**Операции и выражения**

### Выражения

*Программа* оперирует с данными. Числа можно складывать, вычитать, умножать, делить. Из разных величин можно составлять *выражения*, результат вычисления которых – новая величина. Приведем примеры *выражений*:

X \* 12 + Y // значение X умножить на 12 и к результату прибавить значение Y

val < 3 // сравнить значение val с 3

-9 // константное выражение -9

*Выражение*, после которого стоит точка с запятой – это *оператор-выражение*. Его смысл состоит в том, что *компьютер* должен выполнить все действия, записанные в данном *выражении*, иначе говоря, вычислить *выражение*.

x + y – 12; // сложить значения x и y и затем вычесть 12

a = b + 1; // прибавить единицу к значению b и запомнить результат в переменной a

***Выражения*** – это переменные, функции и константы, называемые операндами, объединенные знаками операций. **Операции** могут быть ***унарными*** – с одним операндом, например, минус; могут быть *бинарные* – с двумя операндами, например *сложение* или *деление*. В *Си*++ есть даже одна операция с тремя операндами – условное *выражение*. Чуть позже мы приведем *список* всех операций языка *Си*++ для встроенных типов данных. Подробно каждая операция будет разбираться при описании соответствующего типа данных. Кроме того, ряд операций будет рассмотрен в разделе, посвященном определению операторов для классов. Пока что мы ограничимся лишь общим описанием способов записи *выражений*.

В типизированном языке, которым является *Си*++, у переменных и констант есть определенный тип. Есть он и у результата *выражения*. Например, *операции* *сложения* ( + ), *умножения* ( \* ), *вычитания* ( - ) и *деления* ( / ), примененные к целым числам, выполняются по общепринятым математическим правилам и дают в результате целое *значение*. Те же *операции* можно применить к вещественным числам и получить вещественное *значение*.

Операции ***сравнения***: больше ( > ), меньше ( < ), равно ( == ), не равно ( != ) сравнивают значения чисел и выдают логическое значение: истина ( true ) или ложь ( false ).

### Операция присваивания

***Присваивание*** – это тоже операция, она является частью *выражения*. Значение правого операнда присваивается левому операнду.

x = 2; // переменной x присвоить значение 2

cond = x < 2; // переменной cond присвоить значение true, если x меньше 2,

// в противном случае присвоить значение false

3 = 5; // ошибка, число 3 неспособно изменять свое значение

Последний пример иллюстрирует требование к левому операнду *операции присваивания*. Он должен быть способен хранить и изменять свое *значение*. Переменные, объявленные в программе, обладают подобным свойством. В следующем фрагменте программы

int x = 0;

x = 3;

x = 4;

x = x + 1;

вначале объявляется *переменная* x с начальным значением 0. После этого *значение* x изменяется на 3, 4 и затем 5. Опять-таки, обратим внимание на последнюю строчку. При вычислении *операции присваивания* сначала вычисляется правый *операнд*, а затем левый. Когда вычисляется *выражение* x + 1, *значение переменной* x равно 4. Поэтому *значение* *выражения* x + 1 равно 5. После вычисления *операции присваивания* (или, проще говоря, после *присваивания* ) *значение переменной* x становится равным 5.

У *операции присваивания* тоже есть результат. Он равен значению левого операнда. Таким образом, *операция присваивания* может участвовать в более сложном *выражении*:

z = (x = y + 3);

В приведенном примере переменным x и z присваивается *значение* y + 3.

Очень часто в программе приходится *значение переменной* увеличивать или уменьшать на единицу. Для того чтобы сделать эти действия наиболее эффективными и удобными для использования, применяются предусмотренные в *Си*++ специальные знаки операций: ++ (увеличить на единицу) и -- (уменьшить на единицу). Существует две формы этих операций: префиксная и постфиксная. Рассмотрим их на примерах.

int x = 0;

++x;

*Значение* x увеличивается на единицу и становится равным 1.

--x;

*Значение* x уменьшается на единицу и становится равным 0.

int y = ++x;

*Значение* x опять увеличивается на единицу. Результат *операции* ++ – новое *значение* x, т.е. переменной y присваивается *значение* 1.

int z = x++;

Здесь используется *постфиксная запись* *операции* увеличения на единицу. *Значение переменной* x до выполнения *операции* равно 1. Сама операция та же – *значение* x увеличивается на единицу и становится равным 2. Однако результат постфиксной *операции* – это *значение* аргумента до увеличения. Таким образом, переменной z присваивается *значение* 1. Аналогично, результатом постфиксной *операции* уменьшения на единицу является начальное *значение* операнда, а префиксной – его конечное *значение*.

Подобными мотивами оптимизации и сокращения записи руководствовались создатели языка *Си* (а затем и *Си*++), когда вводили новые знаки операций типа "выполнить операцию и присвоить". Довольно часто одна и та же *переменная* используется в левой и правой части *операции присваивания*, например:

x = x + 5;

y = y \* 3;

z = z – (x + y);

В *Си*++ эти *выражения* можно записать короче:

x += 5;

y \*= 3;

z -= x + y;

Т.е. *запись* oper= означает, что левый *операнд* вначале используется как левый *операнд* *операции* oper, а затем как правый *операнд* *операции присваивания* результата *операции* oper. Кроме краткости *выражения*, такая *запись* облегчает оптимизацию программы компилятором.

### Все операции языка Си++

Наряду с общепринятыми арифметическими и *логическими операциями*, в языке *Си*++ имеется набор операций для работы с битами – поразрядные И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и НЕ, а также *сдвиги*.

Особняком стоит операция sizeof. Эта операция позволяет определить, сколько памяти занимает то или иное *значение*. Например:

sizeof(long);

// сколько байтов занимает тип long

sizeof (b);

// сколько байтов занимает переменная b

Операция sizeof в качестве аргумента берет имя типа или *выражение*. *Аргумент* заключается в скобки (если *аргумент* – *выражение*, скобки не обязательны). Результат *операции* – *целое число*, равное количеству байтов, которое необходимо для хранения в памяти заданной величины.

Ниже приводятся все *операции* языка *Си*++.

#### Арифметические операции

+ сложение

- вычитание

\* умножение

/ деление

Операции *сложения*, *вычитания*, *умножения* и *деления* целых и вещественных чисел. Результат операции – число, по типу соответствующее большему по разрядности операнду. Например, *сложение* чисел типа short и long в результате дает число типа long.

% остаток

Операция нахождения *остатка от деления* одного целого числа на другое. Тип результата – целое число.

- минус

+ плюс

Операция "минус" – это *унарная операция*, при которой знак числа изменяется на противоположный. Она применима к любым числам со знаком. Операция "плюс" существует для симметрии. Она ничего не делает, т.е. примененная к целому числу, его же и выдает.

++ увеличить на единицу, префиксная и

постфиксная формы

-- уменьшить на единицу, префиксная и

постфиксная формы

Эти операции иногда называют ***"автоувеличением"*** (инкремент) и ***"автоуменьшением"*** (декремент). Они увеличивают (или, соответственно, уменьшают) операнд на единицу. Разница между постфиксной (знак операции записывается после операнда, например x++ ) и префиксной (знак операции записывается перед операндом, например --y ) операциями заключается в том, что в первом случае результатом является значение операнда до изменения на единицу, а во втором случае – после изменения на единицу.

#### Операции сравнения

== равно

!= не равно

< меньше

> больше

<= меньше или равно

>= больше или равно

Операции *сравнения*. Сравнивать можно операнды любого типа, но либо они должны быть оба одного и того же встроенного типа ( *сравнение* на равенство и неравенство работает для двух величин любого типа), либо между ними должна быть определена соответствующая операция *сравнения*. Результат – логическое значение true или false.

#### Логические операции

&& логическое И

|| логическое ИЛИ

! логическое НЕ

***Логические операции*** конъюнкции, дизъюнкции и отрицания. В качестве операндов выступают логические значения, результат – тоже логическое значение **true** или **false** .

#### Битовые операции

& битовое И

| битовое ИЛИ

^ битовое ИСКЛЮЧАЮЩЕЕ ИЛИ

~ битовое НЕ

***Побитовые операции*** над целыми числами. Соответствующая операция выполняется над каждым битом операндов. Результатом является целое число.

<< сдвиг влево

>> сдвиг вправо

Побитовый ***сдвиг*** левого операнда на количество разрядов, соответствующее значению правого операнда. Результатом является целое число.

#### Условная операция

операнд1?операнд2:операнд3

***Тернарная операция*** ; если значение первого операнда – истина, то результат – второй операнд; если ложь – результат – третий операнд. Первый операнд должен быть логическим значением, второй и третий операнды могут быть любого, но одного и того же типа, а результат будет того же типа, что и третий операнд.

#### Последовательность

, последовательность

Выполнить *выражение* до запятой, затем *выражение* после запятой. Два произвольных *выражения* можно поставить рядом, разделив их запятой. Они будут выполняться **последовательно**, и результатом всего *выражения* будет результат последнего *выражения*.

#### Операции присваивания

= присваивание

Присвоить значение правого операнда левому. Результат *операции присваивания* – это значение правого операнда.

+=, -=, \*=, /=, %=, |=, &=, ^=, <<=, >>=

выполнить операцию и присвоить

Выполнить соответствующую операцию с левым операндом и правым операндом и присвоить результат левому операнду. Типы операндов должны быть такими, что, во-первых, для них должна быть определена соответствующая арифметическая операция, а во-вторых, результат может быть присвоен левому операнду.

### Порядок вычисления выражений

У каждой *операции* имеется *приоритет*. Если в *выражении* несколько операций, то первой будет выполнена операция с более высоким *приоритетом*. Если же *операции* одного и того же *приоритета*, они выполняются слева направо.

Например, в *выражении*

2 + 3 \* 6

сначала будет выполнено *умножение*, а затем *сложение*, соответственно, *значение* этого *выражения* — число 20.

В *выражении*

2 \* 3 + 4 \* 5

сначала будет выполнено *умножение*, а затем *сложение*. В каком порядке будет производиться *умножение* – сначала 2 \* 3, а затем 4 \* 5 или наоборот, не определено. Т.е. для *операции* *сложения* порядок вычисления ее операндов не задан.

В *выражении*

x = y + 3

вначале выполняется *сложение*, а затем *присваивание*, поскольку *приоритет*    *операции присваивания* ниже, чем *приоритет* *операции* *сложения*.

Для данного правила существует *исключение*: если в *выражении* несколько *операций присваивания*, то они выполняются справа налево. Например, в *выражении*

x = y = 2

сначала будет выполнена *операция присваивания* значения 2 переменной y. Затем результат этой *операции* – *значение* 2 – присваивается переменной x.

Ниже приведен *список* всех операций в порядке понижения *приоритета*. *Операции* с одинаковым *приоритетом* выполняются слева направо (за исключением нескольких *операций присваивания* ).

::(разрешение области видимости имен)

. (обращение к элементу класса), -> (обращение к элементу класса по указателю), [] (индексирование), *вызов функции*, ++ (постфиксное увеличение на единицу), -- (постфиксное уменьшение на единицу), typeid (нахождение типа), dynamic\_cast static\_cast reinterpret\_cast const\_cast (преобразования типа)

sizeof (*определение* размера), ++ (префиксное увеличение на единицу), -- (префиксное уменьшение на единицу), ~ (битовое НЕ), ! (логическое НЕ), – (изменение знака), + (плюс), & (взятие адреса), \* (обращение по адресу), new (создание объекта), delete (удаление объекта), (type) (преобразование типа)

.\*, ->\* (обращение по указателю на элемент класса)

\* ( *умножение* ), / ( *деление* ), % ( *остаток* )

+ ( *сложение* ), – ( *вычитание* )

<< , >> ( *сдвиг* )

< <= > >= ( *сравнения* на больше или меньше)

== != (равно, неравно)

& (поразрядное И)

^ (поразрядное исключающее ИЛИ)

| (поразрядное ИЛИ)

&& (логическое И)

|| (логическое ИЛИ)

= ( *присваивание* ), \*= /= %= += -= <<= >>= &= |= ^= (выполнить операцию и присвоить)

?: (условная операция)

, ( *последовательность* )

Для того чтобы изменить *последовательность вычисления выражений*, можно воспользоваться круглыми скобками. Часть *выражения*, заключенная в скобки, вычисляется в первую *очередь*. Значением

(2 + 3) \* 6

будет 30.

Скобки могут быть вложенными, соответственно, самые внутренние выполняются первыми:

(2 + (3 \* (4 + 5) ) – 2)