ЕН.Ф.02 – Информатика и программирование

Лекция 2. Операции и выражения в C++

Конова Елена Александровна
E_Konova@mail.ru

Операции над данными

Назначение программы – обработка данных, независимо от формата их представления.

Операция – символ или лексема, описывающая правило вычисления одного значения для встроенных типов данных.

Назначение операций – изменение значений данных.

Классификация операций может быть различной.

Одна из важных классификаций по типу возвращаемого значения.

Арифметические операции

\+ сложение;

– вычитание;

* умножение;

/ деление (для целых чисел целая часть от деления);

% остаток от деления.

Арифметические операции имеют обычный смысл, и обычный порядок выполнения.

Для целого типа данных деление имеет особенности.

См. пример.

Логические операции

> больше; // Отношения.

< меньше;

== равно;

!= не равно;

>= больше или равно;

<= меньше или равно;

&& логическое И; // Логические операции.

|| логическое ИЛИ;

! логическое НЕ.

Логического типа данных в C++ нет, логические значения представляет тип **`int`**: значение, равное 0, это «ложь», значение, отличное от 0, это «истина».

Отношения

Операции отношения связывают данные числовых типов и символы, и возвращают логическое значение.

`3 > 1` (истина !=0)

`x >= 0` (зависит от x)

`y != x` (зависит от x и от y)

Сравнение данных типа **`char`** происходит по значению кода.

`'a'<'A'` = истина

Замечание.

Операции == и != вернут правильное значение для данных **`int`** (**`char`**) или перечислимых.

Для данных вещественных типов в силу неточности представления == или != выполняются не всегда корректно.

```
float y = 1.5; // 1.4999999999  или 1.5000000001
```

Логические операции

Логические операции связывают данные логического типа (**int**) и возвращают логическое значение.

&& "И", истинно, когда оба операнда истинны.

|| "ИЛИ" , истинно, когда хотя бы один операнд истинен.

! "НЕ", инверсия исходного значения.

Построение сложных выражений

```
x>0 && y>0      // Переменные x, y одновременно
                // положительны.
x%2==0 || y%2==0 // Хотя бы одно из x, y четно.
!(x*x+y*y <= r*r) // Точка лежит вне окружности.
```

См. пример.

Поразрядные операции

<< сдвиг влево;

>> сдвиг вправо;

& поразрядное "и";

| поразрядное "или";

~ поразрядная инверсия.

Операции сдвига применяются к любому значению.

```
x >> 1    x << 2
```

Выполняется сдвиг побитно содержимого левого операнда на значение правого операнда:

```
001 << 1  =  010      1 → 2
001 << 2  =  100      1 → 4
```

Сдвиг влево равносилен *2.

Сдвиг вправо равносилен /2.

Классификация по числу операндов

1. Унарные (один операнд) + – ++ — & *.
2. Бинарные (два операнда) * / % + – и другие.
3. Тернарная – операция условия, имеет три операнда.

Синтаксис условной операции

Логическое_выражение ? Выражение1 : Выражение2;

Пример

```
int  Mod =  (i<=0) ?  -i  :  i;
```

Это аналог записи:

```
if (i<=0)
    Mod = -i;
    else
    Mod = i;
```

Операции изменения значения

1 группа – операция присваивания и ее клоны.

Символ операции – знак равно "=".

Синтаксис операции присваивания:

Имя = Выражение;

Семантика:

Вычисленное значение выражения изменяет переменную левой части.

Операции после

+= сложение с присваиванием,

–= вычитание с присваиванием,

*= умножение с присваиванием,

/= деление с присваиванием,

%= остаток от деления с присваиванием,

а также операции сдвига: <<= и >>=

Синтаксис – как и у операции присваивания:

Имя Знак= Выражение;

Семантика – как у операции присваивания.

```
x = x+5;
x+=5;
```

См. пример.

Инкремент и декремент

2 группа – операции увеличения и уменьшения на единицу.

Синтаксис

Инкремент: `x++` или `++x`

Декремент: `x--` или `--x`

Пример.

```
x = x+1;
x++;      // Применительно к целым типам.
```

Замечание. Операции ++ и – – имеют 2 формы:

- префиксная `++x;`

// Сначала увеличение x, потом следующие операции.

- постфиксная `x --;`

// Уменьшение x после всех прочих операций.

Выражения в С ++

Выражение (терм) – правило вычисления одного значения.

Формально, это несколько операндов, объединенных знаками операций.

Операндами могут быть:

имена переменных,

константы,

именованные константы,

вызовы функций,

выражения.

Механизм выражений

Важно правильно записать выражение, чтобы получить требуемое значение, потому что механизм выражений, это вычисление одного значения по определенным правилам.

Исключительно важны требования:

1) соблюдения порядка,

2) соблюдение соответствия типов.

Пример.

```
y = a+b-2.7*sin(w*x/2.0)*pow(x,0.5);
```

// Выражение записывается в одну строку.

// Итоговое значение присваивается `y`, присваивание выполняется после всего.

Общие правила построения и вычисления выражений

Поскольку все данные в С (переменные, константы и функции) имеют тип, то значение, вычисленное выражением, также имеет тип, который определяется типом операндов и операциями.

операнды	**операции**	**результат**
целые	+ * /	целый
вещественные	* / –	вещественный
числовые	> != ==	логический (`int`)
логические	&& ! ||	логический

Присваивание в С – тоже операция, является частью выражения, имеет низкий приоритет.

Семантика операции присваивания

Семантика: Вычисляется выражение правой части, присваивается левому операнду.

Слева может быть только переменная (величина, способная хранить и изменять свое значение) – левостороннее выражение (LValue Required).

Пример. `sin(x) = 5;` // ошибка

```
x = 2;
cond = x<=2;        // int cond;
3 = 5;              // ошибка
x = x+1;
x+=5;               // присваивание после
z-=x+y;             //
```

Порядок вычисления выражений

Порядок вычисления выражения (а, значит, значение) определяется:

1) рангом (приоритетом) операции и

2) правилами ассоциативности для операций одного приоритета (слева направо или справа налево).

См. таблицу приоритетов.

Приоритет и ассоциативность операций

1	() [] -> :: .	→
2	! ~ + – ++ – – & * (тип) sizeof	←
3	* / % (бинарные)	→
4	+ – (бинарные)	→
5	<< >> (сдвиг)	→
6	< <= > >= (отношения)	→
7	== != (отношения)	→
8	&	→
9	^ (поразрядное исключающее и)	→
10	| (поразрядное исключающее или)	→
11	&& (логическое и)	→
12	|| (логическое или)	←
13	?: (условная операция)	←
14	= *= /= %= += –= &= ^= |= <<= >>=	→

Приоритет и ассоциативность операций

1.Ранг и ассоциативность арифметических операций близки к математическому:

* / % + –

2. Операция присваивания и ее клоны младше прочих, что позволяет выполнить присваивание только после того, как вычислено значение выражения.

3. Отношения младше арифметических операций:

x>0 && y>0

4. Для инкремента и декремента в префиксной форме операция старше прочих, в постфиксной – младше.

5. Ассоциативность для унарных операций, в основном, справа налево, для бинарных – слева направо.

Преобразование и приведение типов в выражениях

Приведение типов – механизм, который автоматически включается при смешивании типов в выражении.

Механизм приведения: на время вычислений значения данные меньшего типа приводятся к большему типу.

Размер типа определяется по размеру выделенной памяти.

Замечание.

В библиотеках по умолчанию тип **`double`** (не **`float`**), поэтому при вычислении значений функций выполняется приведение типа **`float`** к **`double`**, о чем выдается предупреждение (**`Warning`**).

Приведение типов – пример

Механизм приведения автоматически включается при смешивании типов в выражении.

```
float x,y;
int a,b;
x = a%b;   // Целочисленное деление.
y = x+b;      // b приводится к типу float
a = b+2.5;   // b приводится к float
```

Вывод: не рекомендуется смешение типов в выражениях.

См. пример.

Преобразование типов в выражениях

Преобразование типов выполняется при присваивании или при вычислении значения выражения.

1. Неявное преобразование происходит при смешении типов в выражениях, когда С может это сделать.
2. Явное преобразование происходит при смешении типов в выражениях по указанию программиста.

Синтаксис явного преобразования типов:

(тип) выражение;

Или при присваивании:

имя = (тип) выражение;

Преобразование типов – пример

```
float x,y;
x = sin(3.14*y+1);//

int a,b;
x = (float) a * (float) b  + 1;

float x = 4;
y = pow (x,1/2);  //
```

См. пример.

Правила неявного преобразования

Преобразование от меньшего типа к большему происходит без потери данных.

```
float = int
int   = char
```

Преобразование от большего типа к меньшему происходит с потерей данных.

```
int   x=5;
float y = 7.99;
y = x;    // y=5.0;
x = y;    // Потеря дробной части x=7, 0.99 потеряны.
```

Рекомендации

1. Строго относиться к типам данных, не смешивать типы в выражениях, следить, чтобы тип левого операнда присваивания соответствовал типу выражения правой части.
2. Помнить о приоритетах операций, использовать скобки.
3. Сложные выражения разделять на простые и вычислять по частям.
4. Повторяющиеся части выражения вычислять отдельно.
5. Справка по математическим функциям <**math.h**>