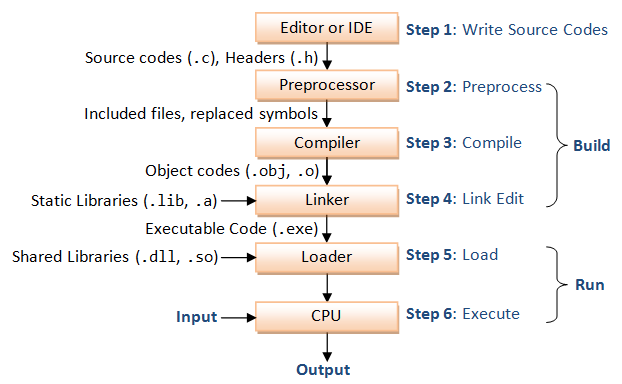
# The process of writing A C program



**Step 1:** Write the source codes (.c) and header files (.h).

**Step 2:** Pre-process the source codes according to the *preprocessor directives*. The preprocessor directives begin with a hash sign (#), such as #include and #define. They indicate that certain manipulations (such as including another file or replacement of symbols) are to be performed BEFORE compilation.

**Step 3:** Compile the pre-processed source codes into object codes (.obj, .o).

**Step 4:** Link the compiled object codes with other object codes and the library object codes (.lib, .a)to produce the executable code (.exe).

**Step 5:** Load the executable code into computer memory.

**Step 6:** Run the executable code.

# C - if...else statement

An **if** statement consists of a Boolean expression followed by one or more statements.

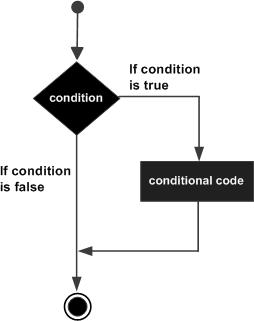
## Syntax

The syntax of an 'if' statement in C programming language is −

if(boolean\_expression) {  
 /\* statement(s) will execute if the boolean expression is true \*/  
}

If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

C programming language assumes any **non-zero** and **non-null** values as**true** and if it is either **zero** or **null**, then it is assumed as **false** value.



#include <stdio.h>  
   
int main () {  
  
 /\* local variable definition \*/  
 int a = 10;  
   
 /\* check the boolean condition using if statement \*/  
   
 if( a < 20 ) {  
 /\* if condition is true then print the following \*/  
 printf("a is less than 20\n" );  
 }  
   
 printf("value of a is : %d\n", a);  
   
 return 0;  
}

When the above code is compiled and executed, it produces the following result −

a is less than 20;  
value of a is : 10

# C - if...else statement

An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

## Syntax

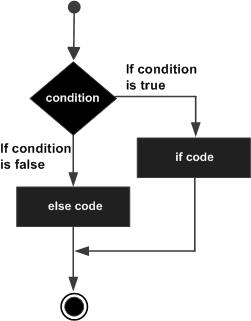
The syntax of an **if...else** statement in C programming language is −

if(boolean\_expression) {  
 /\* statement(s) will execute if the boolean expression is true \*/  
}  
else {  
 /\* statement(s) will execute if the boolean expression is false \*/  
}

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

C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

## Flow Diagram



## Example

#include <stdio.h>  
   
int main () {  
  
 /\* local variable definition \*/  
 int a = 100;  
   
 /\* check the boolean condition \*/  
 if( a < 20 ) {  
 /\* if condition is true then print the following \*/  
 printf("a is less than 20\n" );  
 }  
 else {  
 /\* if condition is false then print the following \*/  
 printf("a is not less than 20\n" );  
 }  
   
 printf("value of a is : %d\n", a);  
   
 return 0;  
}

When the above code is compiled and executed, it produces the following result −

a is not less than 20;  
value of a is : 100

## If...else if...else Statement

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

When using if...else if..else statements, there are few points to keep in mind −

* An if can have zero or one else's and it must come after any else if's.
* An if can have zero to many else if's and they must come before the else.
* Once an else if succeeds, none of the remaining else if's or else's will be tested.

### Syntax

The syntax of an **if...else if...else** statement in C programming language is −

if(boolean\_expression 1) {  
 /\* Executes when the boolean expression 1 is true \*/  
}  
else if( boolean\_expression 2) {  
 /\* Executes when the boolean expression 2 is true \*/  
}  
else if( boolean\_expression 3) {  
 /\* Executes when the boolean expression 3 is true \*/  
}  
else {  
 /\* executes when the none of the above condition is true \*/  
}

### Example

#include <stdio.h>  
   
int main () {  
  
 /\* local variable definition \*/  
 int a = 100;  
   
 /\* check the boolean condition \*/  
 if( a == 10 ) {  
 /\* if condition is true then print the following \*/  
 printf("Value of a is 10\n" );  
 }  
 else if( a == 20 ) {  
 /\* if else if condition is true \*/  
 printf("Value of a is 20\n" );  
 }  
 else if( a == 30 ) {  
 /\* if else if condition is true \*/  
 printf("Value of a is 30\n" );  
 }  
 else {  
 /\* if none of the conditions is true \*/  
 printf("None of the values is matching\n" );  
 }  
   
 printf("Exact value of a is: %d\n", a );  
   
 return 0;  
}

When the above code is compiled and executed, it produces the following result −

None of the values is matching  
Exact value of a is: 100

# C - Loops

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −



C programming language provides the following types of loops to handle looping requirements.

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

## Syntax

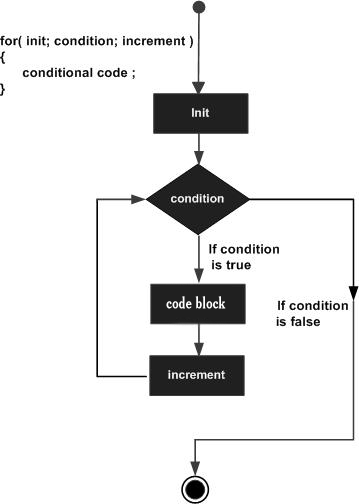
The syntax of a **for** loop in C programming language is −

for ( init; condition; increment ) {  
 statement(s);  
}

Here is the flow of control in a 'for' loop −

* The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
* After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
* The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

## Flow Diagram



## Example

#include <stdio.h>  
   
int main () {  
  
 int a;  
   
 /\* for loop execution \*/  
 for( a = 10; a < 20; a = a + 1 ){  
 printf("value of a: %d\n", a);  
 }  
   
 return 0;  
}

When the above code is compiled and executed, it produces the following result −

value of a: 10  
value of a: 11  
value of a: 12  
value of a: 13  
value of a: 14  
value of a: 15  
value of a: 16  
value of a: 17  
value of a: 18  
value of a: 19