Conditional statement

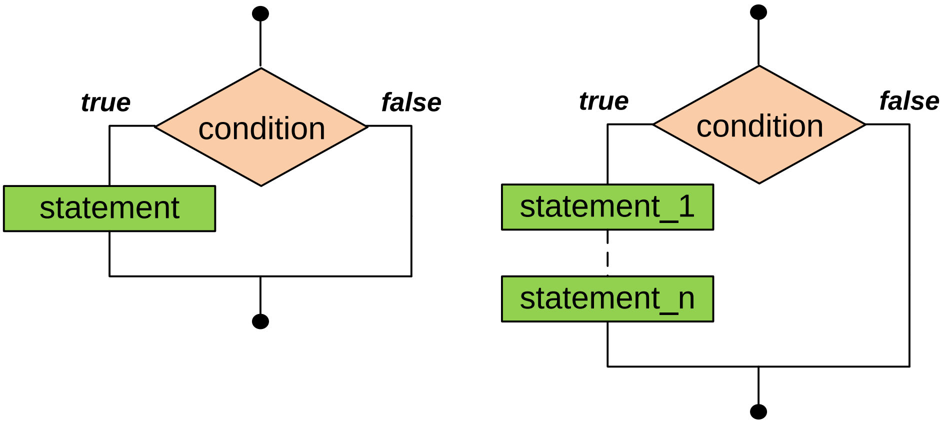
The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer. The *if statement* allows you to control if a program enters a section of code or not based on whether a given condition is true or false. One of the important functions of the *if statement* is that it allows the program to select an action based upon the user's input. For example, by using an *if statement* to check a user-entered password, your program can decide whether a user is allowed access to the program.

An *if statement* consists of a Boolean expression (*condition*) followed by one or more statements:

```
if (condition) statement;
```

```
if (condition)
{
  statement_1;
  . . .
  statement_n;
}
```

If the *condition* evaluates to **true**, then the *if block* will be executed



The next program prints `"Less than 10"` if the input value of *x* is less than 10. If the value of *x* is greater or equal to 10, nothing will be printed.

```
#include <stdio.h>

int x;

int main(void)
{
  scanf("%d", &x);
  if (x < 10) printf("Less than 10\n");
  return 0;
}
```

Real world expression	C notation
if $x > 4$, then . . .	`if (x > 4) . . .`
if $x \geq 4$, then . . .	`if (x >= 4) . . .`

if $x < 6$, then . . .	`if (x < 6) . . .`
if $x \leq 6$, then . . .	`if (x <= 6) . . .`
if $x = 7$, then . . .	`if (x == 7) . . .`
if $x \neq 9$, then . . .	`if (x != 9) . . .`

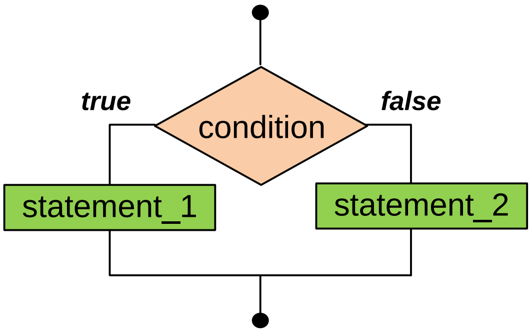
Consider some samples of conditional statements:

`if (x <= 0) y = x + 2;`	if x is less or equal to 0, assign to y the value of x + 2
`if (a == b) y = a + b;`	if a and b are equal, assign to y the sum of a and b
`if (x != a + 3) y = a;`	if x does not equal to a + 3, assign to y the value of a

An *if statement* can be followed by an optional ***else*** statement, which executes when the *condition* is false:

```
if (condition)
  statement_1;
else
  statement_2;
```

If the *condition* evaluates to **true**, then the *if block* will be executed, otherwise, the *else block* will be executed.



```
#include <stdio.h>

int main(void)
{
  int a = 10, b = 4;
  printf("a = %d, b = %d\n",a,b);

  // greater than example
  if (a > b)
    printf("a is greater than b\n");
  else
    printf("a is less than or equal to b\n");

  // lesser than equal to
  if (a <= b)
    printf("a is lesser than or equal to b\n");
  else
```

```
    printf("a is greater than b\n");

  // not equal to
  if (a != b)
    printf("a is not equal to b\n");
  else
    printf("a is equal b\n");

  return 0;
}
```

The next program evaluates the expression:

$$y = \begin{cases} x+4, x<0 \\ x^2, x \geq 0 \end{cases}$$

```
#include <stdio.h>

int x, y;

int main(void)
{
  scanf("%d", &x);
  if (x < 0) y = x + 4; else y = x * x;
  printf("%d\n",y);
  return 0;
}
```

E-OLYMP 8520. Conditional statement - 1 Find the value of y according to condition:

$$y = \begin{cases} x^2 - 3x + 4, \; x < 5 \\ x + 7, x \geq 5 \end{cases}$$

► Use conditional statement. As -1000 $\leq x \leq$ 1000, int type is enough.

E-OLYMP 8521. Conditional statement - 2 Find the value of y according to condition:

$$y = \begin{cases} x^3 + 5x, \; x \geq 10 \\ x^2 - 2x + 4, x < 10 \end{cases}$$

► Use conditional statement. As $x \leq 10000 = 10^4$, then $x^3 \leq 10^{12}$. So we need to use long long type.

E-OLYMP 8612. Conditional statement - 4 Find the value of y according to condition:

$$y = \begin{cases} x^3 + 2x^2 + 4x - 6, \; x \geq 0 \\ x^3 - 7x, x < 0 \end{cases}$$

► Use conditional statement.

E-OLYMP 8613. Conditional statement - 5 Find the value of y according to condition:

$$y = \begin{cases} 3x^3 + 4x^2 + 5x + 6, x \ge 13 \\ 3x^3 - 2x^2 - 3x - 4, x < 13 \end{cases}$$

► Use conditional statement.

E-OLYMP 2606. Minimum and maximum Find minimum and maximum between two positive integers.

► Use conditional statement to compare *a* and *b*. If *a* is bigger than *b*, assign *res* to *a*. Otherwise assign *res* to *b*.

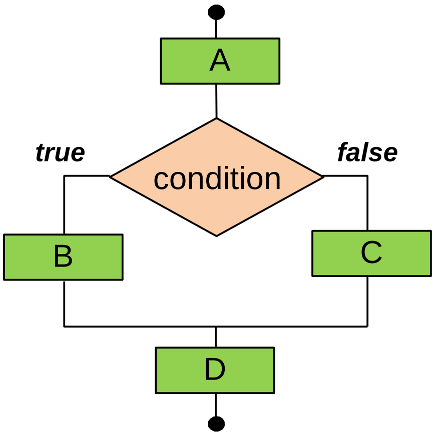
E-OLYMP 8611. Water and Ice The temperature of the air is *t* degrees. Print “Water” if *t* is positive and “Ice” otherwise.

► If *t* > 0 print “Water”, otherwise print “Ice”.

A conditional branch is a statement that causes the program to change the path of execution based on the value of an expression. Consider the following program:

```
int main(void)
{
  // do A
  if (condition)
    // do B
  else
    // do C

  // do D
  return 0;
}
```



This program has two possible paths. If `condition` evaluates to ***true***, the program will execute A, B, and D. If `condition` evaluates to ***false***, the program will execute A, C, and D. As you can see, this program is no longer a straight-line program – its path of execution depends on the value of expression.

Here is a simple program that uses both *if* and *else* block:

```
#include <stdio.h>

int x;

int main(void)
{
  printf("Enter the number: ");
  scanf("%d",&x);
  if (x < 10)
    printf("%d is less than 10\n",x);
  else
    printf("%d is not less than 10\n",x);
  return 0;
}
```

Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration.

operator	*C notation*
x **and** *y*	*x* **&&** *y*
x **or** *y*	*x* || *y*
not *x*	!*x*
x **xor** *y*	*x* ^ *y*

- **Logical AND:** The **'&&'** operator returns *true* when both the conditions in consideration are satisfied. Otherwise it returns *false*. For example, ***a* && *b*** returns *true* when both *a* and *b* are true (i.e. non-zero).
- **Logical OR:** The '||' operator returns *true* when one (or both) of the conditions in consideration is satisfied. Otherwise it returns *false*. For example, ***a* || *b*** returns true if one of *a* or *b* is true (i.e. non-zero). Of course, it returns true when both *a* and *b* are true.
- **Logical NOT:** The **'!'** operator returns *true* the condition in consideration is not satisfied. Otherwise it returns *false*. For example, **!*a*** returns *true* if *a* is *false*, i.e. when *a* = 0.

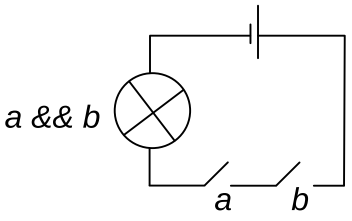
The ***truth tables*** for logical operators are given below:

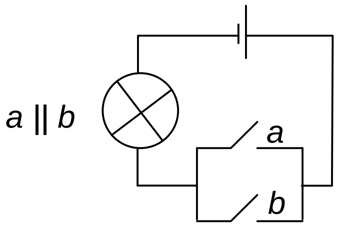
x	*y*	*x* and *y*
0	0	0
0	1	0
1	0	0
1	1	1

x	*y*	*x* or *y*
0	0	0
0	1	1
1	0	1
1	1	1

x	not *x*
0	1
1	0

x	*y*	*x* xor *y*
0	0	0
0	1	1
1	0	1
1	1	0





Check if the value $x \in (1; 5)$:

```
if (x > 1 && x < 5) ...
```

Check if the value $x \in (-\infty; 1] \cup [5; +\infty)$:

```
if (x <= 1 || x >= 5) ...
```

Check if the value $x \in \{3, 4, 8\}$:

```
if (x == 3 || x == 4 || x == 8) ...
```

Check if the value of variables *a*, *b*, *c* are the same:

```
if (a == b && b == c)
```

E-OLYMP 8614. Inside the interval Determine whether the number **x** belongs to the interval [*a*; *b*]. Number *x* belongs to the interval [*a*; *b*] if $a \le x \le b$.

► In C language its *not* possible to write a condition $a \le x \le b$ directly. Use **and** (&&) notation for conditions $a \le x$ and $x \le b$.

E-OLYMP 8615. Outside the interval Determine whether the number *x* is located outside the interval [*a*; *b*]. Number *x* is located outside the interval [*a*; *b*] if either $x < a$ or $x > b$.

► Use **or** (||) notation for conditions $x < a$ and $x > b$.

E-OLYMP 8873. One-digit number Integer *n* is given. Print **Ok**, if *n* is one-digit number and **No** otherwise.

► *n* is one-digit number if $-9 \le n \le 9$. Implement this condition.

Compound Conditional Statement

This allows you to create expressions that contain order-of-precedence grouping without having to use parentheses. The evaluative **or** statement is hidden inside the conditional statement, as long as that conditional statement can evaluate against multiple criteria.

Check if triange with sides *a*, *b*, *c* is right (use Pythagorean theorem: the sum of squares of two sides equals to the square of the third side):

```
if ((a * a + b * b == c * c) ||
    (a * a + c * c == b * b) ||
    (b * b + c * c == a * a))
```

Check if there exists a non degenerate triangle with sides *a*, *b*, *c* (the sum of any two sides must be more than the third side):

```
if (a < b + c && b < a + c && c < a + b)
```

E-OLYMP 8372. Create a triangle Can we construct a triangle from segments of length *a*, *b*, *c*?

► The triangle is non-degenerative if the sum of any two sides is more than the third side.

E-OLYMP 915. Rectangular or not? There is a triangle with sides *a*, *b*, *c*. Is this triangle rectangular?

► The triangle is rectangular if the sum of squares of two sides equals to the square of the third side (Pythagorean theorem).

E-OLYMP 8874. Two-digit number Integer *n* is given. Print **Ok**, if *n* is two-digit number and **No** otherwise.

► *n* is two-digit number if $-99 \le n \le -10$ or $10 \le n \le 99$. We need to write the compound condition:

```
if ((n >= -99 && n <= -10) || (n >= 10 && n <= 99))
```

E-OLYMP 6278. City numbers Determine if the houses with numbers *n* and *m* are located on one side of the street.

► The answer is affirmative if *n* and *m* have the same parity: either both *even* or both *odd*. The conditional statement looks like:

```
if ((n is even and m is even) || (n is odd and m is odd))
```

Second solution is based on the fact that two numbers have the same parity if their sum is even.

E-OLYMP 8864. Numbers of the same sign Determine if numbers *n* and *m* have the same sign.

► Numbers *n* and *m* have the same sign if either they both *positive* or both *negative*. This condition can be simplified: the answer is affirmative if the product of *n* and *m* is positive.

Using if with multiple statements

Note that the ***if*** statement only executes a single statement if the expression is true, and the else only executes a single statement if the expression is false. In order to execute multiple statements, we can use a ***block***:

```
#include <stdio.h>

int main(void)
{
  int x;
  scanf("%d",&x);
  if (x < 10)
  {
    printf("You entered %d\n",x);
    printf("%d is less than 10\n",x);
  }
  else
  {
    printf("You entered %d\n",x);
    printf("%d is not less than 10\n",x);
  }
  return 0;
}
```

Chaining if statements

It is possible to chain ***if-else*** statements together:

$$y(x) = \begin{cases} x+1, x<0 \\ x^2, 0 \le x < 10 \\ x-4, x \ge 10 \end{cases}$$

```
#include <stdio.h>

double x, y;

int main(void)
{
  scanf("%lf", &x);
  if (x < 0) y = x + 1; else
  if (x < 10) y = x * x; else y = x - 4;
  printf("%lf\n", y);
  return 0;
}
```

E-OLYMP 8526. Conditional statement - 3 Find the value of *y* according to condition:

$$y = \begin{cases} x+5, x < -4 \\ x^2 - 3x, -4 \le x \le 7 \\ x^3 + 2x, x > 7 \end{cases}$$

► Use chained ***if-else*** statements.

E-OLYMP 8608. sgn function Find the value of *sgn* function:

$$\operatorname{sgn}(x) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases}$$

► Use chained ***if-else*** statements.

Nesting if statements

It is also possible to nest if statements within other if statements.

Three numbers are given. Find and print the maximum among them.

```
#include <stdio.h>

int a, b, c, max;

int main(void)
{
  scanf("%d %d %d",&a,&b,&c);
  if (a > b)
   if (c > a) max = c; else max = a;
  else
    if (c > b) max = c; else max = b;
  printf("%d\n",max);
  return 0;
}
```

Divisibility

Number *x* is divisible by 2 if the remainder after dividing *x* by 2 is 0:

```
#include <stdio.h>

int x;

int main(void)
{
  scanf("%d", &x);
  if (x % 2 == 0)
    printf("%d is even\n",x);
  else
    printf("%d is odd\n",x);
  return 0;
}
```

Check if the number *x* is divisible by *a* and by *b*:

```
if (x % a == 0 && x % b == 0)
```

E-OLYMP 8371. Even or Odd Given positive integer *n*. Determine is it *even* or *odd*.

► *n* is even if it is divisible by 2.

E-OLYMP 8522. Divisibility Given positive integers *a* and *b*. Check if *a* is divisible by *b*.

► *a* is divisible by *b* means that the remainder after dividing *a* by *b* is 0.

E-OLYMP 8531. Divisibility by numbers Given positive integer *n*. Is is divisible simultaneously by *a* and by *b*?

► Use and (&&) for conditions that *n* is divisible by *a* and *n* is divisible by *b*.

Minimum and maximum

Write a code to find the maximum of four numbers *a*, *b*, *c*, *d*.

Let *res* = max(*a*, *b*, *c*, *d*). Assign initially *a* to res. Then compare each of the next numbers with *res*. If some number is greater than *res*, update *res*.

```
#include <stdio.h>

int a, b, c, d, res;

int main(void)
{
  scanf("%d %d %d %d",&a,&b,&c,&d);
  res = a;
  if (b > res) res = b;
  if (c > res) res = c;
  if (d > res) res = d;
  printf("%d\n",res);
  return 0;
}
```

E-OLYMP 7812. Maximum among four numbers Four numbers *a*, *b*, *c*, *d* are given. Find the maximum among them.

► Let *res* be the maximum. Initialize *res* with *a*. Compare *b*, *c* and *d* with *res* and update *res*.

E-OLYMP 3867. Lazy Misha Three integers t_1, t_2, t_3 are given. Find the minimum among them.

► Use conditional statement to find minimum among three numbers.

Ceiling & floor operations

If x and y are integers, the floor operation $\lfloor x/y \rfloor$ is just simply x / y (integer division). For example 15 / 4 = 3, 15 / 7 = 2.

The ceiling operation can be calculated like $\lceil x/y \rceil = \lfloor (x+y-1)/y \rfloor$. Another way to find *res* = $\lceil x/y \rceil$ is:

- assign `res = x / y;`
- if x is not divisible by y, add 1 to *res*: `if (x % y > 0) res++;`

```
#include <stdio.h>

int x, y, res;

int main(void)
{
  x = 16; y = 3;
  res = x / y;
  if (x % y > 0) res++;
  printf("%d\n",res); // ceil(x/y)
  return 0;
}
```

Ceiling operation can be written in C like

$$\lceil x/y \rceil = x/y + \text{bool}(x \,\%\, y),$$

where bool(x) equals to

- 0 (false), if $x = 0$;
- 1 (true), if $x \neq 0$;

Conditional ?: operator

The ternary operator (?:) is a very useful conditional expression used in C. It's effects are similar to the ***if*** statement but with some major advantages.

The basic syntax of using the ternary operator is thus:

```
(condition) ? (if_true) : (if_false)
```

Which is basically the same as:

```
if (condition)
  if_true;
else
  if_false;
```

The value of a ?: expression is determined like this: `condition` is evaluated. If it is ***true***, then `if_true` is evaluated and becomes the value of the entire ?: expression. If `condition` is ***false***, then `if_false` is evaluated and its value becomes the value of the expression.

The ?: is called a ***ternary operator*** because it requires three operands and can be used to replace if-else statements.

For example, consider the following code:

```
if (y < 10)
  var = 30;
else
  var = 40;
```

Above code can be rewritten like this:

```
var = (y < 10) ? 30 : 40;
```

Here *var* is assigned the value of 30 if *y* is less than 10 and 40 if it is not.

Let’s evaluate the expression

$$y = \begin{cases} x+4, x<0 \\ x^2, x \geq 0 \end{cases}$$

using the ?: operator:

```
#include <stdio.h>

int x, y;

int main(void)
{
  scanf("%d", &x);
  y = (x < 0) ? x + 4 : x * x;
  printf("%d\n",y);
  return 0;
}
```

Find the minimum and maximum of two numbers:

```
#include <stdio.h>

int a, b, min, max;

int main(void)
{
  scanf("%d %d",&a,&b);
  min = (a < b) ? a : b;
  max = (a > b) ? a : b;
  printf("%d %d\n",min,max);
  return 0;
}
```