1. Operators

C offers us a wide variety of operators that we can use to operate on data.In

particular, we can identify various groups of operators:

- arithmetic operators
- comparison operators
- logical operators
- compound assignment operators
- bitwise operators
- pointer operators
- structure operators
- miscellaneous operators

In this section I'm going to detail all of them, using 2 imaginary variables `a` and `b` as examples.

> I am keeping bitwise operators, structure operators and pointer operators out of this list, to keep things simpler

1.1. Arithmetic operators

In this macro group I am going to separate binary operators and unaryoperators.

Binary operators work using two operands:

Operator	Name	Example
`=`	Assignment	`a = b`
`+`	Addition	`a + b`
`-`	Subtraction	`a - b`
`*`	Multiplication	`a * b`
`/`	Division	`a / b`
`%`	Modulo	`a % b`

Unary operators only take one operand:

Operator	Name	Example
`+`	Unary plus	`+a`

`-`	Unary minus	`-a`
`++`	Increment	`a++` or `++a`
`--`	Decrement	`a--` or `--a`

The difference between `a++` and `++a` is that `a++` increments the `a` variable after using it. `++a` increments the `a` variable before using it.For example:

```
int a = 2;
int b;
b = a++ /* b is 2, a is 3 */
b = ++a /* b is 4, a is 4 */
```

The same applies to the decrement operator.

1.2. Operator precedence

With all those operators (and more, which I haven't covered in this post, including bitwise, structure operators and pointer operators), we must pay attention when using them together in a single expression.

Suppose we have this operation:

```
int a = 2;
int b = 4;
int c = b + a * a / b - a;
```

What's the value of `c` ? Do we get the addition being executed before the multiplication and the division?

There is a set of rules that help us solving this puzzle.

In order from less precedence to more precedence, we have:

- the `=` assignment operator
- the `+` and `-` **binary** operators
- the `*` and `/` Operators

  the `+` and `-` unary operator

-

Operators also have an associativity rule, which is always left to right exceptfor the unary operators and the assignment.

In:

```
int c = b + a * a / b - a;
```

We first execute `a * a / b` , which due to being left-to-right we can separateinto `a * a` and the result `/ b` : `2 * 2 = 4` , `4 / 4 = 1` .

Then we can perform the sum and the subtraction: 4 + 1 - 2. The value of `c` is `3` .

In all cases, however, I want to make sure you realize you can useparentheses to make any similar expression easier to read and comprehend.

Parentheses have higher priority over anything else.

The above example expression can be rewritten as:

```
int c = b + ((a * a) / b) - a;
```

1.3. Comparison operators

Operator	Name	Example
`==`	Equal operator	`a == b`
`!=`	Not equal operator	`a != b`
`>`	Bigger than	`a > b`
`<`	Less than	`a < b`
`>=`	Bigger than or equal to	`a >= b`
`<=`	Less than or equal to	`a <= b`

1.4. Logical operators

- `!` NOT (example: `!a`)
- `&&` AND (example: `a && b`)

- `||` OR (example: `a || b`)

Those operators are great when working with boolean values.

1.5. Compound assignment operators

Those operators are useful to perform an assignment and at the same timeperform an arithmetic operation:

Operator	Name	Example
`+=`	Addition assignment	`a += b`
`-=`	Subtraction assignment	`a -= b`
`*=`	Multiplication assignment	`a *= b`
`/=`	Division assignment	`a /= b`
`%=`	Modulo assignment	`a %= b`

1.6. Miscellaneous operators

1.6.1. The ternary operator

The ternary operator is the only operator in C that works with 3 operands,and it's a short way to express conditionals.

This is how it looks:

```
<condition> ? <expression> : <expression>
```

Example:

```
a ? b : c
```

If `a` is evaluated to `true` , then the `b` statement is executed, otherwise `c` is.

The ternary operator is functionality-wise same as an if/else conditional,except it is shorter to express and it can be inlined into an expression.

1.6.2. sizeof

The `sizeof` operator returns the size of the operand you pass. You can passa variable, or even a type.

Example usage:

```c
#include <stdio.h>

int main(void) {
  int age = 37;
  printf("%ld¥n", sizeof(age));
  printf("%ld", sizeof(int));
}
```