы	1004
25.2. Основные концепции	1008
25.2.1. Предсказуемость	1011
25.2.2. Идеалы	1012
25.2.3. Сохранение работоспособности после сбоя	1013
25.3. Управление памятью	1015
25.3.1. Проблемы с динамической памятью	1017
25.3.2. Альтернативы динамической памяти	1020
25.3.3. Пример пула	1021
25.3.4. Пример стека	1023
25.4. Адреса, указатели и массивы	1024
25.4.1. Непроверяемые преобразования	1024
25.4.2. Проблема: дисфункциональный интерфейс	1025
25.4.3. Решение: интерфейсный класс	1029
25.4.4. Наследование и контейнеры	1032
25.5. Биты, байты и слова	1036
25.5.1. Операции с битами и байтами	1036
25.5.2. Класс <code>bitset</code>	1041
25.5.3. Целые числа со знаком и без знака	1042
25.5.4. Работа с битами	1047
25.5.5. Битовые поля	1049

25.5.6. Пример: простое шифрование	1051
25.6. Стандарты кодирования	1056
25.6.1. Каким должен быть стандарт кодирования?	1058
25.6.2. Примеры правил	1059
25.6.3. Реальные стандарты кодирования	1065
<b>Глава 26. Тестирование</b>	<b>1073</b>
26.1. Чего мы хотим	1074
26.1.1. Предостережение	1076
26.2. Доказательства	1076
26.3. Тестирование	1076
26.3.1. Регрессивные тесты	1078
26.3.2. Модульные тесты	1078
26.3.3. Алгоритмы и не алгоритмы	1086
26.3.4. Системные тесты	1094
26.3.5. Поиск предположений, которые не выполняются	1095
26.4. Проектирование с учетом тестирования	1097
26.5. Отладка	1098
26.6. Производительность	1099
26.6.1. Измерение времени	1101
26.7. Ссылки	1103
<b>Глава 27. Язык программирования C</b>	<b>1107</b>
27.1. Языки C и C++: братья	1108
27.1.1. Совместимость языков C и C++	1111
Ссылки	1112
27.1.2. Возможности C++, отсутствующие в C	1112
27.1.3. Стандартная библиотека языка C	1114
27.2. Функции	1115
27.2.1. Отсутствие перегрузки имен функций	1116
27.2.2. Проверка типов аргументов функций	1116
27.2.3. Определения функций	1118
27.2.4. Вызов C-функций из C++-программы и наоборот	1120
27.2.5. Указатели на функции	1122
27.3. Второстепенные языковые различия	1123
27.3.1. Дескриптор пространства имен <b>struct</b>	1124
27.3.2. Ключевые слова	1125
27.3.3. Определения	1125
27.3.4. Приведение типов в стиле языка C	1127
27.3.5. Преобразование указателей типа <b>void*</b>	1128
27.3.6. Перечисление	1129
27.3.7. Пространства имен	1130
27.4. Динамическая память	1130
27.5. Строки в стиле C	1132
27.5.1. Строки в стиле C и ключевое слово <b>const</b>	1135
27.5.2. Операции над байтами	1136

27.5.3. Пример: функция <code>strcpy()</code>	1136
27.5.4. Вопросы стиля	1137
27.6. Ввод-вывод: заголовочный файл <code>stdio.h</code>	1138
27.6.1. Вывод	1138
27.6.2. Ввод	1139
27.6.3. Файлы	1141
27.7. Константы и макросы	1142
27.8. Макросы	1143
27.8.1. Макросы, похожие на функции	1144
27.8.2. Синтаксические макросы	1145
27.8.3. Условная компиляция	1146
27.9. Пример: интрузивные контейнеры	1147

## Часть V. Приложения 1159

Приложение А. Обзор языка	1161
А.1. Общие сведения	1162
А.1.1. Терминология	1163
А.1.2. Запуск и завершение программы	1164
А.1.3. Комментарии	1164
А.2. Литералы	1165
А.2.1. Целочисленные литералы	1165
А.2.2. Литералы с плавающей точкой	1167
А.2.3. Булевы литералы	1168
А.2.4. Символьные литералы	1168
А.2.5. Строковые литералы	1169
А.2.6. Указательные литералы	1169
А.3. Идентификаторы	1169
А.3.1. Ключевые слова	1170
А.4. Область видимости, класс памяти и время жизни	1170
А.4.1. Область видимости	1171
А.4.2. Класс памяти	1172
А.4.3. Время жизни	1173
А.5. Выражения	1174
А.5.1. Операторы, определенные пользователем	1178
А.5.2. Неявное преобразование типа	1179
А.5.3. Константные выражения	1181
А.5.4. Оператор <code>sizeof</code>	1182
А.5.5. Логические выражения	1182
А.5.6. Операторы <code>new</code> и <code>delete</code>	1182
А.5.7. Операторы приведения	1183
А.6. Инструкции	1184
А.7. Объявления	1186
А.7.1. Определения	1187
А.8. Встроенные типы	1187
А.8.1. Указатели	1188

А.8.2. Массивы	1190
А.8.3. Ссылки	1191
А.9. Функции	1191
А.9.1. Разрешение перегрузки	1192
А.9.2. Аргументы по умолчанию	1193
А.9.3. Неопределенные аргументы	1194
А.9.4. Спецификации связей	1194
А.10. Типы, определенные пользователем	1195
А.10.1. Перегрузка операций	1195
А.11. Перечисления	1196
А.12. Классы	1197
А.12.1. Доступ к членам класса	1197
А.12.2. Определения членов класса	1200
А.12.3. Создание, уничтожение и копирование	1201
А.12.4. Производные классы	1204
А.12.5. Битовые поля	1208
А.12.6. Объединения	1209
А.13. Шаблоны	1209
А.13.1. Аргументы шаблонов	1210
А.13.2. Инстанцирование шаблонов	1211
А.13.3. Шаблонные типы членов-классов	1212
А.14. Исключения	1213
А.15. Пространства имен	1215
А.16. Псевдонимы	1216
А.17. Директивы препроцессора	1216
А.17.1. Директива <code>#include</code>	1217
А.17.2. Директива <code>#define</code>	1217
<b>Приложение Б. Обзор стандартной библиотеки</b>	<b>1219</b>
Б.1. Обзор	1220
Б.1.1. Заголовочные файлы	1221
Б.1.2. Пространство имен <code>std</code>	1224
Б.1.3. Стиль описания	1224
Б.2. Обработка ошибок	1225
Б.2.1. Исключения	1225
Б.3. Итераторы	1227
Б.3.1. Модель итераторов	1227
Б.3.2. Категории итераторов	1229
Б.4. Контейнеры	1231
Б.4.1. Обзор	1233
Б.4.2. Члены-типы	1234
Б.4.3. Конструкторы, деструкторы и присваивания	1234
Б.4.4. Итераторы	1235
Б.4.5. Доступ к элементам	1235
Б.4.6. Операции над стеком и очередью	1236
Б.4.7. Операции со списком	1237

Б.4.8. Размер и емкость	1237
Б.4.9. Другие операции	1238
Б.4.10. Операции над ассоциативными контейнерами	1238
Б.5. Алгоритмы	1239
Б.5.1. Немодифицирующие алгоритмы для последовательностей	1240
Б.5.2. Алгоритмы, модифицирующие последовательности	1241
Б.5.3. Вспомогательные алгоритмы	1244
Б.5.4. Сортировка и поиск	1244
Б.5.5. Алгоритмы для множеств	1246
Б.5.6. Пирамиды	1247
Б.5.7. Перестановки	1248
Б.5.8. Функции <code>min</code> и <code>max</code>	1249
Б.6. Утилиты библиотеки STL	1250
Б.6.1. Итераторы вставки	1250
Б.6.2. Функциональные объекты	1250
Б.6.3. Классы <code>pair</code> и <code>tuple</code>	1252
Б.6.4. Список инициализации	1253
Б.6.5. Указатели управления ресурсами	1254
Б.7. Потоки ввода-вывода	1255
Б.7.1. Иерархия потоков ввода-вывода	1256
Б.7.2. Обработка ошибок	1258
Б.7.3. Операции ввода	1258
Б.7.4. Операции вывода	1259
Б.7.5. Форматирование	1260
Б.7.6. Стандартные манипуляторы	1260
Б.8. Работа со строками	1261
Б.8.1. Классификация символов	1262
Б.8.2. Строки	1262
Б.8.3. Регулярные выражения	1264
Б.9. Работа с числами	1266
Б.9.1. Предельные значения	1266
Б.9.2. Стандартные математические функции	1267
Б.9.3. Комплексные числа	1268
Б.9.4. Класс <code>valarray</code>	1269
Б.9.5. Обобщенные численные алгоритмы	1269
Б.9.6. Случайные числа	1270
Б.10. Работа со временем	1270
Б.11. Функции стандартной библиотеки языка C	1271
Б.11.1. Файлы	1271
Б.11.2. Семейство функций <code>printf()</code>	1272
Б.11.3. C-строки	1277
Б.11.4. Память	1278
Б.11.5. Дата и время	1279
Б.11.6. Другие функции	1280
Б.12. Другие библиотеки	1281



<b>Приложение В. Начала работы с Visual Studio</b>	1283
В.1. Запуск программы	1284
В.2. Инсталляция Visual Studio	1284
В.3. Создание и запуск программ	1285
В.3.1. Создание нового проекта	1285
В.3.2. Использование заголовочного файла <code>std_lib_facilities.h</code>	1285
В.3.3. Добавление в проект исходного файла на языке C++	1286
В.3.4. Ввод исходного кода	1286
В.3.5. Создание выполнимой программы	1286
В.3.6. Выполнение программы	1287
В.3.7. Сохранение программы	1287
В.4. Что дальше	1287
<b>Приложение Г. Установка FLTK</b>	1289
Г.1. Введение	1290
Г.2. Загрузка библиотеки FLTK	1290
Г.3. Установка библиотеки FLTK	1291
Г.4. Использование библиотеки FLTK в среде Visual Studio	1292
Г.5. Тестирование, все ли работает	1292
<b>Приложение Д. Реализация графического пользовательского интерфейса</b>	1295
Д.1. Реализация обратных вызовов	1296
Д.2. Реализация класса <code>Widget</code>	1297
Д.3. Реализация класса <code>Window</code>	1298
Д.4. Реализация класса <code>Vector_ref</code>	1300
Д.5. Пример: работа с объектами <code>Widget</code>	1301
<b>Глоссарий</b>	1305
<b>Библиография</b>	1313
<b>Предметный указатель</b>	1316
<b>Фотографии</b>	1327



# Hello, World!

*Программирование изучается  
с помощью написания программ.  
— Брайан Керниган (Brian Kernighan)*

**В** этой главе приводится простейшая программа на языке C++, которая что-то делает. Цели создания этой программы следующие.

- Дать вам возможность поработать с интегрированной средой разработки программ.
- Дать вам почувствовать, как можно заставить компьютер сделать что-то для вас.

Словом, в этой главе мы представим понятие программы, идею о преобразовании программ с помощью компилятора из текстовой формы, понятной для человека, в машинные команды для последующего выполнения компьютером.

---

## 2.1. Программы

### 2.2. Классическая первая программа

### 2.3. Компиляция

### 2.4. Редактирование связей

### 2.5. Среды программирования

---

## 2.1. Программы

Для того чтобы заставить компьютер сделать что-то, вы (или кто-то еще) должны точно рассказать ему — со всеми подробностями, — что именно от него требуется. Описание того, “что следует сделать”, называется *программой*, а *программирование* — это вид деятельности, который заключается в создании и отладке таких программ.

В некотором смысле мы все уже программисты. В конце концов, мы множество раз получали описательные задания, которые должны были выполнить, например “как идти в школу на уроки” или “как поджарить мясо в микроволновой печи”. Разница между такими описаниями и программами заключается в степени точности: люди стараются компенсировать неточность инструкций, руководствуясь здравым смыслом, а компьютеры этого сделать не могут. Например, “по коридору направо, вверх по лестнице, а потом налево” — вероятно, прекрасная инструкция, позволяющая найти нужный кабинет на верхнем этаже. Однако если вы внимательно посмотрите на эти простые инструкции, то увидите, что они являются грамматически неточными и неполными. Человек может легко восполнить этот недостаток. Представим, например, что вы вошли в здание и спрашиваете, как найти кабинет нужного вам человека. Отвечающий вам совершенно не обязан говорить, чтобы вы прошли через дверь, открыв ее поворотом ручки (а не выбив ее ногой), не толкнули в холле вазон с цветами и т.д. Вам также никто не скажет, чтобы вы были осторожны, поднимаясь по лестнице, и постучали, добравшись до нужной двери. Как открыть дверь в кабинет, прежде чем войти в него, вам, вероятно, также не будут рассказывать.

В противоположность этому компьютер *действительно* глуп. Ему все необходимо точно и подробно описать. Вернемся к инструкциям “по коридору направо, вверх по лестнице, а потом налево”. Где находится коридор? Что такое коридор? Что значит “направо”? Что такое лестница? Как подняться по лестнице? По одной ступеньке? Через две ступеньки? Держась за перила? Что находится слева от меня? Когда это окажется слева от меня? Для того чтобы подробно описать инструкции для компьютера, необходим точно определенный язык, имеющий специфическую грамматику (естественный язык слишком слабо структурирован), а также хорошо

определенный словарь для всех видов действий, которые мы хотим выполнить. Такой язык называется языком *программирования*, и язык программирования C++ — один из таких языков, разработанных для решения широкого круга задач.

Более широкие философские взгляды на компьютеры, программы и программирование изложены в главе 1. Здесь же мы рассмотрим код, начиная с очень простой программы, а также несколько инструментов и методов, необходимых для ее выполнения.

## 2.2. Классическая первая программа

Приведем вариант классической первой программы. Она выводит на экран сообщение `Hello, World!`.

```
// Эта программа выводит на экран сообщение "Hello,World!"
#include "std_lib_facilities.h"
int main() // Программы на C++ начинаются с выполнения функции main
{
    cout << "Hello, World!\n"; // Вывод "Hello,World!"
    return 0;
}
```

Рассматривайте этот текст как набор команд, которые должен выполнить компьютер; это напоминает кулинарный рецепт или инструкции по сборке новой игрушки. Посмотрим, что делает каждая из строк программы, начиная с самого начала:

```
cout << "Hello, World!\n"; // Вывод "Hello,World!"
```



Именно эта строка выводит сообщение на экран. Она выводит символы `Hello, World!`, за которыми следует символ перехода на новую строку; иначе говоря, после вывода символов `Hello, World!` курсор будет установлен на начало новой строки. *Курсор* — это небольшой мерцающий символ или линия, показывающая, где будет выведен следующий символ.

В языке C++ строковые литералы выделяются двойными кавычками (""); т.е. `"Hello, Word!\n"` — это строка символов. `\n` — это "специальный символ", означающий переход на новую строку. Имя `cout` относится к стандартному потоку вывода. Символы, "помещенные в поток `cout`" с помощью оператора вывода `<<`, будут отображены на экране. Имя `cout` произносится как "see-out", но является аббревиатурой от "**character output stream**" ("поток вывода символов"). Аббревиатуры довольно широко распространены в программировании. Естественно, все эти сокращения на первых порах могут показаться неудобными для запоминания, но, привыкнув, вы уже не сможете от них отказаться, так как они позволяют создавать короткие и управляемые программы.

## Конец строки

```
// Вывод "Hello,World!"
```

является комментарием. Все, что написано после символа `//` (т.е. после двойной косой черты (`/`), которая называется слэшем), считается комментарием. Он игнорируется компилятором и предназначен для программистов, которые будут читать программу. В данном случае мы использовали комментарий для того, чтобы сообщить вам, что именно означает первая часть этой строки.

Комментарии описывают предназначение программы и содержат полезную информацию для людей, которую невозможно выразить в коде. Скорее всего, человеком, который извлечет пользу из ваших комментариев, окажетесь вы сами, когда вернетесь к своей программе на следующей неделе или на следующий год, забыв, для чего вы ее писали. Итак, старайтесь хорошо документировать свои программы. В разделе 7.6.4 мы обсудим, как писать хорошие комментарии.



Программа пишется для двух аудиторий. Естественно, мы пишем программы для компьютеров, которые будут их выполнять. Однако мы долгие годы проводим за чтением и модификацией кода. Таким образом, второй аудиторией программ являются другие программисты. Поэтому создание программ можно считать формой общения между людьми. В действительности имеет смысл главными читателями своей программы считать людей: если они с трудом понимают, что вы написали, то вряд ли программа когда-нибудь станет правильной. А потому нельзя забывать, что код предназначен для чтения — необходимо делать все, чтобы программа легко читалась. В любом случае комментарии нужны только людям; компьютеры их полностью игнорируют.

Первая строка программы — это типичный комментарий, который сообщает читателям, что будет делать программа.

```
// Эта программа выводит на экран сообщение "Hello,World!"
```

Такие комментарии очень полезны, так как по исходному тексту программы можно понять, что она делает, но нельзя выяснить, что мы на самом деле хотели. Кроме того, в комментариях мы можем намного лаконичнее объяснить цель программы, чем в самом коде (как правило, более подробном). Часто такие комментарии размещаются в начальной части программы и напоминают, что мы пытаемся сделать в данной программе.

## Строка

```
#include "std_lib_facilities.h"
```

представляет собой директиву `#include`. Она заставляет компьютер сделать доступными (“включить”) функциональные возможности, описанные в файле `std_lib_facilities.h`. Этот файл упрощает использование

возможностей, предусмотренных во всех реализациях языка С++ (стандартной библиотеке языка С++).

По мере продвижения вперед мы объясним эти возможности более подробно. Они написаны на стандартном языке С++, но содержат детали, в которые сейчас не стоит углубляться, отложив их изучение до следующих глав. Важность файла `std_lib_facilities.h` для данной программы заключается в том, что с его помощью мы получаем доступ к стандартным средствам ввода-вывода языка С++. Здесь мы используем только стандартный поток вывода `cout` и оператор вывода `<<`. Файл, включаемый в программу с помощью директивы `#include`, обычно имеет суффикс `.h` и называется *заголовком* (header), или *заголовочным файлом* (header file). Заголовок содержит определения терминов, таких как `cout`, которые мы используем в нашей программе.

Как компьютер узнает, с чего начинать выполнение программы? Он ищет функцию с именем `main` и начинает выполнять инструкции, содержащиеся в ней. Вот как выглядит функция `main` нашей программы:

```
int main() // Программы на С++ начинаются с выполнения функции main
{
    cout << "Hello, World!\n"; // Вывод "Hello,World!"
    return 0;
}
```

Для того чтобы указать отправную точку выполнения, каждая программа на языке С++ должна содержать функцию с именем `main`. По сути, функция представляет собой именованную последовательность инструкций, которую компьютер выполняет в порядке их перечисления. Функция состоит из четырех частей.

- *Тип возвращаемого значения*; в этой функции — тип `int` (т.е. integer, целое число), определяет, какой результат возвращает функция в точку вызова (если она возвращает какое-нибудь значение). Слово `int` является зарезервированным в языке С++ (*ключевым словом*), поэтому его нельзя использовать в качестве имени чего-нибудь иного (см. раздел А.3.1).
- *Имя*; в данном случае `main`.
- *Список параметров*, заключенный в круглые скобки (см. разделы 8.2 и 8.6); в данном случае список параметров пуст и имеет вид `()`.
- *Тело функции*, заключенное в фигурные скобки `{ }` и перечисляющее действия (*инструкции*), которые функция должна выполнить.

Отсюда следует, что минимальная программа на языке С++ выглядит так:

```
int main() { }
```


Пользы от такой программы мало, так как она ничего не делает. Тело функции `main` программы “Hello, World!” содержит две инструкции:

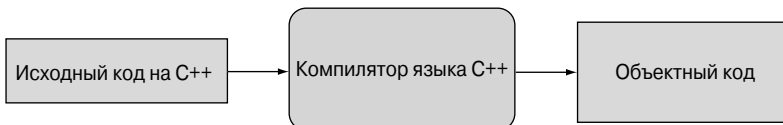
```
cout << "Hello, World!\n"; // Вывод "Hello,World!"
return 0;
```

Сначала она выводит на экран строку `Hello, World!`, а затем возвращает значение `0` (ноль) в точку вызова. Поскольку функция `main()` вызывается системой, мы возвращаемое значение использовать не будем. Однако в некоторых системах (в частности, Unix/Linux) это значение можно использовать для проверки успешности выполнения программы. Ноль (`0`), возвращаемый функцией `main()`, означает, что программа выполнена успешно.

Часть программы на языке C++, определяющая некоторое действие и не являющаяся директивой `#include` (или другой директивой препроцессора; см. разделы 4.4 и А.17), называется *инструкцией* (statement).

## 2.3. Компиляция

 C++ — компилируемый язык. Это означает, что для запуска программы сначала необходимо транслировать ее из текстовой формы, понятной для человека, в форму, понятную для машины. Эту задачу выполняет особая программа, которая называется *компилятором*. То, что вы пишете и читаете, называется *исходным кодом* или *исходным текстом программы*, а то, что выполняет компьютер, называется *выполняемым, объектным или машинным кодом*. Обычно файлы с исходным кодом программы на языке C++ имеют суффикс `.cpp` (например, `hello_world.cpp`) или `.h` (например, `std_lib_facilities.h`), а файлы с объектным кодом имеют суффикс `.obj` (в Windows) или `.o` (в Unix). Следовательно, простое слово *код* является двусмысленным и может ввести в заблуждение; его следует употреблять с осторожностью и только в ситуациях, когда недоразумение возникнуть не может. Если не указано иное, под словом *код* подразумевается “исходный код” или даже “исходный код за исключением комментариев”, поскольку комментарии предназначены для людей и компилятор не переводит их в объектный код.



Компилятор читает исходный код и пытается понять, что вы написали. Он проверяет, является ли программа грамматически корректной, определен ли смысл каждого слова. Обнаружив ошибку, компилятор сообщает о ней, не пытаясь выполнить программу. Компиляторы довольно придирчивы к синтаксису. Пропуск какой-нибудь детали, например директивы

`#include`, двоеточия или фигурной скобки, приводит к ошибке. Кроме того, компилятор точно так же абсолютно нетерпим к опечаткам. Продемонстрируем это рядом примеров, в каждом из которых сделана одна небольшая ошибка. Каждая из этих ошибок является довольно типичной.

```
// пропущен заголовочный файл
int main()
{
    cout << "Hello, World!\n";
    return 0;
}
```

Мы не сообщили компилятору о том, что представляет собой объект `cout`, поэтому он сообщает об ошибке. Для того чтобы исправить программу, следует добавить директиву `#include`.

```
#include "std_facilities.h"
int main()
{
    cout << "Hello, World!\n";
    return 0;
}
```

К сожалению, компилятор снова сообщает об ошибке, так как мы сделали опечатку в строке `std_lib_facilities.h`. Компилятор заметил это.

```
#include "std_lib_facilities.h"
int main()
{
    cout << "Hello, World!\n";
    return 0;
}
```

В этом примере мы пропустили закрывающую двойную кавычку ("). Компилятор указывает нам на это.

```
#include "std_lib_facilities.h"
integer main()
{
    cout << "Hello, World!\n";
    return 0;
}
```

Теперь мы вместо ключевого слова `int` использовали слово `integer`, которого в языке C++ нет. Компилятор таких ошибок не прощает.

```
#include "std_lib_facilities.h"
int main()
{
    cout < "Hello, World!\n";
    return 0;
}
```



Здесь вместо символов << (оператор вывода) использован символ < (оператор “меньше”). Компилятор это заметил.

```
#include "std_lib_facilities.h"
int main()
{
    cout << 'Hello, World!\n';
    return 0;
}
```

Здесь вместо двойных кавычек, ограничивающих строки, по ошибке использованы одинарные. Приведем заключительный пример.

```
#include "std_lib_facilities.h"
int main()
{
    cout << "Hello, World!\n"
    return 0;
}
```

В этой программе мы забыли завершить строку, содержащую оператор вывода, точкой с запятой. Обратите внимание на то, что в языке C++ каждая инструкция завершается точкой с запятой (;). Компилятор распознает точку с запятой как символ окончания инструкции и начала следующей. Трудно коротко, неформально и технически корректно описать все ситуации, в которых нужна точка с запятой. Пока просто запомните правило: точку с запятой следует ставить после каждого выражения, которое не завершается закрывающей фигурной скобкой }.

Для чего мы посвятили две страницы и несколько минут вашего драгоценного времени демонстрации тривиальных примеров, содержащих тривиальные ошибки? Для того, чтобы в будущем вы не тратили много времени на поиск ошибок в исходном тексте программы. Большую часть времени программисты ищут ошибки в своих программах. В конце концов, если вы убеждены, что некий код является правильным, то вы, скорее всего, обратитесь к анализу некоторого другого кода, чтобы не тратить время зря. На заре компьютерной эры первые программисты сильно удивлялись, насколько часто они делали ошибки и как долго их искали. И по сей день большинство начинающих программистов удивляются этому не меньше.



Компилятор нередко будет вас раздражать. Иногда будет казаться, что он придирается к несущественным деталям (например, к пропущенным точкам с запятыми) или к вещам, которые вы считаете абсолютно правильными. Однако компилятор, как правило, не ошибается: если уж он выводит сообщение об ошибке и отказывается создавать объектный код из вашего исходного кода, то это значит, что ваша программа не в порядке; иначе говоря, то, что вы написали, не соответствует стандарту языка C++.



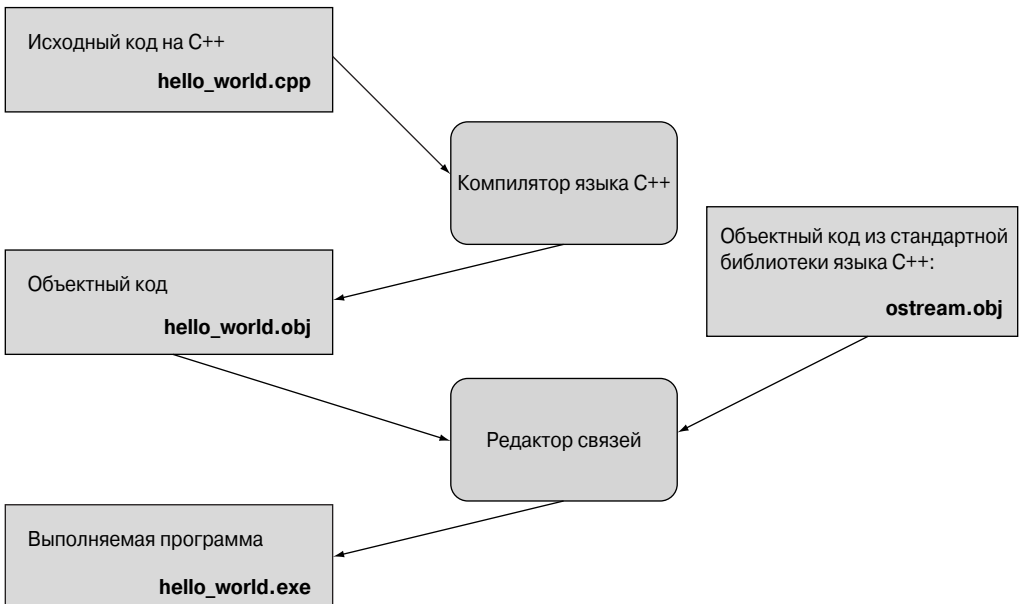
Компилятор не руководствуется здравым смыслом (это не человек!) и очень придирчив к деталям. Поскольку здравый смысл ему чужд, он не пытается угадать, что на самом деле вы имели в виду, написав фрагмент программы, который выглядит абсолютно правильным, но не соответствует стандарту языка C++. Если бы он угадывал смысл программы, но при этом результат оказался бы для вас совершенно неожиданным, вы провели бы очень много времени, пытаясь понять, почему про-



грамма не делает то, что требуется. Компилятор — наш друг, позволяющий избежать множества проблем, которые мы сами же создаем. Никогда не забывайте об этом: компилятор — не просто ваш друг; пожалуй, когда вы занимаетесь программированием, — это наилучший ваш друг.

## 2.4. Редактирование связей

Программа обычно состоит из нескольких отдельных частей, которые часто разрабатываются разными людьми. Например, программа “Hello, World!” состоит из части, которую написали вы, и частей стандартной библиотеки языка C++. Эти отдельные части (иногда называемые *единицами трансляции*) должны быть скомпилированы, а файлы с результирующим объектным кодом должны быть скомпонованы в единое целое, образуя выполняемый файл. Программа, связывающая эти части в одно целое, называется (вполне ожидаемо) компоновщиком или *редактором связей*.



Заметьте, что объектные и выполняемые коды являются *не* переносимыми из одной системы в другую. Например, когда вы компилируете программу для Windows, то получите объектный код именно для Windows, и этот код не будет работать в Linux.

*Библиотека* — это просто некоторый код (обычно написанный другими), доступ к которому можно получить с помощью объявлений, содержащихся в файле, включенном директивой `#include`. *Объявление* — это инструкция программы, указывающая, как можно использовать некоторый фрагмент кода; объявления будут подробно описаны позднее (см., например, раздел 4.5.2).

Ошибки, обнаруженные компилятором, называются *ошибками времени компиляции*; ошибки, обнаруженные компоновщиком, называются *ошибками времени компоновки*, а ошибки, не обнаруженные на этих этапах и проявляющиеся при выполнении программы, называются *ошибками времени выполнения* или *логическими ошибками*. Как правило, ошибки времени компиляции легче понять и исправить, чем ошибки времени компоновки. В свою очередь, ошибки времени компоновки легче обнаружить и исправить, чем ошибки времени выполнения. Ошибки и способы их обработки более детально обсуждаются в главе 5.

## 2.5. Среды программирования

Для программирования необходим язык программирования. Кроме того, для преобразования исходного кода в объектный нужен компилятор, а для редактирования связей нужен редактор связей. Кроме того, для ввода и редактирования исходного текста в компьютере также необходима отдельная программа. Эти инструменты, крайне необходимые для разработки программы, образуют среду разработки программ.

Если вы работаете с командной строкой, как многие профессиональные программисты, то должны самостоятельно решать проблемы, связанные с компилированием и редактированием связей. Если же вы используете IDE (interactive (integrated) development environment — интерактивные (интегрированные) среды разработки), которые также весьма популярны среди профессиональных программистов, то достаточно щелкнуть на соответствующей кнопке. Описание компиляции и редактирования связей приведено в приложении В.

Интегрированные среды разработки включают в себя редактор текстов, позволяющий, например, выделять разными цветами комментарии, ключевые слова и другие части исходного кода программы, а также помогающий отладить, скомпилировать и выполнить программу. *Отладка* — это поиск и исправление ошибок в программе (по ходу изложения мы еще не раз вспомним о ней).

Работая с этой книгой, вы можете использовать любую систему, предоставляющую современную, соответствующую стандарту реализацию C++. Большинство из того, о чем мы говорим, с очень малыми модификациями справедливо для всех реализаций языка C++, и приводимый нами код будет работать везде. В нашей работе мы используем несколько разных реализаций.



### **Задание**

До сих пор мы говорили о программировании, коде и инструментах (например, о компиляторах). Теперь нам необходимо выполнить программу. Это очень важный момент в изложении и в обучении программированию вообще. Именно с этого начинается усвоение практического опыта и овладение хорошим стилем программирования. Упражнения в этой главе предназначены для того, чтобы вы освоились с вашей интегрированной средой программирования. Запустив программу “Hello, World!” на выполнение, вы сделаете первый и главный шаг как программист.

Цель задания — закрепить ваши навыки программирования и помочь вам приобрести опыт работы со средами программирования. Как правило, задание представляет собой последовательность модификаций какой-нибудь простой программы, которая постепенно “вырастает” из совершенно тривиального кода в нечто полезное и реальное. Для выявления вашей инициативы и изобретательности предлагаем набор традиционных упражнений. В противоположность им задания не требуют особой изобретательности. Как правило, для их выполнения важна последовательность пошагового выполнения действий, каждое из которых должно быть простым (и даже тривиальным). Пожалуйста, не умничайте и не пропускайте описанные шаги, поскольку это лишь тормозит работу или сбивает с толку.

Вам может показаться, что вы уже все поняли, прочитав книгу или прослушав лекцию преподавателя, но для выработки навыков необходимы повторение и практика. Этим программирование напоминает спорт, музыку, танцы и любое другое занятие, требующее упорных тренировок и репетиций. Представьте себе музыканта, который репетирует от случая к случаю. Можно себе представить, как он играет. Постоянная практика — а для профессионала это означает непрерывную работу на протяжении всей жизни — это единственный способ развития и поддержания профессиональных навыков.



Итак, никогда не пропускайте заданий, как бы вам этого ни хотелось; они играют важную роль в процессе обучения. Просто начинайте с первого шага и продолжайте, постоянно перепроверяя себя.



Не беспокойтесь, если вы не понимаете все тонкости используемого синтаксиса, и не стесняйтесь просить помощи у преподавателей или друзей. Работайте, выполняйте все задания и большинство упражнений, и со временем все прояснится.

Итак, вот первое задание.

1. Откройте приложение В, и выполните все шаги, необходимые для настройки проекта. Создайте пустой консольный проект на С++ под названием `hello_world`.
2. Введите текст файла `hello_world.cpp` в точности таким, как показано ниже, сохраните его в рабочем каталоге и включите его в проект `hello_world`.

```
#include "std_lib_facilities.h"
int main() // Программы на С++ начинаются с выполнения функции main
{
    cout << "Hello, World!\n"; // Вывод строки "Hello, World!"
    keep_window_open(); // Ожидание ввода символа
    return 0;
}
```

Вызов функции `keep_window_open()` нужен при работе под управлением некоторых версий операционной системы Windows для того, чтобы окно не закрылось прежде, чем вы прочитаете строку вывода. Это особенность вывода системы Windows, а не языка С++. Для того чтобы упростить разработку программ, мы поместили определение функции `keep_window_open()` в файл `std_lib_facilities.h`.

Как найти файл `std_lib_facilities.h`? Если вы учитесь с преподавателем, спросите у него. Если работаете самостоятельно, загрузите его с сайта [www.stroustrup.com/Programming](http://www.stroustrup.com/Programming). А если у вас нет ни преподавателя, ни доступа к вебу? В этом (и только в этом!) случае замените директиву `#include` строками

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;
inline void keep_window_open() { char ch; cin>>ch; }
```

В этих строках непосредственно используется стандартная библиотека. Подождите до главы 5 и еще более подробного изложения в разделе 8.7.

3. Скомпилируйте и выполните программу “Hello, World!”. Вполне вероятно, что у вас это сразу не получится. Очень редко первая попытка использовать новый язык программирования или новую среду

разработки программ завершается успехом. Найдите источник проблем и устраните его! В этот момент целесообразно заручиться поддержкой более опытного специалиста, но при этом вы должны убедиться, что понимаете, что именно он сделал и почему, и сможете повторить эти действия в дальнейшем самостоятельно.

4. Возможно, вы столкнетесь с наличием ошибок в программе и будете должны их исправить. Тогда самое время познакомиться с тем, как ваш компилятор находит ошибки и сообщает о них программисту! Посмотрите, как отреагирует компилятор на шесть ошибок, описанных в разделе 2.3, внося их и пытаясь скомпилировать программу. Придумайте еще как минимум пять других ошибок в вашей программе (например, пропустите вызов функции `keep_window_open()`, наберите ее имя в верхнем регистре или поставьте запятую вместо точки с запятой) и посмотрите, что произойдет при попытке скомпилировать и выполнить эту программу.

### Контрольные вопросы

Основная идея контрольных вопросов — дать вам возможность выяснить, насколько хорошо вы усвоили основные идеи, изложенные в главе. Вы можете найти ответы на эти вопросы в тексте главы; это нормально и вполне естественно, можете перечитать все разделы, и это тоже нормально и естественно. Но если даже после этого вы не можете ответить на контрольные вопросы, то вам следует задуматься о том, насколько правильный способ обучения вы используете? Возможно, вы слишком торопитесь. Может быть, имеет смысл остановиться и попытаться поэкспериментировать с программами? Может быть, вам нужна помощь друга, с которым вы могли бы обсуждать возникающие проблемы?

1. Для чего предназначена программа “Hello, World!”?
2. Назовите четыре части функции.
3. Назовите функцию, которая должна иметься в каждой программе на языке C++.
4. Для чего предназначена строка `return 0` в программе “Hello,World!”?
5. Для чего предназначен компилятор?
6. Для чего предназначена директива `#include`?
7. Что означает суффикс `.h` после имени файла в языке C++?
8. Что делает редактор связей?
9. В чем заключается различие между исходным и объектным файлами?
10. Что такое интегрированная среда разработки и для чего она предназначена?
11. Если в книге вам все понятно, то зачем нужна практическая работа?

Обычно контрольный вопрос имеет ясный ответ, явно сформулированный в главе. Однако иногда мы включаем в этот список вопросы, связанные с информацией, изложенной в других главах и даже в других книгах. Мы считаем это вполне допустимым; для того чтобы научиться писать хорошие программы и думать о последствиях их использования, мало прочитать одну главу или книгу.

## Термины

Приведенные термины входят в основной словарь по программированию и языку C++. Чтобы понимать, что люди говорят о программировании, и озвучивать собственные идеи, следует понимать их смысл.

<code>#include</code>	библиотека	компилятор
<code>//</code>	вывод	объектный код
<code>&lt;&lt;</code>	выполняемый файл	ошибка времени компиляции
<code>C++</code>	заголовок	программа
<code>cout</code>	инструкция	редактор связей
IDE	исходный код	функция
<code>main()</code>	комментарий	

Можете пополнять этот словарь самостоятельно, выполняя приведенное ниже пятое упражнение для каждой прочитанной главы.

## Упражнения

Мы приводим задания отдельно от упражнений; прежде чем приступить к упражнениям, необходимо выполнить все задания. Тем самым вы сэкономите время.

1. Измените программу так, чтобы она выводила две строки:

```
Hello, programming!
Here we go!
```

2. Используя приобретенные знания, напишите программу, содержащую инструкции, с помощью которых компьютер нашел бы кабинет на верхнем этаже, о котором шла речь в разделе 2.1. Можете ли вы указать большее количество шагов, которые подразумевают люди, а компьютер — нет? Добавьте эти команды в ваш список. Это хороший способ научиться думать, как компьютер. Предупреждаем: для большинства людей “иди в указанное место” — вполне понятная команда. Для людей, которые никогда не видели современного строения (например, для перемещенных во времени неандертальцев), этот список может оказаться *очень* длинным. Пожалуйста, не делайте его больше страницы. Для удобства читателей можете изобразить схему строения.

3. Напишите инструкции, как пройти от входной двери вашего дома до двери вашей аудитории (будем считать, что вы студент; если нет, выберите другую цель). Покажите их вашему другу и попросите уточнить их. Для того чтобы не потерять друзей, неплохо бы сначала испытать эти инструкции на себе.
4. Откройте хорошую поваренную книгу и прочитайте рецепт изготовления булочек с черникой (если в вашей стране это блюдо является экзотическим, замените его каким-нибудь более привычным). Обратите внимание на то, что, несмотря на небольшое количество информации и инструкций, большинство людей вполне способны выпекать эти булочки, следуя рецепту. При этом никто не считает этот рецепт сложным и доступным лишь профессиональным поварам или искусным кулинарам. Однако, по мнению автора, лишь некоторые упражнения из нашей книги можно сравнить по сложности с рецептом по выпечке булочек с черникой. Удивительно, как много можно сделать, имея лишь небольшой опыт!
  - ◆ Перепишите эти инструкции так, чтобы каждое отдельное действие было указано в отдельном абзаце и имело номер. Подробно перечислите все ингредиенты и всю кухонную утварь, используемую на каждом шаге. Не пропустите важные детали, например желательную температуру, предварительный нагрев духовки, подготовку теста, время выпекания и средства защиты рук при извлечении булочек из духовки.
  - ◆ Посмотрите на эти инструкции с точки зрения новичка (если вам это сложно, попросите об этом друга, ничего не понимающего в кулинарии). Дополните рецепт информацией, которую автор (разумеется, опытный кулинар) счел очевидной.
  - ◆ Составьте словарь использованных терминов. (Что такое противень? Что такое предварительный разогрев? Что подразумевается под духовкой?)
  - ◆ Теперь приготовьте несколько булочек и насладитесь результатом.
5. Напишите определение каждого из терминов, включенных в раздел “Термины”. Сначала попытайтесь сделать это, не заглядывая в текст главы (что маловероятно), а затем перепроверьте себя, найдя точное определение в тексте. Возможно, вы обнаружите разницу между своим ответом и нашей версией. Можете также воспользоваться каким-нибудь доступным глоссарием, например, размещенным по адресу [www.stroustrup.com/glossary.html](http://www.stroustrup.com/glossary.html). Формулируя свое описание, вы закрепите полученные знания. Если для этого вам пришлось перечитывать главу, то это пойдет вам только на пользу. Можете пересказывать смысл термина своими словами и уточнять его по своему разумению.



Часто для этого полезно использовать примеры, размещенные после основного определения. Целесообразно записывать свои ответы в отдельный файл, постепенно добавляя в него новые термины.

### **Послесловие**



Почему программа “Hello, World!” так важна? Ее цель — ознакомить вас с основными инструментами программирования. Мы стремились использовать для этого максимально простой пример. Так мы разделяем обучение на две части: сначала изучаем основы новых инструментов на примере тривиальных программ, а затем исследуем более сложные программы, уже не обращая внимания на инструменты, с помощью которых они написаны. Одновременное изучение инструментов программирования и языка программирования намного сложнее, чем овладение этими предметами по отдельности. Этот подход, предусматривающий разделение сложной задачи на ряд более простых задач, не ограничивается программированием и компьютерами. Он носит универсальный характер и используется во многих областях, особенно там, где важную роль играют практические навыки.