Операции и выражения

Операция присваивания

После вычисления значения выражения нужно сохранить результат в переменной. Для этих целей используется операция присваивания.

v = e

Формально алгоритм работы операции присваивания в языке Си можно описать следующим образом:

- Вычислить l-значение первого операнда (v) операции.
- Вычислить r-значение второго операнда (e) операции.
- Присвоить вычисленное r-значение вычисленному l-значению объекта данных;
- Возвратить вычисленное r-значение как результат выполнения операции.

Операция присваивания: примеры

```
int i, j;

i = 5;              // Значение i равно 5
j = i;              // Значение j равно 5
k = 10 * i + j;     // Значение k равно 55
```

```
int i;
float f;

i = 72.99f;         // Значение i равно 72
f = 136;            // Значение f равно 136.0
```

Операция присваивания: особенности

Во многих языках программирования присваивание – это оператор. В языке Си присваивание – это операция такая же как, например, сложение.

Значением выражения a = b будет значение переменной b , приведенное к типу переменной a, и в качестве побочного эффекта это значение будет присвоено переменной a.

Благодаря тому, что присваивание - это операция, несколько присваиваний могут быть объединены в «цепочку»:

```
i = j = k = 0;
```

Операция присваивания: особенности

Операция присваивания правоассоциативна:

```
i = (j = (k = 0));
```

Присваивание в форме v = e разрешено везде, где допустимо использовать значение типа v. Например,

```
int i,j,k;

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

Составное присваивание

В операции присваивания «старое» значение переменной часто используется для вычисления «нового» значения этой же переменной:

```
i = i + 2;
```

Операции составного присваивания позволяют нам короче записать этот оператор:

```
i += 2;
```

Операция составного присваивания обладает теми же свойствами, что и обычная операция присваивания. Например, она обладает правой ассоциативностью:

```
i += j += k;   =>  i += (j += k);
```

Составное присваивание

`v += e`	Прибавить e к v, сохранить результат в v
`v -= e`	Вычесть e из v, сохранить результат в v
`v *= e`	Умножить v на e, сохранить результат в v
`v /= e`	Разделить v на e, сохранить результат в v
`v %= e`	Вычислить остаток от деления v на e, сохранить результат в v

Составное присваивание

Обратите внимание, что v += e не эквивалентно v = v + e.

- Причина 1: приоритете операции составного присваивания:

  i *= j + k НЕ i = i * j + k

  i *= j + k ЭТО i = i * (j + k)

- Причина 2: случай когда при вычислении v есть побочный эффект.

Составное присваивание

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
`=`	Присваивание	`x = y`	Инфиксные	2	Справа налево
`*=`	Присваивание с умножением	`x *= y`			
`/=`	Присваивание с делением	`x /= y`			
`%=`	Присваивание с остатком	`x %= y`			
`+=`	Присваивание со сложением	`x += y`			
`-=`	Присваивание с вычитанием	`x -= y`			

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 «%»

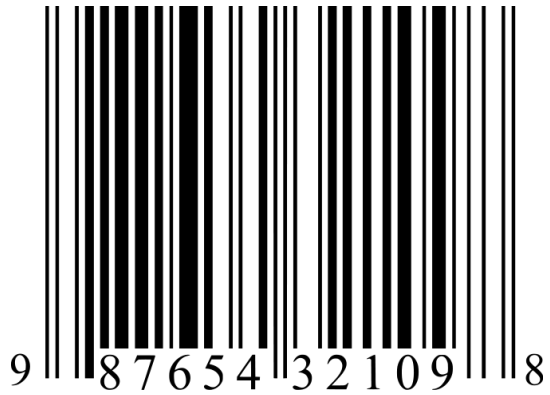
- Если оба операнда операций «/» и «%» являются целыми и один из них отрицательный, результат зависит от используемого стандарта.
  - В C89 результат может округляться как в большую, так и в меньшую сторону (так называемое *поведение определяемое реализацией*):

```
-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
```

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.