

3. Операции

Рассмотрим операции в языке Си в порядке **убывания** их приоритетов.

Операции	В Паскале	В Си	Порядок выполнения
15. Скобки, адресные операции и выбор элемента структуры	() [] .	() [] .	---->
14. Унарные унарный минус унарный плюс логическое НЕ	A := - B A := + B not Flag	a = - b a = + b !flag	<----

Замечание: эта операция отрицания ненулевой операнд (истина) преобразует в 0, а 0 (ложь) - в 1. Эту операцию часто используют для замены конструкции

if (выражение == 0) //в стиле Паскаля

на

if (!выражение) //в стиле Си

в операторах if (...)

постфиксное	или @B	
дополнение	A := not B;	a = ~ b;
адрес	A := Addr(B);	a = &b;
разыменование	A := IntPtr^;	a = *intptr;
размер	A := SizeOf(B);	a = sizeof(b);
увеличение	A := A + 1; или A := Succ(A); или INC(A); процедура	a++ и ++a

Замечание: a++ и ++a являются двумя формами операции увеличения значения на 1. **Отличие** в следующем: когда применяется **постфиксная** форма (a++), то вначале переменная участвует в выражении со своим текущим значением и только после выполнения действий (вычисления значения выражения) значение переменной меняется. В случае **префиксной** формы (++a) - наоборот.

Пример:

a = 2;

b = 3;

a = b++; или a = ++b;

a == 3

a == 4

b == 4

b == 4

операция

уменьшение	A := A - 1; A := Pred(A); или DEC(A); процедура	a-- и --a
приведение типов	b := char(a);	b = (char) a;

Замечание по приведению (преобразованию) типов

Преобразование типов выполняется:

- 1) явно (см. выше --- `char *s = (char *) malloc (3);`);
 - 2) при присваивании;
 - 3) при вызове функции
- } при этом более «короткий» тип автоматически приводится к более «широкому».

Так же, как и в других языках программирования, в Си производится автоматическое (неявное) преобразование типов переменных, встречающихся в арифметических выражениях. Если над операндами различных типов необходимо выполнить операцию, они преобразуются к одному (общему) типу.

К каждой двуместной арифметической операции применяется следующее правило: операнды разных типов **приводятся к одному более длинному** типу.

Типы в Си в порядке убывания длины:

`long double - double - float - long long - unsigned long - signed long - unsigned int - signed int - unsigned short - signed short - unsigned char - signed char.`

При этом преобразование типов **в арифметических выражениях** происходит так:

- 1) переменные `char` и `short` преобразуются в `int`, а `float` - в `double`;
- 2) если один из операндов `double`, то другой преобразуется в `double`.
Результат операции - `double`;
- 3) если один из операндов типа `long`, другой преобразуется в `long`. Результат - `long`;
- 4) если один из операндов `unsigned`, другой преобразуется в `unsigned`.
Результат - `unsigned`;
- 5) если операнды типа `int`, то результат - `int`.

Числа с плавающей точкой (вещественные) перед выполнением операции автоматически преобразуются в `double`. Все операции над такими числами в языке Си выполняются с двойной точностью.

Преобразования автоматически производятся и **в случае операции присваивания**. При этом значение операнда справа от знака присваивания преобразуется к типу операнда слева от этого знака. В этом случае могут возникнуть ситуации преобразования старшего (более длинного) типа в младший (более короткий) и наоборот.

Так как фактические аргументы **при обращении к функции** являются выражениями, то производится преобразование типов фактических параметров (более короткие в более широкие).

В языке Си, как и в Паскале, имеется операция **явного преобразования типов**, с помощью которой можно производить явные преобразования и задавать тип результата. Вид этой операции:

`(type) exp`

где `type` - тип, а `exp` – выражение (результат которого приводится).

Результатом выполнения этой операции (преобразования) будет значение выражения, преобразованное в соответствующий тип по приведенным выше правилам. Так, например, если `k` - целое, а функция `bess` требует аргумента типа `double`, то обращение к ней должно осуществляться так:

`bess((double)k)`

Переменная *k* преобразуется в переменную типа *double*, которая и передается как аргумент функции. При этом **значение самой *k*** (в месте ее хранения в памяти) **не изменяется**. То есть преобразование значения переменной не всегда осуществляется фактически (вместо этого изменяется только точка зрения на это значение). Так, если имеется указатель на целое

`int *pk;`

и нужно проверить четность адреса *pk*, то использовать для этого операцию

`pk & 01`

нельзя, поскольку эту операцию можно применять **только к переменной целого типа** (нельзя к указателям). Однако

`((int) pk) & 01`

правильно, так как *pk* здесь приводится к целому типу. Фактического преобразования значения *pk* не происходит, просто содержимое *pk* в рамках операции `&` рассматривается как целое.

13. Умножение	<code>A := B * C;</code>	<code>a = b * c;</code>	
Деление целочисленное	<code>A := B div C;</code>	<code>a = b / c;</code>	--->
Деление	<code>X := B / C;</code>	<code>x = b / c;</code>	
Модуль (остаток)	<code>A := B mod C;</code>	<code>a = b % c;</code>	

Замечание: в отличие от Паскаля операция `/` в Си с целыми операндами производит целый результат. Чтобы заставить ее для целых операндов получать вещественный результат, надо выполнить приведение:

`(float)a/(float)b` или хотя бы `((float)a)/b`

12. Сложение	<code>A := B + C;</code>	<code>a = b + c;</code>	--->
вычитание	<code>A := B - C;</code>	<code>a = b - c;</code>	
11. Сдвиг вправо	<code>A := B shr C;</code>	<code>a = b >> c</code>	--->
сдвиг влево	<code>A := B shl C;</code>	<code>a = b << c</code>	
10. Больше чем	<code>A > B</code>	<code>a > b</code>	
больше или равно	<code>A >= B</code>	<code>a >= b</code>	--->
меньше чем	<code>A < B</code>	<code>a < b</code>	
меньше или равно	<code>A <= B</code>	<code>a <= b</code>	
9. Равно	<code>A = B</code>	<code>a == b</code>	--->
не равно	<code>A <> B</code>	<code>a != b</code>	
8. Поразрядное И	<code>A := B and C;</code>	<code>a = b & c;</code>	--->
7. Поразрядное исключающее ИЛИ	<code>A := B xor C;</code>	<code>a = b ^ c;</code>	--->
6. Поразрядное ИЛИ	<code>A := B or C</code>	<code>a = b c;</code>	--->
5. Логическое И	<code>Flag1 and Flag2</code>	<code>flag1 && flag2</code>	--->
4. Логическое ИЛИ	<code>Flag1 or Flag2</code>	<code>flag1 flag2</code>	--->

Замечание: часто путают логическую и поразрядную операции ИЛИ

```
int x, y;
void main (void)
{
    x = 0x0001;
    y = x | 0x1110; ----- y == 0x1111 (поразрядное ИЛИ)
    x = 0x0001;
    y = x || 0x1110; ----- y == 0x1 (логическое ИЛИ)
```

выполняется
не над числом
целиком, а над
отдельными
разрядами

Логическое ИЛИ

Паскаль			Си		
x	y	x or y	x	y	x y
False(0)	False(0)	False(0)	0	0	0
False(0)	True(1)	True(1)	0	не 0	не 0
True(1)	False(0)	True(1)	не 0	0	не 0
True(1)	True(1)	True(1)	не 0	не 0	не 0

В операторе IF значение $\neq 0$ трактуется как Истина.

3. Условие

Эта операция - единственная, имеющая 3 операнда (E1,E2,E3), каждый из которых является выражением. Операция условия интерпретирует их следующим образом:

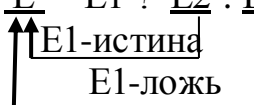
E1 ? E2 : E3;

возвращаемые значения
анализируемое выражение

4

Зачем это нужно? Как известно, оператор if не возвращает значений. А значения записанного выше выражения равны (после проверки условия E1):

- E2, если значение E1 - истина;
- E3, если значение E1 - ложь.

E = E1 ? E2 : E3


Пример: пусть требуется увеличить значение переменной S на величину наибольшего из X и Y. На языке Си это можно сделать так:

$S = (X > Y) ? S + X : S + Y \equiv S += (X > Y) ? X : Y.$

В большинстве других языков программирования решение этой задачи потребует явного нахождения максимального среди X и Y и присваивания этого значения переменной S:

$$S = \begin{cases} S + X, & \text{если } X > Y \\ S + Y, & \text{если } Y \geq X. \end{cases}$$

На языке программирования это выглядит так:

Паскаль	Си
<pre> Var x,y,s: integer; begin if (X > Y) then S = X + S else S = Y + S; end.</pre>	<pre> int x,y,s; void main(void) { s = (x > y)? x+s : y+s; или s += (x > y)? x : y; между + и = не должно быть пробела! }</pre>

2. Присваивание	$A := B;$ $A := A <ОП> B;$	$a = b;$ $a <ОП> = b;$ $a = b = c;$	<---
------------------------	-------------------------------	---	------

одно и то же имя и слева, и справа от знака присваивания

Замечание 1: источником ошибок в программах на Си является использование присваивания вместо проверки на равенство

if a = b на Паскале

if (a == b) на Си

5

Замечание 2: особенностью операции присваивания в Си является то, что она рассматривается как обычная операция, которая может участвовать в выражении (со своим приоритетом) и вызывает не только некоторые действия, но и возвращает значение (то, что справа от знака равенства) в качестве результата. Это значение затем может использоваться другими операциями.

С учетом сказанного следующая запись является корректной:

a=b=c=d=1;

<----- порядок вычисления.

Все присваивания имеют одинаковый приоритет и выполняются справа налево:

(a=(b=(c=(d=1))))).

Здесь выражение (d = 1) имеет значение 1, которое используется при присваивании переменной, и т.д. Таким образом, цепочка присваиваний в ходе выполнения данной записи действий может быть рассмотрена следующим образом:

d=1;

c=d;

b=c;

a=b;

Указанная особенность операции присваивания (она возвращает результат) позволяет предельно кратко (в одну строку) записывать громоздкие вычисления (здесь мы искусственно повысили **приоритет** операции присваивания с помощью **скобок**)

$s1=a+b;$
 $s2=c+d;$
 $s3=s1+s2;$

$\longrightarrow s3 = (s1=a+b) + (s2=c+d).$

сложения приоритет выше, чем у присваивания

Замечание 3: Си позволяет общее выражение вида

$A = A <op> B,$

где $<op>$ любая бинарная операция (кроме $\&\&$ и $\|$), заменять на выражение вида $A <op> = B.$

Так, например, вместо $A = A * B$ можно писать $A * = B.$ При этом следует учитывать, что в краткой записи значение переменной вычисляется лишь один раз, тогда как в полной записи - два раза. Это бывает важно помнить в случаях, аналогичных данному:

краткая форма

$a[i++] += 2 \longrightarrow$ это эквивалентно $a[i]=a[i]+2; i++;$ или $a[i]=a[i++]+2;$
 или $a[i++] = a[i++] + 2;$

это постфиксная форма, т.е. порядок вычисления следующий: сначала $a[i]$, а потом $i++$

1. Запятая	$S = (a = 1, b = 2);$	\longrightarrow
-------------------	-------	-----------------------	-------------------

Замечание: эта операция объединяет два и более выражений в одно выражение, результатом которого (если он нужен) будет значение самого правого выражения. Значения всех других (кроме правого) операндов вычисляются лишь для получения побочного эффекта.

$s3 = (s1 = a+b) + (s2=c+d); \longrightarrow s3 = (s1=a+b, s2=s1 + c + d).$

\longrightarrow

порядок вычисления

Эта операция используется для вычисления нескольких выражений там, где по синтаксису **допускается лишь одно выражение** и нельзя использовать составной оператор.

Например, если надо вычислить

9

$s = (a_1 + a_2 + a_3 + \dots + a_{10}) = \sum_{i=0} (a[i]).$

то, что мы обычно пишем (на Си)	а можно написать так
<p>подготовка цикла</p> <p>$s = 0;$</p> <p>for ($i = 0$; $i < 10$; $i++$)</p> <p>$s += a[i];$</p> <p>условие продолжения</p> <p>тело цикла</p> <p>то, что выполняется перед каждым повторением тела</p>	<p>два действия в подготовке цикла</p> <p>for ($i = 0, s=0; i < 10; s += a[i], i++$)</p> <p>тело цикла перешло в заголовок</p> <p>окончательно можно $S += a[i++]$</p>

В случае если операция запятая используется для других целей [например, для отделения аргументов функции $f(a, b, c)$], выражение, включающее операцию запятая, должно заключаться в скобки:

$f(a, (b=1, b+2), c);$

↘ значение 2-го аргумента функции равно 3, хотя значение b равно 1

Замечание:

$a[1, 2] = 25; \quad \neq \quad a[1][2] = 25;$

$a[1, 2] = 25; \quad == \quad a[2] = 25.$