Операторы

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.

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 $i \in [0; n)$

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

- Чтобы проверить противоположное условие $i \notin [0; n)$

```
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");
```

Условная операция

Условная операция состоит из двух символов «?» и «:», которые используются вместе следующим образом

```
expr_1 ? expr_2 : expr_3
```

Сначала вычисляется выражение expr_1. Если оно отлично от нуля, то вычисляется выражение expr_2, и его значение становится значением условной операции. Если значение выражение expr_1 равно нулю, то значением условной операции становится значение выражения expr_3.

Условная операция

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
?:	Условие	**Z?X:Y**	Инфиксная	3	Справа налево

```
#include <stdio.h>

int main(void)
{
  int x = 5, y = 10, max = x > y ? x : y;

  printf("Max of %d and %d is: %d\n", x, y, max);

  // Можно обойтись без переменной max
  printf("Max of %d and %d is: %d\n", x, y, x > y ? x : y);

  return 0;
}
```

Оператор switch

```
int mark = 4;

if (mark == 5)
  printf("Excellent\n");
else if (mark == 4)
  printf("Good\n");
else if (mark == 3)
  printf("Averadge\n");
else if (mark == 2)
  printf ("Poor\n");
else
  printf("Illegal mark\n");
```

```
int mark = 4;

switch (mark)
{
  case 5:  printf("Excellent\n");
           break;
  case 4:  printf("Good\n");
           break;
  case 3:  printf("Averadge\n");
           break;
  case 2:  printf("Poor\n");
           break;
  default: printf("Illegal mark\n");
           break;
}
```

Оператор switch

В общей форме оператор switch может быть записан следующим образом

```
switch (выражение)
{
    case констнатное_выражение : операторы
    ...
    case констнатное_выражение : операторы
    default : операторы
}
```

- Управляющее выражение, которое располагается за ключевым словом switch, обязательно должно быть целочисленным (не вещественным, не строкой).

Оператор switch

- Константное выражение – это обычное выражение, но оно не может содержать переменных и вызовов функций.

  5 константное выражение

  5 + 10 константное выражение

  n + 10 НЕ константное выражение

- После каждого блока case может располагаться любое число операторов. Никакие скобки не требуются. Последним оператором в группе таких операторов обычно бывает оператор *break*.

Оператор switch

- Только одно константное выражение может располагаться в case-метке. Но несколько case-меток могут предшествовать одной и той же группе операторов.

```
switch (mark)
{
   case 5:
   case 4:
   case 3:  printf("Passing\n");
            break;
   case 2:  printf("Failing\n");
            break;
   default: printf("Illegal mark\n");
            break;
}
```

Оператор switch

- case-метки не могут быть одинаковыми.
- Порядок case-меток (даже метки default) не важен.
- case-метка default не является обязательной.

Роль оператора break в switch

- Выполнение оператора break «внутри» оператора switch передает управление за оператор switch. Если бы оператор break отсутствовал, то стали бы выполняться операторы расположенные в следующих case-метках.

```
int mark = 4;

switch (mark)
{
  case 5:  printf("Excellent\n");
  case 4:  printf("Good\n");
  case 3:  printf("Averadge\n");
  case 2:  printf("Poor\n");
  default: printf("Illegal mark\n");
}
```

```
# На экране увидим

Good
Averadge
Poor
Illegal mark
```

Оператор while

В языке Си цикл с предусловием реализуется с помощью оператора while.

В общей форме этот оператор записывается следующим образом

```
while (выражение) оператор
```

Выполнение оператора while начинается с вычисления значение выражения. Если оно отлично от нуля, выполняется тело цикла, после чего значение выражения вычисляется еще раз. Процесс продолжается в подобной манере до тех пор, пока значение выражения не станет равным 0.

Оператор while

```
#include <stdio.h>

int main(void)
{
    int sum, i, n = 5;

    // Сумма первых n натуральных чисел
    i = 1;
    sum = 0;
    while (i <= n)
    {
        sum += i;
        i++;

        // Можно обойтись одним оператором: sum += i++;
    }

    printf("Total of the first %d numbers is %d\n", n, sum);

    return 0;
}
```

Оператор do-while

В языке Си цикл с постусловием реализуется с помощью оператора do-while.

В общей форме этот оператор записывается следующим образом

```
do оператор while (выражение);
```

Выполнение оператора do-while начинается с выполнения тела цикла. После чего вычисляется значение выражения. Если это значение отлично от нуля, тело цикла выполняется опять и снова вычисляется значение выражения. Выполнение оператора do-while заканчивается, когда значение этого выражения станет равным нулю.

Оператор do-while

```
#include <stdio.h>

int main(void)
{
    int digits = 0, n = 157;

    do
    {
        digits++;
        n /= 10;
    }
    while (n != 0);

    printf("The number has %d digit(s).\n", digits);

    return 0;
}
```

Оператор for

Оператор for обычно используют для реализации цикла со счетчиком.

В общей форме этот оператор записывается следующим образом

```
for (expr_1; expr_2; expre_3) оператор
```

Оператор цикла for может быть заменен (за исключением редких случаев) оператором while

`expr_1;`
`while (expr_2)`
`{`
`  оператор`
`  expr_3;`
`}`
	expr_1 – шаг инициализации, который выполняется только один раз.
	expr_2 – выражение отношения или логическое выражение. Управляет завершением цикла.
	epxr_3 – выражение, которое выполняется в конце каждой итерации цикла.

Оператор for

```
#include <stdio.h>

int main(void)
{
  int sum, i, n = 5;

  i = 1;
  sum = 0;
  while (i <= n)
  {
    sum += i;
    i++;
  }

  ...
```

```
#include <stdio.h>

int main(void)
{
  int sum = 0, i, n = 5;

  for (i = 1; i <= n; i++)
    sum += i;

  ...
```

Оператор for

Любое из трех выражений expr_1, expr_2, expr_3 можно опустить, но точки с запятой должны остаться на своих местах.

- Если опустить expr_1 или expr_3, то соответствующие действия выполняться не будут.

```
i = 1;
for ( ; i <= n; )
  sum += i++;
```

- Если же опустить проверку условия expr_2, то по умолчанию считается, что условие продолжения цикла всегда истинно.

```
for ( ; ; )
  printf("Infinity loop\n");
```

Оператор for: идиомы

```
// Считать в прямом направлении от 0 до n-1
for (i = 0; i < n; i++) ...

// Считать в прямом направлении от 1 до n
for (i = 1; i <= n; i++) ...

// Считать в обратном направлении от n-1 до 0
for (i = n-1; i >= 0; i--) ...

// Считать в обратном направлении от n до 1
for (i = n; i > 0; i--) ...
```

Оператор for и стандарт C99

В C99 первое выражение expr_1 в цикле for может быть заменено определением. Эта особенность позволяет определять переменные для использования в цикле

```
for (int i = 0; i < n; i++)
```

Переменную i не нужно объявлять до оператора for.

```
for (int i = 0; i < n; i++)
{
    ...
    printf("%d", i);          // OK
    ...
}
printf("%d", i);            // ОШИБКА
```

Операция запятая

Иногда бывает необходимо написать оператор for с двумя или более выражениями инициализации или изменить несколько переменных в конце цикла. Это можно сделать с помощью операции запятая.

```
выражение_1, выражение_2
```

Эта операция выполняется следующим образом: сначала вычисляется *выражение_1* и его значение отбрасывается, затем вычисляется *выражение_2*. Значение этого выражения является результатом операции всей операции.

выражение_1 всегда должно содержать побочный эффект. В противном случае от этого выражения не будет никакого толка.

Операция запятая

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
,	Запятая	**X,Y**	Инфиксная	1	Справа налево

```
for (sum = 0, i = 1, n = 5; i <= n; i++, sum += i)
    ;   // пустой оператор
```

Оператор break

Оператор *break* может использоваться для принудительного выхода из циклов while, do-while и for. Выход выполняется из ближайшего цикла или оператора switch.

```
for (d = 2; d < n; d++)
    if (n % d == 0)
        break;

if (d < n)
    printf("%d is divisible by %d\n", n, d);
else
    printf("%d is prime\n", n);
```

Оператор continue

Оператор *continue* передает управление в конец цикла.

В циклах while и do-while это означает переход к проверке управляющего выражения, а в цикле for – выполнение expr_3 и последующую проверку expr_2.

Оператор *continue* может использоваться только внутри циклов.

```
sum = 0;
i = 0;
while (i < 10)
{
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
    i++;
}
```

Оператор goto

Оператор *goto* способен передать управление на любой оператор (в отличие от операторов *break* и *continue*) функции, помеченный *меткой*.

Метка – это идентификатор, расположенный вначале оператора:

```
идентификатор : оператор
```

Оператор может иметь более одной метки. Сам оператор *goto* записывается в форме

```
goto индетификатор;
```

Оператор goto

```
#include <stdio.h>

// Определение "простоты" числа
int main(void)
{
    int d, n = 17;

    for (d = 2; d < n; d++)
        if (n % d == 0)
            goto done;

  done:

    if (d < n)
        printf("%d is divisible by %d\n", n, d);
    else
        printf("%d is prime\n", n);

    return 0;
}
```

Оператор goto

- Считается, что оператор *goto* источник потенциальных неприятностей. Этот оператор на практике практически никогда не бывает необходим и почти всегда легче обходится без него.

- Есть несколько ситуаций, в которых без *goto* удобно использовать. Например, когда необходимо сразу выйти из двух и более вложенных циклов.

Пустой оператор

Пустой оператор состоит только из символа «;». Основная «специализация» пустого оператора – реализация циклов с пустым телом:

```
for (d = 2; d < n; d++)
    if (n % d == 0)
        break;
```

```
for (d = 2; n % d != 0 && d < n; d++)
    ;
```

Пустой оператор легко может стать источником ошибки:

```
if (d == 0);        // <-
    printf("ERROR: division by zero!\n");
```