Introduction to Programming

What is Program?

A program is a set of instructions that tells the computer (Central Processing Unit) what to do. Since CPU can't perform any task at its own. Hence set of instructions are provided to CPU in the form of programs to perform specific tasks.

Source code:

Human readable format of a program is called its Source code. We can modify the source code any **number of times.**

Executable:

Binary format or machine readable format of any program is called as its Executable. They are the compiled format of source code and are generally in binary format. We cannot modify the executable of any program.

hello.c

```
#include <stdio.h>

int main()
{
  printf("Hello");
  return 0;
}
```

hello.exe

```
10101100010110101
10001010011000000
00100000100011011
00101110100101110
01111001010010011
11101111111110011
10101010111100001
01010101100100101
11111111100010101
```

Source code Executable

What is Programming?

Programming can be described as the process of writing an algorithm in a sequence of computer language instructions or we may simply say, the process of writing or editing any program. This may seems surprising that programming are in practice from the days when there were no concepts of computer, and the first programmer of the world was Ada Lovelace.

What is Programming Language?

Programming language is a language that provides a medium of communication between computer and human. They provide a way through which humans can interact with the computer with the help of programs. Programming language are often used for writing Programs. Some of popular programming languages are: C, C++, Java, C#, PHP, Javascript, Python, Ruby etc.

Categorization of programming language:

Low Level Language(LLL)

- ✓ Low level languages are programming languages that are directly understood by the computer or requires less interpretation. Or we may say programming language that provides little or no abstraction. Under this category falls
  Machine language and Assembly language.
- ✓ Low level languages are easily understood by computers but are hard to figure out by humans.
- ✓ Processing of any low level languages is much faster than High level languages.
- ✓ Examples: BCPL, ALGOL etc.

Mid Level Language: Assembly

High Level Language(HLL)

- ✓ High level languages are programming languages that are easily understood by humans and are mostly written in English like statements. Or we may say programming languages with strong abstraction.
- ✓ High level languages requires translation into machine language or low level language hence requires compilers, interpreters to accomplish this translation job.
- ✓ Processing of any high level language takes more time compared to low level language because of time elapsed in converting high level instructions into low level instructions.
- ✓ Examples: C, C++, Java, C# etc.

Compiler:

Compilers are the software program that translates or converts the source code into specific machine code. Compilers also checks for syntax errors and ensures proper grammar of the source code. They may or may not convert the source code into an intermediate code called Object code. Full program is compiled at once and on any error compilation process fails.

Interpreter:

Interpreter are the software programs that directly executes instructions written in programming language line by line. They need not any prior compilation of the whole program instead they compile each line when they are about to execute. If any line is encountered with error, then the whole process is terminated and execution of program stops.

Tokens in programming

Tokens:

Smallest individual element of a program is called as Token. Everything you see inside a program is a token. There are generally five types of tokens:

- ✓ Keyword
- ✓ Identifier
- ✓ Operator
- ✓ Separator
- ✓ Literal

Keyword:

Keyword is a reserved word whose meaning is already defined by the programming language. We cannot use keyword for any other purpose inside programming. Every programming language have some set of keywords.

Examples: int, do, while, void, return etc(Note: These keywords are common to C and C influenced languages).

Identifier

Identifiers are the name given to different programming elements. Either name given to a variable or a function or any other programming element, all follow some basic naming conventions listed below:

Keywords must not be used as an identifier.

Identifier must begin with an alphabet(a-z A-Z) or an underscore(_) symbol.

Identifier can contains alphabets(a-z A-Z), digits(0-9) and underscore(_) symbol.

Identifier must not contain any special character(e.g. !@$*.'[] etc) except underscore(_).

Examples of some valid identifiers:

num, Num, _num, _Num, num1, Num1, _num1, _Num1, _1num, _1Num, _num_, number_to_add etc.

Examples of some invalid identifiers:

1num, number to add, 1_num, num-to-add, num@ etc.

Operator:

Operators are the symbol given to any arithmetical or logical operations. Various programming languages provides various sets of operators some common operators are:

Let's suppose two variables a=10, b=5

Arithmetic operator

Arithmetic operator are used to perform basic arithmetic operations.

Operator	Description	Example
+	Adds two operands.	a + b gives 15
-	Subtracts second operand from first.	a - b gives 5
*	Multiplies two operands.	a * b gives 50
/	Divides two operands.	a / b gives 2
%	Modulus operator divides the first operand from second and returns the remainder. It is generally used for checking divisibility.	a % b gives 0 (As 10/5 will have 0 remainder)

Assignment operator

Assignment operator is used to assign value to a variable. The value is assigned from right to left.

Operator	Description	Example
=	Assigns value from right operand to left operand.	a = 10 will assign 10 in a

Relational operator

Relational operator is used to check relation between any two operands. Whether any of them is greater, equal or not equal.

Operator	Description	Example
<	If value of left operand is greater than right, returns true else returns false	(a > b) will return true

>	If value of right operand is greater than left, returns true else returns false	(a < b) will return false
= =	If both operands are equal returns true else false	(a == b) will return false
!=	If both operands are not equal returns true else false.	(a != b) will return true
>=	If value of left operand is greater or equal to right operand, returns true else false	(a >= b) will return true
< =	If value of right operand is greater or equal to left operand, returns true else false	(a <= b) will return false

Logical operator

Logical operator are used to combine two boolean expression together and results in a single boolean value according to the operand and operator used.

Operator	Description	Example
&&	Used to combine two expressions. If both operands are true or Non-Zero, returns true else false	((a>=1) && (a<=10)) will return true since (a>=1) is true and also (a<=10) is true.
||	If any of the operand is true or Non-zero, returns true else false	((a>1) || (a<5)) will return true. As (a>1) is true. Since first operand is true hence there is no need to check for second operand.
!	Logical NOT operator is a unary operator. Returns the complement of the boolean value.	(!(a>1)) will return false. Since (a>1) is true hence its complement is false.

Increment/Decrement operator

Increment/Decrement operator is a unary operator used to increase an integer value by 1 or decrease it by 1. Increment/decrement operator are of two types Postfix and Prefix.

Operator	Description	Example
++	Increment operator will add 1 to an integer value.	a++ will give 11 ++a will also give 11
--	Decrement operator will subtract 1 from an integer value.	a-- will give 9 --a will also give 9

Separator

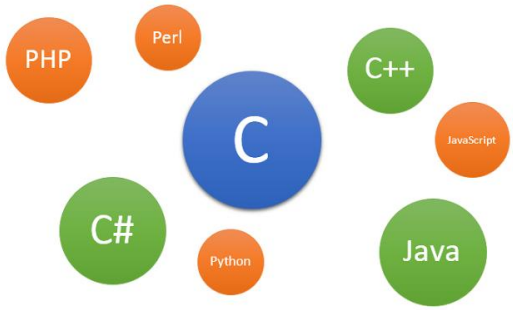
Separators are used to separate different programming elements. The various types of separators used in programming are:

(Space) \t(Tab) \n(New line) . , ; () {} []

Introduction to C programming

C PROGRAMMING LANGUAGE

C is a procedural programming language developed by Dennis Ritchie in 1972 at AT&T Bell Labs. C is considered as the most powerful programming languages ever. It was originally developed to re-implement UNIX operating system. C is a High Level Programming language but often referred as Middle level programming language because of its ability to provide rich support to low level programming.
C is also called as mother of all modern programming languages as many of current programming languages such as C++, Java, C#, PHP, Python, Perl, JavaScript etc. are directly or indirectly influenced from C. It is the also the most popular programming language ever and is widely used for Operating Systems implementations and Embedded programming's.

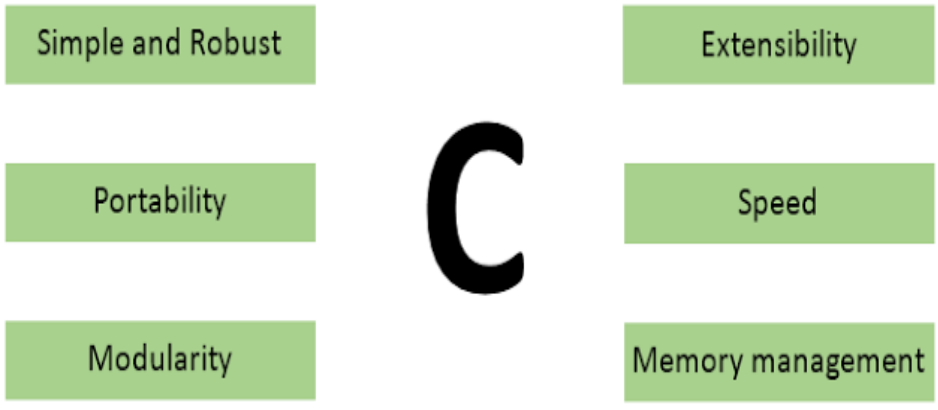


Languages originated from C

FEATURE OF C

C is powerful language and its real power comes from number of features it provides. Here are some features that C provides:

- Simple and Robust
- Portability
- Modularity
- Extensibility
- Speed
- Memory management



Features of C programming language

- **Simple and Robust**
  C is considered as simplest and easiest language ever. Its simplicity lies in the lesser number of programming constructs that are used to create any complex program.
  C is a robust programming language with rich set of in-built library functions and operators that can even write the most complex programs ever.

- **Portability**
  C is a machine independent programming language i.e. C program can run on any machine that has C compiler with few or no modification. Although C is doesn't provide platform independency as Java provides.

- **Modularity**
  C programs are modular in nature i.e. they are basically divided into modules by modules. **Modular programming** is a programming construct where we divide our program into different modules and combine them together in a single module to form a Program.

- **Extensibility**
  C provides a rich set of in-build library functions. But we can also write our custom library function and attach with the C programs to use them in future.

- **Speed**
  C programs compiles and executes faster than any other modern programming languages. And this is also the fact that it is used for Operating System development.

- **Memory management**
  C provides an efficient way to access and manage memory through the use of Pointers.

ADVANTAGES OF LEARNING C

Whether you are beginning your programming with C or you have learnt some other programming languages earlier. Learning C has its own importance, here are few advantages of C, over other programming languages.

- C is much simpler language as compared to any other programming language. It has less number of constructs which makes easier to learn.
- Almost every hardware have a C compiler.
- Compilation and execution of a C program is way faster than other modern programming languages.

- Being a high level programming language, its ability to support low level operations makes it different from other programming languages.
- C language is considered as a backbone of the **Operating System** development. As all popular OS are fully or partially written in C.
- C language is also widely used for developing **Compilers, Assemblers, Language interpreters, Device drivers, Databases** etc.
- All modern programming languages are directly or indirectly inherited from C. Hence learning C will make easier to learn C influenced programming language.

WHAT ARE TOKENS

Tokens are the smallest individual elements of a program. Tokens are the basic building blocks of any program.
For example let's consider this English paragraph

"Programming in C is really very interesting! I am programming since I was 14. "

The above paragraph is made of several basic fundamentals things. Can you figure out all those fundamentals things?
Actually the paragraph contains
13 words, 2 digits, 13 blank spaces, 2 special character (Exclamation mark and full stop)
These are the basic building blocks of any English paragraph. Likewise five fundamental things are used to construct any simple or complex program that are **keywords, identifiers, operators, separators, literals.**

- **Keywords**
  Keywords are the reserved words whose meaning is pre-defined by the programming language specification. They convey some special meaning in the programming.
  Let's take an example of C keyword: **int** in C is used to declare an integer type variable.
  C has total 32 keywords.

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Identifier

In programming we need some data to work upon, since program without data is worthless. And if we have defined some data in memory then there must be some easiest way to access them later.

As the name sounds Identifier might be used for identifying something. Yes you are right, identifiers are used for identifying various programming elements (they might be some variables, or some function or any other programming element). Or in simple words Identifiers are the name given to various programming element.

There are some rules that must be followed by an identifier:

- It must not be a keyword.
- It can contain alphabets a-z A-Z, digits 0-9, and underscore _ symbol.
- It must not contain any special character other than_.
- It must begin with an alphabet or underscore _ symbol.

Examples of some valid identifiers: num, num1, _num1, first_name, _name_
Examples of some invalid identifiers: 1num, first name, email@id, name.

Operators

Operators are symbols given to any mathematical or logical operation. C language provides a rich set of operators. There are basically seven types of operators in C:

- Arithmetic operator **:** + - * / %
- Assignment operator**:** =
- Relational operator**:** > , < , >=, <=, = =, **!=**
- Logical operator **:** **&&** || **!**
- Bitwise operator **:** >> << **&** | ^ ~
- Increment/Decrement operator **:** ++ --
- Conditional/Ternary operator **: ? :**
- Miscellaneous operator sizeof()

- **Separators**
  Separators are used for separating various programming element from one another. Separators in programming are as vital as they are in English paragraphs, as I don't think anyone could ever read an English sentence clearly without separator (Spaces) and punctuation marks. Various types of separators used in C programming are:

  . , : ; ( ) { } [ ] White spaces (Space, \t, \n)

  Amongst all semicolon ; is the most important one as you will find everywhere in all C programs. Semicolon ; is used to terminate any C statement and is also used for separating one statement from another.

- **C CHARACTER SET**
  C character set defines the valid set of alphabets, digits and special characters that are allowed in C. C character set consists of:

- Alphabets: a-z A-Z
- Digits: 0-9
- Special characters: ` ~ ! @ # $ % ^ & * ( ) _ - + = { } [ ] | \ : ; " ' < , > . ? /

Data types in C programming

Data type defines the type of the data that any variable will store. It is a system of declaring variables of different types. Data types are broadly categorized into three categories which are:

1. Primitive/Pre-defined
2. User defined
3. Derived

Primitive/Pre-defined data type:

Primitive data types are the basic data types that are needed for performing general operations.
Example: int, char, float, double

User defined data type:

User defined data types are defined by user according to their need. They are defined or created from the primitive data types.
Example: struct, union, array, pointer

Derived data types:

Derived data types are basically the primitive data type with increased range of values it can store.
Example: short, long long int, double Below is a list of C data types:

Data type	Size	Range	Format specifier
char	1 byte	-127 to +127	%c
unsigned char	1 byte	0 to 255	%c
short	2 byte	-32,767 to 32,767	%hi, %d, %i
unsigned short	2 byte	0 to 65,535	%hu, %d, %i
int	4 bytes 10^4	-2,147,483,847 to +2,147,483,847	%d, %i, %u
unsigned int	4 bytes	0 to 4,294,967,295	%u
long	4 bytes	-2,147,483,847 to +2,147,483,847	%d, %i, %u, %ld, %l, %li
unsigned long	4 bytes	0 to 4,294,967,295	%lu, %u
long long	8 bytes 10^18	-9,223,372,036,854,775,807 to +9,223,372,036,854,775,807	%lli
unsigned long long	8 bytes	0 to 18,446,744,073,709,551,615	%llu
float	4 bytes	1.2E-38 to 3.4E+38	%f
double	8 bytes	2.3E-308 to 1.7E+308	%lf
long double	10 bytes	3.4E-4932 to 1.1E+4932	%Lf

Hello World Program in C

Here we will write our first C program. A Hello World program.

Program:

```
1       #include <stdio.h>
2
3       int main()
4       {
5          printf("Hello World!");
6          return 0;
7       }
```

Know it:

- #include <stdio.h> is a pre-processor directive. When the above program is compiled the compiler replaces the first line with entire contents of stdio.h file. The stdio.h (Standard Input/Output) is called as header file and contains functions for basic input/output operations.
- int main() is a function and defines the starting point of our program. It contains two parts:
  int It is the return type of the function. It specifies that the main() function should return an integer value when its work is finished.
  main() It is an identifier or name of the function.
  Note: The starting point of any program in C is defined by main function.
- The next line contains pair of curly braces. In C we use a pair of curly braces to define a block of statements. An opening curly brace is always followed by a closing curly brace.
- printf(); is a function used to print any data on screen. Anything inside the double quotes " " will be printed as it is on screen.
- return 0; returns an integer value to the operating system or the Runtime environment of C acknowledging OS/C Runtime Environment that the program has terminated successfully.

If you have written the above program same as in the source code without making any mistake then it should print Hello World! on the screen.

Variables and constants in C

Variables:

Variables are the storage location in memory that holds some value. We define a variable to make sure a stored value in memory is accessible by user later. Variables are accessed by their identifiers (Symbolic name that refer to the value stored in the memory) to which they are linked with. In order to use any variable and the value it contains, the variable must be declared first.

Declaring variables in C:

Syntax: data-type variable-name;
Here data-type refers to valid C Data type.
And variable-name refers to valid C identifier.
Note: In C all variables must be declared on the top.

Example:

int num1;

The above code in C will declare an integer variable in memory with Garbage value. Garbage value is a random value of specific type (int in our case) stored in the variable. Hence, variables in C must be initialized prior to their use.

Initializing a variable:

Assigning a value to the variable is called as initializing a variable. Values are initialized or assigned to any variable using assignment operator =.
Example:

int num1 = 10;

Above code will now declare an integer variable num1 in memory with 10 stored in it.

Declaring multiple variables:

Multiples variables in C can be declared one after another declaration.

```
int num1;
int num2 = -30;
unsigned int num3 = 10;
float num4 = 1.2;
char ch = 'a';
```

The above code will declare an integer variable num1 with garbage value, num2 with value -30, an unsigned integer variable with value 10, float variable num4 with value 1.2 and a character variable ch with value 'a'.
Note: Values of character variable must be assigned inside a single quote. And must only contain single character.

Declaring multiple variables of same type:

Multiple variables of same type can be declared and initialized by separating them with comma,.

int num1, num2, num3;
int num4 = 10, num5, num6 = -1290;

The above code will declare 6 integer variables num1, num2, num3, num5 uninitialized containing garbage values and num4 with value 10, num6 with value -1290.

Taking input from user using scanf() function

We use scanf() function in C to take formatted input from user. It is defined in stdio.h header file. Here we will see how to take simple inputs from scanf() function.

Syntax:

int scanf(const char * format, ...);

How to Use:

scanf("format-specifier", &variable-name);

Here **format-specifier** denotes the format specifier of input. And **variable-name** specifies the name of variable in which input is to be taken. The variable name is prefixed by &(Address of operator) which returns the address of the variable. If & is somewhat confusing you leave it we will see in-depth use of & later in pointers. Lets see basic input program.

Program:

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int num;
6
7     printf("Enter any number : ");
8     scanf("%d", &num);
```

```
9
10      printf("Number = %d", num);
11
12      return 0;
13  }
```

Output:

Enter any number: 120
Number = 120

Taking multiple inputs:

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int num1, num2,num3;
6      float f1, f2;
7      printf("Enter two number:\n");
8
9      //Multiple scanf() can be used to take multiple inputs.
10     scanf("%d", &num1);
11     scanf("%d", &num2);
12
13     printf("Enter a number and two float values:\n");
14
15     //All inputs can also be taken in a single scanf()
16     scanf("%d%f%f",&num3,&f1,&f2);
17
18     printf("Num1 = %d, Num2 = %d, Num3 = %d\n", num1,num2,num3);
19     printf("f1 = %f, f2 = %f\n",f1,f2);
20     return 0;
21  }
```

Output:

Enter two number:
12
10
Enter a number and two float values:
130
1.2
1.6
Num1 = 12, Num2 = 10, Num3 = 130
f1 = 1.200000, f2 = 1.600000

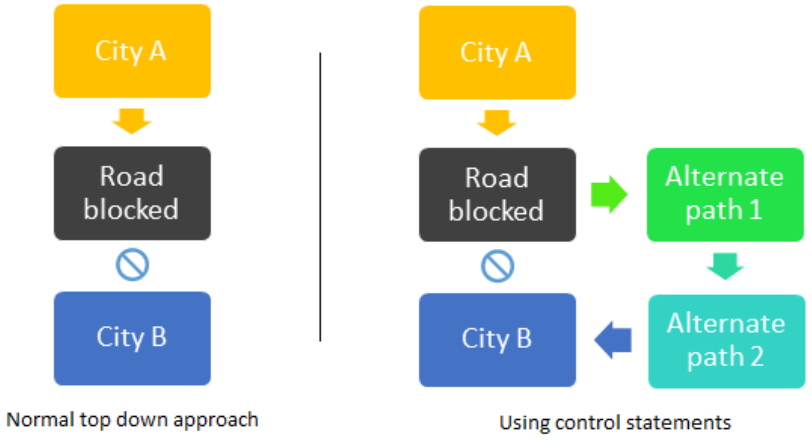
Control Statements in C

Control statements are the heart of any programming language. And as a developer I can say that almost 70-80 percent of the program contains control statements. Hence mastering over control statements means mastering the art of programming. Let's gain control over control statements.

WHAT ARE CONTROL STATEMENTS

Since program are a sequence of statements that are executed linearly one after another. Sometimes we need to change the normal program's flow and here control statements comes in action.
Control statements are statements that are intended to change the normal program's flow.
To understand better let's take an example: Suppose we are travelling from city A to city B. There can be two approaches used to travel from A to B:

1. First approach (Uses normal program flow) will travel until he finds any obstacle(Road block). In case of any road blocks it terminates the travel program.
2. The second approach (Uses control statements) will travel until he finds any obstacle (Road block). If he finds any road blocks it checks for an alternate path. If an alternate path is available then selects the alternate path to city B else terminates the travel program.



As you can see that control statements are highly useful in certain circumstances.
There are basically three types of control statements:

- Conditional statements
- Repetitive statements
- Jump statements

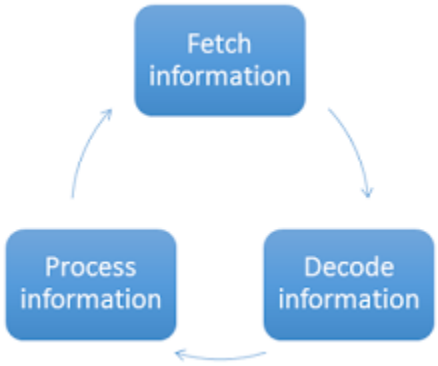
Conditional statements:

Conditional statements are also called as ***If-Else*** Statements. They checks for one or more conditions and if the condition is satisfied i.e true one or more statements are executed. They are most commonly used statements in programming. And if any consequence occurs in programming to run any condition based statement (Statements that requires some condition to be fulfilled prior to their execution) we use conditional statements.
Conditional statements can be of five types:

- If else
- Switch statement

Repetitive statements:

As the name specifies repetitive statements are used to repeat something. Repeat something but what, repetitive statements are used to repeat one or more statements n times. But whats the need of repetition, the need for repeating statements can be explained via a general and basic example.
Let's take an example of functioning of a processor:



Above diagram illustrates the core functioning of a processor **Fetch-Decode-Execute**, this core functioning needs to run till we shut down our computer. Hence requires any mechanism to repeat these three core functioning of processor until the computer is awake. And here **Repetitive statements** comes in action.
There are basically three types of repetitive statements provided in C but languages like C++, Java, C# etc provides an extra iterative statement.

- For loop
- While loop
- Do-while loop

Jump statements:

Jump statements are also one of the important control statements and are sometimes used in combination with repetitive statements. Unlike conditional and repetitive statements jump statements provide an unconditional way to transfer the program's flow from one part to another. There are three types of jump statements:

- Break
- Continue
- Goto

ADVANTAGES OF USING TERNARY OPERATOR

- Reduces the code to single line or just few lines.
- Best for short conditions.
- If used with proper formatting it can more appropriate than simple if else

If else statement in C

If else statements are the conditional control statements. They are used to perform any condition based operation. For example: Program to print a number if it is even, requires a condition to be checked before the number is printed.

If else statements works on a boolean value. If the boolean value is true then the statements under if are executed else they are skipped. In C any non-zero integer value is treated as true and zero is treated as false.

Basically there are four types of if-else:

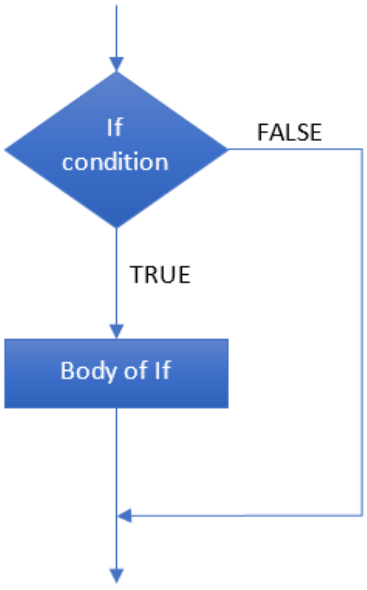
- Simple If
- If-Else
- If-Else-If (Ladder if)
- Nested if

Simple if:

As the name specifies Simple If is the simplest version of If. They contain single condition expression needs to be checked and if the condition is satisfied (true) one or more statements are executed. They do not contain any statement for else part (Part if the condition fails).

Syntax

```
if (condition)
{
   statement/s that will execute if the condition is true.
}
```



Flow chart of Simple if statement

Let's see a simple program to understand:

Write a C program to input any number and print Hello, World! if the number is greater than 1.

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n;
6      printf("Enter any number: ");
7      scanf("%d", &n);
8
9      if( n>1 ) //Checks the condition that n > 1 or not.
10     {
11        printf("Hello, World!");
12     }
13
14     return 0;
15  }
```

Output 1:

Enter any number: 10
Hello, World!

Output 2:

Enter any number: 0

In the output 1 users enters 10, that is greater than 1 hence the if condition will execute like
if (10>1) which is true
Since the if condition is true body of if will execute and prints "Hello, World!".

In output 2 user enters 0, that is less than 1 hence the if condition will execute like
if (0>1) which is false
Since the if condition is false the body of if is skipped printing nothing.

Note: You can also write if else without curly brace if there is only one statement under if else.
Example:

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n = 5;
6
7      if(n>0)
8      {
9         printf("This is positive number");
```

```
10      }
11      return 0;
12  }
```

Above program can also be written as

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n = 5;
6
7      if(n>0)
8         printf("This is positive number");
9
10     return 0;
11  }
```

Since it contains only one statement under if-else hence curly brace { } can be removed.

Let's consider another example

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n = 5;
6
7      if(n>0)
8      {
9         printf("This is positive number");
10        printf("More things to be done");
11     }
12     return 0;
13  }
```

In above program both printf() will be executed only and only if n>0
Let's see what happens when we remove curly brace{ }

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n = 5;
6
7      if(n>0)
8         printf("This is positive number");
9         printf("More things to be done");
10     return 0;
11  }
```

When considering the above program the first printf() will be executed only and only if n>0 But the second printf() will always execute whether the if condition is true or false, as it is not the part of if statement.

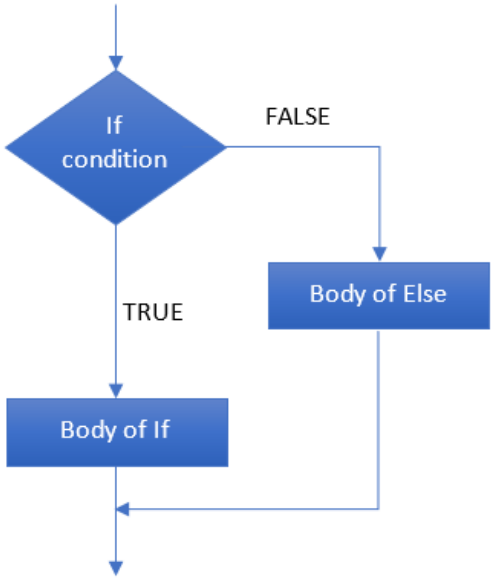
Hence you should be careful while using curly braces { }.

Using curly brace { } with if else is also consider as a Good programming practice.

If-else:

If else statement are the enhanced version of simple if. In simple if one or more statement/s are executed if the condition is true but doesn't describes any statement/s, if the condition is false. If-else describes statements for both whether the condition is true or false. In both case one or more statement/s are executed.
If the condition is true body of if is executed skipping the body of else.
If the condition is false body of else is executed skipping the body of if.

Syntax of if elseif (condition)
{
   statement/s that will execute if the condition is true.
}
else
{
   statement/s that will execute if the condition is false.
}



Flow chart of if-else statement

Here is a sample program of if else:

Write a program to input any number and check whether it is negative or positive.

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n;
6      printf("Enter any number: ");
7      scanf("%d", &n);
8
9      if( n<0 ) //If any number is less than 0 it is negative
10     {
11        printf("Negative");
12     }
13     else //If any number is not less than 0 it is not negative.
14     {
15        printf("Positive");
16     }
17
18     return 0;
19  }
```

Output 1:

Enter any number: -5
Negative

Output 2:

Enter any number: 6
Positive

In output 1, when user enters -5 that is less than 0. The if statement will execute like if(-5 < 0) which is true executing the body of if and skipping the body of else. Hence will print Negative.

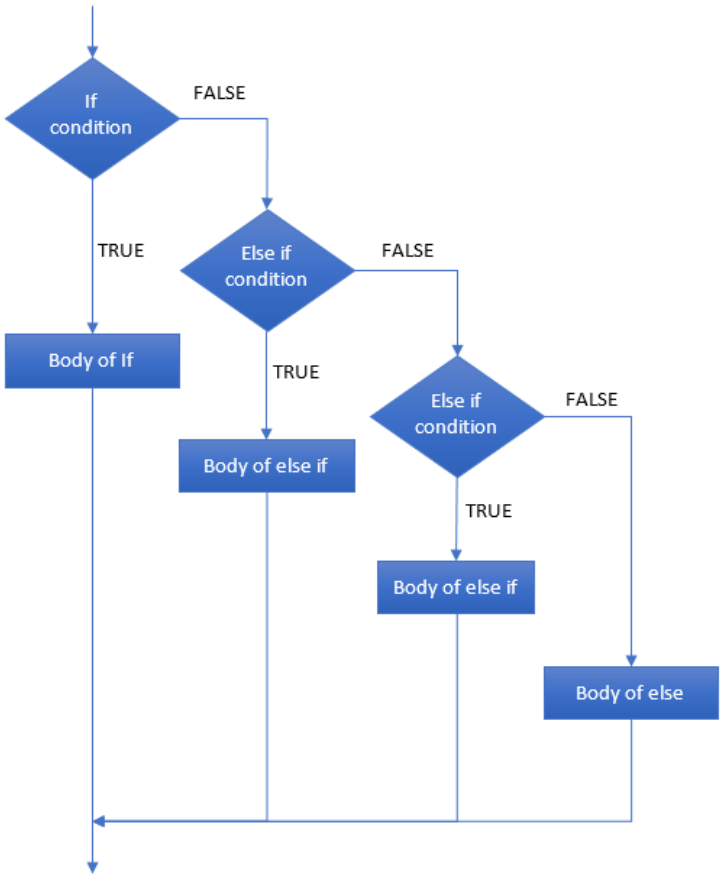
In output 2, when user enters 6 that is greater than 0. The if statement will execute like if(6 < 0) which is false executing the body of else and skipping the body of if. Hence will print Positive.

If-else-if (Ladder if):

If-else-if statements are also called as ladder if due to their control flow. They are generally the enhanced version of if-else.

Syntax of ladder ifif (condition1)
{
   statement/s that will execute if condition 1 is true.
}
else if (condition2)

```
{
   statement/s that will execute if the condition 2 is true.
}
else if (condition3)
{
   statement/s that will execute if the condition 3 is true.
}
...
...
...
else if (condition n)
{
   statement/s that will execute if the condition n is true.
}
else
{
   statement/s that will execute if all conditions are flase.
}
```



Flow chart of if-else-if (ladder if)

Simple example of ladder if

Wite a program to input any number between 1-7 and print equivalent day name. For example: 1-Monday, 2-Tuesday etc.

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n;
6      printf("Enter day number between 1-7: ");
7      scanf("%d", &n);
8
9      if( n == 1)
10     {
11        printf("Monday");
12     }
13     else if(n==2)
14     {
15        printf("Tuesday");
16     }
17     else if(n==3)
18     {
19        printf("Wednesday");
20     }
21     else if(n==4)
22     {
23        printf("Thursday");
24     }
25     else if(n==5)
26     {
27        printf("Friday");
28     }
29     else if(n==6)
30     {
31        printf("Saturday");
32     }
33     else if(n==7)
34     {
35        printf("Sunday");
36     }
37     else
38     {
39        printf("Invalid input");
40     }
41
42     return 0;
43  }
```

Output 1:

Enter day number between 1-7: 4
Thursday

Output 2:

Enter day number between 1-7: 8
Invalid input

In output 1 when user enters 4 the if statements will be executed like
if (4==1) which is false hence moves to the next condition
if (4==2) which is again false hence moves to the next condition
if (4==3) which is also false hence moves to the next condition
if (4==4) which is true hence statements of condition 4 will execute and will print Thursday

In output 2 when user enters 8 the if statements will execute like
if (8==1) which is false hence moves to the next condition
if (8==2) which is also false hence moves to the next condition
if (8==3) which is also false hence moves to the next condition
if (8==4) which is also false hence moves to the next condition
if (8==5) which is also false hence moves to the next condition
if (8==6) which is also false hence moves to the next condition
if (8==7) which is also false hence moves to the next condition
Since all if conditions are false hence the body of else will execute which will print Invalid choice.

Nested if

Nested if is the combined version of all if and is the most complex format of all if. In nested if more than one conditions are checked one after another to executed a specific statement.
Syntax of nested if:if (condition1)

```
{
   statement/s that will execute if the condition 1 is true.
   if (condition2)
   {
      statement/s that will execute if both condition 1 and condition 2 are true.
   }
   else
   {
      statement/s that will execute when condition 1 is true but condition 2 is false.
   }
}
else
{
   statement/s that will execute if the condition 1 is false.
```

```
    {
        Statement/s that will execute if condition 1 is false but condition 2 is true.
    }
    else
    {
        statement/s that will execute when both condition 1 and condition 2 are false.
    }
}
```

Let's see a program to demonstrate nested if:

Write a program to input three numbers and find maximum between all.

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int n1, n2, n3;
6      printf("Enter any three numbers :\n");
7      scanf("%d %d %d", &n1, &n2, &n3);
8
9      if(n1>n2)
10     {
11        if(n1>n3)
12        {
13           printf("n1 = %d is max.", n1);
14        }
15        else
16        {
17           printf("n3 = %d is max.", n3);
18        }
19     }
20     else
21     {
22        if(n2>n3)
23        {
24           printf("n2 = %d is max.", n2);
25        }
26        else
27        {
28           printf("n3 = %d is max.", n3);
29        }
30     }
31
32     return 0;
33  }
```

Output 1:

Enter any three numbers:
30
12
10
n1 = 30 is max.

Output 2:

Enter any three numbers:
10
20
9
n2 = 20 is max.

Output 3:

Enter any three numbers:
10
20
30
n3 = 30 is max.

In output 1 user enters n1=30, n2=12, n3=10. The if statements will execute like.
The first outer if condition
if (n1>n2) i.e (30>12) is true. Hence the first if i.e. outer if will execute skipping the outer else part.
Then the inner condition
if (n1>n3) which is also true. Hence the body of inner if will execute. And prints n1 = 30 is max.

In output 2 user enters n1=10, n2=20, n3=9. The if statements will execute like.
The first outer if condition
if (n1>n2) i.e. (10>20) is false. Hence the first outer if will be skipped and the outer else part will execute.
Then the inner condition
if (n2>n3) i.e. (20>9) which is true. Hence the inner if will execute skipping the inner else part. And prints n2 = 20 is max.

In output 3 user enters n1=10, n2=20, n3=30. The if statements will execute like.
The first outer if condition
if (n1>n2) i.e. (10>20) is false. Hence the first outer if will be skipped and outer else part will execute.
Then the inner condition
if (n2>n3) i.e. (20>30) is also false. Hence the inner if part will also be skipped executing the inner else part. And prints n3 = 30 is max.

Conditional / Ternary operator in C

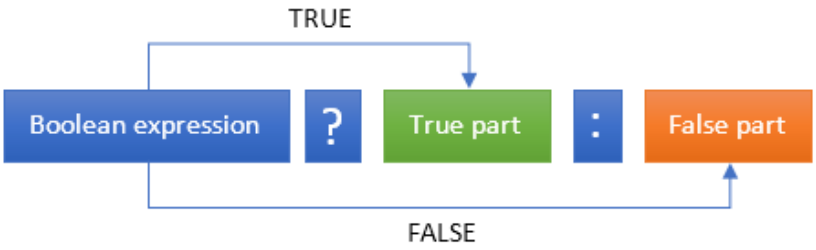
Ternary operator is a shorthand operator for if else and can be used to replace short if else statements in single line. Ternary operator is also known as conditional operator, inline if, ternary if.

As the name specifies ternary operator consist of three operand

- First is the boolean expression (condition) upon which the ternary operator works.
- Second is the statements which are executed if the boolean expression is true, and may be treated as body of if.
- Last is the statements which are executed if the boolean expression if false, and may be treated as body of else.

Syntax:

Basic syntax of ternary operator
(boolean expression) ? (true-part) : (false-part)



Let's see a basic example of ternary/conditional operator:
max = (a>b) ? a : b
The above expression returns the maximum value between a and b and stores in max. If the boolean expression is true i.e. if (a>b) then the expression will return a, else the will return b.

The above code is similar to

```
if(a>b)
{
   max = a;
}
else
{
   max = b;
}
```

Note: In C any non-zero integer value represents true and zero represents false.

ADVANTAGES OF USING TERNARY OPERATOR

- Reduces the code to single line or just few lines.
- Best for short conditions.
- If used with proper formatting it can more appropriate than simple if else.

Sample Program:

C program to find maximum between two numbers using ternary operator.

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int num1, num2, max;
6      printf("Enter two numbers: ");
7      scanf("%d %d", &num1, &num2);
8
9      max = (num1>num2) ? num1 : num2;
10
11     printf("Max is %d", max);
12
13     return 0;
14  }
```

Output:

Enter two numbers: 20
30
Max is 30

NESTED TERNARY OPERATOR

Nesting of ternary operator is often dis-regarded by most of the programmer's and even I also never recommend it. Nesting of ternary operators will definitely reduce the code length in characters but will also cost a poor readable code and more chances of error. Hence you must avoid a nested ternary operator as much as you can.

Sample program of finding maximum between three numbers using a nested ternary operator

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int num1, num2, num3, max;
6
7      printf("Enter three numbers: ");
8      scanf("%d %d %d", &num1, &num2, &num3);
9
10     max = ((num1>num2) ? ((num1>num3) ? num1 : num3) : ((num2>num3) ? num2 : num3));
11
12     printf("Max is %d", max);
13
14     return 0;
15  }
```

In above program you can see that the ternary operator may had reduced the code length but had also made our program less readable. Hence to ensure readability of code we must use proper formatting for the ternary operator.

PROPER FORMATTING OF TERNARY OPERATOR

A proper formatted code will enhance your code readability and will also decrease the probability of errors in your code. A ternary operator must be well formatted in order to make it more readable and clear.

Have a look to the poorly formatted ternary operator.
max = ((num1>num2) ? ((num1>num3) ? num1 : num3) : ((num2>num3) ? num2 : num3));

The above code can be formatted as

```
max = (num1>num2) ?
          ((num1>num3) ? num1 : num3) :
      (num2>num3) ? num2 : num3;
```

The above code demonstrates that how a simple formatted code has increased the readability of our program.

Switch case in C programming:

Switch case is a conditional control statement that allows us to make decisions from number of available choices. It uses an integer value or an enumeration type value for making decisions. Switch case is similar to ladder if else.

In C, an enumeration (or enum) is a user-defined data type used to assign names to integral constants. These named constants make programs easier to read and maintain.

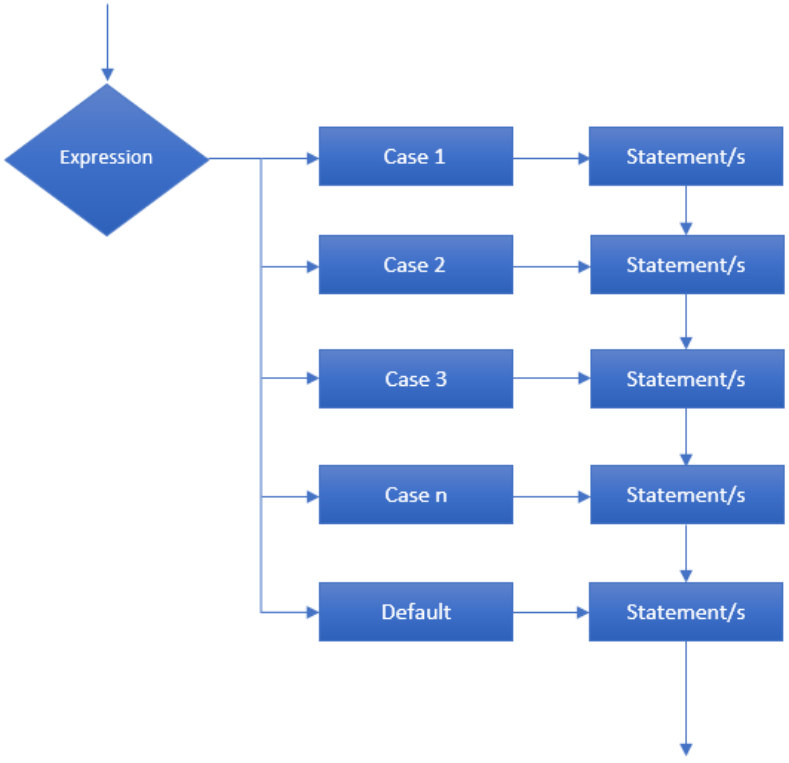
an enumeration type (also called enum) is a data type that consists of integral constants. To define enums, the enum keyword is used.

Syntax:

```
switch(expression)
{
   case 1: statement/s;
   case 2: statement/s;
   case 3: statement/s;
   ...
   ...
   ...
   case n: statement/s;
   default: statement/s;
}
```

Here expression is an integer variable, character or an enumeration type whose value is to be matched with case.

Flow chart:



Switch case flow chart

Rules for using switch case:

- Expression should be an integer, character or an enumeration type.
- Case values should always be an integer, character or enumeration constant. And must match the expression type.
- default case is optional.
- Cases need not to be written in order.
- No two case should have same value
  For example:

```
case 1: statment/s;
case 2: statment/s;
case 1+1: statment/s; //Invalid
case 2: statment/s; //Invalid
```

- When none of the cases are matched default case executes.

Let's see a program to understand switch case:

Program:

Write a program to input any week number and print corresponding week name using switch.

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int week;
6
7      printf("Enter week number: ");
8      scanf("%d", &week);
9
10     switch(week)
11     {
12        case 1: printf("MONDAY\n");
13        case 2: printf("TUESDAY\n");
14        case 3: printf("WEDNESDAY\n");
15        case 4: printf("THURSDAY\n");
16        case 5: printf("FRIDAY\n");
17        case 6: printf("SATURDAY\n");
18        case 7: printf("SUNDAY\n");
19        default: printf("Invalid Input. Please enter week number between 1-7");
20     }
21     return 0;
22  }
23
```

Output:

Enter week number: 4
THURSDAY
FRIDAY
SATURDAY
SUNDAY
Invalid Input. Please enter week number between 1-7

The output of the above program is little weird. When the user input 4 the output should be THURSDAY. But our program is printing all the statements below case 4. In the above program we need something that terminates the control flow from switch when all statements under matched case are executed. And here comes break statement in action.

break: Break statement is used to terminated control flow from inner switch. When a break statement is executed, the control is transferred to next statement after switch.

The above program can be re-written as:

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int week;
6
7      printf("Enter week number: ");
8      scanf("%d", &week);
9
10     switch(week)
11     {
12        case 1: printf("MONDAY\n");
13              break;
14        case 2: printf("TUESDAY\n");
15              break;
16        case 3: printf("WEDNESDAY\n");
17              break;
18        case 4: printf("THRUSDAY\n");
19              break;
20        case 5: printf("FRIDAY\n");
21              break;
22        case 6: printf("SATURDAY\n");
23              break;
24        case 7: printf("SUNDAY\n");
25              break;
26        default: printf("Invalid Input. Please enter week number between 1-7");
27     }
28
```

```
29      return 0;
30  }
```

Output:

Enter week number: 4
THURSDAY

For loop in C

Loops are as valuable control statements in programming as are the conditional statements. Loops are used for performing repetitive tasks. Read more about how loops work.
For loop is the most commonly used and popular loop amongst all three loops present in C. The reason behind its popularity lies in the simplicity. For loop is the easiest loop to begin programming with loops.

Syntax of for loop:

```
for ( initialization ; condition ; update )
{
   //Body of loop
}
```

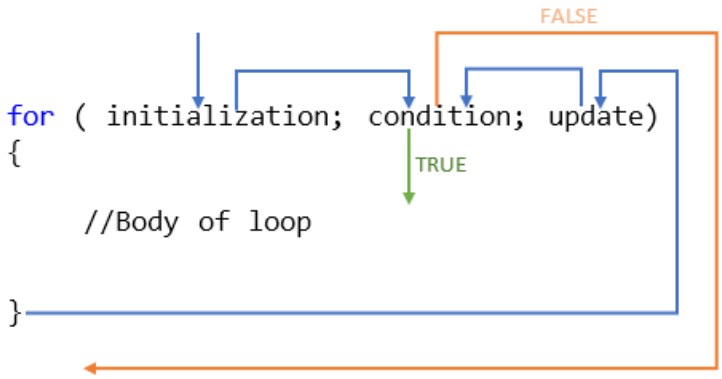
PARTS OF THE FOR LOOP

For loop generally contains four parts:

- First is the initialization. Here we initialize the counter variables.
- Second is the condition. Condition part contains one or more boolean expressions that determines whether the loop will further repeat or not. If the condition satisfies (condition is true) then the statement/s under body of loop will be executed else the loop terminates.
- Third is the body of loop. Body of loop generally contains one or more statement/s that are executed if the loop condition is true.
- And last is the updation. Updation part generally contains expressions to update the loop counter. Updation part is sometimes also referred as Increment/Decrement part.

Note: All the four parts of a for loop are optional. We may or may not specify any of the loop part.

CONTROL FLOW OF FOR LOOP

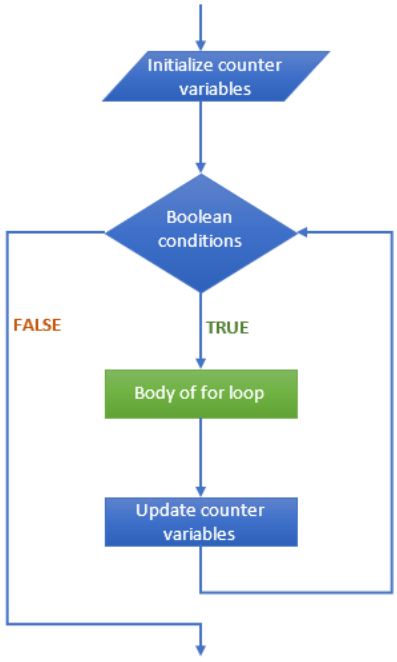


Working mechanism of for loop

Few more things to note about for loop:

- Initialization part runs only once.
- The condition, body of loop and updation part runs repeatedly until the loop terminates.

FLOW CHART OF FOR LOOP



Flow chart of for loop

SAMPLE PROGRAM TO DEMONSTRATE FOR LOOP

Write a program in C to print Hello, World! 10 times.

Program:

```
1   #include <stdio.h>
2
3   int main()
4   {
5      int i; //Used as a counter variable.
6
7      for( i=1 ; i<=10 ; i++)
8      {
9         printf("Hello, World!\n");
10     }
11
12     return 0;
13  }
```

Output:

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Note: For updation you may use any expressions that increments the value of i by 1. Such as you are free to use: i = i + 1; or i+=1; or i++;

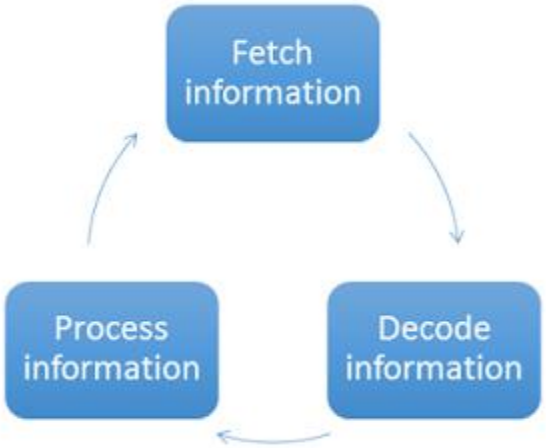
Also if the body of loop contains single statement then there is no need to use curly braces { } for the for loop. As the above loop can also written as.

```
for ( i=1 ; i<=10 ; i++)
   printf("Hello, World!\n");
```

But it is always recommended that use pair of curly braces { } with the for loop whether it contains single or multiple statements.

INFINITE FOR LOOP

An infinite for loop is a loop without any termination condition. Infinite for loop runs infinite times without terminating from the loop. They are generally used in programming when the number of repetitions are unknown by the programmer, for example: The CPU works in an infinite loop with its task as **FETCH-DECODE-PROCESS**. And keeps running infinitely until the user shuts down the PC.



Processor working in an infinite loop

Various examples of infinite for loop

Loop structure	Desccription
for (i=1 ; i<=10 ;) printf("Hello, World!\n");	Here the updation part of the counter variable i is missing. Hence the value of i will remain 1 forever and the loop runs infinite times printing Hello, World!
for (i=1 ; ; i++) printf("Hello, World!\n");	Here the condition part of the loop is missing. Due to which the loop is forced to run repeatedly infinite times printing Hello, World! As there isn't any condition to terminate.
for (i=1 ; ;) printf("Hello, World!\n");	Here the condition as well as updating part is missing from the loop. Hence it will also iterate infinite times printing Hello, World!
for (; ;) printf("Hello, World!\n");	Here all the three parts initialization, condition as well as updating part is missing from the loop. Hence it will also run infinite times printing Hello, World!

EMPTY FOR LOOP

As I said earlier all the four parts of the for loop are optional. Which means we may skip any part of the for loop. An empty loop is a loop without body of loop. Which means it's a loop that does nothing instead of iterating certain times.

Various examples of empty for loop

Loop structure	Description
for (i=1 ; i<=10 ; i++) ;	Here the body of the loop is missing from the loop. The loop may run 10 times but does nothing.

for (i=1 ; i<=10 ; i++)
{
}	

for (; ;) ;	An empty for loop that runs infinite times.