Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №5

Приоритеты операций

Приоритет операции определяет порядок вычисления операций в выражении.

Ассоциативность определяет порядок вычисления операций, имеющих равный приоритет.

Приоритет операции определяется ее положением в *таблице приоритетов*. Самыми приоритетными являются первичные операции в верхней строке, затем идут унарные операции и т.д.

Унарная операция - операция, имеющая один операнд.

Если одна операция имеет более высокий приоритет, чем другая операция, она теснее связывает свои операнды, образуя неявные скобки.

Таблица 1

Вид операции	Знак операции	Ассоциативность
первичные	`( ) [ ] -> .`	слева направо
унарные	`! ~ - + ++ -- (type) * & sizeof`	справа налево
мультипликативные	`* / %`	слева направо
аддитивные	`+ -`	слева направо
сдвиг	`<< >>`	слева направо
отношение	`< <= > >=`	слева направо
равенство	`== !=`	слева направо
побитовая	`&`	слева направо
побитовая	`^`	слева направо
побитовая	`|`	слева направо
логическая	`&&`	слева направо
логическая	`||`	слева направо
условная	`?:`	справа налево
присваивание	`= op=`	справа налево
запятая	`,`	слева направо

Пример 1

```
X = 5 + 3 * 2
```

Данное выражение включает три операции: **=**, **+** и *****. Наиболее приоритетной является операция умножения *****, откуда вытекает следующее отношение:

```
X = 5 + (3 * 2)
```

Операция сложения **+** имеет более высокий приоритет, чем операция присваивания **=**, что приводит к следующему соотношению:

```
X = (5 + (3 * 2))
```

И окончательно получается выражение:

```
(X = (5 + (3 * 2)))
```

Последнее выражение показывает, как компилятор Си будет интерпретировать исходное выражение. Переменной **X** будет присвоено значение 11.

Скобки можно использовать для повышения удобочитаемости или для изменения стандартной группировки операндов.

Пример 2

```
X = (5 + 3) * 2
```

Это выражение даст результат, отличный от предыдущего примера. В данном случае переменной **X** присваивается значение 16.

Что произойдет, если в выражении две операции имеют одинаковый приоритет?

```
X = 12 / 2 * 3
```

Будет ли это выражение интерпретироваться так, что даст результат 18

```
X = (12 / 2) * 3
```

или так, что даст результат 2?

```
X = 12 / (2 * 3)
```

Ответ содержится в третьем столбце таблицы, названном ассоциативность. "Слева направо" означает, что самая левая операция в выражении применяется первой. Предыдущее выражение следует интерпретировать как **X = (12 / 2) * 3**.

Арифметические операции

Арифметические операции язык Си — операции, реализующие арифметические действия.

- *Аддитивные операции*
  - сложение (**+**)
  - вычитание (**-**)
- *Мультипликативные операции*
  - умножение (*****)
  - деление (**/**)
  - получение остатка от деления (**%**)
- *Изменение знака* (унарный минус) (**-**)

Пример 1

```
x+y
num-20
200*val
```

```
10/3
10.0/3.0
clock%60
-(prev+current)
```

Приоритеты арифметических операций определяются таблицей приоритетов операций языка Си.

Если операнды арифметических операций имеют различные типы, то выполняется преобразование типов в соответствии с некоторыми правилами.

Результат операции деления зависит от типов операндов.

При делении целых дробная часть результата отбрасывается (10/3 равно 3). Стандарт ANSI языка Си требует, чтобы округление производилось в сторону нуля. Если один или оба операнда – числа с плавающей точкой, то округления не происходит. Результат — вещественное число (10.0/3.0 равно 3.3333…).

Операция получение остатка от деления (%) выполняет деление и выдает в качестве результата остаток от деления.

10%3 (произносится "10 по модулю 3") равно 1. Операнды данной операции не могут быть вещественными числами.

Операцию изменения знака иногда называют унарным минусом, так как она имеет только один операнд.

Операция приведения

Операция приведение — это операция, используемая для временного преобразования типа операнда, который является выражением, в данные другого типа. Операция приведения является унарной операцией:

```
(тип)операнд
```

Тип задается ключевым словом, определяющим тип данных.

Подобно другим операциям языка Си операция приведение **(type)** указана в таблице приоритетов.

Пример 1

```
int main ( )
{
  float func(float);
  int x;
. . .
  func ((float)x);
  func (x);
. . .
}
float func(float y)
{
  float z;
```

```
. . .
  return z;
}
```

При вызове функции **func()** ее фактический параметр (переменная целого типа **x**) явно приводится к вещественному типу **float**, так как функция **func()** требует вещественного значения аргумента. Хорошим стилем программирования является передача функции параметров ожидаемого ею типа. Если при вызове функции не выполняется явное приведение типа фактического параметра, как во втором вызове функции **func()**, компилятор выполнит его сам некоторым способом (ему известно, что функция **func()** ожидает параметр типа **float** из прототипа объявления функции). Использование приведения делает текст программы более понятным, поскольку пожелания программиста указаны явно.

Другое использование операции приведения показано в прим. 2.

Пример 2

```
int x=15, y=4;
float result;
result = (float)x / (float)y;
```

Значение переменной **result** будет иметь дробную часть. Если бы операция приведения не использовалась, то дробная часть результата была бы потеряна.

0,000000000000000000000000000000,0

Преобразования при выполнении арифметических операций

Язык Си допускает выражения с операндами различных типов. Например, можно делить переменную типа **float** на константу типа **int**, прибавлять константу типа **int** к переменной типа **char** и т.д. Если выражение имеет операнды различных типов, низкий тип всегда преобразуется к более высокому. Один тип ниже другого, если он занимает меньше памяти.

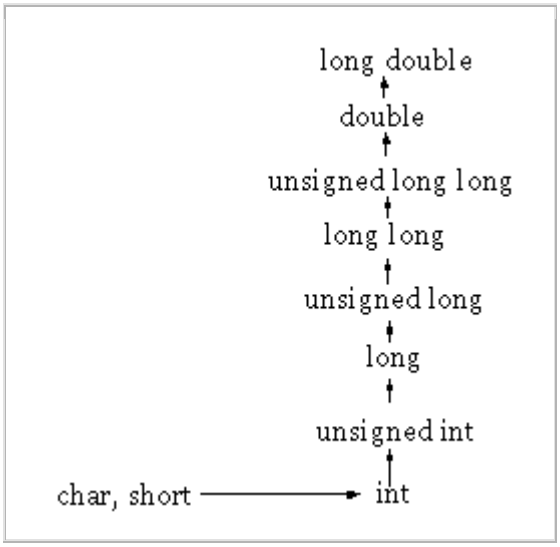

Рис. 1. Преобразования при выполнении арифметических операций

На рис. 1 показано, как будет выполняться преобразование при вычислении значения выражения. Преобразования по горизонтали выполняются автоматически. Например, объект типа **char** в выражении всегда временно преобразуется в тип **int**. Преобразования из младшего в старший тип (снизу вверх) выполняются в случае необходимости. Например, если выражение содержит операнды типов **int** и **double**, то операнд типа **int** будет временно преобразован в тип **double**.

Преобразование из одного типа данных в другой при выполнении арифметических операций выполняется также, как при выполнении операции присваивания.

В стандарте ANSI языка Си арифметика с операндами типа **float** может выполняться с одинарной точностью. Обычно же операнд типа **float** автоматически преобразуется в тип **double**.

Для устранения неоднозначности преобразования можно использовать операцию приведения типа **(type)**.

Пример 1

```
y = (int)7.2 / 3;
```

Примечание 1

По возможности лучше использовать операнды одного типа.

Операция присваивания

Именующие выражение = Выражение

Операция присваивания помещает значение выражения (своего правого операнда) в область памяти, определяемую левым операндом, который должен быть именующим выражением.

Именующее выражение — выражение, определяющее объект, которому может быть присвоено какое-либо значение, например, переменную или элемент массива.

Приоритет операций присваивания определяется таблицей приоритетов.

Пример 1

```
count = 1   /* верно */
   (a - b) = 6 /* неверно */
```

Пример 2

```
int  main ()
{
   unsigntd int        quantity;
   float               total, price = 150.00;

   quantity = 4U;
   total = quantity * price;
   . . .
}
```

Как видно из прим. 2, начальное значение переменной может быть присвоено как при объявлении, так и операцией присваивания.

Составные операции присваивания

Составные операции присваивания вида **op=** работают следующим образом:

выражение1 op= выражение2

эквивалентно выражению

выражение1 = выражение1 op выражение2

Знак **op** может быть знаком одной из перечисленных ниже арифметических операций или одной из бинарных побитовых операций.

Таблица 1

Составная операция присваивания	Эквивалентное выражение
`x += 50`	`x = x + 50`
`x -= 50`	`x = x - 50`
`x *= 50`	`x = x * 50`
`x /= 50`	`x = x / 50`
`x %= 50`	`x = x % 50`
`x *= a + b`	`x = x * (a + b)`

Использование операций присваивания в форме **op=** повышает эффективность программы, так как выражение слева вычисляется только один раз. Применение этой формы может повысить удобочитаемость и упростить отладку, особенно если операцией связаны сложные выражения.

Такая составная форма не только лаконична, но и позволяет с первого взгляда заметить, что используемая в ней переменная также изменяется.

Преобразования при присваивании

Если в операции присваивания оба операнда имеют один и тот же тип данных, то никакого *преобразования данных* не требуется. Однако, выражение слева (именующее выражение) не обязано иметь тот же тип, что и выражение справа. В этом случае делается попытка преобразования значения правого выражения к типу левого операнда.

Возможны следующие варианты использования операции присваивания, которые потребуют преобразования данных:

1. Объект типа **float** или **double** = значение выражения типа **char** или **int**.
2. Объект типа **char** или **int** = значение выраженией типа **float** или **double**.
3. Объект меньшего размера = значение выражения большего размера.
4. Объект большего размера = значение выражения меньшего размера.

В первом варианте присваивание обычно не влечет никакой потери данных. Значение выражение типа **char** или **int** будет помещено в область памяти, отведенную для **float** или **double**. При этом объект, представленный в формате целых чисел, преобразуется в формат с плавающей точкой для вещественных чисел.

Пример 1

```
int_var = 27;
float_var = int_var; /* float_var теперь равно 27.0*/
```

Гипотетически этот вид присваивания может привести к потере данных. Например, на некотором компьютере формат целых чисел типа **int** может содержать 10 цифр, а формат с плавающей точкой типа **float** – 7 значащих цифр. Присваивание переменной типа **float** 10-значащего целого приведет к потере точности. Для большинства компьютеров эта проблема преодолевается путем присваивания целых переменным типа **double**.

Во втором варианте присваивание вызывает отбрасывание дробной части. Если результат не помещается в формат типа **char** или **int**, часть значения теряется.

В третьем варианте в присваиваниях вида

- объект типа **char** = значение выражение типа **int**
- объект типа **float** = значение выражение типа **double**

данные могут быть потеряны.

В дальнейшем термин "младшие байты" и "старшие байты" относятся к двоичному представлению значения, независимо от того, в каком порядке эти байты хранятся в памяти различных процессоров.

В четвертом варианте присваивания вида

- объект типа **int** = значение выражение типа **char**
- объект типа **long** = значение выражение типа **short**

преобразование выполняется следующим образом.

Значение переносится в младший байт(ы). Обработка старшего байта(ов) зависит от знакового бита меньшего объекта и от используемого процессора.

Пример 2

Если у значения правого выражения знаковый бит равен 0, старшие байты заполняются нулями.

```
int_var = char_var
```

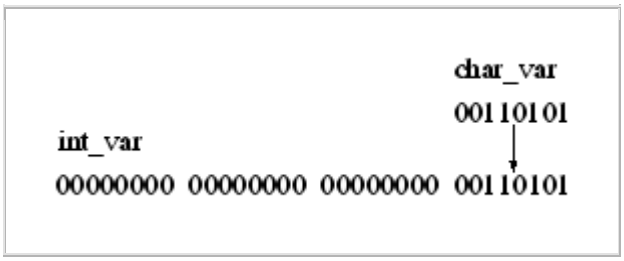

Рис. 1.

Пример 3

Если у значения правого выражения знаковый бит равен 1 и правый операнд имеет знаковый тип, старшие байты заполняются единицами.

int_var = char_var

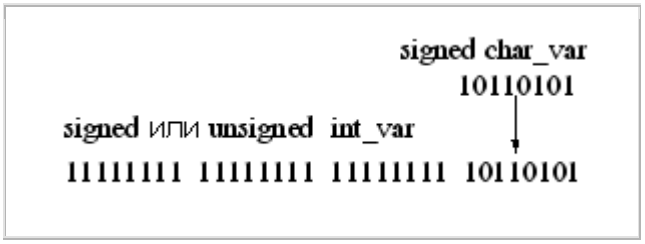

Рис. 2.

Пример 4

Если у значения правого выражения знаковый бит равен 1 и правый операнд имеет беззнаковый тип, старшие байты заполняются нулями.

int_var = char_var

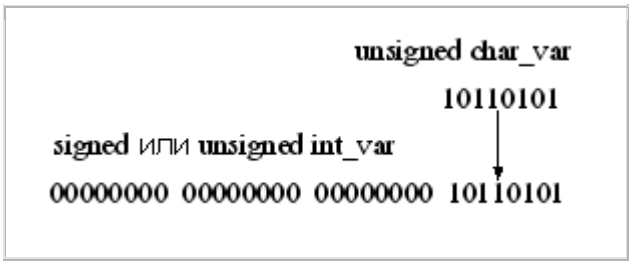

Рис. 3.

Операции инкрементации и декрементации

Операции инкрементации и декрементации являются унарными операциями, то есть операциями, имеющими один операнд.

операнд++

++операнд

Операция инкрементации **++** добавляет к операнду единицу.

операнд--

--операнд

Операция декрементации **--** вычитает из операнда единицу.

Операндом может быть именующее выражение, например, имя переменной.

Приоритет этих операций определяется таблицей приоритетов языка Си.

Следующие три строки увеличивают переменную **x** на 1:

```
x = x + 1;

++x;

x++;
```

Операции инкрементации и декрементации имеют *префиксную* (**++x**, **--x**) и *постфиксную* (**x++**, **x--**) форму записи. Они удобны при использовании в более сложных, чем приведенный пример, выражениях.

При префиксной форме записи операнд увеличивается или уменьшается сразу же, а после этого используется.

Пример 1

```
x = 3;
y = ++x;
```

Переменная **x** сразу же увеличивается до 4, и это значение присваивается переменной **y**.

При постфиксной форме записи операнд увеличивается или уменьшается после того, как он используется.

Пример 2

```
x = 3;
y = x++;
```

Переменной **y** присваивается значение 3, а затем переменная **x** увеличивается до 4.

Эти операции используются для сокращения текста программы и облегчения набора программы с клавиатуры, и для большинства компьютеров повышают эффективность работы программы. Они часто используются для увеличения или уменьшения индекса массива при переборе его элементов.

Но при работе с операциями инкрементации и декрементации надо быть осторожными: их операнд, являющийся именующим выражением, не следует использовать более одного раза в одном и том же выражении.

Пример 3

```
/*Пример плохого стиля:*/
array[num] = ++num;
```

Различные компиляторы могут интерпретировать это выражение по-разному и нельзя точно сказать, какой результат будет получен. Возьмите себе за правило: переменную, увеличиваемую или уменьшаемую в данном выражении, используйте в нем только один раз.

Пример 4

```
/*Более ясный стиль:*/
array[num] = num + 1;
++num;
```

Пример 5

```
i = 0;
while (i < size)
  {
   table[i] = 0;
   i = i+1;
```

```
}
```

Пример 6

```
/*Эквивалентный примеру 5*/
i = 0;
while (i < size)
   table[i++] = 0;
```

Операции отношений

Операции отношений — операции языка Си, которые используются для сравнения двух числовых значений.

Приоритет данных операций определяется таблицей приоритетов языка Си.

Значением операции отношения (например, **x < max**) может быть либо истина, либо ложь. Язык Си присваивает значение 0 ложному отношению и 1 — истинному отношению. На самом деле, любое ненулевое выражение в Си считается истинным, а нулевое — ложным.

Таблица 1

знак операции	смысл операции	пример
<	меньше	x < max
<=	меньше или равно	x <= max
>	больше	x > max
>=	больше или равно	x >= max
==	равно	x == max
!=	не равно	x != max

При использовании операций, обозначаемых двумя символами (например, <=, >=, == или !=), нельзя вставлять пробел между символами.

Примечание 1

Будьте осторожны с операцией "равно" ==, это не то же, что операция присваивания =.

Логические операции

В языке Си существуют следующие *логические операции*: логическое умножение И — *коньюнкция* (&&), логическое сложение ИЛИ — *дизьюнкция*(||), *логическое отрицание* НЕ (!).

Приоритеты логических операций определяются таблицей приоритетов.

Значением логического выражения является 0 (ложь) или 1 (истина). Значения логических операций определяются в соответствии с таблицами истинности этих операций.

Таблица 1

операнд 1	операнд 2	коньюнкция
ложь	ложь	ложь
ложь	истина	ложь
истина	ложь	ложь
истина	истина	истина

Таблица 2

операнд 1	операнд 2	дизьюнкция
ложь	ложь	ложь
ложь	истина	истина
истина	ложь	истина
истина	истина	истина

Таблица 3

операнд	отрицание
ложь	истина
истина	ложь

Пример 1

Логическое И

```
(x >= 1) && (x <= 500)
```

Пример 2

Логическое ИЛИ

```
(input == 'q') || (input == 'Q')
```

Пример 3

Логическое отрицание

```
!(x>y && y>z)
```

Для эффективности в языке Си вычисляется логическое выражение слева направо и прекращает вычисление, когда истинность значения определена.

Пример 4

```
(x<y) && (y<z)
```

Это выражение истинно, если истинно как выражение **(x<y)**, так и выражение **(y<z)**. Если **(x<y)** ложно, то все выражение должно быть ложно и выражение **(y<z)** игнорируется.

Пример 5

```
(x<y) || (y<z)
```

Это выражение истинно, если истинно либо выражение **(x<y)**, либо выражение **(y<z)**, или если истинны оба выражения. Если **(x<y)** истинно, все выражение истинно, так что в этом случае не вычисляется истинность выражения **(y<z)**.

Унарная операция **!** (НЕ) обращает логическое значение. Если **(x< y)** ложно, то **!(x< y)** истинно.

Условная операция

Условная операция- операция языка Си, которая по действию аналогична оператору if-else.

Синтаксис условной операции:

выражение1 ? выражение2 : выражение3

Если **выражение1** истинно, значением всего выражения будет значение **выражения2**, иначе — **выражения3**.

Вместо оператора **if-else** иногда может быть использована условная операция **?:**. Она выполняет проверку и выбор между двумя значениями внутри одного выражения. В отличие от конструкции **if-else**, являющейся оператором, эта операция может использоваться там, где синтаксис требует указывать выражение. Поэтому использование условной операции зачастую является более кратким и эффективным.

Подобно другим операциям языка Си, условная операция указана в таблице приоритетов.

Это единственная операция, составленная из двух символов, разделенных между собой выражениями (в отличии от операций **&&**, **||**, **>=**, **<=** и т.д.). Это единственная операция, требующая три операнда.

Пример 1

```
/* Функция вычисления абсолютной величины*/
int abs(int x)
{
  if ( x >= 0)
     return x;
  else
     return -x;
}
```

Пример 2

```
/* Альтернативный вариант*/
int abs(int x)
{
  return (x >= 0) ? x : -x;
}
```

Пример 3

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

Побитовые операции

Побитовые операции — это операции, работающие с отдельными битами операндов. Операнд должен быть целочисленным, т.е. иметь тип **longlong**, **long**, **unsigned**, **int**, **short** или **char**.

Приоритеты побитовых операций приведены в таблице приоритетов.

Поразрядное дополнение (**~**) – это унарная операция, результатом которой является значение, полученное поразрядным дополнением операнда, т.е. каждый бит со значением 0 заменяется на 1 и наоборот.

Поразрядное умножение И(**&**) – значением выражения, использующего эту операцию, является поразрядное логические умножение двух операндов.

Исключающее ИЛИ (**^**) — значением выражения, использующего эту операцию, является поразрядное исключающее ИЛИ ее операндов.

Поразрядное сложение ИЛИ(**|**) — значением выражения, использующего эту операцию, является поразрядное логические сложение двух операндов.

Знаки этих побитовых операций являются односимвольными. Не путайте их с операциями логического умножения **&&** (И) и логического сложения**||**(ИЛИ). Значением выражений, использующих логические операции может быть только истина или ложь. Значением побитовых операций обычно является новое битовое значение.

Сдвиг влево (**<<**) — значением выражения, использующего эту операцию, является битовое представление левого операнда, сдвинутого влево на число битов, определяемое значением правого операнда.

Сдвиг вправо (**>>**) — значением выражения, использующего эту операцию, является битовое представление левого операнда, сдвинутого вправо на число битов, определяемое значением правого операнда.

Пример 1

```
int num1, num2;
num1 = 5;  /* 00101 */
```

```
num2 = 11; /* 01011 */
```

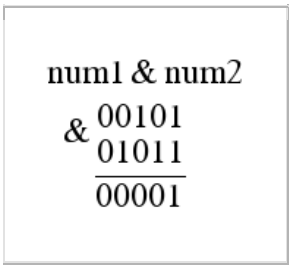

Рис. 1. Поразрядное И

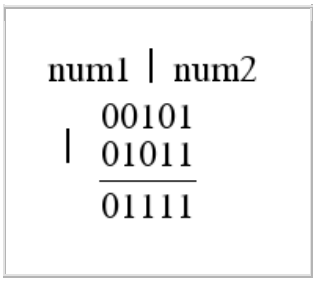

Рис. 2. Поразрядное ИЛИ

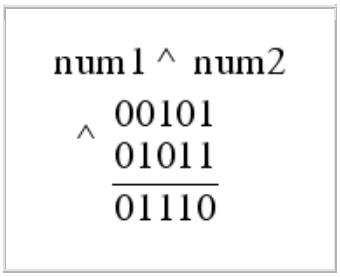

Рис. 3. Исключающее ИЛИ

Значением выражения, содержащего операцию поразрядного дополнения, является логическое отрицание значения операнда. Значения битов инвертируются: единицы заменяются нулями, нули – единицами.

Пример 2

```
num1 = 5;     /* 00101 */
num1              00101
~num1             11010
```

Операции сдвига << и >>

Целочисленное_выражение << Целочисленное_выражение

Целочисленное_выражение >> Целочисленное_выражение

При сдвиге значения операнда влево биты справа заполняются нулями.
При сдвиге вправо возможны следующие ситуации:

- при сдвиге положительных и беззнаковых значений биты слева заполняются нулями;
- при сдвиге отрицательных значений результат совпадает с результатом арифметического или логического сдвига в зависимости от аппаратного обеспечения компьютера: логический сдвиг вправо заполняет освобождающиеся слева биты нулями; арифметический сдвиг размножает знаковый разряд, т.е. самый левый бит.

Пример 3

Предположим, что программа выполняется на 8-разрядном компьютере. Первый (старший) разряд является знаковым.

x = x<<2; /* сдвиг значения переменной x влево на 2 бита */

исходное **x = 00011000**

Результат операции **x<<2** есть **01100000**.

y >>= 3; /* сдвиг значения переменной y вправо на 3 бита */

исходное **y =11011001**

Результат операции **y>>3** есть **00011011** для логического сдвига и **11111011** для арифметического сдвига.

Операция продолжения ("запятая")

Операция продолжения ("запятая") — бинарная операция языка Си, значением которой является значение правого операнда.

Операция "запятая" используется следующим образом:

выражение_A, выражение_B

Сначала вычисляется **выражение_A**, затем **выражение_B**. Значением всего выражения является значение **выражение_B**.

Приоритет этой операции определен в таблице приоритетов.

В общем случае эта операция используется тогда, когда синтаксис языка требует одного выражения, а программисту хочется выполнить два или более действия путем задания нескольких выражений. Операция "запятая" чаще всего используется для включения дополнительных выражений в заголовок цикла **for**.

В прим. 1 приводится корректный цикл **for**, в котором инициализируются и увеличиваются две переменные.

Пример 1

```
for (i = 0, j = 10; i<max; i++, j++)
  list[i] = name[j];
```

Здесь **выражением1** оператора **for** является следующее выражение: **i = 0, j = 10**, а **выражением3** является выражение **i++, j++**.

Выражение типа **(выражение1, выражение2, выражение3, выражение4)** корректно. Поскольку операция "запятая" левоассоциативна, т.е. связывает свои операнды слева направо, значением всего выражения будет значение **выражение4**.

Пример 2

```
x = 0;
while (printf("x is %d\n",x), x<5000)
  tryit (x++);
```

В языке программирования Си любое выражение имеет значение, например значение выражения **5+2** равно 7, значением выражения **x =**

200 является 200, значением выражения **x < y** есть 1 (истина) или 0 (ложь). В прим. 2 в цикле **while** значение выражения в круглых скобках есть значение второго выражения **x < 5000**, которое будет либо 0 (ложью), либо 1 (истиной). Функция **printf()** вычисляется каждый раз при вычислении выражения, определяющего условие продолжения цикла, однако не влияет на истинность значения проверки.