

Разработка приложений на С++ в среде MATLAB



Подкур М. Л.
Подкур П. Н.
Смоленцев Н. К.

М. Л. Подкур, П. Н. Подкур, Н. К. Смоленцев

Разработка приложений на C++ в среде MATLAB

2-е издание, электронное



Москва, 2023

УДК 519.68

ББК 32.913

П44

Подкур, М. Л.

П44 Разработка приложений на C++ в среде MATLAB / М. Л. Подкур, П. Н. Подкур, Н. К. Смоленцев. — 2-е изд., эл. — 1 файл pdf : 497 с. — Москва : ДМК Пресс, 2023. — Систем. требования: Adobe Reader XI либо Adobe Digital Editions 4.5 ; экран 10". — Текст : электронный.

ISBN 978-5-89818-465-0

Данная книга посвящена изложению вопросов совместного использования богатой библиотеки визуальных компонент Borland C++ Builder и возможностей языка C++ с мощными математическими процедурами MATLAB®. Книга содержит основы программирования в Borland C++ Builder и на MATLAB, описание математических библиотек MATLAB для C/C++ и компиляторов MATLAB. Рассматриваются примеры программ на Borland C++ Builder, которые используют математические библиотеки MATLAB C++. Освоение технологии использования функций математических библиотек MATLAB в Borland C++ Builder позволит создавать полноценные Windows-приложения с развитой графической средой, в которых возможна реализация сложных математических алгоритмов.

Книга предназначена преподавателям и студентам вузов по специальностям, близким к прикладной математике, профессиональным программистам C++, которые сталкиваются с проблемами реализации математических алгоритмов на C++, и MATLAB-программистам, которые хотят использовать гибкость языка C++ и богатую библиотеку визуальных компонент Borland C++ Builder для реализации алгоритмов MATLAB в виде законченных и независимых от MATLAB приложений.

Файлы, содержащие примеры программ вместе с исходными текстами, а также архив с математическими библиотеками C/C++ MATLAB от компании MathWorks, можно найти на сайте издательства.

УДК 519.68

ББК 32.913

Электронное издание на основе печатного издания: Разработка приложений на C++ в среде MATLAB / М. Л. Подкур, П. Н. Подкур, Н. К. Смоленцев. — Москва : ДМК Пресс, 2008. — 496 с. — ISBN 978-5-94074-445-0. — Текст : непосредственный.

MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.

В соответствии со ст. 1299 и 1301 ГК РФ при устранении ограничений, установленных техническими средствами защиты авторских прав, правообладатель вправе требовать от нарушителя возмещения убытков или выплаты компенсации.

ISBN 978-5-89818-465-0

© Подкур М. Л., Подкур П. Н., Смоленцев Н. К.
© Оформление, издание. ДМК Пресс, 2008



Содержание

Предисловие	10
-------------------	----

Глава 1. Основы программирования на Borland C++ Builder

1.1. Среда разработки	13
1.1.1. Панель инструментов	14
1.1.2. Палитра компонентов	14
1.1.3. Редактор форм	14
1.1.4. Редактор кода	14
1.1.5. Инспектор объектов	15
1.1.6. Менеджер проекта	16
1.1.7. Контекстное меню	16
1.1.8. Контекстная помощь	16
1.2. Компоненты C++ Builder	17
1.2.1. Свойства компонентов	18
1.2.2. События компонентов	18
1.2.3. Методы компонентов	19
1.3. Основы C++ как языка создания программ в C++ Builder	19
1.3.1. Комментарии	19
1.3.2. Типы и описания	20
1.3.3. Основные типы	21
1.3.4. Константы	22
1.3.5. Производные типы	23
1.3.6. Базовые операторы	27
1.3.7. Выражения и операторы	28
Описания	29
1.3.8. Функции	32
1.3.9. Классы и объектно-ориентированное программирование	33
1.3.10. Структура программы на C++	51
1.3.11. Событийное программирование	54
1.4. Использование основных классов библиотеки VCL	58

1.4.1. Классы описания данных	58
1.4.2. Компоненты интерфейса	69
1.4.3. Системные компоненты	83
TForm	86
1.4.4. Обработка исключений	98
1.5. Пример построения графиков функций в C++ Builder	99

Глава 2. Основы работы в системе MATLAB®

2.1. Система компьютерной математики MATLAB	109
2.1.1 Основные компоненты системы MATLAB	110
2.1.2. Инструментальные средства рабочего стола MATLAB ...	111
2.1.3. Константы и системные переменные MATLAB	117
2.1.4. Типы данных MATLAB	117
2.2. Основы работы с MATLAB	120
2.2.1. Запуск MATLAB и начало работы	120
2.2.2. Задание массивов	122
2.2.3. Операции над массивами	125
2.2.4. Решение систем линейных уравнений	128
2.2.5. М-файлы	131
2.2.6. Чтение и запись текстовых файлов	133
2.2.7. Операции с рабочей областью и текстом сессии	136
2.3. Массивы символов	137
2.3.1. Задание массива символов	137
2.3.2. Общие функции	138
2.3.3. Проверка строк	139
2.3.4. Операции над строками	139
2.3.5. Преобразование чисел в символы и обратно	141
2.3.6. Функции преобразования систем счисления	142
2.3.7. Вычисление строковых выражений	143
2.4. Массивы ячеек	143
2.4.1. Создание массивов ячеек	144
2.4.2. Доступ к данным в ячейках	146
2.4.3. Вложенные массивы ячеек	147
2.4.4. Массивы ячеек, содержащих структуры	149
2.4.5. Многомерные массивы ячеек	149
2.5. Массивы структур	150
2.5.1. Построение структур	150
2.5.2. Доступ к полям и данным структуры	151
2.4.3. Многомерные массивы структур	154
2.6. Программирование в среде MATLAB	154

2.6.1. М-функции	155
2.6.2. Операторы системы MATLAB	160
2.6.3. Управление последовательностью исполнения операторов	163
2.6.4. Вычисление символьных выражений	167
2.6.5. Ошибки и предупреждения	168
2.6.6. Повышение эффективности обработки М-файлов	169

Глава 3. Математическая библиотека MATLAB® для языка C

3.1. Введение	171
3.1.1. Установка Математической библиотеки MATLAB	172
3.1.2. Каталоги Математической библиотеки	174
3.1.3. Документация Математической библиотеки MATLAB C	174
3.1.4. Сходство и различие MATLAB и C	175
3.1.5. Пример написания простой программы	176
3.1.6. Компиляция и компоновка приложения	178
3.2. Работа с массивами mxArray	182
3.2.1. Поддерживаемые типы массивов MATLAB	182
3.2.2. Числовые массивы	183
3.2.3. Разреженные матрицы	188
3.2.4. Массивы символов	191
3.2.5. Массивы ячеек	193
3.2.6. Структуры MATLAB	196
3.2.7. Выполнение обычных задач программирования с массивами	198
3.3. Управление памятью массива	202
3.3.1. Автоматизированное управление памятью против явного	202
3.3.2. Использование автоматизированного управления памятью	205
3.4. Индексация в массивах	210
3.4.1. Введение	211
3.4.2. Функции индексации	212
3.4.3. Индексация числовых массивов	214
3.4.4. Индексация массива ячеек	220
3.4.5. Индексация массива структур	223
3.5. Вызов процедур библиотеки	225
3.5.1. Как вызывать функции MATLAB	226

3.5.2. Передача функций в качестве аргументов	231
3.5.3. Замена списков параметров массивом ячеек	238
3.6. Импорт и экспорт данных	239
3.7. Процедуры Математической библиотеки С	241

Глава 4. Математическая библиотека MATLAB® для языка C++

4.1. Введение	244
4.1.1. Установка Математической библиотеки	245
4.1.2. Каталоги Математической библиотеки	246
4.1.3. Документация Математической библиотеки MATLAB C++	248
4.1.4. Сходство и различие MATLAB и C++	249
4.1.5. Знакомство с Математической библиотекой MATLAB C++	250
4.1.6. Создание автономных C++-приложений	252
4.1.7. Компиляция и компоновка приложения	255
4.2. Работа с массивами mxArray	257
4.2.1. Числовые массивы	258
4.2.2. Разреженные матрицы	263
4.2.3. Массивы символов	265
4.2.4. Массивы ячеек	267
4.2.5. Структуры	270
4.2.6. Выполнение обычных задач программирования с массивами	271
4.3. Индексация в массивах	278
4.3.1. Индексация числовых массивов	279
4.3.2. Индексация в массивах ячеек	283
4.3.3. Индексация в массивах структур MATLAB	286
4.3.4. Некоторые методы индексации	288
4.4. Вызов функций библиотеки	289
4.4.1. Как вызывать функции библиотеки C++	289
4.4.2. Представление входных аргументов как массива ячеек	293
4.4.3. Передача функций в качестве аргументов	294
4.4.4. Использование математических операторов	296
4.4.5. Конфликты имен с функциями стандартной библиотеки С	297
4.5. Ввод и вывод массивов	299
4.5.1. Использование потока ввода-вывода массива	299

4.5.2. Использование функций ввода/вывода файлов	301
4.5.3. Импорт и экспорт данных MAT-файла	305
4.6. Интерфейс класса mxArray	307
4.6.1. Конструкторы	308
4.6.2. Индексация	309
4.6.3. Операторы	311
4.6.4. Размер массива	312
4.6.5. Извлечение данных из mxArray	312

Глава 5. Компилятор MATLAB® версии 3.0

5.1. Введение в Компилятор MATLAB	315
5.1.1. Инсталляция и конфигурация	316
5.1.2. Использование Компилятора msc	318
5.1.3. Конечные продукты Компилятора MATLAB	320
5.1.4. Ограничения компиляторов	323
5.1.5. Пример создания MEX-файла	324
5.2. Опции mbuild	325
5.3. Опции Компилятора msc	328
5.3.1. Обычное использование Компилятора	328
5.3.2. Опции msc	329
5.3.3. Использование опций и макроопций msc	332
5.3.4. Опции создания кода	335
5.3.5. Опции настройки Компилятора и среды разработки	337
5.3.6. Опции mbuild/mex	339
5.4. Создание автономных приложений и библиотек	340
5.4.1. Создание автономных приложений	341
5.4.2. Создание общедоступных библиотек	346
5.4.3. Распространение автономных приложений	352
5.5. Создаваемые Компилятором коды	353
5.5.1. Типы создаваемых файлов	353
5.5.2. Внутренние функции интерфейса	356
5.5.3. Поддерживаемые выполнимые типы	360
5.5.4. Управление видом кода программы	362
5.5.5. Использование псевдокомментариев	364
5.6. Выполнение оптимизации	367

Глава 6. Компилятор MATLAB® версии 4

6.1. Введение	372
6.1.1. Системные требования и ограничения	372
6.1.2. Различия между MATLAB Компилятором 4 и предыдущими версиями	375

6.1.3. Использование Компилятора MATLAB	378
6.2. Процесс трансляции	379
6.2.1. Обзор технологии Компилятора MATLAB	379
6.2.2. Файлы, создаваемые mсс	381
6.2.3. Функция mbuild	382
6.3. Работа с программой mсс	384
6.3.1. Опции mсс	385
6.3.2. Использование файлов групп	392
6.3.3. Использование файлов обертки	394
6.3.4. Использование псевдокомментариев	394
6.3.5. Скрипт-файлы	395
6.4. Автономные приложения	396
6.4.1. Создание кода только из m-файлов	396
6.4.2. Объединение M-файлов и кода C или C++	398
6.4.3. Управление путями при компиляции	400
6.4.4. Пропущенные функции	402
6.4.5. Автономное C-приложение для пользователя	402
6.5. Библиотеки	404
6.5.1. Библиотека совместного использования C	405
6.5.2. C++-библиотека совместного использования	409
6.5.3. Функции, созданные из m-файлов	412
6.5.4. Использование varargin и varargout в интерфейсе m-функции	413

Глава 7. Создание приложений на Borland C++ Builder с использованием Математической библиотеки C++ MATLAB®

7.1. Чтение, обработка и запись данных	415
7.2. Построение графиков данных mxArray	421
7.3. Пакетное вейвлет-разложение сигнала	425
7.3.1. Описание пакетного вейвлет-разложения	425
7.3.2. Описание функций программы	426
7.3.3. Описание работы программы	428

Приложение 1. Функции Математической библиотеки C++ MATLAB®

П.1.1. Операторы	437
П.1.2. Функции Математической библиотеки MATLAB C++	439
1.2.1. Функции ввода/вывода файлов	439
1.2.2. Операторы и специальные функции	440

1.2.3. Элементарные матрицы и управление матрицами	442
1.2.4. Элементарные математические функции	443
1.2.5. Специализированные математические функции	444
1.2.6. Численная линейная алгебра	445
1.2.7. Анализ данных и преобразования Фурье	446
1.2.8. Полиномы и интерполяционные функции	447
1.2.9. Функции от функций и решение обыкновенных дифференциальных уравнений	448
1.2.10. Типы данных	449
1.2.11. Функции многомерных массивов	449
1.2.12. Функции строк символов	449
1.2.13. Функции массива ячеек	450
1.2.14. Функции структур	451
1.2.15. Функции разреженных матриц	451
1.2.16. Время и даты	452
П.1.3. Сервисные функции	452
П.1.4. Функции доступа к массивам	455

Приложение 2. Библиотека классов C++ Компилятора 4 MATLAB®

П.2.1. Основные типы данных	458
П.2.2. Класс mwString	458
2.2.1. Конструкторы	459
2.2.2. Методы	460
2.2.3. Операторы	460
П.2.3. Класс mwException	462
2.3.1. Конструкторы	462
2.3.2. Методы	464
2.3.3. Операторы	464
П.2.4. Класс mxArray	465
2.4.1. Конструкторы	466
2.4.2. Методы	470
2.4.3. Операторы	481
2.4.4. Статические методы	482

Содержание компакт-диска	485
---------------------------------------	------------

Литература	489
-------------------------	------------

Предметный указатель	490
-----------------------------------	------------



Предисловие

Данная книга посвящена изложению вопросов совместного использования богатой библиотеки визуальных компонент Borland C++ Builder и возможностей языка C++ с мощными математическими процедурами MATLAB®. Книга содержит основы программирования в Borland C++ Builder и на MATLAB, описание математических библиотек MATLAB для C/C++ и компиляторов MATLAB (включая последнюю версию). Рассматриваются примеры программ на Borland C++ Builder, которые используют математические библиотеки MATLAB. Хотя книг по основам системы MATLAB и по программированию на C/C++ и на Borland C++ Builder достаточно много, в настоящее время нет ни одной книги, в которой были бы изложены вопросы совместного использования MATLAB и C/C++ и Borland C++ Builder.

Книга предназначена преподавателям и студентам вузов по специальностям, близким к прикладной математике (математическая подготовка читателя предполагается в пределах технического вуза), профессиональным программистам C++, которые сталкиваются с проблемами реализации математических алгоритмов на C++, и MATLAB-программистам, которые хотят использовать гибкость языка C++ и богатую библиотеку визуальных компонент Borland C++ Builder для реализации алгоритмов MATLAB в виде законченных и независимых от MATLAB приложений.

Углубленное программирование на MATLAB предполагает использование языка C для создания MEX-файлов, создание автономных приложений на MATLAB, создание библиотек (dll) функций. Это направление в MATLAB представлено тремя пакетами расширений: MATLAB C Math Library, MATLAB C++ Math Library и MATLAB Compiler – математические библиотеки C/C++ MATLAB и Компилятор MATLAB. На сегодняшний день эти пакеты не отражены ни в одной книге по MATLAB.

Стандартная математическая библиотека C++ содержит около 20 элементарных функций. Поэтому создание программы на C++, где даже немного используется математика, возможно, но требует серьезной работы по программированию математических функций и процедур. Даже такая простая задача, как вычисление ранга матрицы, требует от программиста очень больших усилий. Система MATLAB в своих пакетах MATLAB C/C++ Math Library и MATLAB Compiler предлагает богатую коллекцию математических функций (более 400), которые могут быть эффективно использованы программистами C/C++. При программировании на C++ имеются определенные трудности использования математических библиотек MATLAB. Предлагаемая книга рассказывает, как решить эти проблемы. Освоение технологии использования математических библиотек

MATLAB в Borland C++ Builder позволит создавать полноценные Windows-приложения с развитой графической средой, в которых возможна реализация сложных математических алгоритмов.

Рассмотрим кратко содержание данной книги по главам.

Первая глава представляет краткое введение в программирование на Borland C++ Builder 6. Она предназначена для читателей, которые немного владеют программированием, например на MATLAB, но не на C/C++. Дается общая характеристика системы, ее возможности, инструментальные средства Borland C++ Builder. Рассматриваются файлы, создаваемые Borland C++ Builder при построении проекта, их характеристики и взаимосвязи. Дается краткий обзор программирования на C++.

Вторая глава содержит первоначальные сведения о системе MATLAB. Она предназначена для читателей, которые немного владеют программированием, но не на MATLAB. Глава содержит описание работы с числовыми массивами, массивами символов, массивами ячеек и структур и основы программирования в среде MATLAB.

Глава 3 является введением в Математическую библиотеку C. Данная библиотека представляет собой набор более чем 400 математических подпрограмм, написанных на языке C. Библиотека предлагает определенные типы массивов, аналогичных массивам MATLAB, и свой язык для использования математических функций и массивов в программах на C. Глава содержит детальный анализ ряда тестовых примеров C-программ, в которых используются функции Математической библиотеки C (все эти примеры входят в инсталляционный пакет Математической библиотеки C).

Глава 4 посвящена описанию Математической библиотеки MATLAB C++. Данная библиотека основана на Математической библиотеке C, главное, что изменилось, – это синтаксис вызова математических процедур библиотеки. Язык Математической библиотеки C++ стал почти таким же интуитивно понятным и кратким, как в MATLAB. Математическая библиотека C++ использует класс `mwArray` для представления всех типов массивов MATLAB.

Пятая глава посвящена описанию пакета расширения MATLAB Compiler версии 3 (для MATLAB 6.x). Компилятор MATLAB из m-файлов MATLAB может создать MEX-файлы, C или C++ исходные коды, автономные приложения, библиотеки общего доступа, C-коды S-функций для использования с Simulink, дополнения Excel и COM-объекты. Изложение материала сопровождается обсуждением многочисленных тестовых примеров MATLAB 6 (все эти примеры входят в инсталляционный пакет MATLAB Compiler).

Глава 6 является введением в Компилятор версии 4 (для MATLAB R14) и его использование. Данная версия Компилятора существенно отличается от предыдущей. В частности, Компилятор 4 включает все математические библиотеки MATLAB (они в MATLAB R14 даже не выделены в отдельный пакет расширения). Возможности Компилятора огромны. Компилятор MATLAB поддерживает почти все функциональные возможности MATLAB.

В последней главе 7 на примерах рассматривается создание Windows-приложений на Borland C++ Builder с использованием Математической библиотеки MATLAB C++. Полные тексты этих примеров записаны на компакт-диске, прилагаемом к данной книге.

Книга имеет три приложения. Первое приложение содержит полный перечень функций и процедур Математической библиотеки MATLAB C++. Приложение 2 содержит описание библиотеки классов C++ для Компилятора MATLAB последней версии 4. Третье приложение есть компакт-диск с примерами программ, рассматриваемых в данной книге.

Книга написана при содействии корпорации MathWorks в соответствии с программой MathWorks поддержки книг, посвященных MATLAB®. Авторы благодарят компанию MathWorks за предоставленную возможность использования документации и лицензионного программного обеспечения MATLAB® для написания этой книги.

Глава 1. Основы программирования на Borland C++ Builder

Borland C++ Builder является средством быстрой разработки приложений (Rapid Application Development Environment), позволяющим создавать приложения на языке C++ и использующим в качестве основы библиотеку визуальных компонентов (Visual Component Library), позаимствованную у Borland Delphi. Сведения, приведенные в данной главе, ориентированы на читателя, уже имеющего навыки программирования. В главе будут рассмотрены: основы языка C++, среда разработки C++ Builder, основные приемы, применяемые при проектировании пользовательского интерфейса, а также простые примеры построения математических графиков.

1.1. Среда разработки

C++ Builder представляет собой однооконное приложение (SDI), главное окно которого (верхняя часть) содержит панель меню программы, настраиваемую панель инструментов (левая часть главного окна) и палитру компонентов (правая часть главного окна). Помимо этого, по умолчанию при запуске C++

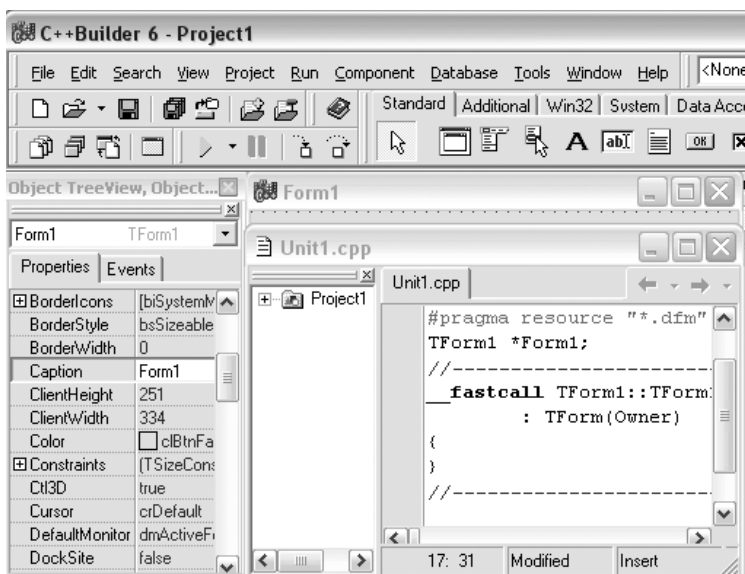


Рис. 1.1.1. Среда разработки C++ Builder

Builder отображаются окно дерева объектов (Object Tree View), окно инспектора объектов (Object Inspector), форма нового приложения (например, Form1) и окно редактора кода (например, Unit1.cpp).

1.1.1. Панель инструментов

Рассмотрим панель инструментов, на которой стандартно размещены 16 кнопок быстрого доступа (все эти кнопки соответствуют определенным пунктам меню). Этими кнопками являются: «новый файл», «открыть проект», «сохранить», «сохранить все», «добавить к проекту», «выбрать модуль», «выбрать форму», «запустить», «пауза» и т. д. Панель инструментов можно полностью перестроить под собственные предпочтения, внося в нее требуемые команды меню или удалив ненужные.

1.1.2. Палитра компонентов

Палитра компонентов предназначена для выбора компонентов или элементов управления (таких как ActiveX), размещаемых на форме разрабатываемого программного приложения. Все компоненты C++ Builder располагаются в *палитре компонентов*, выполненной в виде многостраничного блокнота, каждая закладка которого объединяет компоненты в смысловую группу. Особенностью среды разработки C++ Builder является то, что она позволяет создавать собственные компоненты и добавлять их в палитру компонентов, а также настраивать палитры компонентов для разных проектов.

1.1.3. Редактор форм

Формы являются основой для разработки интерфейса приложений C++ Builder. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы компонентов (объектов) C++ Builder и задания некоторой реакции компонента на события. Для помещения компонента в форму необходимо щелкнуть на его пиктограмме в палитре компонентов, а затем щелкнуть на нужном месте в вашей форме, при этом новому компоненту (экземпляру класса) будет автоматически присвоено имя и заполнены некоторые свойства, например отвечающие за местоположение компонента в форме. При этом в заголовочном файле формы появится строка типа `TLabel *Label1` (в `public` разделе объявлений). Если вы хотите разместить несколько копий компонентов одного типа, то, выбрав в палитре необходимый компонент и удерживая клавишу Shift, кликая мышкой, помещайте в форму сколько угодно компонентов. При завершении процесса тиражирования щелкните на быструю кнопку «стрелка» в палитре компонентов.

1.1.4. Редактор кода

Окно редактора кода разделено на расположенный слева проводник классов (ClassExplorer) и непосредственно сам редактор кода справа. Проводник классов позволяет оперативно следить за развитием собственной программы, а также ви-

деть поля и методы используемых программой классов. При необходимости проводник классов может быть закрыт. Все открытые в среде исходные файлы отображаются непосредственно в редакторе кода, который представляет собой окно редактирования текста с закладками, соответствующими именам открытых файлов. В левой части редактора кода расположено поле для установки *точек прерывания* выполнения программы. Они отображаются жирными красными точками. Устанавливать их можно, просто щелкая мышью по полю слева от текста, причем делать это можно и перед запуском, и во время выполнения программы. Недействующие точки останова помечаются крестиком. Также стоит отметить то, что редактор кода C++ Builder позволяет выделять цветом ключевые слова и синтаксические конструкции. При удачном задании цветов (по умолчанию оно таковым не является) чтение кода становится намного легче.

1.1.5. Инспектор объектов

Инспектор объектов работает совместно с редактором форм. Инспектор объектов предназначен для установки значений свойств, доступных на этапе разработки. Окно инспектора объектов состоит из: Селектора компонентов, Закладки Properties (свойства) и Закладки Events (события). О свойствах и событиях мы подробнее поговорим при изучении компонентов.

Селектор представляет собой выпадающий комбинированный список, в котором отображены все компоненты формы. Выбранный в списке компонент будет выделен в форме.

Закладка *Properties* отображает все свойства текущего компонента, доступные на этапе разработки. Если вы выбрали в форме несколько компонентов, то инспектор объектов отобразит их общие *свойства*. В левом столбце вы видите имя свойства, а в правом – установленное значение. Практически все значения можно поменять прямым вводом значения в правом столбце, но во многих случаях редактор свойств содержит список возможных значений. У некоторых свойств (например, Font) стоит знак плюс. Это означает, что данное свойство представляет собой другой объект, также содержащий набор свойств. Щелкнув на плюс, вы раскроете весь набор. Для свойств, представляющих собой объекты, есть второй способ редактирования свойств: нужно щелкнуть в столбце значений на многоточие, при этом раскроется соответствующее окно редактора свойств. Для редактирования некоторых свойств можно пользоваться только многоточием.

Закладка *Events* содержит список событий, которые может обрабатывать данный компонент. Использование этой закладки заключается в том, чтобы соответствующим событиям назначить имена функций, которые будут выполняться при возникновении этого события. Это позволяет нескольким компонентам, в случае указания одной и той же функции, одинаково реагировать на одно и то же *событие*. Если вы хотите написать отдельный код для этого события, то самый простой вариант – это просто дважды щелкнуть в столбце значений соответствующего события. При этом автоматически будет создана функция для этого события.

1.1.6. Менеджер проекта

Файлы, образующие программное приложение (формы и модули), собраны в проект. Менеджер проекта показывает список файлов и модулей приложения, позволяя осуществлять навигацию между ними. Менеджер проектов можно вызвать, выбрав пункт меню View/Project Manager. По умолчанию вновь созданный проект получает имя Project1.cpp.

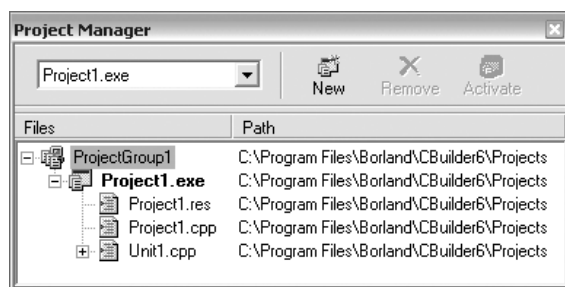


Рис. 1.1.2. Менеджер проекта

Первоначально проект содержит файлы для одной формы и исходного кода одного модуля. Однако большинство проектов содержат несколько форм и модулей. Можно добавлять существующие формы и модули к проекту: для этого нужно щелкнуть правой кнопкой мыши по проектному файлу (в окне менеджера проекта отображается жирным шрифтом) и выбрать пункт Add контекстного меню менеджера проектов, затем выбирать модуль или форму, которую нужно добавить. Аналогичным образом формы и модули можно удалять в любой момент из разработки проекта. Однако, из-за того что форма с модулем всегда связаны, нельзя удалять одно без удаления другого, за исключением случая, когда модуль изначально не имеет связи с формой. Удалить модуль из проекта можно, используя кнопку Remove менеджера проектов или через контекстное меню.

Если выбрать пункт меню **Options** в менеджере проектов, откроется диалоговая *панель опций* проекта (рис. 1.1.3), в которой можно выбрать главную форму приложения, определить, какие формы будут создаваться динамически, каковы будут параметры компиляции и компоновки модулей.

1.1.7. Контекстное меню

Важным элементом среды разработки C++ Builder является контекстное меню, появляющееся при нажатии на правую клавишу мыши и предлагающее быстрый доступ к наиболее часто используемым командам.

1.1.8. Контекстная помощь

Разумеется, C++ Builder обладает встроенной системой контекстно-зависимой помощи, доступной для любого элемента интерфейса и являющейся обшир-

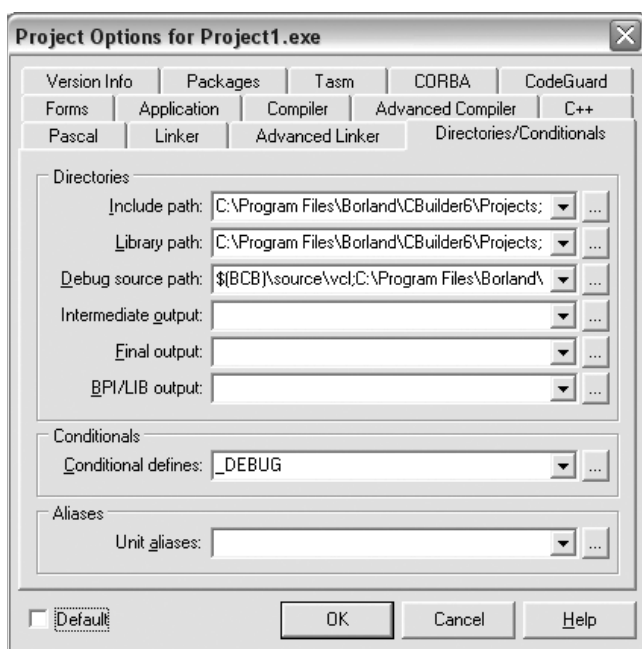


Рис. 1.1.3. Установка опций проекта

ным источником справочной информации о C++ Builder. Достаточно установить курсор на неизвестный элемент или часть кода и нажать клавишу <F1>. Особенно полезна помощь во время написания программ, так как в ней описаны все объекты библиотеки визуальных компонентов (VCL). Практически все описанные свойства и методы классов сопровождаются примерами использования.

1.2. Компоненты C++ Builder

Под компонентами C++ Builder понимаются классы C++, которые размещены в палитре компонентов и могут быть перенесены в разрабатываемую форму. Компоненты C++ Builder разделяются на визуальные и невидимые. *Визуальные компоненты* во время выполнения разработанного приложения отображаются на форме точно так же, как и во время проектирования. Этот класс компонент, исходя из названия, необходим для представления данных на экране компьютера. Примерами таких компонент являются поля редактирования, статический текст, кнопки, списки и т. д. Следует также отметить, что во время выполнения программы, при необходимости, они могут быть видимыми и невидимыми. *Невизуальные компоненты* во время исполнения программы не отображаются, а в процессе проектирования отображаются на форме в виде соответствующих иконок. Такие компоненты обладают определенной функциональностью (например, вызывают стандартные диалоги открытия или закрытия файлов, обрабатывают системные события, обеспечивают доступ к базам данных и т. д.)



Рис. 1.1.4. Пример отображения визуальных и невидимых компонентов

1.2.1. Свойства компонентов

Свойства являются атрибутами компонента, определяющими его внешний вид и поведение во время работы программы. Многие свойства компонента, отображаемые в колонке свойств инспектора объектов, имеют значения, устанавливаемые по умолчанию (например, высота кнопок). Инспектор объектов отображает только опубликованные (*published*) свойства компонентов. Помимо *published*-свойств, компоненты могут и чаще всего имеют общие (*public*) свойства, которые доступны только во время выполнения приложения. Можно определять свойства во время проектирования или написать код для видоизменения свойств компонента во время выполнения приложения.

1.2.2. События компонентов

Закладка событий инспектора объектов (*Events*) показывает список всех возможных событий, распознаваемых компонентом. Программирование в операционных системах с графическим пользовательским интерфейсом (в частности, *Microsoft Windows*) предполагает описание реакций создаваемого приложения на те или иные события, в то время как сама операционная система занимается постоянным опросом компьютера с целью выявления наступления какого-либо события. Каждый компонент имеет свой собственный набор событий. Создание программ в *C++ Builder* в основном и заключается в написании функций (называемых обработчиками событий), связываемых с событиями компонентов, помещенных в форму компонентов. Создавая обработчик того или иного события, вы поручаете программе выполнить написанную функцию, если это событие произойдет.

Для того чтобы добавить *обработчик событий*, необходимо выбрать компонент на форме с помощью мыши (или Селектор компонентов Инспектора объектов), которому необходим обработчик событий, затем открыть страницу событий инспектора объектов и дважды щелкнуть левой клавишей мыши на колонке значений рядом с событием, чтобы заставить *C++ Builder* сгенерировать прототип обработчика событий и показать его в редакторе кода. При этом автоматически

генерируется текст пустой функции, и редактор открывается в том месте, где следует вводить код. Курсор позиционируется внутри операторных скобок { и }. Далее нужно ввести код, который должен выполняться при наступлении события. Обработчик событий может иметь параметры, которые указываются после имени функции в круглых скобках.

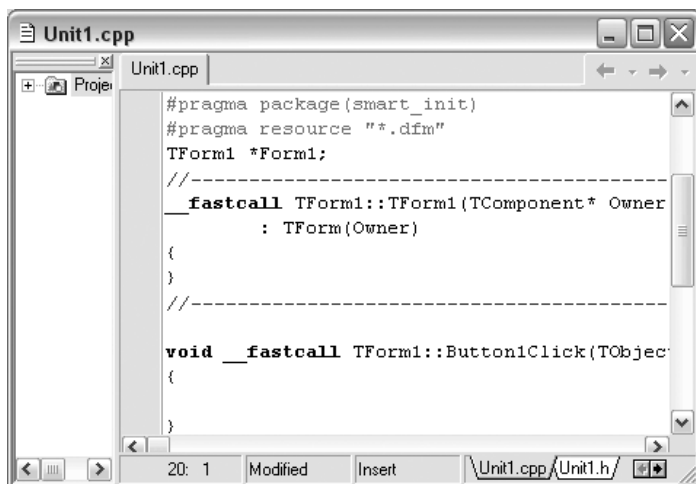


Рис. 1.1.5. Пример пустой функции обработчика событий

1.2.3. Методы компонентов

Методами компонента являются все функции, которые связаны с компонентом и которые объявляются в классе C++, описывающем компонент. Создавая обработчик события, можно вызывать методы компонентов, описанных в модуле формы, используя нотацию «->», например:

```
Edit1->Show();
```

1.3. Основы C++ как языка создания программ в C++ Builder

Этот раздел представляет собой краткий обзор языка программирования C++, являющегося основным языком создания программ в C++ Builder. Эта глава всего лишь является справочным пособием для людей, обладающих знанием других процедурных языков программирования, а приведенные в примерах конструкции языка программирования C++ демонстрируют основные особенности языка.

1.3.1. Комментарии

Часто бывает полезно вставлять в программу игнорируемый компилятором текст, который предназначен в качестве комментария для человека, стараю-

щегося впоследствии разобраться с тем, что же ваша программа все-таки делает. В C++ это можно сделать одним из двух способов:

Символы `/*` начинают комментарий, заканчивающийся символами `*/`. Вся эта последовательность символов эквивалентна символу пропуска (например, символу пробела). Чаще всего они используются для многострочных комментариев, но иногда и для изъятия частей программы (отключения части функциональности кода) при отладке. Следует помнить, что комментарии `/* */` не могут быть вложенными.

Символы `//` указывают на наличие однострочного комментария, который заканчивается в конце той же самой строки, на которой появились эти символы. Используется он для коротких комментариев. Его также можно использовать для отключения работы единичной строки кода программы.

Символы `//` можно использовать для того, чтобы закомментировать символы `/*` или `*/`, а символами `/*` можно закомментировать последовательность `//`.

Если вы используете комментарии при отладке кода, не забывайте после доведения программы до рабочего состояния прибирать «строительный мусор», удаляя закомментированные куски кода. По личному опыту не раз убедился, что они вносят существенное замешательство при разборе чужих, да и собственных программ, особенно если это делается спустя годы и в редакторах без подсветки синтаксических конструкций языка.

1.3.2. Типы и описания

Каждая поименованная переменная в программе и каждое выражение имеют тип, определяющий операции, которые могут над ними производиться. Например, описание

```
int MaxSize;
```

определяет, что `MaxSize` имеет тип `int`, то есть `MaxSize` является целочисленной переменной. *Описание переменной* – это оператор, который сообщает программе имя переменной и задает ее тип. Тип определяет правила использования имени или выражения. Для целых определены такие *операции*, как `+`, `-`, `*` и `/`. Например, после подключения файла `stream.h`, переменная или выражение типа `int` может также быть вторым операндом `<<`, при выводе в некоторый поток `ostream`.

Для переменных инициализация необязательна, но настоятельно рекомендуется. Тип объекта определяет не только то, какие операции могут к нему применяться, но и смысл этих операций. В C++ есть несколько основных типов и несколько способов создавать собственные.

1.3.3. Основные типы

Основные типы данных строятся из следующих ключевых слов:

<code>char</code>	<code>__int8</code>	<code>long</code>
<code>double</code>	<code>__int16</code>	<code>signed</code>
<code>float</code>	<code>__int32</code>	<code>short</code>
<code>int</code>	<code>__int64</code>	<code>unsigned</code>

Из этих ключевых слов вы можете создать любые целочисленные типы данных и типы с плавающей точкой, которые больше известны как арифметические типы. К целочисленным типам могут быть применены модификаторы `long`, `short`, `signed` и `unsigned`. Подключаемый файл `limits.h` содержит определения диапазонов значений для всех базовых типов данных.

Целочисленные типы

`char`, `short`, `int` и `long` вместе со своими беззнаковыми вариантами описывают целочисленные типы данных. Целочисленные типы указаны ниже вместе с условиями образования синонимов:

- `char`, `signed char`, будут являться синонимами, если по умолчанию `char` является знаковым;
- `unsigned char`;
- `char`, `unsigned char`, будут являться синонимами, если по умолчанию `char` является беззнаковым;
- `signed char`;
- `int`, `signed int`;
- `unsigned`, `unsigned int`;
- `short`, `short int`, `signed short int`;
- `unsigned short`, `unsigned short int`;
- `long`, `long int`, `signed long int`;
- `unsigned long`, `unsigned long int`.

Модификаторы `signed` и `unsigned` могут быть использованы с `char`, `short`, `int` и `long`. Если при описании переменных используются только ключевые слова `signed` и `unsigned`, то это соответственно означает `signed int` и `unsigned int`.

Ключевые слова `long` и `short` могут использоваться только с `int`, означая соответственно длинное и короткое целое значение.

Так как ANSI C не диктует размера или внутреннего представления таких типов, указывая только формы (`short`, `int` и `long`), то не меняет значимости «`sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)`». И поэтому вполне законно, что все целочисленные типы могут быть одним и тем же. В 32-битных программах на C++ Builder типы `int` и `long` идентичны и занимают 32 бита. Для типа `char`, по определению, имеет размер единица. Соотношение между типами с плавающей точкой можно записать «`sizeof(float) <= sizeof(double)`». Предполагать что-либо еще насчет основных типов неразумно. В частности, то, что целое достаточно для хранения указателя, верно не для всех платформ.

Типы с плавающей точкой

Представления и наборы значений для типов с плавающей точкой являются зависящими от реализации, то есть каждая реализация C может определить их по-своему. Авторы C++ Builder использовали форматы с плавающей точкой

стандарта IEEE. float и double являются соответственно 32- и 64-битными типами данных с плавающей точкой. long может использоваться с double, чтобы объявить идентификатор с плавающей точкой 80-битной точности, например long double test_case.

Таблица 1.3.1. Типы данных и их применение

Тип	Размер (бит)	Границы	Примеры применения
unsigned char СИМВОЛОВ	8	0 <= X <= 255	Маленькие числа и таблица
char	8	-128 <= X <= 127	Очень маленькие числа и таблица ASCII символов
short int	16	-32'768 <= X <= 32'767	Счетчики циклов и маленькие числа
unsigned int	32	0 <= X <= 4'294'967'295	Большие числа и счетчики циклов
int	32	-2'147'483'648 <= X <= 2'147'483'647	Счетчики циклов и маленькие числа
int	32	-2'147'483'648 <= X <= 2'147'483'647	Счетчики циклов и маленькие числа
unsigned long	32	0 <= X <= 4'294'967'295	Астрономические дистанции
enum	32	-2'147'483'648 <= X <= 2'147'483'647	Упорядоченные наборы значений
long	32	-2'147'483'648 <= X <= 2'147'483'647	Большие числа, население
float	32	1.18 10^-38 < X < 3.40 10^38	Научные расчеты с 7 знаками точности
double	64	2.23 10^-308 < X < 1.79 10^308	Научные расчеты с 15 знаками точности
long double	80	3.37 10^-4932 < X < 1.18 10^4932	Финансовые расчеты с 18 знаками точности

1.3.4. Константы

К основным типам можно применять декларатор const. Это даст тип, имеющий те же свойства, что и исходный, за исключением того, что значение переменных типа const не может изменяться после инициализации.

```
const float pi = 3.14;
const char plus = '+';
```

Символ, заключенный в одинарные кавычки, является *символьной константой*. Заметьте, что константа, определенная таким образом, не занимает память, а просто там, где требуется ее значение, компилятор подставит непосредственно само значение, а не оператор чтения данных, как это происходит с переменными. Константа должна всегда инициализироваться при описании.

Структуры

Если массивы – это совокупность элементов одного типа, то `struct` является совокупностью элементов (практически) произвольных типов. Например:

```
struct    address {    // почтовый адрес
char* name;           // имя
long  number;         // номер дома
    char* street;     // улица
    char* town;       // город
    char* country[2]; // страна
    int  zip;         // индекс
};
```

определяет новый тип, названный `address` (почтовый адрес), состоящий из элементов, требующихся для того, чтобы послать кому-нибудь корреспонденцию.

Обратите внимание на точку с запятой в конце. Это одно из очень немногих мест в C++, где необходимо ставить точку с запятой после фигурной скобки, поэтому люди склонны забывать об этом.

Переменные типа `address` могут быть описаны точно так же, как другие переменные, а доступ к отдельным членам получается с помощью операции «.» (точка). Например:

```
address jd;
jd.name = "Иван Петров";
jd.number = 12;
```

Для инициализации структуры можно использовать запись вида:

```
address jd = {"Петр Иванов", 61, "пр. Ленина", "Москва",
{'R','U'}, 7974};
```

Обратите внимание, что нельзя было бы инициализировать символьный массив `jd.country` строкой «RU», так как строки C++ оканчиваются символом «\0», поэтому в строке «RU» три символа, то есть на один больше, чем влезет в `jd.country`.

Объекты типа структур можно присваивать, передавать как параметры функции и возвращать из функции в качестве результата. Например:

```
address current;

address set_current(address next){
    address prev = current;
    current = next;
    return prev;
}
```

Остальные осмысленные операции, такие как сравнение (`==` и `!=`), не определены. Однако пользователь может самостоятельно определить эти операции.

Размер объекта структурного типа нельзя вычислить просто как сумму длин его элементов. Причина этого состоит в том, что многие платформы требуют, чтобы объекты определенных типов выравнивались в памяти по некоторым, зависящим от архитектуры границам (типичный пример: целое должно быть выровнено

по границе слова), что позволяет гораздо более эффективно обрабатывать такие объекты. Выравнивание приводит к «дырам» в структуре. Например, `sizeof (address)` равен 24, а не 22, как можно было ожидать.

Заметьте, что имя типа структуры становится доступным сразу после того, как оно встретилось, а не только после того, как полностью просмотрено все описание. Например:

```
struct link{
    link* previous;
    link* successor;
}
```

Однако объекты типа структуры не могут быть использованы до тех пор, пока не завершится его полное описание, поэтому выражение

```
struct no_good {
    no_good member;
};
```

является ошибочным. Чтобы дать возможность двум (или более) структурным типам ссылаться друг на друга, можно просто описать имя как имя структурного типа. Например:

```
struct list;           // должна быть определена позднее

struct link {
    link* pre;
    link* suc;
    list* member_of;
};

struct list {
    link* head;
}
```

Без первого описания `list` описание `link` вызвало бы синтаксическую ошибку.

Указатели

Использование указателей в программах на C++ Builder является если не основной, то очень значимой частью. Связано это с тем, что любая программа, оперирующая большим объемом данных, просто обязана бережно относиться к такому ресурсу, как оперативная память. А также то, что основной библиотекой компонентов для C++ Builder является Visual Component Library (Библиотека визуальных компонентов), изначально написанная для среды разработки Borland Delphi, языком разработки которой является Object Pascal. Особенностью Object Pascal является то, что все классы изначально (при описании) являются динамическими объектами. Указатели могут быть описаны, например, так:

```
char *p; // указатель на символ
char *const q; // константный указатель на символ
```

Переменная указатель содержит адрес объекта соответствующего ей типа, например:

```
char c, d;
...
```

```
p = &c; // p получит указатель на переменную c
d = *p; // *p – разыменовывает указатель (получает фактическое значение)
```

Как правило, переменные указатели содержат ссылки не на статически размещенные в памяти объекты (как показано в примере выше), а на динамические, то есть в памяти, выделенной для хранения переменной в процессе работы. Для выделения памяти используется оператор `new`, а для удаления – оператор `delete`. Например:

```
char *buffer;
buffer = new char; // выделение динамической памяти для
                  // хранения одного символа
...
delete buffer; // высвобождение памяти
```

При описании динамических массивов используется несколько другой синтаксис:

```
char *buffer;
buffer = new char[256]; // выделение памяти для хранения 256 символов
...
delete[] buffer; // высвобождение всего объема памяти,
                // выделенной для хранения массива
```

Следует отметить, что такой способ описания динамических массивов несколько несовременен. В C++ Builder введен новый совместимый с Borland Delphi вариант описания динамических массивов, который будет рассмотрен ниже.

При работе с динамическими записями часто используется операция `->`. Например:

```
address *p = new address;
...
p->name = "Петр Иванов";
p->number = 34 ;
```

Операция `->` фактически заменяет собой выполнение двух других последовательно записанных операторов «`*`» и «`.`». Так, указанный выше пример будет эквивалентен:

```
(*p).name = "Петр Иванов";
(*p).number = 34 ;
```

Для работы с классами, размещенными в динамической памяти, используется еще один вариант записи:

```
TList *MyList;
TItem *Item;
MyList = new TList(); // при создании объекта автоматически
                     // вызывается конструктор
Item = new TItem(125, "Hello", 34.032); // в скобках указаны значения
                                       // для передачи конструктору
...
MyList->Add(Item); // использование косвенной адресации
...
delete MyList; // при удалении объекта будет вызван деструктор
               // объекта (подробнее см. ниже)
```

Перечисления

Иногда появляется необходимость описать в программе собственный тип, содержащий ограниченное количество значений. Сделать это можно, используя ключевое слово `enum`, например:

```
enum days {su, mo, tu, we, th, fr, sa};
```

Так как для внутреннего представления элементов перечисления используются целые значения, то можно определить тип

```
enum modes {LASTMODE = -1, BW40=0, C40, BW80, C80, MONO = 7};
```

в котором значение `LASTMODE` равно `-1`, значения `C40`, `BW80` и `C80` соответственно будут равны 1, 2 и 3, а `MONO` равно 7. Использование этих типов настолько же легко, как и целочисленных.

```
days A;  
//...  
A = we;  
//...  
for (days i=su; i<=sa; i++) {...};  
//...  
if (A == th) {...};
```

1.3.6. Базовые операторы

К любой комбинации базовых типов могут применяться арифметические операции:

- `+` – плюс, унарный и бинарный;
- `-` – минус, унарный и бинарный;
- `*` – умножение;
- `/` – деление.

А также операции сравнения:

- `==` – равно;
- `!=` – не равно;
- `<` – меньше;
- `>` – больше;
- `<=` – меньше или равно;
- `>=` – больше или равно.

Обратите внимание, что деление целочисленных значений дает целый результат, например `7/2` есть 3. Над целыми может выполняться операция `%` получения остатка, например, `7%2` равно 1.

Стандартные преобразования типов

Когда вы используете арифметическое выражение типа `a + b`, где `a` и `b` – разные арифметические типы, C++ Builder исполняет некоторые внутренние преобразования, прежде чем вычислить выражение. Эти стандартные преобразования включают преобразование «более низких» типов данных к «более высоким» типам в целях точности и содержательности. Например, это необходимо в случае операций над операндами разного типа.

```
double d = 1;
int i = 1;

d = d + i;
i = d + i;
```

Ниже представлены шаги (по приоритетам), используемые C++ Builder для конвертации операндов при вычислении арифметических выражений.

1. Любые маленькие целочисленные типы преобразовываются так, как показано в табл. 1.1.3. После этого преобразования любые два значения ассоциируются с оператором по приведенному ниже алгоритму, для любого типа, будь то `int` (включая длинные и беззнаковые), `double`, `float` или `long double`.
2. Если один из операндов имеет тип `long double`, другой операнд тоже преобразуется в `long double`.
3. Иначе, если любой из операндов имеет тип `double`, другой операнд преобразуется в `double`.
4. Иначе, если любой из операндов имеет тип `float`, другой операнд преобразуется в `float`.
5. Иначе, если любой операнд имеет тип `unsigned long`, другой операнд преобразуется в `unsigned long`.
6. Иначе, если любой операнд имеет тип `long`, то другой операнд будет преобразован в `long`.
7. Иначе, если любой операнд имеет тип `unsigned`, то другой операнд будет преобразован в `unsigned`.
8. Иначе, оба операнда имеют тип `int` (операнды не преобразуются).

Таблица 1.3.3. Методы, используемые при стандартных арифметических преобразованиях

Тип	Конвертируется в	Метод
<code>char</code>	<code>int</code>	Со знаком или без, в зависимости от указания по умолчанию
<code>unsigned char</code>	<code>int</code>	Старший бит всегда обнуляется
<code>signed char</code>	<code>int</code>	Всегда, как знаковое
<code>short</code>	<code>int</code>	То же самое знаковое значение
<code>unsigned short</code>	<code>unsigned int</code>	То же значение с обнуленным старшим битом
<code>enum</code>	<code>int</code>	То же значение

Тип результата выражения будет тем же, что и у одного из двух операндов.

1.3.7. Выражения и операторы

В C++ имеется богатый набор операций, с помощью которых в выражениях образуются новые значения и изменяются значения переменных. Поток управления в программе задается с помощью операторов, а описания используются для введения в программе имен переменных, констант и т. д. Заметьте, что описания

В C++ имеется операция присваивания `=`, а не оператор присваивания, как в других языках. Таким образом, присваивание может встречаться в неожиданном контексте, например `x=sqrt(a=3*x)`. Это бывает полезно. `a=b=c` означает присвоение значения `c` переменной `b`, а затем переменной `a`. Другим свойством