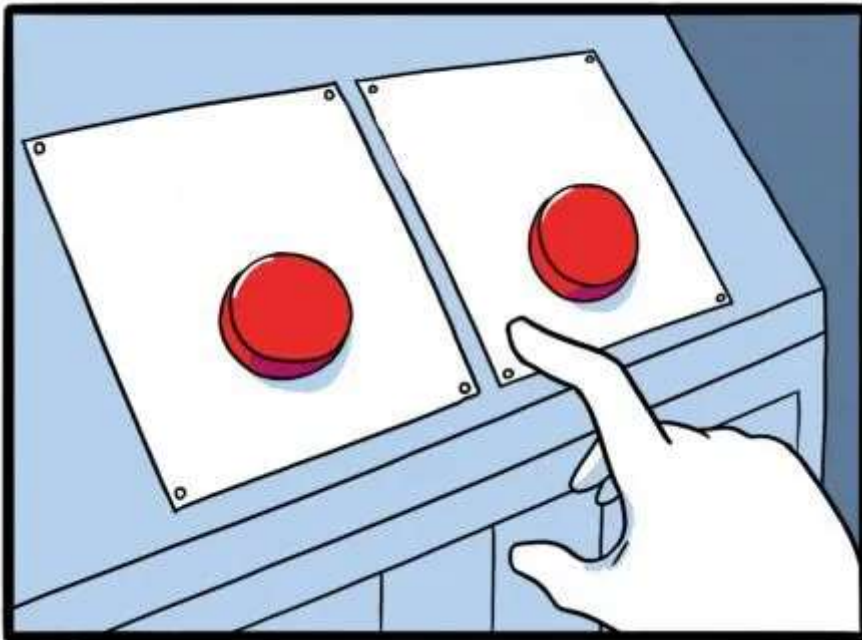


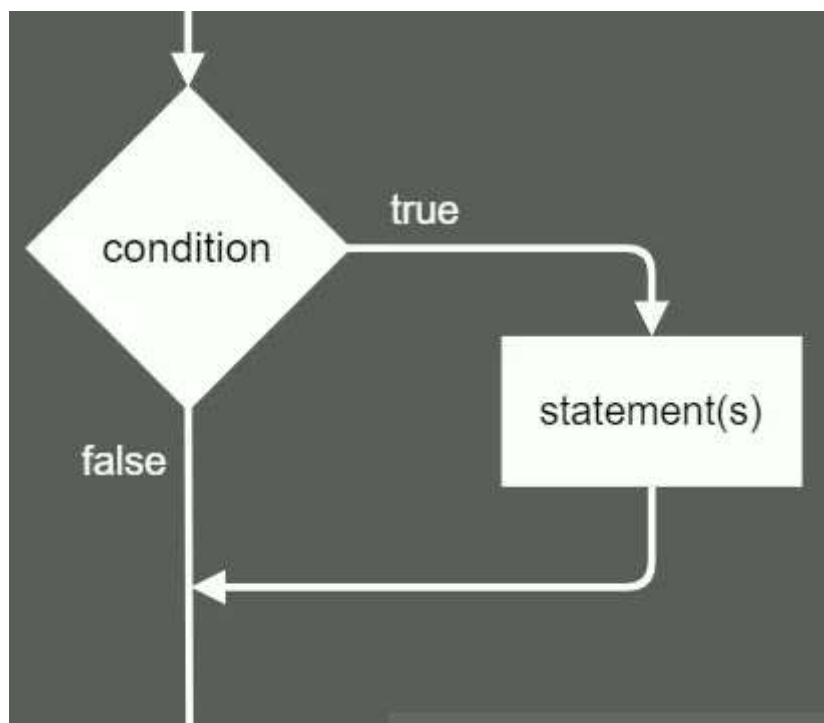
# Conditional Statements

As we discussed in the flowchart (the diamond box), conditional statements are used to define the flow of the program. It is very well explained by this image.



Conditional statements are used for decision making. The process is similar to what we do in real life. If a specified condition is true then a particular block of statements is executed and if not then the other one. Notice how we used 'if' and 'if not' in the previous lines? That's what actually 'if-else' statements are in the world of programming.

**Now that we are approaching the first real programming construct of this blog, one thing you must understand that even though our general syntaxes will be following the C language, these constructs are common to all programming languages in different forms.**



## If statement:

An "if statement" is a well known conditional statement in programming. In an, if statement a condition is given inside the parentheses (), and the code tests and evaluates the condition, and if the condition is true then the if block is executed otherwise not.

Syntax:

```
if(condition)
{
    "Code to be executed if the condition is
    true."
}
```

Let's take an example:

```
13 int num=9;
14
15 if(num>10)
16 {
17     printf("Hello World!");
18 }
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

If the condition is true the code inside the 'if block' will be executed and "Hello World" will be printed. (printf is used to print something to the user in C).

The if statements are usually used in 4 forms if, if-else, if-else ladder, and nested if.

We have understood the If statement above,

## If-else statement:

In if-else statements, the condition is tested and evaluated and if it is true then the block inside the if statement is executed and if it is false then instead of doing nothing, the block inside the else statement is executed.

Let's take an example. Say we have to check whether a given person is above 18 or not and print "Person is above 18" if he/she is and "Person is not above 18" if not.

*Remember: 'scanf' is used to take input from users in the C programming language.*

```
13 int age;
14 printf("Enter age of person: ");
15 scanf("%d",&age);
16
17
18 if(age>=18)
19 {
20     printf("Person is above 18\n");
21 }
22 else
23 {
24     printf("Person is not above 18\n");
25 }
```

```
Enter age of person: 12
Person is not above 18

...Program finished with exit code 0
Press ENTER to exit console.
```

In the above example, the age input given by the user is 12 which is less than 18. So the condition (age >= 18) is false and its block of code is not executed. So the else condition is true and its block of code is executed.

```
13 int age;
14 printf("Enter age of person: ");
15 scanf("%d",&age);
16
17
18 if(age>=18)
19 {
20     printf("Person is above 18\n");
21 }
22 else
23 {
24     printf("Person is not above 18\n");
25 }
```

```
Enter age of person: 12
Person is not above 18

...Program finished with exit code 0
Press ENTER to exit console.
```

When age 20 > 18, if condition is true and its corresponding block of code is executed.

## Nested if:

A "nested if" is an if statement that is the target of another if statement. In other words, nested if statements have an if statement inside another if statement.

```
if(condition1)
{
    // Executes when condition1 is true
    {
        // Executes when condition2 is true
    }
}
```

Try an example:

```
13 int num=10;
14
15     if(num>8)
16     {
17         printf("Hello ");
18
19         if(num>10)
20         {
21             printf("World");
22         }
23     }
24 }
```

Hello

...Program finished with exit code 0  
Press ENTER to exit console.

What will be printed in this case? Will putting num=12 change the output?

Let's check it out.

```
13 int num=10;
14
15     if(num>8)
16     {
17         printf("Hello ");
18
19         if(num>10)
20         {
21             printf("World");
22         }
23     }
24 }
```

```
Hello World
```

```
...Program finished with exit code 0  
Press ENTER to exit console.█
```

## If else ladder:

We know that the if-else statement allows us to check between two possibilities: when the condition is true and when it is false. But what if there are more than two possibilities. Here's where the if-else ladder comes into place.

```
if(condition1)
{
    // Executes when condition1 is true
}
else if(condition2)
{
    // Executes when condition2 is true
}
else
{
    // Executes when no condition is true
}
```

Let's have a look at an example. Suppose we want to grade students on the basis of their marks such that marks greater than 90 are given "A", between 80 - 90 "B" and below 80 "C". Don't worry this is just an example, you won't actually be graded this harshly.

```
13     int marks=85;
14
15     if(marks>90)
16     {
17         printf("Grade : A");
18     }
19     else if(marks>80)
20     {
21         printf("Grade : B");
22     }
23     else
24     {
25         printf("Grade : C");
26     }
```

```
Grade : B
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

**Not that if starting from the top, any condition is found to be true, the rest of the blocks won't be executed, even if their conditions come out to be true.**

Understand this in the context of our example. Say a student has marks 85. Then first the, "if condition" will be checked and found to be false and hence "A", will be not printed. However, "B" will be printed as marks>80 being true. The code will skip the rest of the conditions, once one is found to be true.

**We have now understood the if construct. It is**

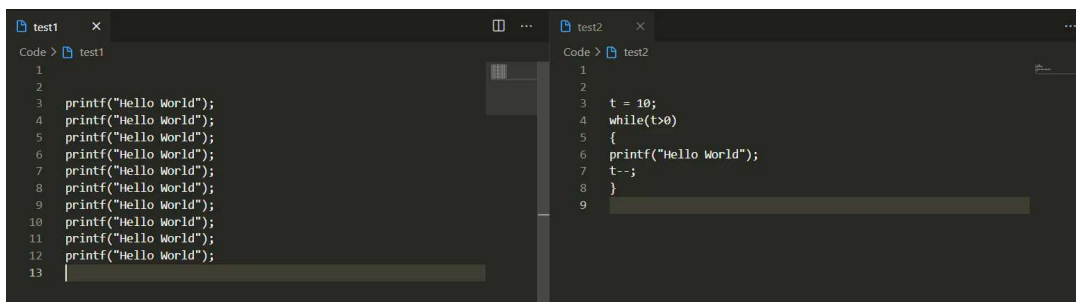


# QUIZ TIME!

```
int zine = 2;
if(!(zine == (1, 2)))
    printf("Hello !");
if(zine>0)
    printf("Hey !");
else
    printf("Hi !");
```

**What should be the output of this code? Note the absence of . What difference does that make? (Do use the concepts of operators we learned previously)**

Before moving to the next programming construct, consider a situation. Say you have to print your name 10 times. How will you do it? Simple, right? Use a print statement 10 times. What about 100 or 1000? Writing the same statement this many times would be a tedious task and a waste of valuable time. This is where loops come in.



```
Code > test1
1
2
3 printf("Hello World");
4 printf("Hello World");
5 printf("Hello World");
6 printf("Hello World");
7 printf("Hello World");
8 printf("Hello World");
9 printf("Hello World");
10 printf("Hello World");
11 printf("Hello World");
12 printf("Hello World");
13

Code > test2
1
2
3 t = 10;
4 while(t>0)
5 {
6 printf("Hello World");
7 t--;
8 }
9
```

Focus on the simplicity and shortness of code on the right side. This will not change much even when we increase the number of required print statements. Don't worry we will discuss the syntax soon. Hence, loops make our work as a coder easier and are an integral part of any programming language.

## Loops



So, Loops are used to repeat a set of statements until a condition is met. Two kinds of loops are commonly used: the for loop and the while loop.

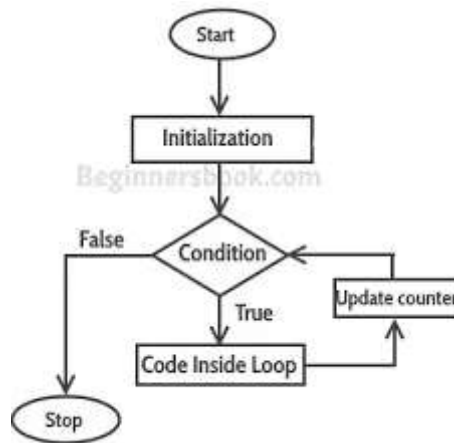
### For Loop

Syntax of for loop:

```
for(initialization statement; condition ; update statement)
```

```
{  
    //statements inside the body of loop  
}
```

How does it work?



The flowchart given above gives us a basic idea about working for loops. For better understanding let's see a few examples.

```
12 {  
13     int i,n=10;  
14  
15     for( i=1/*initialisation*/; i<=n/*condition*/; i++/*incrementation*/)  
16     {  
17         printf("%d ",i);  
18     }  
19     return 0;  
20 }  
21
```

input  
1 2 3 4 5 6 7 8 9 10  
...Program finished with  
Press ENTER to exit cons

- First, we initialized  $i=1$ , this initialization statement is executed only once. Here, we initialize the counter for the loop.
- Then, the condition is evaluated. If the test expression is evaluated to true, statements inside the body of for loop are executed. Here the block of code: `printf("%d", i)` is executed as  $i=1$  which is less than 10.
- The next counter is updated. Here we have to increment  $i$  by  $+1$ .

- Again, the test expression is evaluated. This process goes on until the test expression is false. When the test expression is false, the loop terminates. So all natural numbers from 1 to 10 are printed and when condition  $i \leq 10$  becomes false i.e  $i=11$ , the loop terminates.

Note:

`%d`

**is a format specifier that is used to output an integer in C. Simply put, this is the syntax to print out an integer.**

## While Loop:

Syntax:

```
while(condition)
{
    //statements
    //also change the value here that will make
    the condition false at some point.
}
```

## How does it work?

Here the counter will be initialized before the while loop. A condition is specified in the parenthesis of the 'while' loop that will be checked before each iteration until it becomes false. Make sure to change the value of the counter in the while block otherwise it will become an infinite loop.

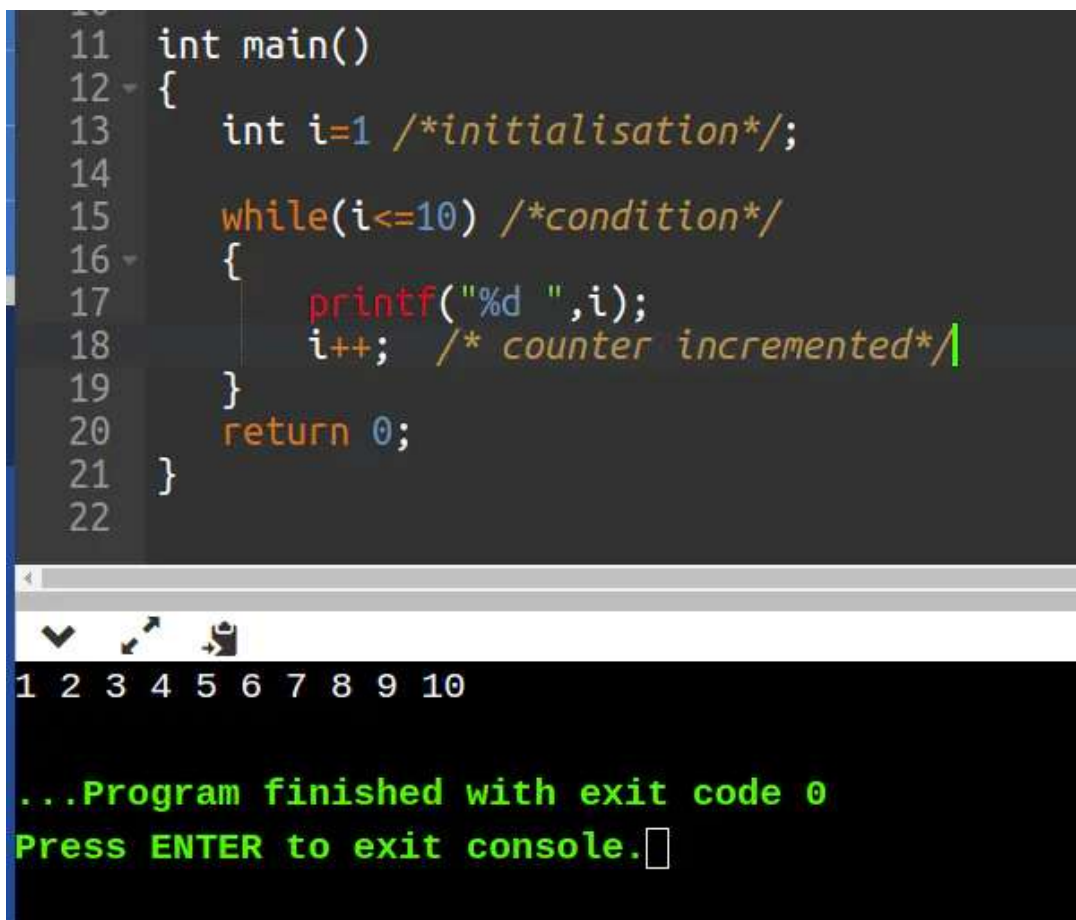
Let's see some basic examples of while loop:

```
int a = 1;
while ( a < 4 )
{
printf ( "Hello World\n" );
a ++;
}
```

---

## Output

Let's print the first 10 natural numbers using a while loop for which we earlier used the for loop.

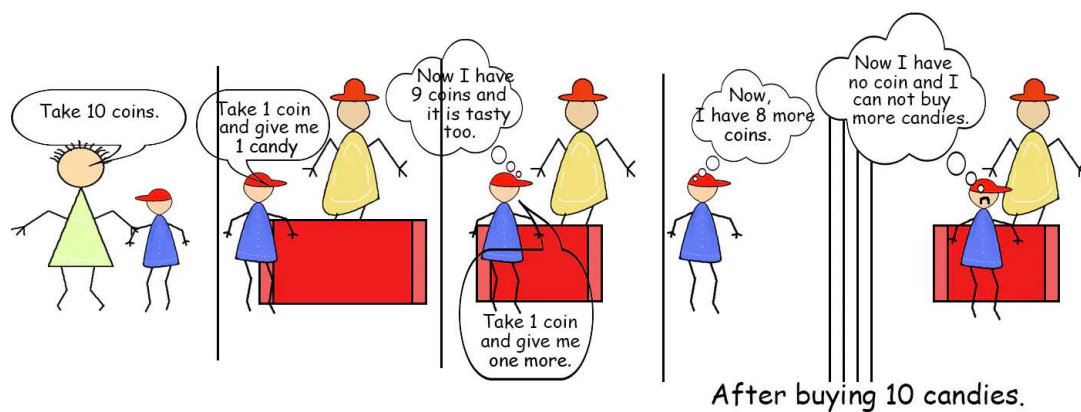
The image shows a screenshot of a code editor and a terminal window. The code editor displays a C program that uses a while loop to print the first 10 natural numbers. The code is as follows:

```
11 int main()
12 {
13     int i=1 /*initialisation*/;
14
15     while(i<=10) /*condition*/
16     {
17         printf("%d ",i);
18         i++; /* counter incremented*/
19     }
20     return 0;
21 }
22
```

The terminal window below the code editor shows the output of the program: the numbers 1 through 10 printed in a single line. Below the output, the terminal displays the message "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor.

**Note that any problem involving loops can be implemented using any of the two loops.**

**Take a look at another illustration of the working of loops.**



The first step is the initialization. Buying candy through coins is the body of the loop and updating of the counter. Finally, when all coins are over, the loop terminates.

Write pseudocode along with the flowchart to store the age of 10 people and count how many of them are above 30 years of age.

Suppose you are driving at a speed of 80km/h. After every 5 min, your speed decreases by 2km/h. Find the maximum distance you can cover at an average speed during the journey. Write pseudocode for this problem and see how you can apply loops here.