

Переменные и операции в C

Лекция 2

Операции в **C** используются для выполнения действий над переменными

Атрибуты операций:

- Арность
 - унарные
 - бинарные
 - тернарные
- Приоритет
- Ассоциативность
 - левая
 - правая

Примеры операций в C

Арифметические

+ - * / %

Сравнения

< <= > >=
== !=

Логические

! && ||

Присваивания

=

Арифм. с
присваиванием

+= -= *=
/= %=

Побитовые

~ & ^

Побит. с
присваиванием

&= ^= `

Адресные

* &

C
пользовательскими
типами

. ->

Инкремент,
декремент

++ --

Разные

?: sizeof
[] ()

Арифметические операции

Название	Знак	Приоритет	Арность	Ассоциативность
Унарный плюс	+	высокий	унарная	правая
Унарный минус	-	высокий	унарная	правая
Умножение	*	средний	бинарная	левая
Деление	/	средний	бинарная	левая
Остаток от деления	%	средний	бинарная	левая
Плюс	+	низкий	бинарная	левая
Минус	-	низкий	бинарная	левая

Унарные операции

Унарный плюс $+$ умножает операнд на $+1$, результат записывается во временную память

```
int test = 17;  
int result = 0;  
result = +test;  
printf("%d\n%d\n", test, result);
```

17
17

Унарный минус $-$ умножает операнд на -1 , результат записывается во временную память

```
int test = 17;  
int result = 0;  
result = -test;  
printf("%d\n%d\n", test, result);
```

17
-17

Умножение

Операция умножения `*` умножает операнды друг на друга, результат записывается во временную память

```
int a = 5, b = 6;  
printf("%d\n", a * b );
```

30

```
float a = 5.5, b = 3.2;  
printf("%f\n", a * b );
```

17.6

```
char a = 'a', b = 'b';  
printf("%d\n", a * b );
```

9506

ASCII код 'a' = 97
'b' = 98

Деление

Операция деления `/` делит операнды друг на друга, результат записывается во временную память.

```
int a = 30, b = 6;  
printf( "%d\n", a / b );
```

5

```
float a = 5.5, b = 3.2;  
printf( "%f\n", a / b );
```

1.71875

Деление на 0

При делении целого числа на 0
поведение неопределено!!!
undefined behavior

```
int a = 30, b = 0;  
printf("%d\n", a / b );
```

При делении вещественного
числа на 0 получается
бесконечность (INFINITY)

```
float a = 30, b = 0;  
printf("%f\n", a / b );
```

inf

```
float a = -30, b = 0;  
printf("%f\n", a / b );
```

-inf

Деление целых чисел

При делении целого числа на целое число
всегда получается целое число

```
int a = 3, b = 4;  
printf("%d\n", a / b );
```

0

```
int a = 4, b = 3;  
printf("%d\n", a / b );
```

1

Явное приведение типов

- Унарная операция, выглядит как (<целевой_тип>)
- Создает временную копию операнда, приводя его к целевому типу
- При приведении вещественных чисел к целым дробная часть отбрасывается
- Рекомендуется всегда использовать операцию явного приведения типов. Этим программист показывает, что ему известно о различии типов, что смешение сделано намеренно

```
int a = 3, b = 4;  
printf("%.2f\n", (float) a / b );
```

0.75

“Остаток от деления” (на самом деле это не он)

- Определена **только для целых чисел**
- Операция получения остатка от деления `%` вычисляет остаток от деления операндов друг на друга, результат записывается во временную память.
- Верно следующее равенство:

$$a = (a / b) * b + a \% b ;$$

```
int a = 30, b = 7;  
printf("%d\n", a % b );
```

2

```
int a = -30, b = 7;  
printf("%d\n", a % b );
```

-2

Программа вычисления объёма сферы

```
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>           /*математическая библиотека для
                             числа M_PI и функции возведения в
                             степень pow */

int main() {
    float radius;
    puts("Enter a radius ");
    scanf("%f", &radius);
    printf("Volume is: %.2f\n",
           (float)4 / 3 * M_PI * pow(radius, 3));
    return 0;
}
```

Получение суммы цифр трехзначного числа

```
#include <stdio.h>
int main() {
    int number = 0, sum = 0;
    puts("Enter number:");
    scanf("%d\n", &number);
    sum = number / 100; //получение старшей цифры числа
    (234/100=2)
    sum += ( number % 100 ) / 10; //средняя цифра числа
    sum += number % 10; //младшая (правая) цифра числа
    printf("Sum of digits is %d\n", sum);
    return 0;
}
```

Логические операции

Название операции		Приоритет	Арность	Ассоциативность / Особенности
Отрицание	!	высокий	унарная	справа налево
Логическое И	&&	низкий	бинарная	выполнение гарантируется слева направо
Логическое ИЛИ			бинарная	ещё ниже, чем &&

- Не изменяют своих операндов
- Результатом операций будет **ИСТИНА** или **ЛОЖЬ**
- **ЛОЖЬ** в **C** — это **0**
- **ИСТИНА** в **C** — все, что не **0**
- Обычно используются в условных выражениях и циклах
- Выполняются **строго слева направо**, если становится известен результат операции, выполнение операции прекращается

Логическое отрицание

a	! a
---	-----

0	1
---	---

1	0
---	---

```
#include <stdio.h>
```

```
int main() {  
    float a = 5.6;  
    int b = 34;  
    printf("%d\n", !a );  
    printf("%d\n", !(b-b) );  
    printf("%d\n", !((b-b)*a) );  
    printf("%f\n", !(b-b)*a );  
    return 0;  
}
```

```
0  
1  
1  
5.6
```

Логическое И

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

```
float a = 5.6, b = 3.4, c = 0;
int d = 3, e = 7, f = 0;
(a - a) && (c = b);
printf("%f\n", c );
(c = b) && (a - a);
printf("%f\n", c );
(e * f) && (f = d);
printf("%d\n", f );
(f = d) && (d = e * f);
printf("%d\n", f );
printf("%d\n", d );
```

```
0
3.4
0
3
21
```


Логическое ИЛИ

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

```
float a = 5.6, b = 3.4, c = 0;
int d = 3, e = 7, f = 0;
a || (c = b);
printf("%f\n", c );
(c = b) || (a - a);
printf("%f\n", c );
e || (f = d);
printf("%d\n", f );
(f = d) || (d = e * f);
printf("%d\n", f );
printf("%d\n", d );
```

```
0
3.4
0
3
3
```

Сложные логические выражения

Результат зависит от операции с более низким приоритетом, поэтому операция с более высоким приоритетом может не выполняться

```
float a = 5.6, b = 3.4, c = 0;
int d = 3, e = 7, f = 0;
d || (f = e && a);
printf("%d\n", f);
(f = e && a) || (c = b);
printf("%d\n", f);
printf("%f\n", c);
((f = e) && a) || (c = b);
printf("%d\n", f);
printf("%f\n", c);
(f && (a-a)) || (c = b);
printf("%f\n", c);
```

```
0
1
0
7
0
3.4
```

Операторы сравнения

- Не изменяют своих операндов
- Результатом операций будет **ИСТИНА** или **ЛОЖЬ**
- Обычно используются в условных выражениях и циклах

Название	Знак	Приоритет	Арность	Ассоциативность
меньше	<	средний	бинарная	левая
меньше равно	<=	средний	бинарная	левая
больше	>	средний	бинарная	левая
больше равно	>=	средний	бинарная	левая
равно	==	низкий	бинарная	левая
не равно	!=	низкий	бинарная	левая

Что выведет программа?

```
int x = 40;  
if ( 30 < x < 50 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

TRUE

```
int x = 40;  
if ( 45 < x < 60 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

TRUE

Что выведет программа?

```
int x = -20;  
if ( -30 < x < -10 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

FALSE

```
int x = -20;  
if ( -10 < x < 1 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

TRUE

Что выведет программа?

```
int x = 40;  
if ( 30 < x && x < 50 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

TRUE

```
int x = 40;  
if ( 45 < x && x < 60 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

FALSE

Что выведет программа?

```
int x = -20;  
if ( -30 < x && x < -10 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

TRUE

```
int x = -20;  
if ( -10 < x && x < 1 )  
    puts( "TRUE" );  
else  
    puts( "FALSE" );
```

FALSE

Операция равно ==

```
const int TEST = 5;
int a = 3;
printf( "%d\n", (a == TEST) );
printf( "%d\n", (a = TEST) );
```

0
5

- Легко перепутать с операцией присваивания
- Если сравнивается выражение и константа, то константу рекомендуется ставить слева

```
const int TEST = 5;
int a = 3;
printf( "%d\n", (TEST == a) );
printf( "%d\n", (TEST = a) );
```

```
...
/Users/danilashubin/CLionProjects/untitled5/main.c:7:26
: error: cannot assign to variable 'TEST' with
const-qualified type 'const int'
    printf( "%d\n", (TEST = a) );
                        ~~~~ ^
/Users/danilashubin/CLionProjects/untitled5/main.c:4:15
: note: variable 'TEST' declared const here
    const int TEST = 5;
```


Побитовые операции

- Выполняются только над **целыми числами**
- Не изменяют своих операндов
- Результатом операций будет **целое число**

Название	Знак	Приоритет	Арность	Ассоциативность
Побитовое логическое отрицание	~	высокий	унарная	правая
Побитовое логическое И	&	средний	бинарная	левая
Побитовое логическое исключающее ИЛИ	^	чуть ниже	бинарная	левая
Побитовое логическое ИЛИ		еще ниже	бинарная	левая
Побитовый сдвиг влево	<<	средний	бинарная	левая
Побитовый сдвиг вправо	>>	средний	бинарная	левая

- Выполняются над каждым битом числа **по отдельности**
- Чтобы узнать результат операции нужно перевести число в двоичную систему счисления

Побитовое логическое И

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Используется при наложении маски на число для получения значения отдельного бита

```
char a = 10, b = 7;  
printf("%d\n", ( a & b ) );
```

2

a	00001010
b	00000111
a & b	00000010

Побитовое логическое ИЛИ

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

Используется при объединении флагов для передачи в функцию в качестве одного параметра

```
char a = 10, b = 7;  
printf("%d\n", ( a | b ) );
```

15

a	00001010
b	00000111
a b	00001111

Побитовое логическое исключающее ИЛИ

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Может использоваться в
алгоритмах симметричного
шифрования

```
char a = 10, b = 7;  
printf("%d\n", ( a | b ) );
```

13

a	00001010
b	00000111
a ^ b	00001101

Побитовое логическое отрицание

a	~ a
0	1
1	0

```
char a = 10, b = -7;  
printf( "%d\n", ~a );  
printf( "%d\n", ~b );
```

```
-11  
7
```

Если отрицается положительное число, то получается отрицательное, по модулю на один больше. И наоборот.

a	00001010	b	11111001
~ a	11110101	~ b	00000110

Побитовый сдвиг влево

a	b	Результат
17	6	$17 * 2^6$
17	6	$6 * 2^{17}$
17		$17 * 2^3$
	6	$6 * 2^3$

- Эквивалентен умножению числа на соответствующую степень двойки
- Освободившиеся справа разряды заполняются нулями

```
int a = 17, b = 6;  
printf( "%d\n", (a << b) );  
printf( "%d\n", (b << a) );  
printf( "%d\n", (a << 3) );  
printf( "%d\n", (b << 3) );
```

```
1088  
786432  
136  
48
```

Побитовый сдвиг вправо

- Эквивалентен делению числа на соответствующую степень двойки
- Освободившиеся слева разряды заполняются нулями, если сдвигается положительное число
- Освободившиеся слева разряды заполняются единицами, если сдвигается отрицательное число

```
int a = 17, b = 6;  
printf( "%d\n", (a >> b) );  
printf( "%d\n", (b >> a) );  
printf( "%d\n", (a >> 3) );  
printf( "%d\n", (b >> 3) );
```

```
0  
0  
2  
0
```

```
int a = 17, b = 6;  
printf( "%d\n", (a >> b) );  
printf( "%d\n", (b >> a) );  
printf( "%d\n", (a >> 3) );  
printf( "%d\n", (b >> 3) );
```

```
0  
0  
2  
0
```

Операция присваивания

Изменяет значение своего левого операнда

Название	Знак	Приоритет	Арность	Ассоциативность
присваивание	=	самый низкий	бинарная	гарантируется выполнение справа налево

L-value и R-value

L-value

Что такое

может стоять слева от знака присваивания

Требования

- не может быть константой
- должно иметь адрес

Чем может быть

- Переменная
- Функция
- Указатель
- Объект
- Указатель на функцию
- Указатель на массив

R-value

может стоять справа от знака присваивания

должно иметь значение

- Переменная
- Функция
- Указатель
- Объект
- Указатель на функцию
- Указатель на массив
- Константа
- Имя массива
- Выражение

Арифметические с присваиванием

Знак	Приоритет	Арность	Ассоциативность
<code>*=</code>	самый низкий	бинарная	правая
<code>/=</code>	самый низкий	бинарная	правая
<code>%=</code>	самый низкий	бинарная	правая
<code>+=</code>	самый низкий	бинарная	правая
<code>-=</code>	самый низкий	бинарная	правая

```
int myVeryVeryVeryVeryLongNameVariable = 5, b = 7;  
myVeryVeryVeryVeryLongNameVariable = myVeryVeryVeryVeryLongNameVariable + b;
```

```
int myVeryVeryVeryVeryLongNameVariable = 5, b = 7;  
myVeryVeryVeryVeryLongNameVariable += b;
```

Побитовые с присваиванием

Знак	Приоритет	Арность	Ассоциативность
<code>&=</code>	самый низкий	бинарная	левая
<code> =</code>	самый низкий	бинарная	левая
<code>^=</code>	самый низкий	бинарная	левая
<code><<=</code>	самый низкий	бинарная	левая

```
#include <stdio.h>

int main() {
    int a = 10, b = 7;
    printf("%d\t%d\n", a, b);           // 10    7
    a ^= b ^= a ^= b;
    printf("%d\t%d\n", a, b);           // 7     10
    return 0;
}
```

Операции **инкремента** / **декремента**

Название	Знак	Приоритет	Арность	Ассоциативность
инкремент	++	высокий	унарная	правая
декремент	--	высокий	унарная	правая

++i	i++	i = i + 1	i += 1
--i	i--	i = i - 1	i -= 1

Все унарные операции существуют в **префиксной** форме (знак операции записывается перед операндом)

Операции инкремента / декремента существуют в двух формах:

- **префиксной**: **++i**;
- **постфиксной**: **i++**;

Отличия префиксной и постфиксной форм

Префиксная форма

Постфиксная форма

Алгоритм

- Увеличивает значение операнда
- Возвращает новое значение

- Создает временную копию текущего значения
- Увеличивает значение
- Возвращает временную копию

Скорость

выше

ниже

Является l-value

да

нет

Пример работы с оператором инкремента

```
int i = 5;  
printf( "%d\n", ++i );
```

6

```
int i = 5;  
printf( "%d\n", i++ );  
printf( "%d\n", i );
```

5

6

```
int i = 5;  
printf( "%d\n", ( ++i = i++ ) );  
printf( "%d\n", i );
```

6

7

```
int i = 5;  
printf( "%d\n", ( ++i + i++ ) );  
printf( "%d\n", i );
```

12

7

```
int i = 5;  
printf( "%d\n", ( ++i + ++i ) );
```

14

Условная операция

Название	Знак	Приоритет	Арность	Ассоциативность
условная	?:	низкий	тернарная	гарантируется выполнение слева направо

Поиск максимума двух чисел:

```
int a = 15, b = 8;  
printf("%d\n", (a > b ? a : b) );
```

15

```
int a = 5, b = 8;  
printf("%d\n", (a > b ? a : b) );
```

8

Условная операция

Условные операции можно вкладывать друг в друга

Поиск максимума трех чисел:

```
int a = 15, b = 8, c = 19;  
printf("%d\n", (a > b ? a > c ? a : c : b) );
```

19

```
int a = 15, b = 8, c = 9;  
printf("%d\n", (a > b ? a > c ? a : c : b) );
```

15

```
int a = 5, b = 18, c = 9;  
printf("%d\n", (a > b ? a > c ? a : c : b) );
```

18

Операция `sizeof`

Название	Знак	Приоритет	Арность	Ассоциативность
Получение размера	<code>sizeof</code>	высокий	унарная	правая

Позволяет получить размер типа или переменной:

```
int a = 15;  
printf("%d\n", sizeof(int) );
```

4

```
long double a = 15.0;  
printf("%d\n", sizeof(a) );
```

8

Приоритет операций в C

-> . () []

++ -- * & ~ ! + - sizeof

* / %

+ -

<< >>

< <= > >=

== !=

&

^

|

&&

||

?:

= += -= *= /= %= <<= >>= &= |= ^=

Короче говоря...

- Операции выполняют действия над операндами
- Существуют различные группы операций
- Операции имеют разные арифметичность, ассоциативность и приоритет
- Для 4 операций строго определен порядок выполнения: &&, ||, =, ?:.
- Только операции присваивания (все виды) и операции инкремента/декремента изменяют свои операнды
- Операции инкремента/декремента существуют в двух формах