Н. А. Литвиненко

Технология программирования на C++

Начальный курс

- ***Основы программирования на C++*** *– описание синтаксиса языка C, рассмотрение приемов и методов программирования в стиле классического C*
- ***Классы*** *– введение понятия класса, рассмотрение шаблонных классов, вопросов наследования и построения сложных классов*
- ***Потоковый ввод/вывод*** *– организация современного подхода к организации ввода/вывода при помощи потоковых классов*
- ***Стандартная библиотека шаблонов STL*** *– описание стандартной библиотеки и многочисленные примеры ее использования*
- ***Организация оконного интерфейса*** *– структура Windows-программы, методы обработки стандартных сообщений. Программирование с использованием API-функций*

УЧЕБНОЕ ПОСОБИЕ

BHV

Н. А. Литвиненко

Технология программирования на C++

Начальный курс

Допущено УМО вузов по университетскому политехническому образованию
в качестве учебного пособия для студентов высших учебных заведений,
обучающихся по направлению 654600
«Информатика и вычислительная техника»

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92C++(075.8)
ББК 32.973.26-018.1я73
Л64

Литвиненко Н. А.

Л64 Технология программирования на C++. Начальный курс. — СПб.: БХВ-Петербург, 2005. — 288 с.: ил.

ISBN 5-94157-655-2

Рассмотрены основы программирования на C++, начиная с описания синтаксиса языка C, приемов и методов программирования в стиле классического C до введения понятий классов, шаблонов классов и вопросов наследования. Уделено особое внимание использованию стандартной библиотеки шаблонов STL. Представлен современный подход к организации ввода/вывода при помощи потоковых классов. Рассматривается техника создания простейших Windows-приложений с использованием API-функций. Материал иллюстрируется многочисленными примерами.

Для студентов и преподавателей технических вузов и самообразования

УДК 681.3.068+800.92C++(075.8)
ББК 32.973.26-018.1я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Дарья Масленникова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.04.05.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 23,22.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 5-94157-655-2

Оглавление

ГЛАВА 1

Основы программирования на C++

Часто формализм языка рассматривается безотносительно конкретной реализации. И все же читателю будет предложено использовать в качестве основного инструмента для построения программ консольное приложение среды MS Visual C++ 6.0 (Win32 Console Application), ориентируясь на приведенные далее примеры, реализованные в этой среде программирования. Язык программирования в классическом стиле, рассмотренный в данной главе, будет дополнен лишь элементами расширения, принципиально не меняющими его сути. Однако для организации ввода/вывода воспользуемся подходом C++, применив потоковые классы, подробнее о которых будет рассказано далее, в *главе 3*.

Структура классической C-программы

#*Директивы препроцессора*	Объявление файлов включения
Объявление функций	Описание прототипов функций
Описание глобальных переменных и констант	

`main` (*параметры*) `//` *заголовок программы*
`{`
Описание локальных переменных и констант
Операторы
`}`
	Головная функция

Описание функций	

В основу любого компилятора для языка C и C++ заложена однопроходовая схема, т. е. процесс построения программы осуществляется за один проход по тексту. Таким образом, все имена переменных и констант программы должны быть объявлены до их первого использования.

Препроцессор

Для создания дополнительного сервиса, а также улучшения переносимости программ, в C++ реализован так называемый *препроцессор*, который выполняет набор инструкций перед началом компиляции программы. Все директивы препроцессора начинаются с символа `#`.

#include

Директива включения файла. Обычно используется для вызова так называемых файлов включений, содержащих *прототипы* (т. е. описания) библиотечных функций. Также позволяет включить часть текста программы, записанного в другой файл. По сути дела, как в текстовом редакторе, объединяются два файла.

Формат директивы:

```
#include <имя файла включения>
```

Здесь угловые скобки `< >` являются частью конструкции и указывают на то, что файл ищется в каталоге файлов включения, прописанном в среде программирования.

Формат директивы:

```
#include "имя файла включения"
```

Двойные кавычки `" "` означают, что поиск файла включения осуществляется в том же каталоге, в котором открыт исходный файл.

```
#include <stdlib.h>   // Включение стандартной библиотеки
#include "meny.h"     // Включение файла meny.h
```

#define

Определение символического обозначения. Устаревшая конструкция, вместо нее более целесообразно определять константные переменные, но об этом чуть позже.

Формат директивы:

```
#define ИМЯ ЗНАЧЕНИЕ
```

В результате действия директивы все символические обозначения `ИМЯ` в тексте программы заменятся на `ЗНАЧЕНИЕ` перед началом компиляции. Исключе-

ние составляют текстовые константы, заключенные в двойные кавычки `"..."`, внутри которых подстановки не производятся. Опять же, проводя аналогию с текстовым редактором, это просто операция *"поиск и замена"*, например:

```
#define PI 3.14159
```

Перед компиляцией все переменные `PI` будут заменены их значением `3.14159`.

Примечание

Компилятор языка C++ различает строчные и прописные буквы, поэтому для выделения в тексте программы символических констант их принято писать заглавными буквами.

Некоторые другие директивы мы рассмотрим по мере надобности, но на данном этапе главная директива препроцессора, с которой будут начинаться все наши программы, — это директива включения `#include`.

Создание простой программы

С-программа имеет ярко выраженный модульный характер. Ее головная функция состоит из множества обращений к функциям, выполняющим те или иные действия, и отличается от прочих только тем, что имеет фиксированное имя `main`.

Листинг 1.1. Простейшая С-программа

```
#include <iostream.h>                // Включение библиотеки
main()
{ int i;                             /* Описание целой переменной*/
       for (i = 0; i < 10; i++)      // Оператор цикла
              cout << i << endl;     // Вывод числа
}
```

Здесь после двойного слэша `//` помещен комментарий, который занимает остаток строки и игнорируется компилятором. Можно также располагать комментарий между парами ограничителей `/*...*/`, в этом случае он может и не ограничиваться одной строкой.

Не вдаваясь пока в детали, прокомментируем приведенный текст программы:

1. Первая строка — подключение файла `iostream.h`, расположенного в стандартном каталоге `include` среды разработки (на это указывают угловые скобки `< >`), который должен быть в ней прописан, что обычно происходит при инсталляции компилятора. Этот файл необходим для того, чтобы

можно было воспользоваться оператором потокового вывода `cout <<`. Вообще-то `cout` — это экземпляр класса `ostream`, но пока не будем углубляться, отметим только, что это работает.

2. `main()` — имя головной функции. Именно ей, согласно стандарту языка, передается управление при запуске программы. В С-программе должна обязательно присутствовать функция `main()`.
3. Тело функции располагается между открывающей и закрывающей скобками `{ }`. Надо иметь в виду, что компилятор очень скрупулезно подсчитывает, сколько скобок открыто и сколько закрыто — если скобку забыли закрыть, он выдает сообщение об ошибке.
4. `int i;` — описываем переменную для хранения целого числа.
5. `for (i = 0; i < 10; i++)` — оператор цикла, обеспечивает выполнение оператора вывода на консоль 10 раз, изменяя значение переменной `i` от 0 до 9.
6. `cout << i << endl;` — вывод числа на экран монитора (консоль вывода), после чего производится переход в начало следующей строки `endl` *(end of line)*. Читаем `cout` как *console output*. Как видно из примера, оператор `cout <<` можно применять агрегатно и выводить сразу несколько значений, а тип выводимого значения оператор определит сам.

Для выполнения рассмотренного примера откроем среду программирования MS Visual C++ 6.0, запустив головной модуль msdev.exe. Пока будем работать с консольными приложениями.

1. Выполнив команду меню **File | New**, выберем **Win32 Console Application** в окне создания проекта New, открытом на вкладке **Projects** (рис. 1.1).
2. В поле **Location** введем вручную (либо воспользуемся поиском по кнопке **...**) имя рабочей папки. Например, `C:\WORK`.
3. В поле **Project name** введем имя нашего будущего проекта — `example1`, после чего нажмем кнопку **ОК**.
4. Появляется диалоговое окно, приведенное на рис. 1.2, в котором мы оставим настройку по умолчанию — **An empty project** (Пустой проект).
5. Получаем диалоговое окно **New Project Information**. Писать программу пока негде, на данном этапе лишь создана папка для нового проекта `C:\WORK\example1`.
6. Выполнив команду меню **File | New**, выберем на вкладке **Files** пункт **C++ Source File** (рис. 1.3) и в поле **File name** введем имя файла, например `main`. (Проверьте, что перед полем **Add to project** выставлен флаг, иначе потом придется "вручную" добавлять файл к проекту.)

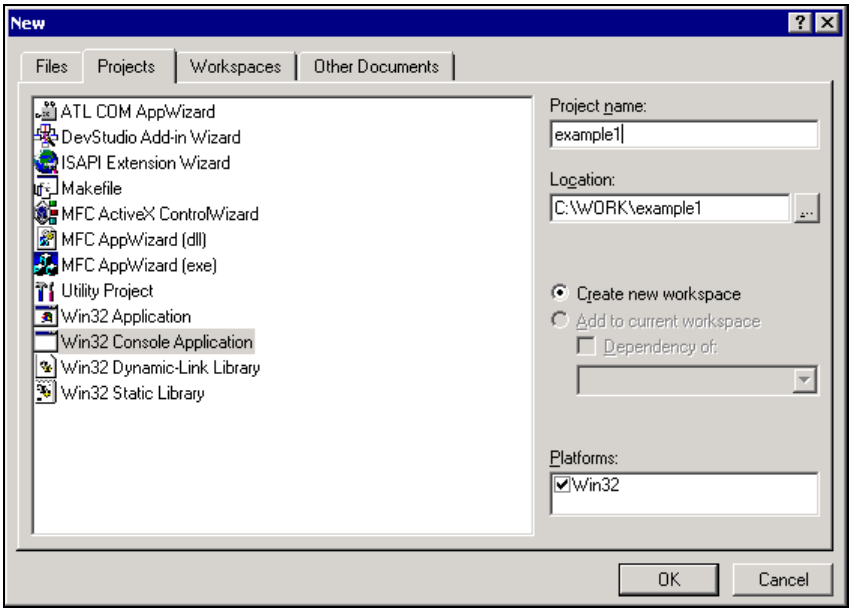

Рис. 1.1. Окно создания проекта

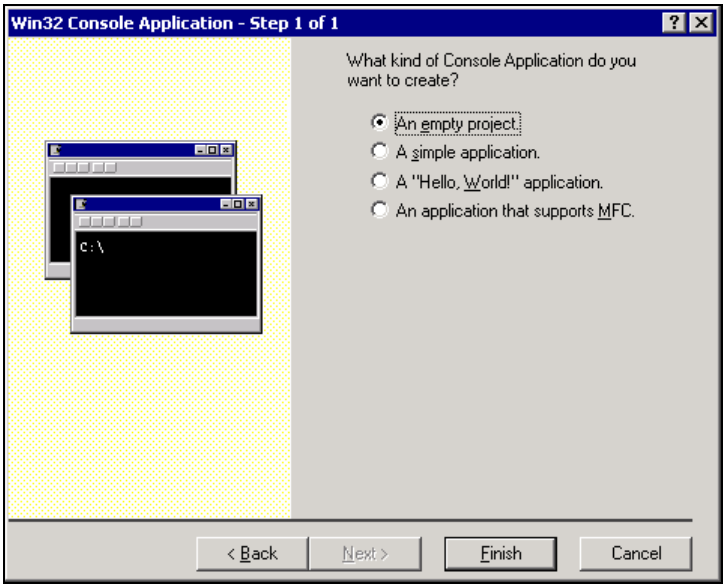

Рис. 1.2. Создание проекта Win32 Console Application. Шаг 1

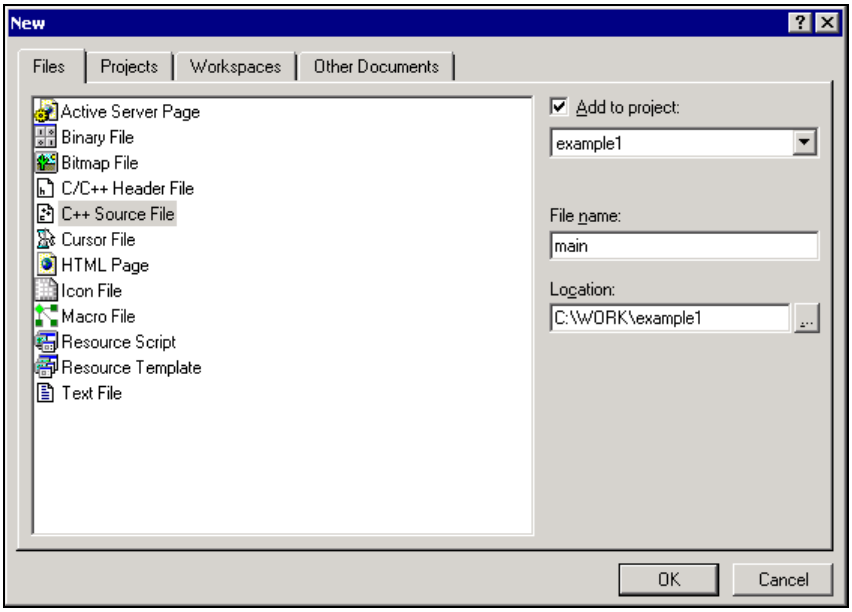

Рис. 1.3. Добавление файла к проекту

7. Введем текст программы и запустим на выполнение нажатием кнопки **!**. Это единственная кнопка со значком красного цвета. Получим результат, приведенный на рис. 1.4.

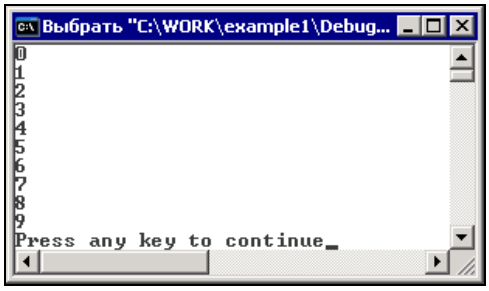

Рис. 1.4. Результат выполнения программы

Набирая текст этой программы, сложно допустить ошибку, но если это все-таки произошло, перейдите в окно вывода **Output** на вкладку **Build** и установите указатель на сообщение об ошибке. После двойного щелчка на нем компилятор отметит строку с ошибкой. Исправьте ошибку и повторите процедуру запуска программы. Если у вас есть навыки работы в текстовых редакторах, то ввод и редактирование текста программы не должны вызывать никаких проблем. Однако, поскольку это все-таки специализированный ре-

дактор, он имеет некоторые дополнительные возможности. Например, если в тексте программы установить курсор рядом со скобкой либо выделить ее, можно использовать одну из следующих полезных команд: поиск парной скобки и выделение блока текста, заключенного в скобки.

Полный список команд редактора можно посмотреть, выполнив команду меню **Help | Keyboard Map**.

Типы данных. Описание переменных

Вначале рассмотрим подробнее, как хранятся переменные различных типов в памяти. Адресуемая единица памяти, *байт*, состоит из 8 разрядов *(бит)*, куда и записывается число в двоичном коде, например:

$00010101 = 2^4 + 2^2 + 2^0 = 16 + 4 + 1 = 21$.

Примечание

Двоичная система счисления — позиционная система с основанием 2.

Правила двоичной арифметики очень просты (табл. 1.1).

Таблица 1.1. *Двоичная арифметика*

`0 + 0 = 0`	`0 * 0 = 0`
`0 + 1 = 1`	`0 * 1 = 0`
`1 + 1 = 10`	`1 * 1 = 1`

В один байт может быть записан код от 0 до 255 (2^8–1).

Таблица 1.2. *Перевод двоичных чисел в восьми-, десяти- и шестнадцатеричное представление*

2	16	8	10	2	16	8	10
0000	0	0	0	1000	8	10	8
0001	1	1	1	1001	9	11	9
0010	2	2	2	1010	A	12	10
0011	3	3	3	1011	B	13	11
0100	4	4	4	1100	C	14	12
0101	5	5	5	1101	D	15	13
0110	6	6	6	1110	E	16	14
0111	7	7	7	1111	F	17	15

Пользуясь табл. 1.2, можно очень просто представить двоичное число в виде его шестнадцатеричного аналога. На самом деле, числа записывают в двоичной форме очень редко, поскольку тогда они занимают много разрядов. Обычно двоичное число записывают более компактно, используя шестнадцатеричную форму записи, пример которой приведен на рис. 1.5.

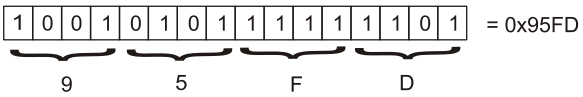

Рис. 1.5. Преобразование двоичного числа в шестнадцатеричное представление

Нетрудно было разбить это число на тетрады и записать, сверившись с таблицей кодов, результат. Обратное преобразование также не вызывает сложности, достаточно вместо каждой шестнадцатеричной цифры записать соответствующую двоичную тетраду.

Размер выделенной памяти под переменные различных типов и диапазон их изменения приведен в табл. 1.3—1.4.

Таблица 1.3. *Размер типа данных Visual C++*

Тип	Размер в битах	Диапазон
`bool`	8	true, false
`char`	8	–128 .. 127
`short`	16	–32768 .. 32767
`int, long`	32	–2147483648 .. 2147483647
`float`	32	3.4E-38 .. 3.4E+38
`double`	64	1.7E-308 .. 1.7E+308
`long double`	80	3.4E-4932 .. 1.1E+4932

Примечание

- Тип `bool` отсутствовал в классическом C и появился лишь в C++.
- Для 32-битных приложений тип `int` совпадает с `long` (полная запись `long int`).
- Целые типы данных могут иметь модификатор `unsigned` (беззнаковый). Диапазон таких переменных отсчитывается от 0.

Таблица 1.4. *Размер целых беззнаковых типов*

Тип	Размер в битах	Диапазон
`unsigned char`	8	0 .. 255
`unsigned short`	16	0 .. 65535
`unsigned int, long`	32	0 .. 4294967295

Рассмотрим подробнее, как в памяти представлена переменная типа `short`. Например, константа 1 в двоичном представлении имеет вид:

```
0000 0000 0000 0001.
```

Чтобы получить число 2, нужно прибавить к 1 еще 1 и т. д., в результате получаются все целые числа.

```
    + 0000 0000 0000 0001
      0000 0000 0000 0001
    = 0000 0000 0000 0010
```

Отрицательные числа для процессора 80×86 представлены в дополнительном коде и образуются инверсией числа с добавлением 1 (при инвертировании кода 1 заменяется на 0, а 0 на 1), например –1 получается по следующей схеме:

```
    + 1111 1111 1111 1110          + ~1
      0000 0000 0000 0001            1
    = 1111 1111 1111 1111          = -1
```

Такой способ образования отрицательных чисел позволяет свести операцию вычитания к операции сложения с отрицательным числом. Таким образом, операция вычитания является излишней для системы команд процессора и может быть исключена. Пример: 1 – 1 = 1 + (–1) = 0.

```
    + 0000 0000 0000 0001
      1111 1111 1111 1111
    = 0000 0000 0000 0000
```

C++ относится к таким языкам программирования, в которых все переменные, используемые программой, должны быть предварительно описаны. В описании переменной задается ее класс памяти и тип для того, чтобы транслятор выделил нужное количество памяти и в нужном разделе. Для обозначения переменной служит имя — символическое обозначение переменной. Начинается имя с буквы или символа подчеркивания "_", может содер-

жать буквы латинского алфавита, цифры и знак подчеркивания. Длина имени может быть произвольной, но распознается по первым 32 символам (это зависит от настроек компилятора). Строчные и прописные буквы различаются компилятором как разные буквы.

Переменные могут быть описаны в любом месте текста программы, но обязательно до их использования. Имена переменных в операторах описания отделяются запятыми. Пример:

```
int i,j,k;
```

Возможна инициализация переменных при описании, например:

```
int i = 0;
```

Переменные целого типа: *bool*, *char*, *short*, *int*, *long*

При описании переменной целого типа необходимо учитывать диапазон ее изменения. Если переменная может принимать одно из двух значений (`true`, `false`), используется тип `bool`. Если же необходим больший диапазон выбора значений, можно использовать переменную одного из следующих типов: `char` (от –127 до +127), `short` (от –32 768 до +32 767) или `int` (от –2 147 483 648 до +2 147 483 647). Тип `long` совпадает с типом `int`. Однако для переменных целого типа необходимо иметь в виду, что при выполнении арифметических операций процессор разворачивает число до длины машинного слова (для 32-битных приложений удвоенное машинное слово — 4 байта), т. е. никакой экономии в скорости выполнения операций не достигается при использовании типа меньшей размерности, так что выбор типа данных определяется лишь экономией памяти при хранении данных.

Если заведомо известно, что переменная принимает только положительные значения, целесообразно использовать модификатор `unsigned`, который обеспечит дополнительный контроль со стороны компилятора на присваивание отрицательного значения и увеличит вдвое диапазон положительных чисел.

enum — перечисляемый тип

Перечисляемый тип используется, как правило, для описания символических констант. Переменным, перечисленным в списке `enum`, присваиваются последовательно целые значения, начиная с 0. Пример:

```
enum { BLACK, BLUE };
```

Здесь описан неименованный объект перечисляемого типа, в котором определены две символические константы: `BLACK = 0` и `BLUE = 1`.