Операции и операторы, часть 1

Арифметические операции

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`+ (унар.)`	Плюс	`+X`	Префиксные	15	Справа налево
`– (унар.)`	Минус	`–X`			
`*`	Умножение	`X * Y`	Инфиксные	13	Слева направо
`/`	Деление	`X / Y`			
`%`	Остаток	`X % Y`			
`+`	Сложение	`X + Y`	Инфиксные	12	Слева направо
`–`	Вычитание	`X – Y`			

Унарные «+» и «-»

- Операция «унарный плюс» в качестве результата возвращает значение своего операнда (т.е. ничего не делает).

```
int a= 5, b = -17;

printf("%d %d\n", +a, +b);          // 5 -17
```

- Операция «унарный минус» в качестве результата возвращает значение операнда с противоположным знаком.

```
int a= 5, b = -17;

printf("%d %d\n", -a, -b);          // -5 17
```

Особенности операций «/» и «%»

- Если оба операнда операции «/» являются целочисленными, выполняется целочисленное деление (т.е. дробная часть просто отбрасывается).

```
1 / 2 => (не 0.5)
```

- В операции «%» оба операнда должны быть целыми.

- Использование 0 в качестве правого операнда операций «/» или «%» приведет к непредсказуемому результату (так называемое *неопределенное поведение*).

Особенности операций «/» и «%»

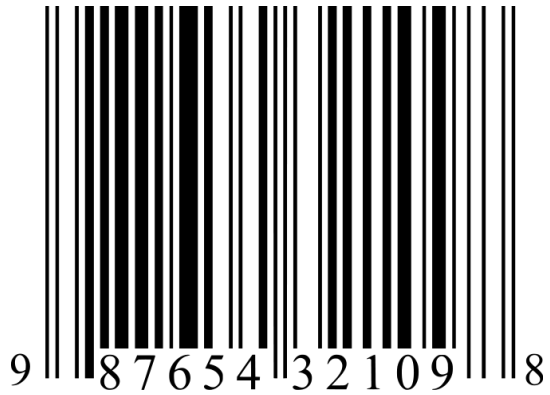
- Если оба операнда операций «/» и «%» являются целыми и один из них отрицательный, результат зависит от используемого стандарта.
  - В C89 результат может округляться как в большую, так и в меньшую сторону (так называемое *поведение определяемое реализацией*):

```
-9 / 7 => -1 или -2      -9 % 7 => -2 или 5
```

  - В C99 результат деления округляется большую сторону (по направлению к нулю):

```
-9 / 7 => -1    -9 % 7 => -2
```

Пример (UPC штрих-код)



- Префикс (1 цифра) – вид продукции
- Код производителя (5 цифр)
- Код товара (5 цифр)
- Контрольное число

Пример (UPC штрих-код)

- Суммируются все цифры на нечётных позициях (первая, третья, пятая, и т. д.) и результат умножается на три.
- Суммируются все цифры на чётных позициях (вторая, четвёртая, шестая, и т. д.).
- Числа, полученные на предыдущих двух шагах, складываются, и из полученного результата оставляется только последняя цифра.
- Эту цифру вычитают из 10.
- Конечный результат этих вычислений и есть контрольная цифра (десятке соответствует цифра 0).

Пример (UPC штрих-код)

```
#include <stdio.h>

int main(void)
{
    int d, i1, i2, i3, i4, i5, j1, j2, j3, j4, j5, s1, s2, total;

    printf("Enter the first (single) digit: ");
    scanf("%1d", &d);
    printf("Enter first group of five digits: ");
    scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4, &i5);
    printf("Enter second group of five digits: ");
    scanf("%1d%1d%1d%1d%1d", &j1, &j2, &j3, &j4, &j5);

    s1 = d + i2 + i4 + j1 + j3 + j5;
    s2 = i1 + i3 + i5 + j2 + j4;
    total = 3 * s1 + s2;

    printf("Check digit: %d\n", 10 - total % 10);

    return 0;
}
```

Операции инкремента и декремента

- Часто в программе можно встретить операцию «инкремента» (т.е. увеличение на единицу) и «декремента» (т.е. уменьшение на единицу) переменной:

```
i = i + 1;          // i += 1;
j = j - 1;          // j -= 1;
```

- В языке Си существуют операции, которые позволяют еще более сократить запись. Это операции инкремента «++» и декремента «––».
  1. Операции инкремента и декремента могут записываться как в префиксной (++i), так и в постфиксной формах (i++).
  2. Операции инкремента и декремента, как и операция присваивания, обладают побочным эффектом: они изменяют значения своих операндов.

Операции инкремента и декремента

- В случае префиксного инкремента ++i вычисление выражения возвращает значение i + 1 и увеличивает на 1 значение переменной i:

```
int i = 1;
printf("i is %d\n", ++i);          // i is 2
printf("i is %d\n", i);            // i is 2
```

- В случае постфиксного инкремента i++ вычисление выражения возвращает значение i и увеличивает на 1 значение переменной i:

```
i = 1;
printf("i is %d\n", i++);          // i is 1
printf("i is %d\n", i);            // i is 2
```

Операции инкремента и декремента

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`++`	Инкремент	`x++`	Постфиксные	16	Слева направо
`--`	Декремент	`x--`			
`++`	Инкремент	`++x`	Префиксные	15	Справа налево
`--`	Декремент	`--x`			

Операции инкремента и декремента

Когда операции инкремента и декремента используются в выражении более одного раза, то бывает сложно понять, какой получится результат.

Исходный фрагмент	Эквивалентный фрагмент	На выходе

`i = 1;`
`j = 2;`
`k = ++i + j++;`
	`i = i + 1;`
	`k = i + j;`
	`j = j + 1;`
		`i = 2`
		`j = 3`
		`k = 4`

`i = 1;`
`j = 2;`
`k = i++ + j++;`
	`k = i + j;`
	`i = i + 1;`
	`j = j + 1;`
		`i = 2`
		`j = 3`
		`k = 3`

Вычисление выражений

```
a=b+=c++-d+--e/-f
```

- Самая приоритетная операция – постфиксный инкремент.

```
a=b+=(c++)-d+--e/-f
```

- Следующие по приоритету операции – это префиксный декремент и унарный минус:

```
a=b+=(c++)-d+(--e)/(-f)
```

- Из оставшихся операций (вычитание, сложение и деление) самой приоритетной является операция деления:

```
a=b+=(c++)-d+((--e)/(-f))
```

Вычисление выражений

- Операции вычитания и сложения имеют одинаковый приоритетны, поэтому необходимо использовать правило ассоциативности этих операций. Операции вычитания и сложения группируются слева направо.

```
a=b+=(((c++)-d)+((--e)/(-f)))
```

- Остались только операция присваивания и операция составного присваивания. Эти операции имеют одинаковый приоритет, поэтому снова используем правило ассоциативности. Операции присваивания группируются справа налево.

```
(a=(b+=(((c++)-d)+((--e)/(-f)))))
```

Порядок вычисления подвыражений

- Язык Си не определяет порядок (в общем), в котором вычисляются подвыражения.

```
(a + b) * (c - d)   // что вычисляется раньше (a+b) или (c-d)?
```

- Большинство выражений имеют одно и то же значение независимо от того, в каком порядке вычисляются подвыражения. Это может быть не так
  - если подвыражение изменяет один из своих операндов

```
a = 5;
c = (b = a + 2) - (a = 1);    // ОШИБКА
```

  - к подобным ситуациям может также привести использование операций инкремента и декремента

```
i = 2;
j = i * i++;   // ОШИБКА
```

Оператор «выражение»

Выражения формируют основные строительные блоки для операторов и определяют, каким образом программа управляет данными и изменяет их. *Операторы* определяют каким образом управление передается из одной части программы другой.

В языке Си любое выражение можно «превратить» в оператор, добавив к этому выражению точку с запятой:

```
++i;
```

В языке Си точка с запятой является элементом оператора и его завершающей частью, а не разделителем операторов.

Оператор «выражение» (примеры)

```
i = 1;
```

1 сохраняется в переменной i, затем значение операции (новое значение переменной i) вычисляется, но не используется.

```
i--;
```

В качестве значения операции возвращается значение переменной i, оно не используется, но после этого значение переменной i уменьшается на 1.

```
i * j - 1;      // warning: statement with no effect
```

Поскольку переменные i и j не изменяются, этот оператор не имеет никакого эффекта и бесполезен.

Операции отношения и сравнения

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`<`	Меньше	`X < Y`	Инфиксные	10	Слева направо
`<=`	Меньше или равно	`X <= Y`			
`>`	Больше	`X > Y`			
`>=`	Больше или равно	`X >= Y`			
`==`	Равно	`X == Y`	Инфиксные	9	Слева направо
`!=`	Не равно	`X != Y`			

Операции отношения и сравнения

Операции отношения эквивалентны соответствующим математическим операциям за одним исключением. В выражении они возвращают целое значение 0, когда операция ложна, или целое значение 1, в противном случае.

```
int a = -5, b = 10;
printf("%d < %d is %d\n", a, b, a < b);      // -5 < 10 is 1
printf("%d > %d is %d\n", a, b, a > b);      // -5 > 10 is 0
```

Приоритет операций отношения меньше приоритета арифметических операций, поэтому

```
i + j < k - 1    ⇔    (i + j) < (k - 1)
```

Операции отношения и сравнения

Выражение

```
i < j < k
```

допустимо в языке Си, но имеет значение отличного от аналогичного математического выражения.

```
i < j < k    ⇔    (i < j) < k
```

В выражении сначала проверяется меньше ли значение переменной i значения переменной j, а затем 0 или 1 (как результат этого сравнения) будут сравниваться со значением k.

```
i < j < k НЕ эквивалентно проверке j∈(i, k)
```

Логические операции

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`!`	Не	`!X`	Префиксные	15	Справа налево
`&&`	И	`X && Y`	Инфиксные	5	Слева направо
`||`	Или	`X || Y`		4	

Операция	Значение
`!expr`	1 («истина»), если выражение имеет значение 0
`expr_1 && expr_2`	1 («истина»), если значения обоих выражений отличны от 0
`expr_1 || expr_2`	1 («истина»), если хотя бы одно из выражений отлично от 0

Особенности вычисления логических операций

1. При вычислении логических операций используется так называемая *сокращенная схема*.

   Сначала вычисляется значение левого операнда, затем правого. Если результат операции может быть определен по значению только левого операнда, то правый операнд не вычисляется.

```
(i != 0) && (j / i > 0)
```

2. Для логических операций «И» и «ИЛИ» порядок вычисления операндов фиксирован: сначала вычисляется левый операнд, затем правый.

Условный оператор if-else

Условный оператор позволяет сделать выбор между двумя альтернативами, проверив значение выражения.

```
if (выражение)
    оператор_1
else
    оператор_2
```

Скобки вокруг выражения обязательны, они являются частью самого условного оператора. Часть else не является обязательной.

```
if (a > b)
    max = a;
else
    max = b;
```

```
if (d % 2 == 0)
    printf("%d is even\n", d);
```

Условный оператор if-else

- Не путайте операцию сравнения «==» и операцию присвоения «=» .

```
if (i == 0) НЕ эквивалентно if (i = 0)
```

- Чтобы проверить, что i∈[0; n)

```
if (0 <= i && i < n) ...
```

- Чтобы проверить противоположное условие i∉[0; n)

```
if (i < 0 || i >= n) ...
```

- Поскольку в выражении условного оператора анализируется числовое значение этого выражения, отдельные конструкции можно упростить.

```
if (выражение != 0)    ⇔    if (выражение)
```

Составной оператор

В нашем «шаблоне» условного оператора указан только один оператор. Что делать, если нужно управлять несколькими операторами? Необходимо использовать составной оператор.

```
{
    операторы
}
```

Заключая несколько операторов в фигурные скобки, мы заставляем компилятор интерпретировать их как один оператор.

```
if (d > 0.0)
{
    x_1 = (-b - sqrt(d)) / (2.0 * a);
    x_2 = (-b + sqrt(d)) / (2.0 * a);
}
```

Вложенный условный оператор

```
if (a > b)
    if (a > k)
        max =  a;
    else
        max = k;
else
    if (b > k)
        max =  b;
    else
        max = k;
```

```
if (a > b)
{
    if (a > k)
        max =  a;
    else
        max = k;
}
else
{
    if (b > k)
        max =  b;
    else
        max = k;
}
```

Вложенный условный оператор

Поскольку часть else условного оператора может отсутствовать, в случае вложенных условных операторов это может приводить к путанице.

```
if (y != 0)
    if (x != 0)
        result = x / y;
else
   printf("y is equal to 0\n");
```

В языке Си else всегда связывается с ближайшим предыдущим оператором if без else.

```
if (y != 0)
    if (x != 0)
        result = x / y;
    else
        printf("y is equal to 0\n");
```

Каскадный условный оператор

```
if (выражение_1)
    оператор_1
else if (выражение_2)
    оператор_2
...
else if (выражение_n)
    оператор_n
else
    оператор
```

```
if (n < 0)
    printf("n is less than 0\n");
else if (n == 0)
    printf("n is equal to 0\n");
else
    printf("n is greater than 0\n");
```