Максим Кузнецов
Игорь Симдянов

bhv®

C++ мастер-класс в задачах и примерах

158 задач и готовых решений!

Тонкости C и C++

Объектно-ориентированное программирование

Библиотека STL

Библиотека ввода-вывода

On-line поддержка

+ CD-ROM

Максим Кузнецов
Игорь Симдянов

C++ мастер-класс
в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.06
ББК 32.973.26-018.2
К89

Кузнецов, М. В.

К89 С++. Мастер-класс в задачах и примерах / М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2007. — 480 с.: ил. + CD-ROM

ISBN 978-5-94157-953-2

Книга разбита на две основные части: задачи и решения. Рассматриваются базовые конструкции языка С++, тонкие моменты низкоуровневых операций, объектно-ориентированное программирование, разработка приложений при помощи стандартной библиотеки шаблонов STL, а также прикладные задачи. Особенностью предлагаемых задач и их решений является независимость от платформы и среды программирования, поэтому книга будет интересна как UNIX-, так и Windows-программистам. Компакт-диск содержит листинги всех готовых решений, представленных в книге.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Наталья Таркова*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.12.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

ISBN 978-5-94157-953-2

Оглавление

Введение

Язык C++ — на сегодняшний день один из самых сложных языков программирования. Тем не менее он де-факто представляет собой промышленный стандарт. Изучение является обязательным требованием для программиста, работающего с использованием нескольких языков. Профессиональный программист, не владеющий C++, вызывает такое же удивление, как полиглот, не владеющий английским языком. Любые методологические приемы, библиотеки в первую очередь рассматриваются на примере C++.

Однако C++ ведет свою историю от C, который изначально является системным языком. C++ — это своеобразное расширение C, привносящее в язык объектно-ориентированный подход, исключения, пространство имен и т. п. Одной из целей при создании C++ было сохранение скорости выполнения конечных программ. Эта цель была успешно достигнута, однако благодаря значительной части системности языка.

Рассмотрим, чем отличается системный язык от обычного языка высокого уровня. Изначально программы создавались при помощи машинных кодов, т. е. номеров инструкций процессора. Инструкции в свою очередь осуществляли различные операции вроде перемещения значения из одного регистра в другой. Путем размещения в регистрах значений и осуществления над ними операций (вызова операции по ее номеру), за несколько вызовов (тактов) можно было выполнить определенное действие — например, сложить два числа. Из простейших арифметических операций можно было построить небольшую программу, например, деления числа (в те далекие времена еще не было сопроцессоров, где операции с плавающей точкой реализованы на аппаратном уровне). Система номеров команд, мягко говоря, неудобна для повседневного использования. Так, для копирования содержимого регистра в другой регистр необходимо было использовать один номер, для копирования в третий регистр предназначен второй номер, для копирования содержимого в ячейку оперативной памяти — третий, причем номера команд и адреса яче-

ек необходимо было писать в двоичной форме (листинг В1), что значительно затрудняло отладку. Программисты со стажем вспоминают, что приходилось помнить до 400 основных кодов команд. Самое страшное начиналось, когда нужно было переходить на процессор новой архитектуры — необходимо было как-то забыть старые команды и изучать новые, при этом не путать их. Это было достаточно сложно, т. к. и там, и там были цифры.

Листинг В1. Машинный код (шестнадцатеричный формат)

```
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
...
73 63 72 69 62 65 64 20 69 6E 20 52 46 43 20 31
33 32 31 2E 20 20 57 68 65 6E 20 53 68 65 63 6B
```

В связи с этим практически повсеместно стал вводиться язык Assembler — команды обозначались буквенными сокращениями и назывались *мнемониками*. Одна мнемоника обозначает сразу несколько машинных кодов, например перемещение значения в регистры, ячейку в Assembler можно при помощи мнемоники `MOV` (от англ. *move* — перемещать). С введением Assembler жизнь программистов значительно облегчилась, даже при смене процессора, неизбежно приводившего к изменению Assembler, основной набор команд оставался прежним. Однако архитектура процессора значительно влияла на программы, программист был вынужден помнить, в какие регистры следует поместить данные, чтобы осуществить сложение, а в какие лучше не надо. Кроме того, процессоры не сопровождались математическим сопроцессором, и алгоритмы численного вычисления синуса угла (реализованные в любом языке высокого уровня в виде стандартной функции) составляли предмет многочисленных диссертаций. Главным же недостатком Assembler оставалась значительная привязка к архитектуре процессора.

Одновременно с Assembler стали появляться языки высокого уровня — идея их заключалась в том, чтобы создать язык, который бы не зависел от архитектуры процессора. Программа, созданная на одной машине и операционной системе, будучи откомпилированной на машине с другой архитектурой и операционной системой, вела бы себя точно так же. Так как различных архитектур немного, а программистов несколько миллионов, проще для каждой из архитектур создать свой собственный компилятор, который сможет адаптировать любую программу, написанную на языке высокого уровня. Введение языков высокого уровня позволило программистам не вникать в особенности

архитектуры компьютера и не выполнять рутинные системные задачи, реализацию которых взял на себя компилятор. Теперь программист оперировал не низкоуровневыми аппаратными понятиями, такими как регистр, ячейка памяти, адрес, а более высокими абстрактными конструкциями — переменная, массив, строка, файл, функция, процедура. Однако для того чтобы имелась возможность решать системные задачи (писать компиляторы, операционные системы), ряд языков в своем синтаксисе оставили системные возможности, указатели, ссылки, битовые поля и т. п. К таким языкам относятся C и C++. Именно поэтому, когда начинающий программист, только приступивший к изучению этих языков, встречает указатель на функцию, который мало того что передается в другую функцию, но и умудряется еще и вызывать ее, это порождает слабый протест и ощущение уходящей из-под ног почвы. Именно поэтому программисты, знакомые с Assembler, изучают C/C++ гораздо быстрее, чем программисты, знающие языки высокого уровня. Знатокам Assembler проще свыкнуться с мыслью, что в C/C++ никаких строк и массивов нет, а их реализация — лишь бутафория, призванная упростить процесс создания программы человеком.

Таким образом, сложность языка C/C++ основывается на том, что язык является системным, и помимо того, что он является языком высокого уровня, его также можно назвать языком низкого уровня, позволяющим разрабатывать системные программы. Однако это еще не все. Рассмотренные системные особенности языка скорее относятся к букве C, но свои два плюса язык C++ получил благодаря дополнению C объектно-ориентированной моделью.

Переход от структурного программирования к объектно-ориентированному (ООП) связан, в первую очередь, с возрастающей сложностью создаваемого программного обеспечения. В первое время программы не превышали нескольких сотен строк. Увеличение количества кода в программах до нескольких тысяч строк привело к внедрению приемов структурного программирования (появились функции), что позволило создавать и сопровождать программы размерами до ста тысяч строк. Стремительное развитие программного обеспечения потребовало создания и сопровождения еще большего объема кода. Ответом на это было создание объектно-ориентированной технологии. Применение данной технологии дает возможность создавать еще большие по объему приложения и позволяет программисту оперировать при создании кода объектами реального мира, а не понятиями программы (переменные, функции и т. п.). ООП позволяет создавать программу на более высоком абстрактном уровне, объединяя данные и методы в классы, и является прототипом языка нового поколения. Таким образом, C++ является языком высокого уровня, имеет системные средства и обладает объектно-ориентированной технологией, занимая три уровня на шкале увеличения уровня абстрактности языков программирования (рис. В1).

Рис. В1. Шкала увеличения абстрактности языков программирования

Однако C++ — это не только основной язык, это также большое число библиотек, которые были написаны за время его существования. Сам по себе язык программирования способен не на много, сила языка C/C++ в огромном числе созданного программного обеспечения, которое можно использовать повторно. В результате сложнейшие математические задачи решаются в несколько строк. Поэтому процесс обучения языку C++ разделяется на два этапа: это изучение базовых конструкций языка и изучение библиотек. Конструкции языка играют роль грамматики — они позволяют участникам процесса программирования (программистам и компьютерам) понимать друг друга. Человек может до тонкостей знать грамматику языка, но если в жизни он прочитал лишь учебник грамматики, он вряд ли сможет написать книгу, которая заинтересует читателей. Поэтому изучение языка не ограничивается ознакомлением с базовыми основами языка, любой опытный программист должен знать библиотеки, которые позволят ему разрабатывать более функциональные программы за более короткие сроки. Библиотеки могут быть совершенно различными, от управления ресурсами операционной системы (API Windows, системные вызовы UNIX) и создания интерфейса (MFC, VCL, DirectX, OpenGL, Qt и т. д.) до математических библиотек символьной алгебры или разбора регулярных выражений.

Среди этих библиотек выделяется стандартная библиотека, которая должна быть реализована в любом компиляторе, именно она позволяет осуществлять ввод/вывод, а также реализовать конструкции, отсутствующие в языке. Одним из элементов стандартной библиотеки является стандартная библиотека шаблонов STL (Standard Template Library), которая позволяет работать с произвольными типами. По словам Н. Джосьютиса, автора книги C++ "Стан-

дартная библиотека"[1]: "Концепция STL в известном смысле противоречит исходным принципам объектно-ориентированного программирования: STL отделяет друг от друга данные и алгоритмы, вместо того чтобы объединять их".

Изучение базовых основ языка C/C++ (грамматики) строго обязательно для любого человека, претендующего на знание языка. Изучение библиотек (чтение литературы, если проводить аналогию с изучением разговорных языков) — дело индивидуальное и зависит от интересов программиста и решаемых им задач.

Замечание

Человек, интересующийся математикой, с большим интересом и эффективностью будет читать "Топологию", чем классическую литературу XIX века. Хотя для чтения книг изучать базовые основы языка все равно придется.

Поэтому в данном задачнике мы сосредоточимся в первую очередь на базовых конструкциях языка, которые актуальны для любой операционной системы, будь то UNIX или Windows, и любого компилятора. Книга заостряет внимание читателя на тонких моментах C++, заставляя глубже копнуть язык и разобраться с особенностями, которые на первый взгляд кажутся странными, но на самом деле открывают новые возможности языка и позволяют решать более широкий круг задач.

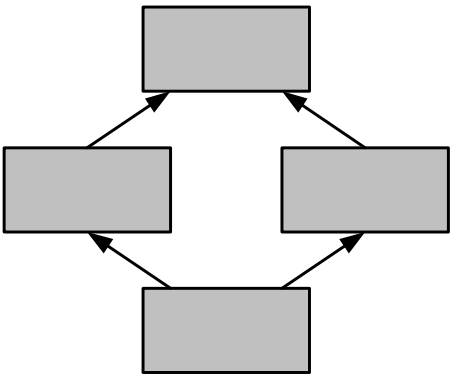
Изучение той или иной библиотеки не будет входить в круг наших задач, т. к., во-первых, почти по каждой из библиотек следует писать отдельную книгу и отдельный задачник, а во-вторых, в силу широкого распространения C++-библиотеки (да и C++-программисты тоже) слишком далеки друг от друга. Windows-программисты будут скучать при обсуждении системных вызовов UNIX, а UNIX-программистов будет только раздражать подробное описание особенностей DirectX.

Для кого предназначена книга?

Книга предназначена для читателей, которые хотят выяснить, владеют ли они языком C++ или нет. C++ интересен огромным числом библиотек, позволяющих создавать удивительные вещи: программное обеспечение для управления хитроумными приборами и роботами, игры, утилиты, языки программирования и все, с чем только можно столкнуться в мире компьютеров. Такая гибкость и мощь привлекает большое число программистов, которые стремятся побыстрее освоить ту или иную библиотеку, интерфейс или создать

[1] Джосьютис Н. C++ Стандартная библиотека. Для профессионалов. — СПб.: Питер, 2004. — 730 с.

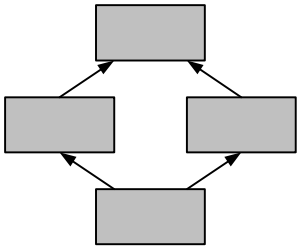
что-то свое. Если программист не обладает глубокими знаниями базовых основ языка C++, то взаимодействие с библиотекой и сам процесс создания программного обеспечения превращается в муку. Книга позволяет ответить на вопрос, готов программист к штурму сторонних библиотек или нет: если задания не вызывают трудностей или вы можете предложить более эффективные решения, чем представлены в ответах — вы знаете C++, если более 20% заданий вас ставит в тупик, то для успешного программирования вам следует еще раз проштудировать пособия по C++. Ничего страшного в том, что вы читаете книгу по программированию 5—6 раз для того, чтобы понять все тонкости языка, нет. Плотность информации, особенно в книгах по C++, очень велика, и усвоить все с первого раза, особенно, если C++ — первый язык, очень сложно. Поэтому стоит возвращаться к описанию языка снова и снова. Иногда возникает ощущение, что наступило понимание всех тонкостей языка — данная книга позволит вам проверить истинное ли это ощущение или ложное.



ЧАСТЬ I

Задачи по основам языка

ГЛАВА I.1



Базовые конструкции языка

В основе любого языка лежат типы, операторы и ключевые слова, которые позволяют формировать составные операторы. Конструкции языка появились не просто так, а в результате длительной эволюции и бурного обсуждения сообществом программистов. Если вам кажется, что какая-то конструкция языка лишняя, вы, скорее всего, ошибаетесь. Все лишние конструкции на сегодняшний день исключены из языка, и присутствие в языке конструкции говорит о том, что она необходима для элегантного решения ряда задач.

Современное поколение программистов практически не застало конструкций и операторов, использование которых приводит к ухудшению качества и устойчивости программ. Почти все такие конструкции были исключены из ранних языков программирования высокого уровня. Например, в ранних вариантах языка FORTRAN был не только множественный выход из функции при помощи оператора `return`, но и множественный вход, т. е. можно было задать несколько точек в функции, и ее выполнение могло осуществляться с начала, с середины или с конца функции, в зависимости от формы вызова. Такие функции, возможно, и позволяли сэкономить пару тактов процессора, но их сопровождение требовало значительных усилий программиста.

Когда компьютеры стоили "миллионы долларов", время работы программиста не имело значения — стоимость работы программиста была каплей в море. С удешевлением компьютеров ситуация резко поменялась, львиную долю бюджета стало составлять не компьютерное время, а время работы программиста. Чем проще ему сопровождать программу, чем быстрее он выполнит свою работу, тем эффективнее протекает сам процесс разработки программного обеспечения. Поэтому в современных программах ценится, в первую очередь, их читабельность и лишь затем скорость выполнения и потребления памяти.

Замечание

Особняком в требованиях стоит отсутствие ошибок, которое ценится всегда. Однако согласно последним исследованиям плотность ошибок не зависит от характера программного обеспечения или языка программирования, а зависит лишь от квалификации программиста. Это означает, что чем короче будет код, тем меньше ошибок он будет содержать. Короткий код, как правило, является более читабельным — именно к нему следует стремиться.

В данной главе мы рассмотрим задачи, предусматривающие использование простых конструкций, таких как операторы цикла, ветвления, приведения типов, логические и битовые операторы.

Замечание

Решения задач данной главы можно найти в каталоге code\1 компакт-диска, прикладываемого к книге.

I.1.1. Включение заголовочных файлов

Три представленные в листинге I.1.1 записи являются правильными и позволяют включить заголовочный файл библиотеки ввода/вывода.

Листинг I.1.1. Три варианта подключения стандартной библиотеки `iostream`

```
#include <iostream>
#include <iostream.h>
#include "iostream.h"
```

Назовите наиболее предпочитаемый в случае библиотеки `iostream` вариант подключения и объясните, для каких целей применяется каждый из случаев.

I.1.2. Сколько байтов занимает каждый из базовых типов?

Определите, сколько байтов отводится под каждый из базовых типов:

- `char`;
- `wchar_t`;
- `unsigned char`;
- `signed char`;
- `int`;
- `unsigned int`;
- `signed int`;
- `unsigned short int`;
- `signed short int`;
- `long int`;
- `signed long int`;
- `unsigned long int`;
- `float`;
- `double`;
- `logn double`;
- `bool`.

I.1.3. Сколько байтов занимает тип *void*?

При определении размера типа `void` компилятор сообщает об ошибке. Объясните, почему это происходит, и сколько байтов отводится под тип `void`?

I.1.4. Равны ли числа?

В листинге I.1.2 между собой сравниваются два числа, каков будет результат сравнения? Числа равны между собой или нет?

Замечание

Предполагается, что выполнение программы производится на компьютере с 32-битной архитектурой.

Листинг I.1.2. Сравнение двух чисел

```
#include <iostream>
using namespace std;

int main()
{
  unsigned short int first = 60000;
  signed short int second = 60000;
  if(second == first) cout << "Числа эквивалентны\n";
  else cout << "Числа не эквивалентны\n";

  return 0;
}
```

I.1.5. Результат сравнения?

В листинге I.1.3 между собой сравниваются два числа: 60 000 и –5536. Разумеется, программа сообщает, что они не равны. Что требуется сделать, чтобы программа посчитала эти две переменных равными, при условии, что изменять значения самих переменных нельзя (т. е. не допускается прибавлять, удалять, делить, умножать и выполнять другие математические преобразования с переменными).

Замечание

Предполагается, что выполнение программы производится на компьютере с 32-битной архитектурой.

Листинг I.1.3. Сравнение двух чисел

```
#include <iostream>
using namespace std;

int main()
{
  unsigned short int first = 60000;
  signed short int second = -5536;
  if(second == first) cout << "Числа эквивалентны\n";
  else cout << "Числа не эквивалентны\n";

  return 0;
}
```

I.1.6. Сравнение инкремента и постинкремента

Каков результат выполнения программы, представленной в листинге I.1.4? Объясните, почему так происходит?

Замечание

Залогом успешного программирования на языке C++ (или на любом другом языке) является автоматическое применение базовых конструкций. Хороший программист заранее видит и знает результат операций, что позволяет ему без усилий создавать свои программы и читать чужие. Если базовые навыки не отработаны, то программирование превращается в эксперимент методом тыка, а сам программист напоминает музыканта, пренебрегавшего гаммами, каждая попытка сыграть какое-либо произведение приводит лишь к тому, что на четвертой ноте у него запутываются пальцы.

Листинг I.1.4. Сравнение инкремента и постинкремента

```
#include <iostream>
using namespace std;

int main()
{
  int i = 10;

  if(++i == i++) cout << "числа равны\n";
  else cout << "числа не равны\n";

  if(i++ == ++i) cout << "числа равны\n";
  else cout << "числа не равны\n";
```

```
  if(++i == ++i) cout << "числа равны\n";
  else cout << "числа не равны\n";

  if(i++ == i++) cout << "числа равны\n";
  else cout << "числа не равны\n";

  if(i++ == --i) cout << "числа равны\n";
  else cout << "числа не равны\n";

  if(i++ == i--) cout << "числа равны\n";
  else cout << "числа не равны\n";

  if(i-- == ++i) cout << "числа равны\n";
  else cout << "числа не равны\n";

  if(i-- == i++) cout << "числа равны\n";
  else cout << "числа не равны\n";
}
```

I.1.7. Четное или нечетное?

Создайте программу, которая запрашивает целое число у пользователя и определяет, является оно четным или нет.

I.1.8. Имя программы

Создайте программу, которая выводит свое собственное имя.

I.1.9. Чем отличается *switch* от конструкции *if-else-if*?

В листингах I.1.5 и I.1.6 представлены две программы, которые в зависимости от введенного пользователем числа (от 1 до 5) выводят английские названия цифр. В листинге I.1.5 для этого используется оператор `switch`, а в листинге I.1.6 конструкция `if-else-if`. Результат работы программ идентичен. Чем отличается оператор `switch` от конструкции `if-else-if` или это одно и то же? Если операторы различны, то когда оправдано применение `switch`, а когда `if-else-if`?

Листинг I.1.5. Использование оператора `switch`

```
#include <iostream>
using namespace std;
```

```
int main()
{
  int number = 0;

  cout << "Input the number ";
  cin >> number;

  switch(number)
  {
    case 1:
      cout << "\nfirst\n";
      break;
    case 2:
      cout << "\nsecond\n";
      break;
    case 3:
      cout << "\nthird\n";
      break;
    case 4:
      cout << "\nfourth\n";
      break;
    case 5:
      cout << "\nfifth\n";
      break;
    default :
      cout << "\nInput a number between 1 and 5\n";
  }

  return 0;
}
```

Листинг I.1.6. Использование оператора `if-else-if`

```
#include <iostream>
using namespace std;

int main()
{
  int number = 0;

  cout << "Введите число ";
  cin >> number;
```

```
  if(number == 1) cout << "\nfirst\n";
  else if(number == 2) cout << "\nsecond\n";
  else if(number == 3) cout << "\nthird\n";
  else if(number == 4) cout << "\nfourth\n";
  else if(number == 5) cout << "\nfifth\n";
  else cout << "\Введите число между 1 и 5\n";

  return 0;
}
```

I.1.10. Вывод случайного числа символов

Создайте программу, которая выводит случайное число символов * от 0 до 10.

I.1.11. Вывод четных чисел

Создайте программу, которая спрашивает у пользователя число и выводит в цикле четные числа от 0 до заданного числа.

I.1.12. Вывод всех видимых ASCII-символов

Создайте цикл, выводящий все ASCII-символы, начиная с 32-го символа, с которого начинаются видимые символы.

I.1.13. Поиск простых чисел

Создайте скрипт, который запрашивает у пользователя число и ищет простые числа до введенной пользователем величины.

Замечание

Простыми называются числа, которые делятся только на 1 и на самих себя.

I.1.14. Упаковка цикла *for*

Пусть имеется программа, которая в цикле `for` увеличивает значение одной переменной `i` и уменьшает значение другой переменной `j` до тех пор, пока их величины не сравняются (листинг I.1.7).

Листинг I.1.7. Цикл `for`

```
#include <iostream>
using namespace std;
```

```
int main()
{
  int number = 10;
  int j = number;

  for(int i = 0; i <= j; i++)
  {
    cout << i << " " << j << "\n";
    j--;
  }
  return 0;
}
```

Результат работы программы выглядит следующим образом:

```
0 10
1 9
2 8
3 7
4 6
5 5
```

Перепишите листинг I.1.7 таким образом, чтобы у цикла `for` отсутствовало тело цикла (листинг I.1.8).

Листинг I.1.8. Результат упаковки цикла `for`

```
#include <iostream>
using namespace std;

int main()
{
  int number, j, i;
  for(???);

  return 0;
}
```

Вместо знаков вопроса в листинге I.1.8 подставьте решение таким образом, чтобы программа была идентична по результату программе из листинга I.1.9.

I.1.15. Преобразование десятичного числа в двоичное

Создайте программу, которая запрашивает у пользователя целое число и выводит его в двоичном представлении.

I.1.16. Преобразование двоичного числа в десятичное

Создайте программу, которая запрашивает у пользователя двоичное число и выводит его в десятичном представлении.

I.1.17. Преобразование десятичного числа в восьмеричное

Создайте программу, которая запрашивает у пользователя целое число и выводит его в восьмеричном представлении.

I.1.18. Преобразование восьмеричного числа в десятичное

Создайте программу, которая запрашивает у пользователя восьмеричное число и выводит его в десятичном представлении.

I.1.19. Преобразование десятичного числа в шестнадцатеричное

Создайте программу, которая запрашивает у пользователя десятичное число и выводит его в шестнадцатеричном представлении.

I.1.20. Преобразование шестнадцатеричного числа в десятичное

Создайте программу, которая запрашивает у пользователя шестнадцатеричное число и выводит его в десятичном представлении.

I.1.21. Исключающее ИЛИ

В C и C++ существуют три логических оператора: `&&` (И), `||` (ИЛИ) и `!` (НЕ), логика использования которых представлена в табл. I.1.1. В этой же таблице описывается логика оператора исключающего ИЛИ — `XOR`, который не пред-