Тема 2

Числа и операторы

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	$2^{32} - 1 = 4294967295$

Замечание 1: Размер типа int зависит от реализации. В Visual Studio размер int 4 байта.

Замечание 2: В таблице представлены «классические» целые типы. В современных стандартах присутствуют и другие дополнительные базовые типы, например, символьные: wchar_t, char8_t, char16_t, char32_t; числовые целые: long long int, unsigned long long int – занимают по 8 байт; целочисленные типы, определяемые корпорацией Microsoft: __int16 __int32 и __int64.

Арифметические операции с целыми

- сложения +,
- вычитания –,
- умножения *,
- деления нацело (с отбрасыванием остатка) /,
- нахождения остатка от целочисленного деления %

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

Замечание: арифметические операции над целыми дают целый результат.

Операции сравнения над целыми

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

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

Замечание: Результатом сравнения целых является целое, равное **1**, если результат сравнения истинный
и **0**, если ложный

Программа 2.1. Операции над целыми

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{                                                    // Определение переменных
int a = 13, b = 7, sum, difference, prod, quotient, remainder; //  Числа a и b,
Сумма
                                                     //  Разность, Произведение, Частное,
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;   }
```

Результат выполнения

```
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
```

Целые константы

Десятичные константы записываются с помощью цифр от 0 до 9 и могут иметь знак.

Пример: 123, -15, +9, -100.

Восьмеричные константы могут иметь знак, начинаются с цифры-приставки «нуль» (0) и должны включать только восьмеричные цифры от 0 до 7.

Пример: 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. 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.

Пример: 0xA (это 10), 0Xf (это 15), 0x41(это $65 = 4*16^1+1*16^0$).

Целые константы
Вывод в разных системах счисления

Манипуляторы			
oct	Вывод величин в восьмеричной системе счисления	**int value = 64; cout << oct << value << endl;**	**100_8**
dec	Вывод величин в десятичной системе счисления (принято по умолчанию)	**int value = 64; cout << dec << value << endl;**	**64_{10}**
hex	Вывод величин в шестнадцатеричной системе счисления	**int value = 64; cout << hex << value << endl;**	**40_{16}**

Числа с плавающей точкой

Числа с плавающей точкой имеют целую и дробную части, могут быть положительными и отрицательными. Они моделируют вещест-венные числа, используемые в математике.

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

Замечание: у Microsoft представление long double и double идентично (**по 8** байт)

Под *точностью* в таблице понимается количество значащих цифр в десятичной записи числа.

Над числами с плавающей точкой можно выполнять операции сложения (+), вычитания (–), умножения (*) и нецелочисленного деления (/).

В результате этих операций получается также число с плавающей точкой.

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.

123.321=1.23321E2=123321e-3.

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,

Вывод чисел

При включения заголовочного файла **iostream** становится доступной потоковая переменная **std::cout**, связанная со стандартным выходным устройством **stdin**.

std::cout надо использовать при выводе результатов в консольное окно.

Числовые данные выводятся в **выходной поток** перегруженным оператором вывода **<<**.

precision() – функция-метод для управления точностью.

cout.precision() – возвращает установленную точность.
cout.precision(n) – задается число выводимых цифр n.

Ввод чисел

При включения заголовочного файла **iostream** становится доступной потоковая переменная **std::cin**, связанная со стандартным входным устройством **stdout**.

std::cin надо использовать при вводе значений с консоли (клавиатурный ввод).

Числовые данные вводятся из входного потока перегруженным оператором ввода **>>**.

При вводе чисел сначала пропускаются начальные пробелы, затем читаются символы числа. Ввод числа прекращается при поступлении пробела, табуляции или символа завершения строки. При вводе данных любых типов оператором **>>** разделителем отдельных порций данных является пробел. При вводе с клавиатуры, набранные символы будут обрабатываться после нажатия **Enter**.

Программа Точность чисел с плавающей точкой

```
#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 yd
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; }
```

Логический тип

В C++ введен специальный логический тип, **bool**, имеющий два значения: true – истина и false – ложь.

Замечание: в языке Си нет встроенного логического типа.

По определению C++, 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.

Пример

bool bb = 7; // 7 преобразуется в bool и bb будет true

int i = true; // true будет преобразуется в 1, и i равно 1

Логический тип

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.

Пример

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

Здесь a_more_b будет true, если a больше b и false в

Логические операторы

&& - логическое умножение И,

! - логическое отрицание НЕ,

|| - логическое сложение ИЛИ.

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

Приоритет операций

1)! - логическое отрицание НЕ,

2) && - логическое умножение И,

3) || - логическое сложение ИЛИ.

Логические операторы

a || b && !c

Сначала будет вычислено !c, затем b && !c и потом логическое сложение в логическое сложение.

(a || b) && !c

Первым будет выполняться логическое сложение в скобках (приоритет скобок).

Замечание: Выражения с использованием логических операторов вычисляются только до тех пор, пока не станет известной истинность или ложность результата.

Проверка существования треугольника по сторонам :

```
double a, b, c; // Стороны треугольника
```

Операции сравнения в C++ имеют более низкий приоритет по сравнению с любых логических операций!

Математические функции

Библиотека математических функций: cmath (или math.h)

Функция	Прототип на C++
cos(x)	double cos(double x)
sin(x)	double sin(double x)
arccos(x)	double acos(double x)
arcsin(x)	double asin(double x)
tg(x)	double tan(double x)
arctg(x)	double atan(double x)
Гиперболический синус	double cosh(double x)
Гиперболический косинус	double sinh(double x)
Гиперболический тангенс	double tanh(double x)

e^x	double exp (double x)
ln(x)	double log (double x)
lg(x)	double log10 (double x)
x^y	double pow (double x, double y)
$\sqrt{x}$	double sqrt (double x)
Вычисляет ближайшее целое, не меньше чем аргумент x	double ceil (double x)
Находит наибольшее целое, не превышающее значение x	double floor (double x)
$\lvert x \rvert$	double fabs (double x)

В **math.h** определены константы **M_PI** для числа

```
#define _USE_MATH_DEFINES
#include <math.h>
```

Математические функции

Примеры:

$$\frac{|x|-|y|}{1+|x\cdot y|}$$

На C++: (fabs(x)- fabs(y))/(1+ fabs(x*y))

$$\frac{\sqrt{|x+1|}-\sqrt[3]{|y|}}{1+\frac{x^2}{2}+\frac{y^2}{4}}$$

Неверно на C++:

(sqrt(fabs(x+1))–pow(fabs(y),1/3))/(1+pow(x, 2.0)/2+pow(y, 2.0)/4)

Верно на C++:

(sqrt(fabs(x+1))–pow(fabs(y),1.0/3))/(1+pow(x, 2.0)/2+pow(y, 2.0)/4)

Математические функции

Треугольник образован тремя вершинами, координаты которого задаются с клавиатуры. Найти площадь и периметр треугольника.

Длина отрезка между двумя точками: $l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Формула Герона $S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$, где p – полупериметр; a,b,c – длины сторон

```
#include “pch.h"
#include <iostream>
#include <cstdlib>
#include <locale>
#include <cmath>
using namespace std;
```

```
int  main()
{   setlocale(LC_ALL, "Russian");
double x1, y1, x2, y2, x3, y3,    // Координаты вершин
a1, a2, a3,                       // Длины сторон
p, s;     // Полупериметр и площадь
  cout<<"Введите координаты вершин"<<endl;
  cin>>x1>>y1>>x2>>y2>>x3>>y3; // ввод переменных
координат
a1=sqrt(pow((x2-x1),2)+ pow((y2-y1),2));    // вычисление
a2=sqrt(pow((x3-x1),2)+ pow((y3-y1),2));    // длин
a3=sqrt(pow((x2-x3),2)+ pow((y2-y3),2));    // сторон треугольника
p=(a1+a2+a3)/2;                           // полупериметр
S=sqrt(p*(p-a1)*(p-a2)*(p-a3));   // площадь
cout<<"Периметр = "<<2*p<<endl<<"Площадь = "<< S <<endl;
system("pause");
return 0;    }
```

Операторы

Операторы выполняют действия над данными, например, производят сложение двух чисел, выводят значения и т.д.

Операторы различаются числом операндов, участвующих в соответствующей операции. Существуют *унарные, бинарные* операторы и один *тернарный* оператор.

Примеры операторов:

Арифметические операторы : +, –,*, /, %.

Оператор вызова функции: ().

Оператор доступа к элементу массива: [].

Унарные операторы

Одноместные (унарные) операторы – операторы, которые применяются к единственному операнду.

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--	

Операторы инкремента и декремента.

Предназначены для увеличения и уменьшения переменных. Оператор инкремента ++ добавляет 1 к своему операнду, а оператор декремента – вычитает 1.

Их можно использовать и *префиксно* (помещается перед переменной: ++i), и *постфиксно* (помещаются после переменной: i++). В обоих случаях значение i увеличивается на 1, но выражение ++i увеличивает i до того, как его значение будет использовано в выражении, а i++ после использования в выражении.

Пример.

```
n=5;                                    n=5;
x=n++;  //Получим x=5, n=6      x=++n;  // Получим x=6, n=6
```

Замечания:

- Операторы инкремента и декремента можно применить только к переменным. Выражение типа (i+j)++ недопустимо.
- Если требуется только увеличить или уменьшить значение переменной (но не получить её значение), то безразлично, какой оператор выбирать – префиксный или постфиксный.

$$n = n + 1 = n + + = + + n$$

Задание: Вычислить значения переменных ***i*** и ***n*** после выполнения операций:

```
i=5;   n=++i+i++;   Замечание: Операция ++ приоритетное
```

Бинарные операторы

Двухместные (бинарные) операторы – операторы требующие двух операндов.

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

Несколько выражений, разделенных запятыми, вычисляются слева направо и рассматриваются как одно выражение. Типом и значением результата является тип и значение правого выражения.

Пример: int k1, k2=1;

```
cout << (k1 = k2 + 2, k2 = 5 % 6, 3.14 + 6 – 4 + k1 + k2);
                    // Будет напечатано 13.14 т.к. 13.14 = 5.14 + 3 + 5
```

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

Пример:

```
double y = 1.;
double x = pow(1 + exp(2) + y, y = 3 + log(4)); //Запятая – не оператор!
```

Замечание: заранее нельзя сказать какое из выражений: *1 + exp(2) + y* или *y = 3+ log(4)* будет вычислено первым, а от

Операторы присваивания

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

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

в более краткой форме:

Знак операции	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

Тернарный оператор «Условное выражение»

Оператор условного выражения – единственный, требующий трех

операндов.

Обозначение: (? :).

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*.

Пример:

В следующей инструкции переменной max присваивается максимум из x и y:

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

Приоритеты операторов

В языке C++ операторы выполняются в очередности, определяемой

их приоритетами. Всего существует 16 приоритетов операторов. 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.

Пример: арифметические операторы выполняются *слева направо*.

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

Замечание: нужный порядок выполнения операторов можно задать явно с помощью скобок, например:

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


Пример: операторы присваивания выполняются *справа*

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