В.Л. Тарасов

Лекции по программированию на C++

Лекция 2

Числа и операторы

2.1. Целые типы

В языке C++ поддерживаются *целые типы*, перечисленные в табл.2.1.

Таблица 2.1. Целые типы

Тип	Размер, байт	Диапазон значений	
		Минимальное	Максимальное
char	1	$-2^7=-128$	$2^7 - 1 = 127$
short	2	$-2^{15}=-32768$	$2^{15} - 1 = 32767$
int, long int	4	$-2^{31} = -2147483648$	$2^{31} - 1 = 2147483647$
unsigned char	1	0	$2^8 - 1 = 255$
unsigned short	2	0	$2^{16} - 1 = 65535$
unsigned int, unsigned long	4	0	$2^{32} - 1 = 4294967295$

Размер типа `int` зависит от реализации. В Visual Studio размер `int` 4 байта.

Над целыми числами можно выполнять операции:

сложения +,	деления с отбрасыванием остатка /,
вычитания -,	нахождения остатка от деления %.
умножения *,	

Эти арифметические операции над целыми дают целый результат.

Примеры выражений с использование операций над целыми и значения этих выражений приведены в табл.2.2.

Таблица 2.2. Операции над целыми типами

Выражение	a	b	a + b	a - b	a * b	a / b	a % b
Значение	13	7	20	6	91	1	6

Целые можно *сравнивать* с помощью операторов *отношения*:

меньше	<,	больше	>,
меньше или равно	<=,	больше или равно	>=,
равно	==,	не равно	!=.

Результат сравнения является либо истинным, либо ложным. *Истинным* считается любое значение, отличное от нуля, нуль считается *ложью*. Результатом сравнения целых является целое, равное 1, если результат сравнения истинный и 0, если ложный. Примеры сравнения целых приведены в табл.2.3.

Таблица 2.3. Сравнение целых

Выражение	a	b	a < b	a <= b	a > b	a >= b	a == b	a != b
Значение	13	7	0	0	1	1	0	1

Обратите внимание на обозначение оператора *равно* с помощью двух знаков: ==. Одним знаком = обозначается *оператор присваивания*. Это важно запомнить, так как использование оператора присваивания = вместо оператора сравнения == часто допускаемая ошибка.

Целые константы

Целые константы могут записываться в *десятичной, восьмеричной* и *шестнадцатеричной* системах счисления.

Десятичные константы записываются с помощью цифр от 0 до 9 и могут иметь знак, например, 123, -15, +9, -100.

Восьмеричные константы могут иметь знак, начинаются с цифры нуль (0) и должны включать только восьмеричные цифры от 0 до 7, например, 0123, -015, но запись +09 или 09 является ошибкой, так как в восьмеричном числе использована недопустимая цифра 9.

Шестнадцатеричные константы могут иметь знак, начинаются с приставок 0x или 0X (цифра нуль и латинская буква “икс”). В их записи можно использовать, кроме обычных цифр от 0 до 9 и латинские буквы a, b, c, d, e, f или A, B, C, D, E, F, имеющие, соответственно, значения 10, 11, 12, 13, 14, 15, например, 0xA (это 10), 0Xf (это 15), 0x41(это 65). В скобках указано числовое значение в десятичной системе счисления. Справедливо равенство: $65_{10} = 0x41_{16} = 0101_{8}$. Здесь индексы для наглядности обозначают основание системы счисления.

Программа 2.1. Операции над целыми

Программа вычисляет выражения, приведенные в таблицах 2.2 и 2.3. При определении переменных a и b им сразу задается начальное

значение. Это называется *инициализация*. Внешне инициализация похожа на присваивание, так как используется тот же знак =, но инициализация выполняется на *этапе компиляции*, а присваивание происходит на *этапе выполнения* программы.

```
// Файл OperOnInt.cpp

// Результаты операций над целыми
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
// Определение переменных
   int a = 13, b = 7,            // Числа
   sum,                          // Сумма
   difference,                   // Разность
   prod,                         // Произведение
   quotient,                     // Частное
   remainder;                    // Остаток
   cout << "a = " <<  a << ", b = " << b << endl; // Вывод a и b
   sum = a + b;                                    // Вычисление суммы
   cout << "a + b = " <<  sum << endl;             // Вывод суммы
   difference = a - b;                             // Вычисление разности
   cout << "a - b = " << difference << endl;       // Вывод разности
   prod = a * b;                                   // Произведение
   cout << "a * b = " << prod<< endl;              // Вывод произведения
   quotient = a / b;                               // Вычисление частного
   cout << "a / b = " << quotient << endl;         // Вывод частного
   remainder = a % b;                              // Вычисление остатка
   cout << "a % b = " << remainder << endl;        // Вывод остатка
   cout << "(a < b) = "  << (a < b)  << endl;      // Вычисление и
   cout << "(a <= b) = " << (a <= b) << endl;      // вывод
   cout << "(a > b) = "  << (a > b)  << endl;      // результатов
   cout << "(a >= b) = " << (a >= b) << endl;      // сравнения
   cout << "(a == b) = " << (a == b) << endl;
   cout << "(a != b) = " << (a != b) << endl;
   system("pause");                                // Ждем нажатия клавиши
   return 0;
}
```

Для вывода результатов и пояснений к ним используются цепочки операторов <<, которые обеспечивают последовательный вывод своих операндов. Переменная `endl` объявлена в заголовочном файле `iostream`. Ее вывод приводит к переводу курсора в начало новой строки экрана.

Результаты арифметических действий присваиваются специальным переменным, значения которых затем печатаются. Результаты сравнения не запоминаются в каких-либо переменных, а сразу после вычисления выводятся.

Результаты, выдаваемые программой, совпадают со значениями выражений из таблиц 2.2 и 2.3:

```
a = 13, b = 7
a + b = 20
a - b = 6
a * b = 91
a / b = 1
a % b = 6
(a < b) = 0
(a <= b) = 0
(a > b) = 1
(a >= b) = 1
(a == b) = 0
(a != b) = 1
```

Чтобы задержать переход от экрана вывода в среду разработки, в конце программы вызывается функция `cin.get()`, которая ждет нажатия клавиши `Enter`.

2.2. Числа с плавающей точкой

Числа с плавающей точкой имеют целую и дробную части, могут быть положительными и отрицательными. Они моделируют вещественные числа, используемые в математике. В табл.2.4 перечислены типы чисел с плавающей точкой языка C++. Для краткости рассматриваемые типы иногда называют просто *плавающими*.

Таблица 2.4. Типы чисел с плавающей точкой

Тип	Размер, байт	Диапазон значений модуля	Точность, цифр
`float`	4	от $3.4 * 10^{-38}$ до $3.4 * 10^{+38}$	6-7
`double`	8	от $1.7 * 10^{-308}$ до $1.7 * 10^{+308}$	15-16
`long double`	10	от $3.4 * 10^{-4932}$ до $1.1 * 10^{+4932}$	19-20

Под *точностью* понимается количество значащих цифр в десятичной записи числа. Точность определяется размером памяти, выделяемой под числа и способом ее использования.

Над числами с плавающей точкой можно выполнять операции сложения (+), вычитания (-), умножения (*) и деления (/). В результате этих операций получается также число с плавающей точкой.

Числа с плавающей точкой можно сравнивать на равенство и неравенство, с помощью тех же операторов, что и целые числа (см. табл.2.3). В результате сравнения получается 1 при истинности результата и 0 при ложности.

Плавающие константы

Числовые константы для типов чисел с плавающей точкой записываются в виде:

```
[s]m.mmmE/e[+/-]pp.
```

Здесь s – знак числа, m.mmm – *мантисса*, количество цифр которой определяет *точность* числа, pp – *порядок* числа, который может быть положительным или отрицательным. Если знак отсутствует, число считается положительным. Символ E (или e) заменяет 10. Следующие константы дают три различных записи одного и того же числа:

```
123.321, 1.23321E2, 123321e-3.
```

В записи числовых констант недопустимы пробелы, так как пробелы считаются разделителями лексем.

По умолчанию считается, что числовые константы с плавающей точкой имеют тип double. Если требуется константа типа float, ее можно явно определить с помощью суффиксов f или F, например,

```
3.14159265f, 2.0F, 1.23E5F.
```

Для числовых констант типа long double используются суффиксы l или L:

```
2.718281828459045235361, 3.141592653589793238461
```

(это числа e и π соответственно).

2.3. Ввод и вывод чисел

Числовые данные выводятся в выходной поток оператором вывода <<, который становится доступным после включения в программу заголовочного файла iostream. Этот оператор может выводить данные любых стандартных типов, в том числе отдельные символы, строки. Приемником данных для выходного потока может быть стандартное устройство для вывода (экран) или дисковый файл. В файле iostream объявлена потоковая переменная cout, связанная со стандартным выходным устройством, которую надо использовать при выводе результатов на экран.

При выводе чисел с плавающей точкой можно управлять точностью, то есть количеством цифр, представляющих число. Для этого используется функция `precision()`. При вызове этой функции в виде `cout.precision()` она возвращает установленную точность. При вызове в виде `cout.precision(n)` задается число выводимых цифр `n`.

Для ввода чисел применяется оператор >> чтения из потока. Источником данных для входного потока может быть стандартное входное устройство (клавиатура) или файл на диске. При вводе чисел сначала пропускаются начальные пробелы, затем читаются символы числа. Ввод числа прекращается при поступлении пробела, табуляции или новой строки. Для ввода с клавиатуры следует использовать стандартный входной поток `cin`, объявленный в заголовочном файле `iostream`.

При вводе данных любых типов оператором >> разделителем отдельных порций данных является пробел.

Оператор ввода >> читает из потока символы, которые допускаются в записи данных для типа вводимой величины. Например, при вводе чисел с плавающей точкой допустима буква `e` как обозначение порядка, а при вводе целых чисел эта буква недопустима. При вводе с клавиатуры, набранные символы будут обрабатываться после нажатия **`Enter`**.

Программа 2.2. Точность чисел с плавающей точкой

В программе переменная `xf` типа `float` инициализируются константой с числом цифр заведомо большим, чем может разместиться в памяти, выделяемой для этой переменной. При переводе констант, записанных в десятичной системе счисления, в двоичную систему, используемую внутри компьютера, лишние двоичные цифры, не умещающиеся в разряды, отведенные для числа, отбрасываются, за счет чего возникает *ошибка усечения*. Значение переменной `yd` типа `double` вводится с клавиатуры. Далее переменные `xf` и `yd` печатаются с различной точностью.

```
//  Файл Precision.cpp

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

int main()
{
   float xf = 12345.678987654321;             // 17 цифр
   double yd;
   cout << "Input yd: ";                      // Приглашение к вводу
   cin >> yd;                                 // Ввод переменной y
   cout << "precision = " << (cout.precision());  // Точность по умолчанию
   cout << "\n xf  = " << xf << ",  yd = " << yd;
   cout.precision(4);                         // Установка точности 4 цифры
   cout << "\nprecision = " << (cout.precision());
```

```
    cout << "\n xf  = " << xf << ",  yd = " << yd;
    cout.precision(18);                          // Установка точности 18 цифр
    cout << "\nprecision = " << (cout.precision());
    cout << "\n xf  = " << xf << ",  yd = " << yd << endl;
    cout.precision(6);                 // Восстановление точности по умолчанию
    system("pause");
    return 0;
}
```

Далее приводится диалог с программой:

```
Input yd: 123456789.87654321
precision = 6
xf  = 12345.7,  yd = 1.23457e+08
precision = 4
xf  = 1.235e+04,  yd = 1.235e+08
precision = 18
xf  = 12345.6787109375,  yd = 123456789.876543224
```

Оператор << для чисел с плавающей точкой выводит по умолчанию 6 цифр, при этом происходит округление, если все цифры числа не помещаются в отведенное число позиций. В зависимости от значения числа используется представление с фиксированной точкой или научный формат с показателем степени. Если точность превышает число цифр в числе, выводятся случайные цифры, заполняющие отведенные позиции.

Точность 18 цифр превышает точность представления чисел `float` и `double`, хотя оператор вывода «добросовестно» рисует на экране заказанные 18 цифр, но видно, что для `float` правильными являются только первые 8 цифр, а для `double` первые 16.

2.4. Логический тип и логические операторы

В языке Си нет логического типа, вместо него используется целый тип, причем нулевое значение принято считать ложью, а отличное от нуля значение – истиной. В C++ введен специальный логический тип, обозначаемый ключевым словом `bool`, имеющий два значения: `true` – истина и `false` – ложь.

По определению, `true` имеет значение 1 при преобразовании к целому типу, а `false` преобразуется в 0.

Целые можно преобразовывать в логические значения, при этом ненулевое значение преобразуется в `true`, а нуль – в `false`, например,

```
bool bb = 7;     // Целое 7 будет преобразовано в bool и bb будет true
int i = true;    // true будет преобразовано в 1, поэтому i будет равно 1
```

В арифметических и логических выражениях логические значения преобразуются в целые (int) и операции выполняются над преобразованными величинами. Если результат приводится обратно к логическому типу, то 0 преобразуется в false, а ненулевое значение – в true.

Логический тип можно использовать для выражения результатов логических операций, например,

```
int a, b;
…
bool amoreb = a > b;
```

Здесь amoreb будет true, если a больше b и false в противном случае.

В C++ имеются три логических оператора:

&& - логическое умножение И, || - логическое сложение ИЛИ.

! - логическое отрицание НЕ,

В табл.2.5 приводятся результаты выполнения логических операторов. Вместо true и false показаны совместимые значения 1 и 0.

Таблица 2.5. Таблица истинности логических операторов

Выражения	a	b	a && b	a || b	! a
Значения	0	0	0	0	1
	0	1	0	1	1
	1	0	0	1	0
	1	1	1	1	0

В языке C++ каждый оператор имеет приоритет, определяющий очередность его выполнения. Приоритет оператора ! выше, чем операторов && и ||, а приоритет && выше, чем у ||. В выражении

```
a || b && !c
```

сначала будет вычислено !c, затем b && !c и, наконец, логическое сложение. С помощью круглых скобок можно изменять очередность вычислений, например, в выражении

```
(a || b) && !c
```

первым будет выполняться логическое сложение в скобках.

Выражения с использованием логических операторов вычисляются только до тех пор, пока не станет известной истинность или ложность результата. Например, в следующем фрагменте программы проверяется существование треугольника со сторонами a, b, c:

```
  double a, b, c;              // Стороны треугольника
…
  if(a + b >c && a + c > b && b + c > a)
    cout << "Существует \n";
  else
```

```
cout << "Не существует \n";
```

Если окажется, что a + b < c, то первый операнд оператора && будет ложным и все выражение будет ложным, независимо от значений выражений a + c > b и b + c > a, поэтому они не будут вычисляться.

2.5. Математические функции

Стандартом языка C++ предусмотрена библиотека математических функций, заголовочный файл которой cmath (или math.h). Упомянем наиболее употребительные математические функции:
sin(x), cos(x), tan(x) – тригонометрические (x в радианах);
asin(x), acos(x), atan(x) – обратные тригонометрические;
exp(x), sinh(x), cosh(x), tanh(x) – экспонента и гиперболические;
log(x) – натуральный логарифм; log10(x) – десятичный логарифм;
sqrt(x) - вычисление квадратного корня $\sqrt{x}$, x≥0;
pow(x, y) – возведение x в степень y (x^y);
fabs(x) - абсолютная величина (модуль) |x|, x – число с плавающей точкой.

Аргументы и возвращаемые значения перечисленных функций имеют тип double.

Для вычисления абсолютной величины целых есть отдельная функция: abs(x).

В math.h определены константы M_PI для числа π и M_E для числа e с точностью 21 цифра. В Visual Studio для доступа к этим константам надо определить макрос _USE_MATH_DEFINES:

```
#define   _USE_MATH_DEFINES
#include <cmath>
using namespase std;
```

2.6. Операторы

Операторы выполняют действия, например, производят сложение двух чисел. Операторы различаются числом операндов, участвующих в соответствующей операции. Существуют *унарные*, *бинарные* операторы, а также единственный оператор с тремя операндами.

Арифметические операторы языка C++, такие как сложение, умножение, похожи на соответствующие математические действия. Но есть операторы, не имеющие прямой аналогии в математике, например, вызов функции (), доступ к элементу массива [].

Унарные операторы

В табл.2.6 перечислены имеющиеся в языке одноместные (унарные) операторы, которые применяются к единственному операнду.

Оператор ++ увеличивает свой операнд на 1, а оператор -- уменьшает на 1. Данные операторы могут быть префиксными (приставочными) и постфиксными (суффиксными). Если оператор стоит *перед* операндом, то *сначала изменяется* операнд, а затем измененное значение используется в вычислениях, например,

```
int i = 1, j = 1, n, m, k;
n = ++i;              // i = 2, n = 2
```

Если оператор стоит *после* операнда, то в вычислениях *используется начальное значение* операнда, а затем операнд изменяется, например,

```
m = j--;              // m = 1, j = 0
k = j-- -1;           // k = -1, j = -1
```

Таблица 2.6. Унарные операторы

Знак оператора	Операция	Пример выражения	Значение выражения
-	Унарный минус	-(-1)	1
+	Унарный плюс	+(-1)	-1
~	Побитовое логическое отрицание	~(0)	-1
!	Логическое отрицание	!(0)	1
sizeof	Размер объекта или типа в байтах	sizeof(char)	1
(тип)	Приведение типа	(int)1.9	1
тип()		int(1.9)	1
*	Доступ к объекту по указателю	*p	
&	Вычисление адреса	&x	адрес x
new	Выделение памяти	p = new char	
delete	Освобождение памяти	delete p	
++	Увеличение на единицу	++k; k++	
--	Уменьшение на единицу	--k; k--	

Бинарные операторы

В табл.2.7 перечислены двухместные (бинарные) операторы, требующие двух операндов. Операторы арифметические, логические и сравнения уже обсуждались. Побитовые операторы применимы к целым типам и действуют на отдельные разряды их двоичного представления, подробнее они будут рассмотрены ниже.

Таблица 2.7. Бинарные операторы

Знак оператора	Операция	Пример выражения	Значение выражения
*	Умножение	(-1) * (-1)	1
/	Деление	3 / 2	1
		3.0 / 2.0	1.5
%	Вычисление остатка	3 % 2	1
+	Сложение	3 + 2	5
-	Вычитание	3 - 2	1
<	Меньше	3 < 2	0
>	Больше	3 > 2	1
<=	Меньше или равно	3 <= 2	0
>=	Больше или равно	3 >= 2	1
==	Тождество	3 == 2	0
!=	Не равно	3 != 2	1
<<	Сдвиг влево	3 << 2	12
>>	Сдвиг вправо	7 >> 2	1
&	Побитовое И	3 & 2	2
|	Побитовое ИЛИ	3 | 2	3
^	Побитовое исключающее ИЛИ	3 ^ 2	1
&&	Логическое И	3 && 2	1
||	Логическое ИЛИ	3 || 2	1
,	Последовательное вычисление	(1, 2, 2+0.5)	2.5

Оператор запятая

Рассмотрим здесь оператор *запятая*. Несколько выражений, разделенных запятыми, вычисляются слева направо и рассматриваются как одно выражение. Типом и значением результата является тип и значение правого выражения. Например,

```
cout << (1 + 2, 5 % 6, 3.14 + 6 - 4); // Будет напечатано 5.14
```

Запятые, разделяющие аргументы функций и переменные в описаниях, операторами *не являются* и не обеспечивают вычислений слева направо, например,

```
double y = 1.;
double x = pow(1 + exp(2) + y, y = 3 + log(4));
```

Заранее нельзя сказать какое из выражений: `1 + exp(2) + y` или `y = 3 + log(4)` будет вычислено первым, а от этого зависит значение `x`.

Условное выражение

Оператор условного выражения – единственный, требующий трех операндов. Для его обозначения используются знак вопроса и двоеточие (`? :`). Условное выражение имеет вид:

```
a ? b : c;
```

Если a есть истина (не нуль), то результатом всего выражения будет значение выражения b, иначе результатом будет значение выражения c. Например, в следующей инструкции переменной max присваивается максимум из x и y:

```
max = x > y ? x : y;
```

Операторы присваивания

В языке C++, кроме обычного оператора присваивания, обозначаемого одним знаком =, существуют операторы присваивания, совмещенные с основными операциями, табл.2.8. Они позволяют писать инструкции вида:

```
a = a + b;
```

в более краткой форме:

```
a += b;
```

Таблица 2.8. Операторы присваивания

Знак оператора	Выполняемое действие	Примеры выражений	Значение k
=	Простое присваивание	k = 2	2
+=	Сложение и присваивание	k += 2	4
-=	Вычитание и присваивание	k -= 2	2
*=	Умножение и присваивание	k *= 3	6
/=	Деление и присваивание	k /= 2	3
%=	Вычисление остатка и присваивание	k %= 2	1
<<=	Сдвиг влево и присваивание	k <<= 2	4
>>=	Сдвиг вправо и присваивание	k >>= 2	1
&=	Побитовое И и присваивание	k &= 2	0
|=	Побитовое ИЛИ и присваивание	k |= 2	2
^=	Побитовое исключающее ИЛИ и присваивание	k ^= 2	0

Приоритеты операторов

В языке C++ операторы выполняются в очередности, определяемой их приоритетами. Всего существует 16 приоритетов операторов. В табл.2.9 перечислены операторы в порядке убывания их приоритетов.

При вычислении выражений сначала выполняются операторы, заключенные в самые внутренние круглые скобки. Если скобок нет, то сначала выполняются операторы с более высоким приоритетом. В случае одинакового приоритета операторы выполняются либо слева направо, либо справа налево. Например, арифметические операторы выполняются *слева направо*. В инструкция

```
a = b + c + d + e;
```

сложения будут выполняться слева направо по умолчанию. Нужный порядок выполнения операторов можно задать явно с помощью скобок:

```
a = b + (c + (d + e) );
```

Операторы присваивания выполняются *справа налево*, поэтому инструкция:

```
x = y = z = 10;
```

присвоит всем трем переменным одно значение 10.

Таблица 2.9. Приоритеты и порядок выполнения операторов

Приоритет	Знаки операторов	Тип операторов	Порядок выполнения
1	`()  []  .  ->`	Вызов функции, выбор элемента массива, доступ к члену структуры	Слева направо
2	`-  ~  !  *  &  ++ -- sizeof  (тип)  тип()`	Унарные с одним операндом	Справа налево
3	`.*  ->*`	Доступ к члену класса через указатель	Слева направо
4	`*  /  %`	Умножение, деление, остаток от деления	Слева направо
5	`+  -`	Сложение, вычитание	Слева направо
6	`<<  >>`	Сдвиг влево и вправо	Слева направо
7	`<  >  <=  >=`	Отношение (неравенство)	Слева направо
8	`==  !=`	Равно, не равно	Слева направо
9	`&`	Побитовое И	Слева направо
10	`^`	Побитовое исключающее ИЛИ	Слева направо
11	`|`	Побитовое ИЛИ	Слева направо
12	`&&`	Логическое И	Слева направо
13	`||`	Логическое ИЛИ	Слева направо
14	`? :`	Условная	Справа налево
15	`=  *=  /=  %=  +=  -= <<=  >>=  &=  |=  ^=`	Простое и составное присваивание	Справа налево
16	`,`	Последовательное вычисление	Слева направо

2.7. Ключевые слова

В табл. 2.10 перечислены ключевые слова языка Си, предусмотренные стандартом 1989 г., которые являются в то же время и ключевыми словами языка C++. Эти идентификаторы имеют предопределенный смысл и не могут использоваться для обозначения объектов, определяемых пользователем.

Таблица 2.10. Ключевые слова стандарта языка Си

Ключевое слово	Значение, использование
auto	Устанавливает автоматический класс памяти для объектов
break	Оператор выхода из цикла или переключателя switch
case	Метка в операторе switch
char	Спецификатор символьного типа
const	Модификатор типа. Запрещает изменение объекта
continue	Оператор перехода к следующей итерации цикла
default	Метка в операторе switch
do	Первое слово цикла do ... while
double	Спецификатор вещественного типа двойной точности
else	Необязательная ветвь в операторе if
enum	Спецификатор перечислимого типа
extern	Класс памяти, указывающий, что идентификатор определяется позже или в другом файле
float	Спецификатор вещественного типа одинарной точности
for	Оператор цикла
goto	Оператор безусловного перехода
if	Оператор выбора
int	Спецификатор целого типа
long	Спецификатор типа длинных целых. Используется также как префикс с int, float, double
register	Указание разместить величину в регистрах процессора
return	Оператор возврата в вызывающую функцию
short	Спецификатор типа коротких целых
signed	Указывает на наличие знака у целочисленных типов
sizeof	Оператор. Определяет размер операнда в байтах
static	Класс памяти статических объектов
struct	Спецификатор типа структура
switch	Оператор выбора
typedef	Используется для создания синонима типа
union	Спецификатор типа объединение
unsigned	Префикс данных целочисленного типа без знака
void	Тип выражения, не имеющего значения
volatile	Модификатор типа объектов, которые могут быть прочитаны или записаны какой-либо другой программой
while	Оператор цикла

В язык C++ введены дополнительные ключевые слова, с помощью которых реализуется концепция объектно-ориентированного программирования, обработка исключений (ошибок) и другие возможности. В табл. 2.11 приведены ключевые слова, предусмотренные стандартом языка C++, принятом в 1998 г.

Таблица 2.11. Ключевые слова стандарта языка C++

Ключевое слово	Значение, использование
asm	Используется для вставки в программу ассемблерного кода
bool	Спецификатор логического типа
catch	Обрабатывает исключение, порожденное оператором throw
class	Используется для объявления классов
const_cast	Переопределение модификаторов const и/или volatile
delete	Оператор освобождения памяти
dynamic_cast	Оператор динамической проверки приведения типа
explicit	Запрещает приведение типов аргументов в конструкторах
export	Разрешает использование шаблона из одного файла в другом
false	Логическая константа со значением *ложь*
friend	Разрешает функциям доступ к закрытым членам класса
inline	Требует встраивать код функции в каждую точку ее вызова
mutable	Разрешает изменять объекты, объявленные как const
namespace	Создает пространство имен
new	Оператор выделения динамической памяти
operator	Создает (перегружает) функции операторы
private	Спецификатор доступа закрытых членов класса
protected	Спецификатор доступа закрытых членов класса, доступных в производных классах
public	Спецификатор доступа открытых членов класса
reinterpret_cast	Оператор преобразования одного типа в другой
static_cast	Оператор приведения типа
template	Используется для объявления шаблонов функций и классов
this	Указатель на объект из функция-члена класса
throw	Оператор генерации исключения
true	Логическая константа со значением *истина*
try	Заголовок блока, где контролируются исключения
typeid	Оператор определения типа объекта
typename	Используется вместо class в шаблонах template
using	Открывает доступ к именам из пространства имен
virtual	Спецификатор создания виртуальных функций
wchar_t	Спецификатор типа многобайтовых символов