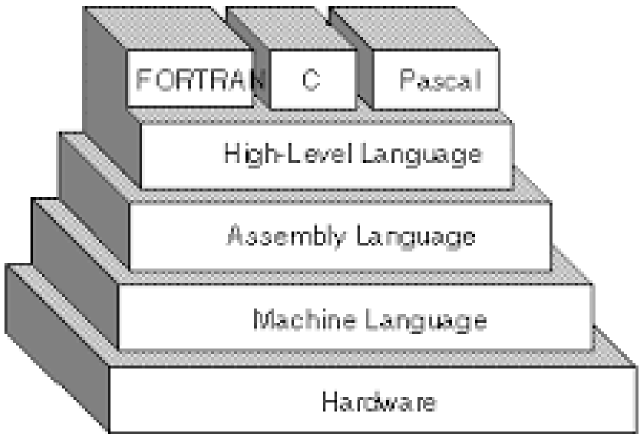
Programming Language

A programming language is a formal language comprising a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms. Most programming languages consist of instructions for computers.

High Level and Low Level Language

The only language understood by computers is binary, also known as machine code.



Programming languages can be classified into low-level and high-level languages.

Low-level languages

Assembly:

To make it easier to program computers a programming language was invented. It was called 'Assembly' and was made up of a small set of command words called mnemonics which programmers typed instead of binary. Examples of mnemonics are "MOV", "ADD" and "PUSH". Computers could not understand Assembly so it had to be converted to machine code by an 'assembler' before it could be run.

- Low-level languages are more similar to machine code than they are to languages spoken by humans
- They are not very developed and just offer quicker ways to write binary
- This means they give the programmer close control over the computer because their instructions are very specific
- Unfortunately they are hard to learn
- Since the machine code for each computer is different, programs translated to machine code on one computer will not work on a different one: Low level languages are not very 'portable'

High-level languages

High level programming languages are more developed than low-level languages so are closer to human spoken language.

- Some examples of high level languages are: C#, Visual Basic, C, C++, JavaScript, Objective C, BASIC and Pascal.
- High-level programming languages are easier for humans to write, read and maintain
- They support a wide range of data types
- They allow the programmer to think about how to solve the problem and not how to communicate with the computer. This is called abstraction

Converting to Machine Code

Translators:

- Just like low-level languages, high-level languages must be converted to machine code before a computer can understand and run them
- This is done using a ‘translator‘
- Different translators convert the same high level code into machine code for different computers
- High level code ready to be translated into machine code is called ‘source code’

There are two different types of translator: Compilers and Interpreters

Compilers:

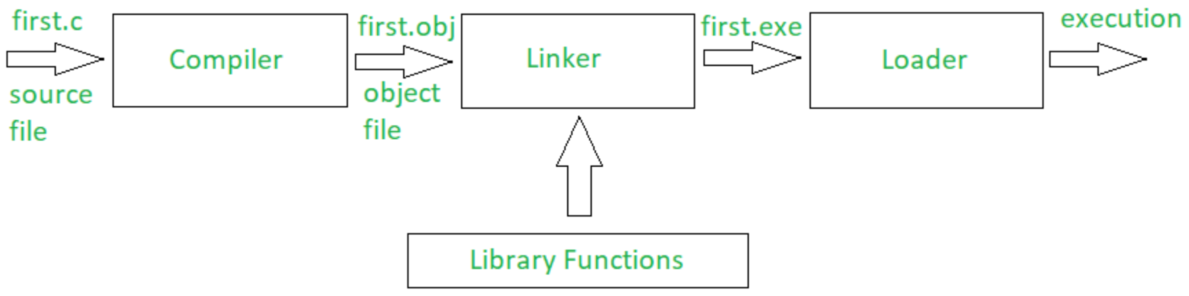
- Compilers convert (or ‘compile’) the source code to machine code all at once
- This is then stored as an executable file which the computer can run (for example, something ending with the ‘.exe’ file extension)
- Errors in the source code can be spotted as the program is compiling and reported to the programmer

Interpreters:

- Interpreters convert the code as it is running
- They take a line of source code at a time and convert it to machine code (which the computer runs straight away)
- This is repeated until the end of the program
- No executable file is created

- If the interpreter comes across an error in the source code the only things it can do is to report the error to the person trying to use the program (or it may just refuse to continue running)

Compiling a C program



Programming C/C++

Basic Combined Programming Language, BCPL is a programming language developed in 1966 by Martin Richards of the University of Cambridge. It quickly became popular, largely because of its high portability.

B is a programming language developed at Bell Labs circa 1969. It is the work of Ken Thompson with Dennis Ritchie.

C is a general purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. It was named 'C' because many of its features were derived from an earlier language called 'B'.

The C++ programming language has a history going back to 1979, when Bjarne Stroustrup was doing work for his Ph. D. thesis. He began work on "C with Classes", which as the name implies was meant to be a superset of the C language. In 1985, C++ was implemented as a commercial product.

C++ is more often used in the programming world today and it is often considered the more robust language, even though C is better suited to some applications. ... If you are new to programming, learning C before C++ will likely be less overwhelming and give you some room to learn and grow.

C is known as a mother language because most of the compilers and Java Virtual Machines (JVMs) are written in C language. Most of the languages which are developed after C language has borrowed heavily from it like C++, Python, Rust, javascript, etc.

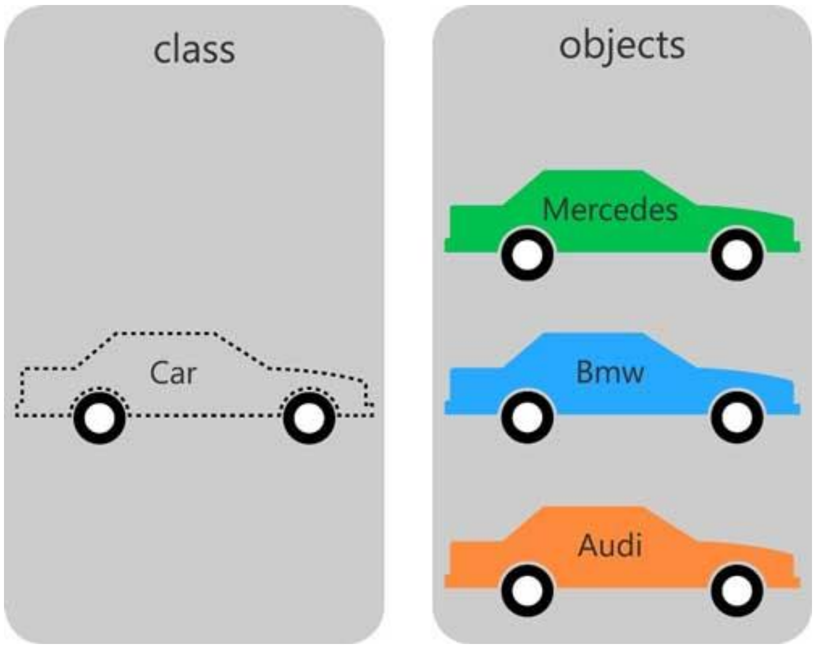
Structured/Procedural vs. Object Oriented Programming:

Structured Programming Language:

This type of programming is based on some instructions and functions. So this type of languages are divides into some functions. A function is a named section of a program that performs a specific task. FORTRAN, C, PHP are structured programming languages.

Object Oriented Programming:

This type of programming language includes some objects and classes. A class is like a template and it defines some behaviors. Objects are the members of this template that have those defined behaviors. So, this shows us that all of these objects are related to each other. In the example below, all of the three cars are a member of the class "car". This means that they all have features and behaviors of a car but they also might have other features that vary them from each other.

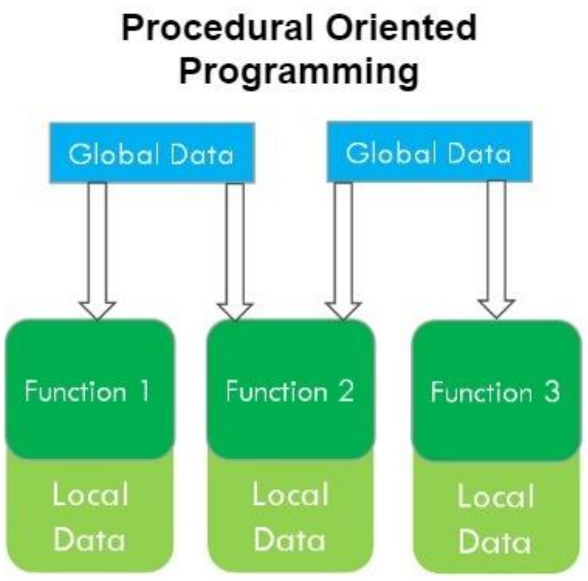


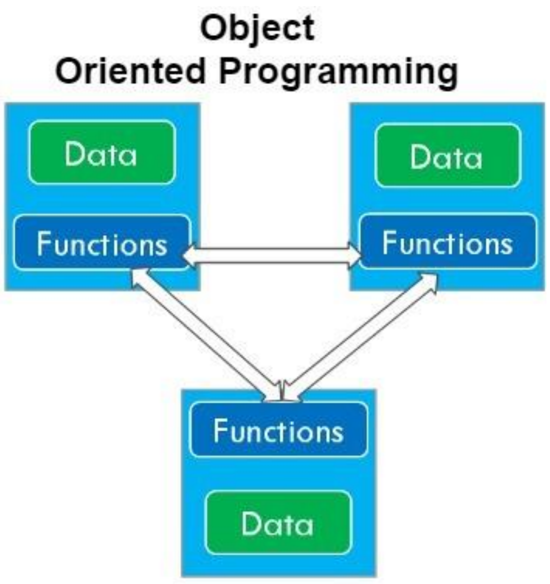
Advantage of OOPs over Structured programming language

OOPs makes development and maintenance easier where as in structured programming language it is not easy to manage if code grows as project size grows.

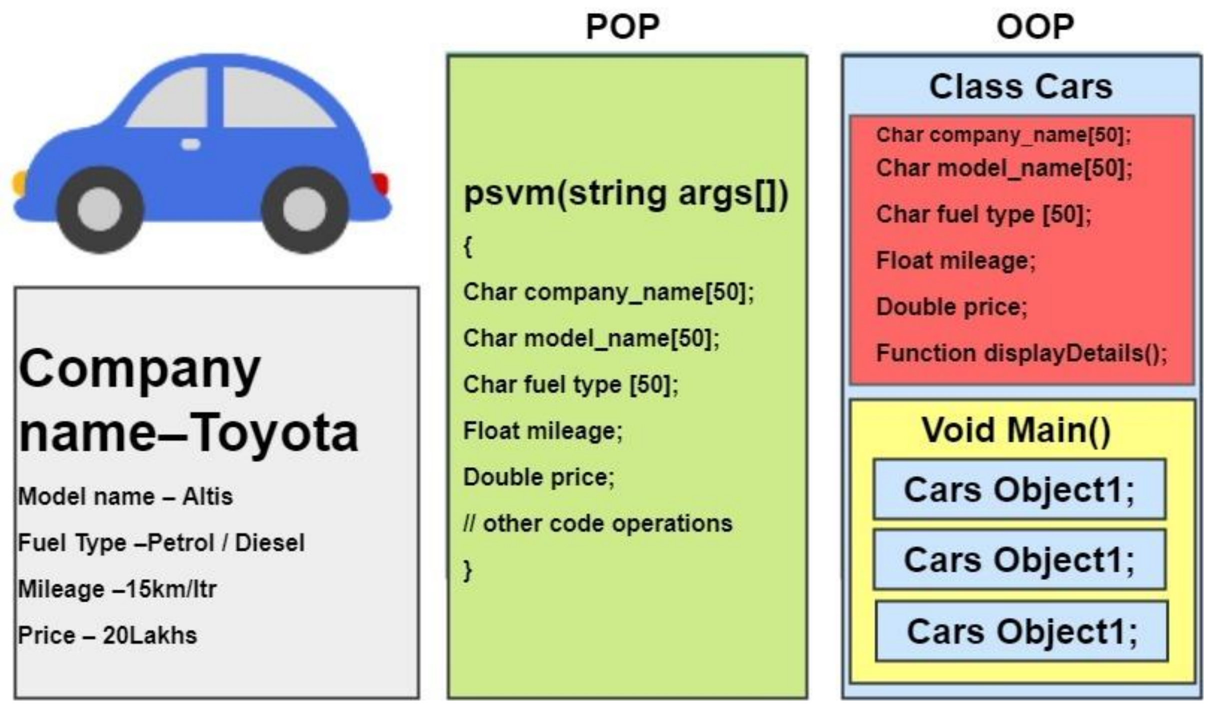
OOPs provides data hiding whereas in structured programming language a global data can be accessed from anywhere.

OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.





Example of Structured/Procedural Oriented Programming vs Object Oriented Programming



```
psvm(string args[])
{
Char company_name[50];
Char model_name[50];
Char fuel type [50];
Float mileage;
Double price;
// other code operations
}
```

```
Char company_name[50];
Char model_name[50];
Char fuel type [50];
Float mileage;
Double price;
Function displayDetails();
```

Let’s understand the difference between POP and OOP and see how OOP is better when it comes to resembling real world scenarios in terms of programming.

Let’s assume we want to store details of Cars and store its Model name, fuel type, mileage and price. Assume that we want to store details about 20 cars.

Now in POP since we don’t have Objects and Classes, we have to create individual variables to hold every detail right? So 5 variables for 1 car (1 variable for name, 1 for fuel type and so on). Now this is just for 1 car right, we have 20 cars. So 5 x 20 = 100 variables. This definitely makes the whole program messy and lengthy right. Also if we were to perform some more tasks on 1 type or car, we would have to create individual functions for each which again complicates the process. Thus as you can see, in this scenario POP doesn’t seem to be very efficient.

Now let’s see what happens in OOP, in OOP we create a class which the required data variables. Then we use this class to create objects. Each object of Cars type has its own copies of variables encapsulated inside them. Thus when we create 1 Car object, by default we create all the data variables that we want to store for that car object. This makes it process easy. Now we can also add methods in the class which takes input from user to store the car details or to print/display those details. This makes it even more efficient as we only have to write the method in the class once and then all the objects of the class will by default get this method.

The Form of a C Program

```
global declarations
main( )
{
        local variables
        statement sequence
}
f1( )
{
        local variables
        statement sequence
}
```

The Library and Linking

C comes with a standard library that provides functions that perform most commonly needed tasks. When we call a function that is not part of the program we write, C “remembers” its name. Later the linker combines the code we write with the object code already found in the standard library. This process is called linking.

Program to add two numbers:

```
#include <stdio.h>
#include <conio.h>

int main() {
   int number1, number2, sum;
   printf("Enter two integers: ");
   scanf("%d %d", &number1, &number2);
   // calculating sum
   sum = number1 + number2;
   printf("%d + %d = %d", number1, number2, sum);
   return 0;
}
```

To Download Turbo C/C++ Compiler go the mentioned link below and follow the instructions: https://developerinsider.co/download-turbo-c-for-windows-7-8-8-1-and-windows-10-32-64-bit-full-screen/

Declaring Variables

A variable is an allocated data storage container. The name given to a variable is determined by the programmer, thus the name is referred to as an identifier.

There are a few limitations on the allowable name for variables.

- Identifiers may contain any alphanumeric characters (a-z, A-Z, 0-9) as well as underscores (_).
- Must not contain spaces. Consider camel or underscore style for multi-word names.
- Must not contains special characters other than the underscore.
- Must not begin with a number.
- Must not be one of the keywords of the language.

Keywords in C

These are words that are explicitly reserved and have strict meaning to the C compiler. They cannot be redefined or redeclared. The meaning of these words is mostly related to the flow of control. There are 32 reserved words (a small number compared to some other languages).

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

Basic Data Types

Name	Use	Size	Range	Format Specifier
int	Integer (whole) numbers	32 bits	-2147483648 to 2147483647	%d
short	Integer (whole) numbers	16 bits	-32768 to 32767	%hd
long	Integer (whole) numbers	64 bits	1.7xE-308 to 1.7xE+308	%ld
char	ASCII letters	8 bits	-128 to 127	%c
float	real numbers with a decimal point	32 bits	3.4xE-38 to 3.4xE+38	%f
double	real numbers with a decimal point	64 bits	1.7xE-308 to 1.7xE+308	%lf

signed and unsigned

In C, signed and unsigned are type modifiers. We can alter the data storage of a data type by using them.

Example 1: C Output

```
#include <stdio.h>
int main()
{
   // Displays the string inside quotations
   printf("C Programming");
   return 0;
}
```

Example 2: Integer Output

```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
   return 0;
}
```

Example 3: float and double Output

```
#include <stdio.h>
int main()
{
   float number1 = 13.5;
   double number2 = 12.4;

   printf("number1 = %f\n", number1);
   printf("number2 = %lf", number2);
   return 0;
}
```

Example 4: Print Characters

```
#include <stdio.h>
int main()
{
   char chr = 'a';
   printf("character = %c", chr);
   return 0;
}
```

Assignment 1:

Write a small C program to perform the calculation of $e = mc^2$. For this assignment, hard code the value of known variables into the program and use an equation to calculate the unknown variable.

Use the printf() function to print the result. The program should print the known values, textual explanation of what the numbers are and the calculated value.

Example 5: Integer Input/Output

```
#include <stdio.h>
int main()
{
   int testInteger;
   printf("Enter an integer: ");
```

```
    scanf("%d", &testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

Example 6: Float and Double Input/Output

```
#include <stdio.h>
int main()
{
    float num1;
    double num2;

    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);

    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);
    return 0;
}
```

Example 7: C Character I/O

```
#include <stdio.h>
int main()
{
    char chr;
    printf("Enter a character: ");
    scanf("%c",&chr);
    printf("You entered %c.", chr);
    return 0;
}
```

Example 8: Arithmetic Operators

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int a = 9, b = 4, c;
    clrscr();
    c = a+b;
    printf("a+b = %d \n", c);
    c = a-b;
    printf("a-b = %d \n", c);
    c = a*b;
    printf("a*b = %d \n", c);
    c = a/b;
    printf("a/b = %d \n", c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);
    getch();
    return 0;
}
```

Assignment 2:

Write a small C program to perform the calculation of $e = mc^2$. For this assignment, prompt the user for the known variables of the equation and use an equation to calculate the unknown variable.

Use the printf() function to print the result. The program should print the calculated value.

Example 9: Working with String

```
#include <stdio.h>
#include <conio.h>
#include<string.h>
int main()
{
   Char f_name[25], l_name[25];
   printf("Enter First Name: ”);
   scanf(“%s”,f_name);
   printf(“Enter Last Name: ”);
   gets(l_name);
   printf(“Your name is: ”);
   printf(“%s”, strcat(f_name,l_name);
   getch();
   return 0;
}
```

prefix and postfix increment

++i is prefix increment	i++ is post increment

#include<stdio.h>
int main()
{
   int i, j;
   i = 9;
   j = ++i;
   printf(“i and j = %d %d,i,j);
}
Output: i and j = 10 10
	#include<stdio.h>
	int main()
	{
	   int i, j;
	   i = 9;
	   j = i++;
	   printf(“i and j = %d %d,i,j);
	}
	Output: i and j = 10 9

Assignment 3:

Write a small C program to calculate the area of a rectangle. For this assignment, prompt the user to enter length and width of the rectangle. Calculate the area.

Use the printf() function to print the result. The program should print the calculated value.

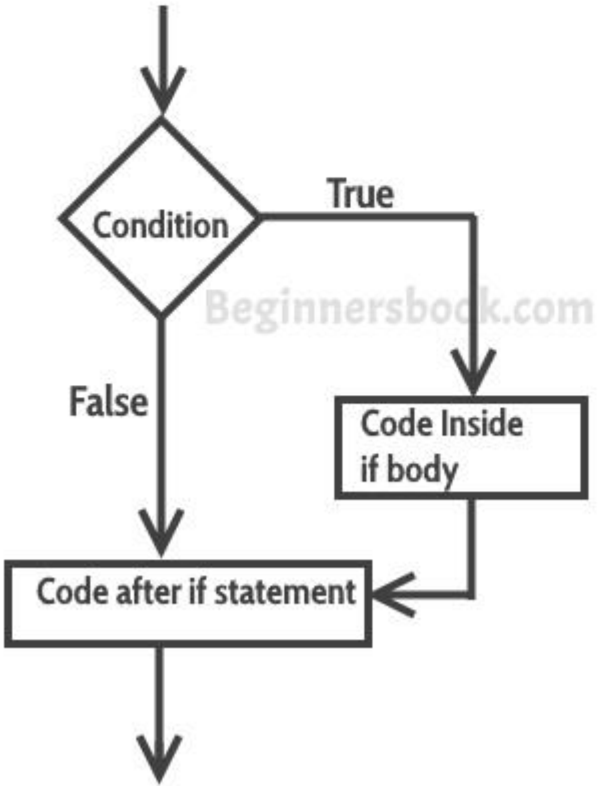
Control Statement:

There are two types of control statement in C.

1. Conditional Control Statement
2. Loop Control Statement

1. Conditional Control Statement
   i. if
   ii. if…..else
   iii. else if
   iv. switch

Flow diagram of if statement



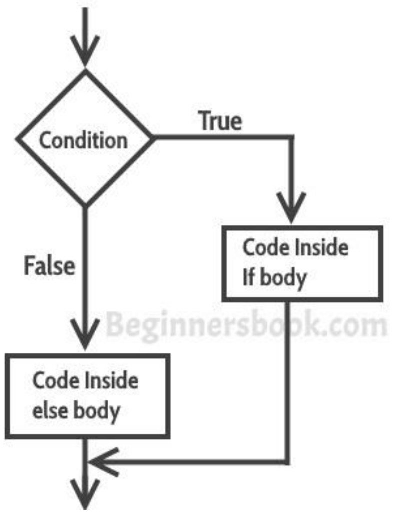
Syntax of if statement:

The statements inside the body of “if” only execute if the given condition returns true. If the condition returns false then the statements inside “if” are skipped.

Example 10: if condition

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int x, y;
   printf("enter the value of x:");
   scanf("%d", &x);
   printf("enter the value of y:");
   scanf("%d", &y);
   if (x>y){
         printf("x is greater than y\n");
   }
   if (x<y){
         printf("x is less than y\n");
   }
   if (x==y){
         printf("x is equal to y\n");
   }
   printf("End of Program");
   return 0;
   getch();
}
```

Flow diagram of if..else statement



Syntax of if else statement:

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.

If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.

Example 11: if……else [Finding whether a number is odd or even]

```
#include <stdio.h>
#include <conio.h>
int main() {
   int num;
   printf("Enter an integer: ");
   scanf("%d", &num);

   // True if num is perfectly divisible by 2
   if(num % 2 == 0)
      printf("%d is even.", num);
   else
      printf("%d is odd.", num);

   return 0;
}
```

Example 12: if…..else [Finding whether a number is positive or negative]

```
#include <stdio.h>
void main()
{
   int num;
   printf("Input a number :");
   scanf("%d", &num);
   if (num >= 0)
      printf("%d is a positive number \n", num);
   else
      printf("%d is a negative number \n", num);
}
```

Example 13: else if condition [Finding greater value]

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int x, y;
   printf("enter the value of x:");
   scanf("%d", &x);
   printf("enter the value of y:");
   scanf("%d", &y);
   if (x>y){
         printf("x is greater than y\n");
   }
   else if (x<y){
         printf("x is less than y\n");
   }
   else{
         printf("x is equal to y\n");
   }
   printf("End of Program");
   return 0;
   getch();
}
```

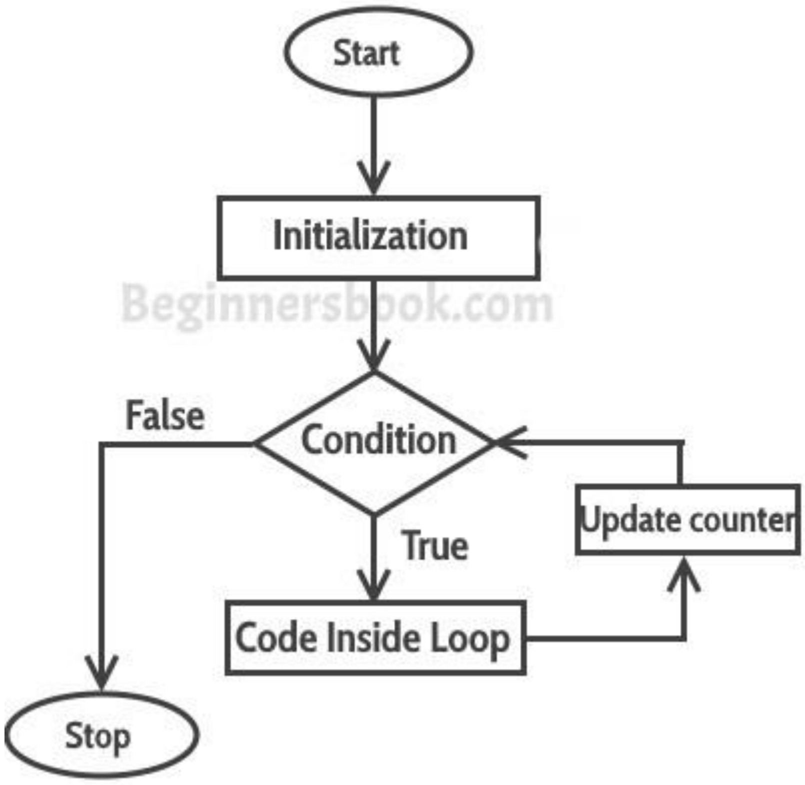
Example 14: else … if condition [Finding leap year]

```
#include <stdio.h>
#include <conio.h>
void main()
{
   int chk_year;
   clrscr();
   printf("Input a year :");
   scanf("%d", &chk_year);
```

```
    if ((chk_year % 400) == 0)
        printf("%d is a leap year.\n", chk_year);
    else if ((chk_year % 100) == 0)
        printf("%d is a not leap year.\n", chk_year);
    else if ((chk_year % 4) == 0)
        printf("%d is a leap year.\n", chk_year);
    else
        printf("%d is not a leap year \n", chk_year);
}
```

2. Loop Control Statement
   i. for loop
   ii. while loop
   iii. do-while loop

Flow Diagram of For loop



Step 1: First initialization happens and the counter variable gets initialized.

Step 2: In the second step the condition is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.

Step 3: After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or –).

Example 15: for loop [First 10 natural numbers]

```
#include <stdio.h>
void main()
{
   int i;
         printf("The first 10 natural numbers are:\n");
         for (i=1;i<=10;i++)
         {
                  printf("%d ",i);
         }
}
```

Example 16: for loop [The sum of first 10 natural numbers.]

```
#include <stdio.h>
void main()
{
   int  j, sum = 0;
   printf("The first 10 natural number is :\n");
   for (j = 1; j <= 10; j++)
   {
      sum = sum + j;
   }
   printf("\nThe Sum is : %d\n", sum);
}
```

Example 17: for loop [Find cube of the number up to a given integer]

```
#include <stdio.h>
void main()
 {
    int i,ctr;
    printf("Input number of terms : ");
    scanf("%d", &ctr);
    for(i=1;i<=ctr;i++)
    {
            printf("Number is : %d and cube of the %d is :%d \n",i,i, (i*i*i));
    }
 }
```

Example 18: for loop [Compute multiplication table of a given integer]

```
#include <stdio.h>
void main()
{
   int j,n;
   printf("Input the number (Table to be calculated) : ");
   scanf("%d",&n);
   printf("\n");
   for(j=1;j<=10;j++)
   {
     printf("%d X %d = %d \n",n,j,n*j);
   }
}
```

Example 19: for loop [Calculate the factorial of a given number]

```
#include <stdio.h>
void main(){
  int i,f=1,num;

  printf("Input the number : ");
  scanf("%d",&num);
  for(i=1;i<=num;i++)
  {
      f=f*i;
   }
  printf("The Factorial of %d is: %d\n",num,f);
}
```

Example 20: for loop [Check whether a given number is prime or not]

```
#include <stdio.h>
#include <conio.h>

int main(){
     int n,i,flag=0;
     printf("Enter a positive number: ");
     scanf("%d", &n);
     for(i=2;i<=n/2;i++){
          if(n%2 == 0){
              flag = 1;
              break;
           }
       }
       if(flag == 0)
              printf("%d is a prime number.", n);
       else
              printf("%d is not a prime number.", n);

       return 0;
       getch();
}
```

Function Call

```c
#include <stdio.h>

int addition(int num1, int num2)
{
     int sum;
     sum = num1+num2;
     /* Function return type is integer so we are returning
      * an integer value, the sum of the passed numbers.
      */
     return sum;
}

int main()
{
     int var1, var2;
     printf("Enter number 1: ");
     scanf("%d",&var1);
     printf("Enter number 2: ");
     scanf("%d",&var2);
     /* Calling the function here, the function return type
      * is integer so we need an integer variable to hold the
      * returned value of this function.
      */
     int result = addition(var1, var2);
     printf ("Output: %d", result);

     return 0;
}
```