

Переменные и операции

Лекция 2

Операции языка C++

Операции используются для выполнения действий над переменными

Атрибуты операций:

- ▶ Арность
 - ▶ унарные
 - ▶ бинарные
 - ▶ тернарные
- ▶ Приоритет
- ▶ Ассоциативность
 - ▶ Левая
 - ▶ правая

Группы операций языка C++

Арифметические	+ - * / % + -
Сравнения	< <= > >= == !=
Логические	! &&
Присваивания	=
Арифметические с присваиванием	*= /= %= += -=
Побитовые	~ & ^ << >>
Побитовые с присваиванием	&= ^= = <<= >>=
Адресные	* &
Работа с пользовательскими типами	. ->
Инкремент, декремент	++ --
Разные	? : sizeof [] () ::

Арифметические

Название	Знак	Приоритет	Арность	Ассоциативность
унарный плюс	+	высокий	унарная	правая
унарный минус	-	высокий	унарная	правая
умножение	*	средний	бинарная	левая
деление	/	средний	бинарная	левая
остаток от деления	%	средний	бинарная	левая
плюс	+	низкий	бинарная	левая
минус	-	низкий	бинарная	левая

Унарные арифметические операции

- ▶ Унарный плюс “+” - умножает operand на +1, результат записывается во временную память

```
int test = 17;  
int result = 0;  
result = +test;  
std::cout << test << std::endl << result << std::endl;
```

Результат:

17
17

- ▶ Унарный минус “-” - умножает operand на -1, результат записывается во временную память

```
int test = 17;  
int result = 0;  
result = -test;  
std::cout << test << std::endl << result << std::endl;
```

Результат:

17
-17

Умножение

Операция умножения “*” - умножает операнды друг на друга, результат записывается во временную память

```
int a = 5, b = 6;  
std::cout << a * b << std::endl;
```

Результат:
30

```
float a = 5.5, b = 3.2;  
std::cout << a * b << std::endl;
```

Результат:
17.6

```
char a = 'a', b = 'b';  
std::cout << a * b << std::endl;
```

Результат:
9506

ASCII код 'a' = 97
'b' = 98

Деление

Операция деления “/” - делит операнды друг на друга, результат записывается во временную память.

```
int a = 30, b = 6;  
std::cout << a / b << std::endl;
```

Результат:
5

```
float a = 5.5, b = 3.2;  
std::cout << a / b << std::endl;
```

Результат:
1.71875

Деление

При делении целого числа на “0” выбрасывается исключение

```
int a = 30, b = 0;  
std::cout << a / b << std::endl;
```

Результат:

Сигнатура проблемы:
Имя события проблемы: APPCRASH
Имя приложения: test.exe
Версия приложения: 0.0.0.0
Отметка времени приложения: 5b0d135c
Имя модуля с ошибкой: test.exe
Версия модуля с ошибкой: 0.0.0.0
Отметка времени модуля с ошибкой: 5b0d135c
Код исключения: c0000094
Смещение исключения: 00011dc7
Версия ОС: 6.1.7601.2.1.0.768.2
Код языка: 1049
Дополнительные сведения 1: 0a9e
Дополнительные сведения 2: 0a9e372d3b4ad19135b953a78882e789
Дополнительные сведения 3: 0a9e
Дополнительные сведения 4: 0a9e372d3b4ad19135b953a78882e789

Деление

При делении вещественного числа на “0” получается бесконечность (INFINITY)

```
float a = 30, b = 0;  
std::cout << a / b << std::endl;
```

Результат: **inf**

```
float a = -30, b = 0;  
std::cout << a / b << std::endl;
```

Результат: **-inf**

Деление

При делении целого числа на целое число всегда получается целое число

```
int a = 3, b = 4;  
std::cout << a / b << std::endl;
```

Результат: 0

```
int a = 4, b = 3;  
std::cout << a / b << std::endl;
```

Результат: 1

Операция явного приведения типа

- ▶ Унарная операция, выглядит как “(<целевой_тип>)”
- ▶ Создает **временную копию** операнда, приводя его к целевому типу
- ▶ При приведении вещественных чисел к целым дробная часть «теряется»
- ▶ **Рекомендуется всегда использовать операцию явного приведения типов.** Этим программист показывает, что ему известно о различии типов, что смешение сделано намеренно

Операция явного приведения типа при делении

Для получения правильного результата при делении целых чисел необходимо выполнить операцию явного приведения типа

```
int a = 3, b = 4;  
std::cout << (float) a / b << std::endl;
```

Результат:

0.75

```
int a = 4, b = 3;  
std::cout << (float) a / b << std::endl;
```

Результат:

1.33333

Остаток от деления

- ▶ Определена только для целых чисел
- ▶ Операция получения остатка от деления “%” - вычисляет остаток от деления операндов друг на друга, результат записывается во временную память.
- ▶ Операция определена следующим образом:

$$a = (a/b) * b + a \% b$$

Остаток от деления

```
int a = 30, b = 7;  
std::cout << a % b << std::endl;
```

Результат:

2

```
int a = -30, b = 7;  
std::cout << a % b << std::endl;
```

Результат:

-2

```
int a = 30, b = -7;  
std::cout << a % b << std::endl;
```

Результат:

2

```
int a = -30, b = -7;  
std::cout << a % b << std::endl;
```

Результат:

-2

Задача: вычислить объем сферы

```
#include <iostream>
```

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

```
int main()  
{  
    float radius;  
  
    std::cout << "Enter a radius " << std::endl;  
  
    std::cin >> radius;  
  
    std::cout << "Volume is: " << (float)4 / 3 * M_PI * pow(radius, 3)  
        << std::endl;  
  
    return 0;  
}
```

ПЯВУ. Лекция 2

Эти строчки нужны для определения числа пи

Включение библиотеки математических функций

Функция возведения в степень

Выражение, рассчитывающее объем сферы

Логические операции

Название	Знак	Приоритет	Арность	Ассоциативность
отрицание	!	высокий	унарная	правая
логическое И	&&	низкий	бинарная	гарантируется выполнение слева направо
логическое ИЛИ		еще ниже	бинарная	гарантируется выполнение слева направо

Логические операции

- ▶ Не изменяют своих operandов
- ▶ Результатом операций будет ИСТИНА или ЛОЖЬ
- ▶ ЛОЖЬ в C++ - это “0”
- ▶ ИСТИНА в C++ - все, что не “0”
- ▶ Обычно используются в условных выражениях и циклах
- ▶ Выполняются строго слева направо, если становится известен результат операции, выполнение операции прекращается

Логическое отрицание

Таблица истинности:

a	! a
0	1
1	0

```
#include <iostream>

int main()
{
    float a = 5.6;
    int b = 34;
    bool c = false;
    std::cout << !a          << std::endl;
    std::cout << !(b-b)      << std::endl;
    std::cout << !((b-b)*a)  << std::endl;
    std::cout << !(b-b)*a   << std::endl;
    std::cout << !c          << std::endl;
    return 0;
}
```

Результат:

```
0
1
1
5.6
1
```

Логическое И

Таблица истинности:

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

```
float a = 5.6, b = 3.4, c = 0;  
int d = 3, e = 7, f = 0;  
(a - a) && (c = b);  
std::cout << c << std::endl;  
(c = b) && (a - a);  
std::cout << c << std::endl;  
(e * f) && (f = d);  
std::cout << f << std::endl;  
(f = d) && (d = e * f);  
std::cout << f << std::endl;  
std::cout << d << std::endl;
```

Результат:

```
0  
3.4  
0  
3  
21
```

Логическое ИЛИ

Таблица истинности:

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

```
float a = 5.6, b = 3.4, c = 0;
int d = 3, e = 7, f = 0;
a || (c = b);
std::cout << c << std::endl;
(c = b) || (a - a);
std::cout << c << std::endl;
e || (f = d);
std::cout << f << std::endl;
(f = d) || (d = e * f);
std::cout << f << std::endl;
std::cout << d << std::endl;
```

Результат:

```
0
3.4
0
3
3
```

Сложные логические выражения

Результат зависит от операции с более низким приоритетом, поэтому операция с более высоким приоритетом может не выполниться

```
float a = 5.6, b = 3.4, c = 0;  
int d = 3, e = 7, f = 0;  
d || (f = e && a);  
std::cout << f << std::endl;  
(f = e && a) || (c = b);  
std::cout << f << std::endl;  
std::cout << c << std::endl;  
((f = e) && a) || (c = b);  
std::cout << f << std::endl;  
std::cout << c << std::endl;  
(f && (a-a)) || (c = b);  
std::cout << c << std::endl;
```

Результат:

0
1
0
7
0
3.4

Операции сравнения

Название	Знак	Приоритет	Арность	Ассоциативность
меньше	<	средний	бинарная	левая
меньше равно	<=	средний	бинарная	левая
больше	>	средний	бинарная	левая
больше равно	>=	средний	бинарная	левая
равно	==	низкий	бинарная	левая
не равно	!=	низкий	бинарная	левая

Операции сравнения

- ▶ Не изменяют своих операндов
- ▶ Результатом операций будет ИСТИНА или ЛОЖЬ
- ▶ Обычно используются в условных выражениях и циклах

Пример неправильного использования операции сравнения

```
int x = 40;
if ( 30 < x < 50 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

TRUE

```
int x = 40;
if ( 45 < x < 60 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

TRUE

```
int x = -20;
if ( -30 < x < -10 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

FALSE

```
int x = -20;
if ( -10 < x < 1 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

TRUE

Исправленный пример использования операций сравнения

```
int x = 40;
if ( 30 < x && x < 50 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

TRUE

```
int x = 40;
if ( 45 < x && x < 60 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

FALSE

```
int x = -20;
if ( -30 < x && x < -10 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

TRUE

```
int x = -20;
if ( -10 < x && x < 1 )
    std::cout << "TRUE" << std::endl;
else
    std::cout << "FALSE" << std::endl;
```

Результат:

FALSE

Операция равно “==”

- ▶ Легко перепутать с операцией присваивания
- ▶ Если сравнивается выражение и константа, то константу рекомендуется ставить слева

```
const int TEST = 5;  
int a = 3;  
std::cout << (a == TEST) << std::endl;  
std::cout << (a = TEST) << std::endl;
```

Легко перепутать = и ==

Результат:

```
0  
5
```

```
const int TEST = 5;  
int a = 3;  
std::cout << (TEST == a) << std::endl;  
std::cout << (TEST = a) << std::endl;
```

Нельзя перепутать = и ==

Результат:

Синтаксическая
ошибка

Побитовые операции

Название	Знак	Приоритет	Арность	Ассоциативность
Побитовое логическое отрицание	\sim	высокий	унарная	правая
Побитовое логическое И	$\&$	средний	бинарная	левая
Побитовое логическое исключающее ИЛИ	\wedge	чуть ниже	бинарная	левая
Побитовое логическое ИЛИ	$ $	еще ниже	бинарная	левая
Побитовый сдвиг влево	$<<$	средний	бинарная	левая
Побитовый сдвиг вправо	$>>$	средний	бинарная	левая

Побитовые операции

- ▶ Выполняются только над целыми числами
- ▶ Не изменяют своих операндов
- ▶ Результатом операций будет целое число
- ▶ Выполняются над каждым битом числа по отдельности
- ▶ Чтобы узнать результат операции нужно перевести число в двоичную систему счисления

Побитовое логическое И

Таблица истинности

(для каждого бита):

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Используется при
наложении маски на
число для получения
значения отдельного бита

```
char a = 10, b = 7;  
std::cout << ( a & b ) << std::endl;
```

Результат:

2

a	00001010
b	00000111
a & b	00000010

Побитовое логическое ИЛИ

Таблица истинности

(для каждого бита):

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

Используется при
объединении флагов для
передачи в функцию в
качестве одного параметра

```
char a = 10, b = 7;  
std::cout << ( a | b ) << std::endl;
```

Результат:

15

a	00001010
b	00000111
a b	00001111

Побитовое логическое исключающее ИЛИ

Таблица истинности

(для каждого бита):

a	b	$a \wedge b$
0	0	0
0	1	1
1	0	1
1	1	0

Может использоваться в
алгоритмах симметричного
шифрования

```
char a = 10, b = 7;  
std::cout << ( a ^ b ) << std::endl;
```

Результат:

13

a	00001010
b	00000111
$a \wedge b$	00001101

Побитовое логическое отрицание или дополнение до единицы

Таблица истинности
(для каждого бита):

a	$\sim a$
0	1
1	0

Если отрицается
положительное число, то
получается отрицательное,
по модулю на один больше.
И наоборот.

```
char a = 10, b = -7;  
std::cout << ~a << std::endl;  
std::cout << ~b << std::endl;
```

Результат:

```
-11  
6
```

a	00001010	b	11111001
$\sim a$	11110101	$\sim b$	00000110

Получение отрицательных чисел

$$\begin{array}{r} + \ 0000001 \\ - 1111111 \\ \hline 100000000 \end{array}$$

Отсекаем
«лишнюю»
единицу

$$\begin{array}{r} - 100000000 \\ - 11111111 \\ \hline 00000001 \end{array}$$

Добавляем
единицу – до
этого мы ее
отsekли

$$\begin{array}{r} - 100000000 \\ - 00000101 \\ \hline 11111011 \end{array}$$

Побитовый сдвиг влево

- Эквивалентен умножению числа на соответствующую степень двойки
- Освободившиеся справа разряды заполняются нулями

```
int a = 17, b = 6;  
std::cout << (a << b) << std::endl;  
std::cout << (b << a) << std::endl;  
std::cout << (a << 3) << std::endl;  
std::cout << (b << 3) << std::endl;
```

Результат:

```
1088  
786432  
136  
48
```

a	b	Результат
17	6	$17 * 2^6$
17	6	$6 * 2^{17}$
17		$17 * 2^3$
	6	$6 * 2^3$

Побитовый сдвиг вправо

- ▶ Эквивалентен делению числа на соответствующую степень двойки
- ▶ Освободившиеся слева разряды заполняются нулями, если сдвигается положительное число
- ▶ Освободившиеся слева разряды заполняются единицами, если сдвигается отрицательное число

```
int a = 17, b = 6;  
std::cout << (a >> b) << std::endl;  
std::cout << (b >> a) << std::endl;  
std::cout << (a >> 3) << std::endl;  
std::cout << (b >> 3) << std::endl;
```

Результат:

```
0  
0  
2  
0
```

```
int a = -17, b = 6;  
std::cout << (a >> b) << std::endl;  
std::cout << (b >> a) << std::endl;  
std::cout << (a >> 3) << std::endl;  
std::cout << (b >> 3) << std::endl;
```

Результат:

```
-1  
0  
-3  
0
```

Операция присваивания

Название	Знак	Приоритет	Арность	Ассоциативность
присваивание	=	самый низкий	бинарная	гарантируется выполнение справа налево

Изменяет значение своего левого операнда

L-value и R-value

	L-value	R-value
Что такое	может стоять слева от знака присваивания	может стоять справа от знака присваивания
Требования	<ul style="list-style-type: none">• не может быть константой• должно иметь адрес	должно иметь значение
Чем может быть	<ul style="list-style-type: none">• Переменная• Функция• Указатель• Объект• Указатель на функцию• Указатель на массив	<ul style="list-style-type: none">• Переменная• Функция• Указатель• Объект• Указатель на функцию• Указатель на массив• Константа• Имя массива• Выражение

Арифметические с присваиванием

Знак	Приоритет	Арность	Ассоциативность
<code>*=</code>	самый низкий	бинарная	правая
<code>/=</code>	самый низкий	бинарная	правая
<code>%=</code>	самый низкий	бинарная	правая
<code>+=</code>	самый низкий	бинарная	правая
<code>--=</code>	самый низкий	бинарная	правая

```
int myVeryVeryVeryVeryVeryLongNameVariable = 5, b = 7;  
myVeryVeryVeryVeryVeryLongNameVariable =  
myVeryVeryVeryVeryVeryLongNameVariable + b;
```

```
int myVeryVeryVeryVeryVeryVeryLongNameVariable = 5, b = 7;  
myVeryVeryVeryVeryVeryVeryLongNameVariable += b;
```

Получение суммы цифр трехзначного числа

```
#include <iostream>

int main()
{
    int number = 0, sum = 0;

    std::cout << "Enter number:" << std::endl;
    std::cin >> number;

    sum = number / 100; //получение старшей цифры числа (234/100=2)
    sum += ( number % 100 ) / 10; //средняя цифра числа
    sum += number % 10; //младшая (правая) цифра числа
    std::cout << "Sum of digits is " << sum << std::endl;

    return 0;
}
```

Побитовые с присваиванием

Знак	Приоритет	Арность	Ассоциативность
<code>&=</code>	самый низкий	бинарная	левая
<code> =</code>	самый низкий	бинарная	левая
<code>^=</code>	самый низкий	бинарная	левая
<code><<=</code>	самый низкий	бинарная	левая
<code>>>=</code>	самый низкий	бинарная	левая

Обмен значениями двух целочисленных переменных

```
#include <iostream>

int main()
{
    int a = 10, b = 7;
    std::cout << a << "\t" << b << std::endl; // 10  7
    a ^= b; ^= a; ^= b;
    std::cout << a << "\t" << b << std::endl; // 7   10
    return 0;
}
```

Операции инкремента / декремента

Название	Знак	Приоритет	Арность	Ассоциативность
инкремент	<code>++</code>	высокий	унарная	правая
декремент	<code>--</code>	высокий	унарная	правая

<code>++i</code>	<code>i++</code>	<code>i = i + 1</code>	<code>i += 1</code>
<code>--i</code>	<code>i--</code>	<code>i = i - 1</code>	<code>i -= 1</code>

Формы унарных операций

- ▶ Все унарные операции существуют в префиксной форме (знак операции записывается перед операндом)
- ▶ Операции инкремента / декремента существуют в двух формах:
 - ▶ префиксной:
`++i;`
 - ▶ постфиксной:
`i++;`

Отличия префиксной и постфиксной форм

	Префиксная форма	Постфиксная форма
Алгоритм	<ul style="list-style-type: none">Увеличивает значение операндаВозвращает новое значение	<ul style="list-style-type: none">Создает временную копию текущего значенияУвеличивает значениеВозвращает временную копию
Скорость	выше	ниже
Является l-value	да	нет

Пример работы с оператором инкремента

```
int i = 5;  
std::cout << ++i << std::endl;
```

Результат:

6

```
int i = 5;  
std::cout << i++ << std::endl;  
std::cout << i << std::endl;
```

Результат:

5
6

```
int i = 5;  
std::cout << ( ++i = i++ ) << std::endl;  
std::cout << i << std::endl;
```

Результат:

6
7

```
int i = 5;  
std::cout << ++i + i++ << std::endl;  
std::cout << i << std::endl;
```

Результат:

12
7

```
int i = 5;  
std::cout << ++i + ++i << std::endl;
```

Результат:

14

Условная операция

Название	Знак	Приоритет	Арность	Ассоциативность
условная	? :	низкий	тернарная	гарантируется выполнение слева направо

Поиск максимума двух чисел:

```
int a = 15, b = 8;  
std::cout << (a > b ? a : b) << std::endl;
```

Результат:
15

```
int a = 5, b = 8;  
std::cout << (a > b ? a : b) << std::endl;
```

Результат:
8

Пример использования условной операции

Условные операции можно вкладывать друг в друга

Поиск максимума трех чисел:

```
int a = 15, b = 8, c = 19;                                     Результат:  
std::cout << (a > b ? a > c? a : c : b) << std::endl; 19
```

```
int a = 15, b = 8, c = 9;                                     Результат:  
std::cout << (a > b ? a > c? a : c : b) << std::endl; 15
```

```
int a = 5, b = 18, c = 9;                                     Результат:  
std::cout << (a > b ? a > c? a : c : b) << std::endl; 18
```

Операция sizeof

Название	Знак	Приоритет	Арность	Ассоциативность
Получение размера	sizeof	высокий	унарная	правая

Позволяет получить размер типа или переменной:

```
int a = 15;  
std::cout << sizeof(int) << std::endl;
```

Результат:

4

```
long double a = 15.0;  
std::cout << sizeof(a) << std::endl;
```

Результат:

8

Приоритет операций в C++

::
-> . () []
++ -- * & ~ ! + - new delete sizeof
* / %
+ -
<< >>
< <= > >=
== !=
&
^
&&
? :
= += -= *= /= %= <<= >>= &= = ^=

Подводя итог

- ▶ Операции выполняют действия над операндами
- ▶ Существуют различные группы операций
- ▶ Операции имеют разные арность, ассоциативность и приоритет
- ▶ Для 4 операций строго определен порядок выполнения: `&&`, `||`, `=`, `?::`.
- ▶ Только операции присваивания (все виды) и операции инкремента / декремента изменяют свои операнды
- ▶ Операции инкремента / декремента существуют в двух формах

Конец