COMPUTER PROGRAMMING

A Beginner's Guide

Programming Language :- It converts the raw input data into information, which is useful to user.

→ It includes 5 steps - (i) Inputting, (ii) Storing (iii) Processing, (iv) Outputting (v) Controlling.

Inputting :- It is the process of entering data & information into the system.

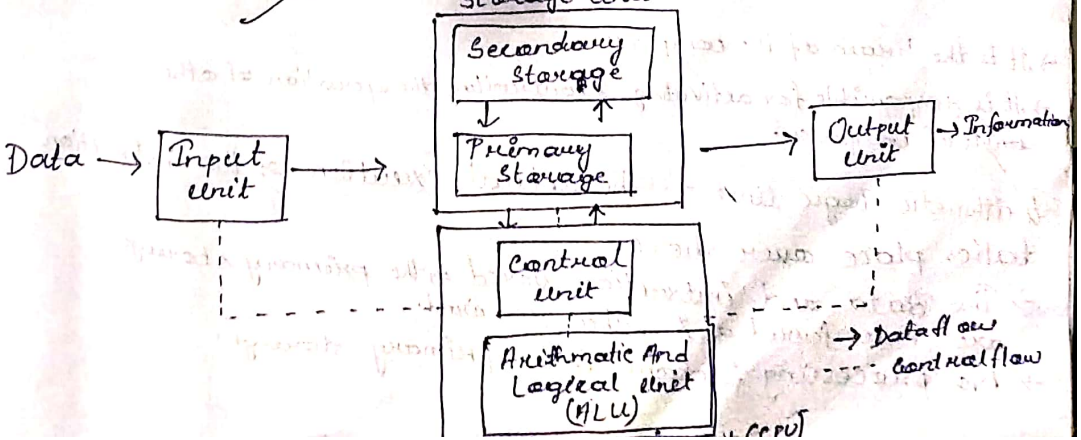
Storing :- The data & information are stored for either initial or additional processing as and when required.

Processing :- It require ~~processing~~ performing arithmetic or logical operation on the saved data to make it into useful information.

Outputting :- It is the process of producing the output data to the end user.

Controlling :- The above operation have to be directed in a particular sequence to be completed.

Block Diagram Of A Computer



Input Unit:- We need to first enter the data & instruction in the computer system, before any computation begins. The device is responsible for linking the system with external environment. The data accepted is in a human readable form. The input device converts it into computer readable language.

Ex:- Keyboard, mouse, scanner, digital cameras etc.

Storage Unit:- It is designed to save the initial data, intermediate and final result.

Primary Storage:- The primary storage, also called as the main memory. It stores the data until the computer is on. When the computer is switched off all the data stored gets deleted.

→ It has limited capacity

→ It is very expensive in nature as it made up of semiconductor material.

Secondary Storage:- It is also known as auxillary storage. It can store the information ~~until it enters~~ even if the computer is switched off.

→ It is basically used for holding the program & instruction & data on which the computer is not working on currently, but needs to process them later.

Central Processing Unit:- It is the ~~process~~ combination of Control unit & Arithmetic logic unit.

→ It is the brain of the computer.

→ It is responsible for activating & controlling the operation of other units of computer.

Arithmetic logic Unit:- The actual execution of the instruction takes place over here.

→ The data and instruction stored in the primary storage are transferred as & when required.

→ No processing is done in the primary storage.

Control Unit:- This unit controls the operation of all parts of the computer but doesnot carry out any actual data processing

→ It is responsible for the transfer of data and instruction among other units of the computer.

Output Unit:- The job of an output unit is just the opposite of an input unit.

→ It converts the coded result to human readable form.

→ Ex- Monitors, printers, projectors etc.

Hardware:- This hardware is responsible for all the physical work of the computer.

Software:- This software commands the hardware what to do & how to do it.

→ It is of 2 types (i) System software
(ii) Application software.

System Software:- These are a set of programs, responsible for running the computer, controlling various operation of computer system and management of computer resources.

→ Ex:- Operating System.

Application Software:- Application software is a set of program designed to solve a particular problem for user.

→ Ex:- Web Browsers, Gaming software etc.

5

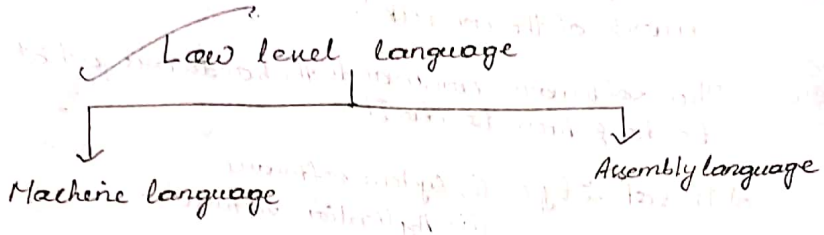
Introduction To Programming

Programming language :- A language i.e acceptable to a computer system is called programming language or computer language.

Coding - The process of creating a sequence of instruction in such a language is called programming or coding.

Program - It is set of instructions, written to perform a specific task by computer.

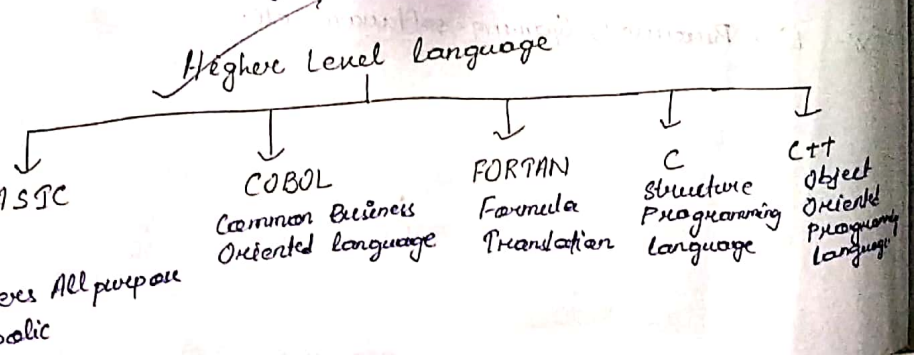
Syntax :- These are the rules which bound the computer language.



Machine language :- This is the language understood directly by the computer.

→ It in the form of 0's and 1's called binary numbers.

Assembly language :- This is the language where the machine code comprising of 0's and 1's are substituted by symbolic code to improve their understanding.



Programming Language Translator

Compiler:- The software that reads a program written in high level language and translate it into an equivalent program in machine language is called as compiler.

→ It convert the entire source code in machine code.

Source Program:- The program written in higher level language is called source program.

Object Program:- The program generated by the compiler after translation is called as object program.

Interpreter:- It convert the high level language into binary instructions, but their method of execution is different.

→ It takes 1 st statement, translate it, execute it & then again take the next statement.

Assembler → The software that reads a program written in assembly language and translates it into an equivalent program in machine is called as assembler.

Linker → It is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file or another object file.

Characteristic Of C Program

Example Of different level of language.

High level language:- Java, Python.

Middle level language:- C, C++

Low level language:- Assembler.

Structure Oriented

→ In this type of language, large programs are divided into small programs called function

→ Prime focus on function and procedure that operates on the data

→ Data moves freely around the system from one function to another.

→ ~~Program structure~~

→ Ex:- Pascal, ALGOL, and Modula-2.

Object Oriented

→ Programs are divided into objects.

→ Prime focus is in the data that is operated and not on the function.

→ Data is hidden and cannot be accessed by external functions.

→ Ex:- C++, Java,

Non Structure

→ There is no specific structure for programming language.

→ Ex:- BASIC, COBOL, FORTRAN

Preprocessor

→ The part of the compiler which actually gets your program from the source file is called preprocessor.

→ # include <stdio.h> is a preprocessor.

<stdio.h>

→ It stands for Standard input and output

→ It is the name of the file in which function that are present here are defined

→ The "h" portion of the filename is the language extension. Which stands for header file

Void

→ It tells to C compiler that a given entity has no meaning and produces no error.

Main

→ C regards the name main as a special case and will run this function first i.e the program execution starts from main.

( )

→ It tells the compiler that it has no parameter.

{ } (Brace)

→ It allow us to group pieces of program together, often called a block.

→ It encloses the working parts of the function main.

Block

It contain the declaration of variable used within it, followed by a sequence of program statements.

; Semicolon

→ It marks the end of the list of variable names, and also the end of that declaration statement.

→ It tells the compiler where a given statements ends.

* Scanf

→ Scan formatted

→ It is used to read character, string, numeric data from keyboard.

→ It allows us to accept input from standard in.

Printf

→ It takes text and values from within the program and sends it out onto the screen.

→ Print formatted

Files Used In C-Program

- Source File:- This file contains the source code of the program.

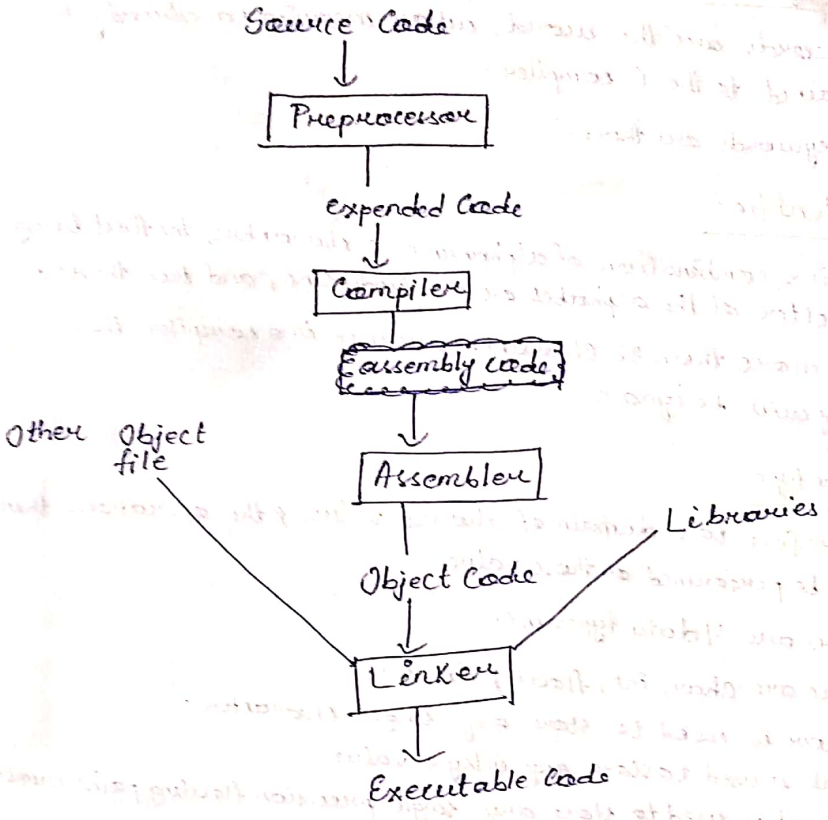
→ Source code is the main function

Header File:- A header file is a file with extension .h which contains the C function declaration and macro definition and to be shared between several source files.

Object file:- These are produced by an assembler, compiler or other language translator, and used as input to the linker.

Executable file:- The binary executable file is generated by the linker.

Compilation And Execution Code



Character Set

Communicating with a computer involves speaking the language the computer understands.

Types	Character Set
Lower case	a-z
Upper case	A-Z
Digits	0-9
Special Character	! @ # $ % ^ & *
White Space	Tab or new lines ouspace.

Keywords

→ Keywords are the words whose meaning has already been explained to the C compiler.

→ 32 Keywords are there.

Identifier

→ It is a combination of alphanumeric characters, the first being a letter of the alphabet or an underline, and then the rest.

→ If more than 32 character are used in a compiler then they will be ignored.

Data type

→ It refers to a domain of allowed values & the operations that can be performed on those values.

→ There are 4 data types in C.

→ These are char, int, float & double.

→ Char is used to store any single character.

→ int is used to store any integer value

→ float is used to store any single precision floating point number.

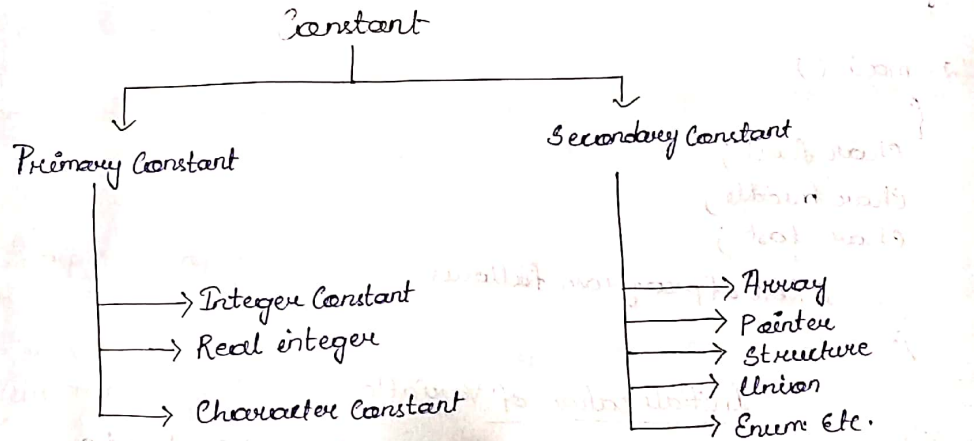
→ double " " " " double " " " " .

The data types in C can be classified as follows:-

Type	Storage size	Value range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	2 or 4 byte	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 byte	0 to 65,535 or 0 to 4,294,967,295
short	2 byte	-32,768 to 32,767

Unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
Unsigned long	4 bytes	0 to 4,294,967,295

Type	Storage Size	Value range	Precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	10 bytes	3.4E-4932 to 1.1E+4932	19 decimal places.



Variables

→ These are the names that are used to store values.
→ It can take different values but one at a time.

Declaring Variable

→ There are 2 places where you can declare a variable.
  → After the opening brace of a block of code.
  → → Before a function name.

Ex:- To Keep track of a person's first, middle and last initial.

```
1. main ()
   {
   char first, middle, last;
   // Rest of program
   }.

2. main ()
   {
    char first;
    char middle;
    char last;
        // Rest of program follows
   }.
```

Initialization of Variable

→ When a variable is declared, it contains undefined value commonly known as garbage value.
→ If we want we can assign some initial value to variables during the declaration itself. This is called initialization of the variable.

```
Eg- int page no = 10;
    char grade = 'A';
    float salary = 20000.50;
```

Expression

→ It consist of a combination of operators, operands, variables & function calls.

→ It can be of 2 types - (i) arithmetic (ii) relational (iii) logical.

a+b → arithmetic operation.
a>b → relational " .
==b → logical " .

Subexpression:- Some expressions are combination of smaller expressions.

Statements:- These are primary building blocks.

Ex:- logs=4;

Compound Statement :- It is two or more statements grouped together by enclosing them in braces.

Semple Program

```
#include <stdio.h>   } This is needed to run printf() function.
int main()
{
printf("C programming");  } displays content inside quotation
return 0;
}.
```

Output

C Programming

Explanation of the previous program

→ Every program starts from main() function.

→ printf is used to display the output.

→ printf only works when #include <stdio.h> is included at the begening.

→ When compiler encounters printf() function and doesn't find stdio.h headerfile, compiler show error.

→ return 0; indicates the successful execution of the program.

Input - Output of integer in C

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

Output

Number = 5.

→ Here %d is is a conversion format string.

```
#include <stdio.h>
int main()
{
    int c;
printf ("Enter a number\n");
scanf ("%d", &c);
printf ("Number = %d", c);
return 0;
}
```

→ The scanf function is used to take input from user.
→ '&c' denotes the address of c and value is stored in that address.

Input - Output of float in C

```
#include <stdio.h>
int main()
{
float a;
printf ("Enter value");
scanf ("%f" &a);
printf ("Value = %f", a);
return 0;
}
```

→ %f is used to take input and to display floating value of a variable.

Output

Enter value: 23.45
Value = 23.450000

Input- Output of characters and ASCII Code

```
#include <stdio.h>
int main(){
{
char var1;
printf("Enter a character");
scanf("%c",&var1);
printf("You entered %c", var1);
return 0;
}
```

Output

```
Enter character: g
You entered : g
```

→ Conversion format string "%c" is used in case of character.

ASCII Code

```
#include <stdio.h>
int main()
{
char var1;
printf("Enter character:");
scanf("%c", &var1);
printf("You entered %c.\n", var1);
printf("ASCII value of %d", var1);
```

```
return 0;
}
```

Output

```
Enter character
g
103
```

→ ASCII value of g is 103

Program-2

```
#include <stdio.h>
int main()
{
int var1 = 69;
printf ("character of ASCII value 69: %c", var1);
return 0;
}
```

Output

Character of ASCII code 69: E

→ The ASCII value of 'A' is 65, 'B' is 66 .... 'Z' is 90.

→ " " " " 'a' is 97, 'b' is 98 - .... 'z' is 122.

Variation in output for integer & floats:

```
#include <stdio.h>
int main()
{
 printf("Case1 %6d\n" 9876);
 printf ("Case2 %3d\n" 9876);
 printf ("Case3 %2f\n" 987.6543);
 printf ("Case4 %f\n" 987.6543);
 printf ("Case5 %e\n", 987.6543);
 return 0;
 }
```

Output

Case 1 : 9876
Case 2: 9876
Case 3: 987.6543
Case 4: 98-7.6543
Case 5: 987.6543

Variation in Input for integer & floats:

```
# include <stdio.h>
 Int main()
 {
 int a, b;
 float c,d;
 printf ("Enter two integers: ");
 scanf ("%d %d," &a, &b);
 printf (" Enter integer and floating point numbers");
 scanf ("%d %f", &a, &c);
 return 0;
  }
```

OPERATORS

→ An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulation.

Arithmetic operator :-

These are used to perform mathematical calculation like addition, substraction, multiplication, division and modulus.

Operator	Description	~~Example~~
+	Add two operands	
−	Substract two operands	
*	Multiply two operands	
/	devide two operands	
%	Modulus operator and remainder of ~~or~~ after an integer division.	
++	Increments operators increases integer value by one	
−−	Decrements operator decreases integer value by one.	

Relational Operators

These operators are used to compare between 2 variables.

Operator	Description
==	Check if the values of two operands are equal or not, if yes then condition becomes true.
!=	Check if the values of operands are equal or not, if equal then the condition will be false.
>	Check if the value of left operands is greater than the value of right operand, if yes then condition becomes true.
<	Check if the value of right operand is greater than the value of left operand, if yes then condition becomes true.
>=	Check if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Check if the value of left operands is less than or equal to the value of right operand, if yes then condition becomes true.

Logical Operator

These operators are used to perform logical operations on the given two variable.

Operator	Description
&&	Called logical AND operator. If both the operands are non zero, then the condition becomes true.
||	Called logical OR operator. If any of the two operand is non-zero, then condition becomes true.
!	Called logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true then Logical Not Operator will make false.

Bitwise Operators

Logic gate

~~Band operal~~

And → Multiplication of two variable.

OR → Addition " " " .

XOR → If two variables are same then the answer will 0.

For And → &
OR → |, ~
XOR → ^

P	q	P & q	P | q	P ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit of it exits in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary left shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right hand operand.

Assignment Operator

→ Values for the variable are assigned using assignment operators.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is C = C - A.
*=	Multiply AND assignment operator, It multiply right operand with the left operand and assign the result to left operand.	C *= A is C = C * A.
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= is equal to C = C / A.

%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C%=A is equal to C=C%A.
<<=	Left shift AND assignment operator	C<<=2 is C=C<<2
>>=	Right " " " "	C>>2
&=	Bitwise AND assignment operator	C&=2 C=C&2
^=	" exclusive OR and assignment operator	C^=2 is same as C=C^2
|=	"	C|=2, C=C|2

Increment And Decrement Operator

- Let a=5.

a++; becomes a=6

a--; becomes a=5

++a; becomes a=6

--a; becomes a=5.

→ When ++var will be used in prefix then it will first increment the value of var and then return it back, if ++ is used as postfix (like: var++), operator will return the value of operand first and then only increment it.

Program

```
# include<stdio.h>
int main()
{
 int c=2, d=2;
 printf ("%d \n", c++);
 printf ("%d", ++c);
 Return 0;
 }
```

Output

2

4

Conditional Operator

→ It executes, different statements according to test condition whether it is either true or false.

Syntax of conditional operators;

→ Conditional_expression? expression1: expression2.

Ex
```
if (x>5){
    y=3;
else
    y=4.
```

Misc Operators:

Operator	Description	Example
~~size of()~~		
sizeof()	It is a unary operator which is used in finding the size of data type, constant arrays, structure etc.	Size of(a), where a is integer, will return 4.
&	Return the address of a variable	&a; will give actual address of the variable.
*	pointer to a variable	*a; will pointer to a variable.

Operators Precedence in C

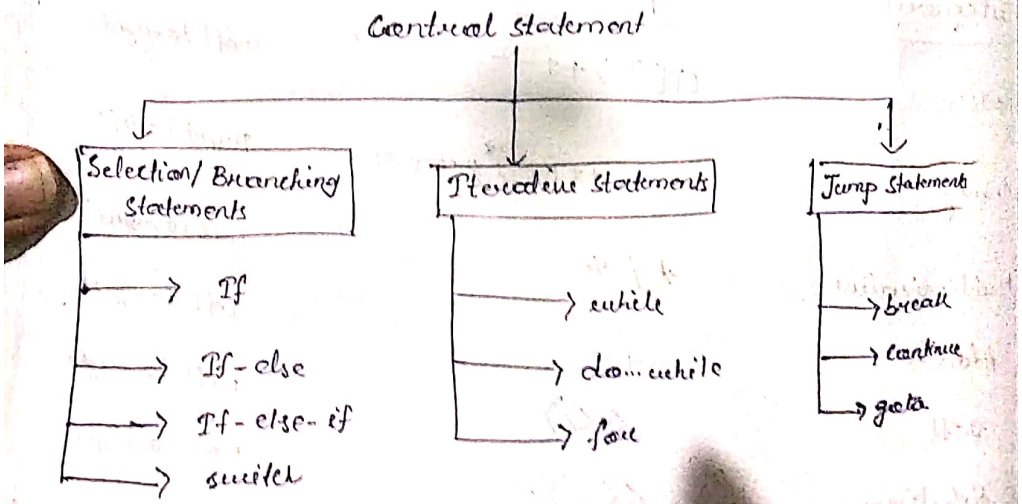
→ Operators precedence determines the grouping of terms in an expression.

Category	Operator	Associativity
Postfix	() [] -> ++ --	Left to right.
Unary	+ - ! ~ ++ (type)* & size of	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right.
shift	<< >>	Left to right
Relational	<< = >> =	Left to right.

Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	=+= -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right.

Control Statement

→ It enable us to specify the order in which the various instruction in the program are to be executed.



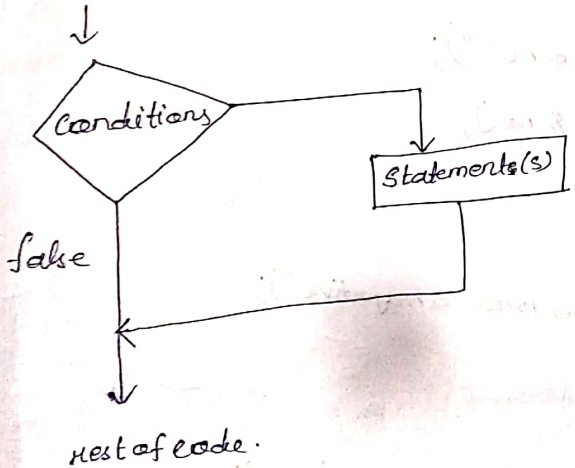
Decision- Control Statement

→ It requires that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be false.

if statements

→ The Keyword if tells the compiler that what follows is a decision control instruction.

→ The if statements allows us to put some decision-making into our programs.



Syntax of if statement

```
if (condition)
{
 Statement 1;
 ----------
 Statement n;
}
```

Q.1 Write a program to print a message if negative no is entered.

```
#include <stdio.h>
int main()
{
  int no;
  printf ("Enter a no: ");
  scanf ("%d", &no);
  if (no<0)
  {
   printf ("no entered is negative");
     no=-no;
}
print f ("Value of no is %d\n", no);
return 0;
}
```

Output

Enter a no : 6

value of no is 6

or

Enter a no: -2

value of no is 2

2. Write a program to perform division of 2 nos

```
#include <stdio.h>
int main()
{
  int a, b;
  float c;
 printf ("Enter 2 nos :");
 scanf ("%d %d", &a, &b);
 if(b==0)
 {
    printf ("Division is not possible");
 }
 c= a/b;
 printf ("quotient is %f \n", c);
 return 0;
}
```

Output

Enter 2 nos : 6 2
quotient is 3

or

Enter 2 nos : 6 0
Division is not possible.

if- else statement

→ When the expression following if evaluates to true.

→ By using else we execute another group of statements if the expression evaluates to false

```
if (a>b)
    {
      z = a;
      printf ("Value of z is : %d", z);
    }
else
    {
      z = b;
      printf ("Value of z is : %d", z);
    }
```

Q. Write a program to check whether the given no. is even or odd

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

```
scanf ("%d", &n);
if (n%2==0)
printf ("Even\n");
else
 printf ("Odd\n");
 return 0;
}
```

Output

```
Enter an integer 3
Odd
  or
Enter an integer 4
 even
```

Q. Write a program to check whether a given year is leap year or not?

```
#include <stdio.h>
int main()
{
int year;
printf ("Enter a year to check if it is a leap year\n");
scanf ("%d", &year);
if (year%4==0) && ((year%100!=0) || (year%400==0))
printf ("%d is a leap year.\n", year);
else
    printf ("%d is not a leap year.\n", year);
```

```
return 0;
}
```

Output

```
Enter a year to check if it is a leap year 1996
1996 is a leap year.
```

or

```
Enter a year to check if it is a leap year 2015
2015 is not a leap year.
```

Nested if-else statement

```
#include <stdio.h>
int main ()
{
int m= 40, n=20;
if ((m>0) && (n>0)){
{
printf ("nos are positive ");
 if(m>n)
{
 printf ("m is greater than n");
}
else
{
 printf ("m is less than n");
}
else
```

```
{
  printf ("nos are negative");
}
return 0;
}
```

Output

40 is greater than 20

Else-if statement

→ The last else part handles the "none of the above" or default case where none of the other condition is satisfied.

Program

```
#include <stdio.h>
int main()
{
int m = 40, n = 20;
if (m>n)
 {
  printf ("m is greater than n");
 }
 else if (m<n)
 {
  printf ("m is less than n");
 }
else
 {
  printf ("m is equal to n");
 }
```

}

Output

m is greater than n.

Switch case:

→ The switch statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values and branches accordingly.

Q. WAP to enter a grade & check its corresponding ~~mark~~ remarks.

```
#include <stdio.h>
int main()
{
 char grade;
 printf ("Enter the grade");
 scanf ("%c", &grade);
 switch (grade)
 {
 case 'A': printf ("Outstanding ! \n");
           break;
 case 'B': printf ("Excellent ! \n");
           break;
 case 'C': printf ("Well done ! \n");
 case 'D': printf (" You have passed ! \n");
 case 'E': printf ("Better turn next time ! \n");
 default : printf ( " Invalid grade \n");
 }
```

```
printf ("Your grade is %c \n", grade);
return 0;
}
```

Iterative Statements

while statement

→ The while statement is used when the program needs to perform repetitive tasks.

→ The program will repeatedly execute the statement inside the while until the condition becomes false.

→ If the condition is initially false, the statement will not be executed.

Program

```
main()
{
 int p, n, count;
 float r, si;
 count = 1;
 while (count <= 3)
 {
  printf ("\n Enter values of p, n and r");
  scanf ("%d %d %f", &p, &n, &r);
  si = p*n*r/100;
  printf ("Simple interest = Rs. %f", si);
  count = count + 1;
 }
}
```

Q. WAP to check whether a given number is a palindrome or not?

```
#include <stdio.h>
int main()
{
int n, reverse = 0, temp;
printf ("Enter a number to check if it is a palindrome
        or not \n");
scanf ("%d", &n);
temp = n;
while (temp != 0)
{
reverse = reverse * 10
reverse = reverse + temp % 10;
temp = temp/10;
}
if(n == reverse)
printf ("%d is a palindrome number. \n", n);
else
printf ("%d is not a palindrome number. \n", n);
return 0;
}
```

Output

```
12321
12321 is a palendrome.
```

do-while loop

- The body of the do-while executes at least once.
- As long as the test is true, the body of the loop continues to execute.

Q. WAP to add all the numbers entered by a user until user enter 0.

```
#include <stdio.h>
int main()
{
 int sum = 0, num;
do
{
 printf("Enter a number\n");
 scanf("%d", &num);
 sum += num;
}
while (num != 0);
printf("Sum = %d", sum);
return 0;
}
```

Output

```
Enter a number
 3
Enter a number
 -2
Enter a number
  0
 sum = 1
```

```
#include <stdio.h>
main()
{
int i=10;
do
{
printf ("Hello %d\n", i);
 i= i-1;
}
 while(i>0)
}
```

Output

Hello 10
" 9
" 8
" 7
" 6
" 5
" 4
" 3
" 2
" 1

Q. Program to count the no. of digits in a number.

```
#include <stdio.h>
int main()
{
int n, count=0;
printf ("Enter an integer:");
scanf ("%d", &n);
do
{
n/=10;
```

```
count++;
} while(n!=0);
printf("Number of digits: %d", count);
}
```

Output

```
Enter an integer: 34523
Number of digits : 5.
```

for Loop

→ It allows us to specify three things about a loop in a single line:

(a) Setting a loop counter to an initial value

(b) Testing the loop counter to determine whether its value has reached the number of repetitions desired.

(c) Updating the value of loop counter either increment or decrement.

Q. WAP to print cubes of number from 1 to 6.

```
int main(void)
{
int num;
printf("n n cubed\n");
for (num = 1; num<=6; num++)
printf("%5d %5d\n", num, num*num*num);
return 0;
}
```

Output

```
1  1
2  8
3  27
```

4 64
5 125
6 216

Q. Program to print the sum of 1st N natural numbers.

```
#include <stdio.h>
int main()
{
int n, i, sum=0;
printf (" Enter the limit: ");
scanf ("%d", &n)
for (i=1; i<=n; i++)
{
  sum = sum+i;
}
printf ("Sum of N natural numbers is %d ", sum);
}
```

Output

Enter the limit: 5
Sum of N natural number is 15.

Q. Program to find the reverse of a number

```
#include <stdio.h>
int main()
{
int num, r, reverse=0;
printf ("Enter any number: ");
scanf ("%d", &num);
for (; num!=0; num = num/10)
{
  r = num % 10;
  reverse = reverse *10 +r;
```

```
}
printf ("Reversed of number : %d", reverse);
return 0;
}
```

Output

```
Enter any number : 123
Reverse of any number : 321
```

Nesting Of Loops

→ We can put any type of loop inside any type of loop

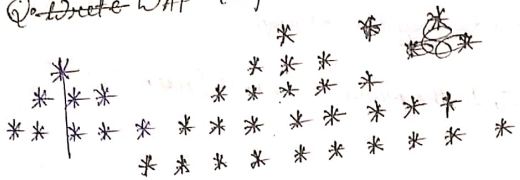
Q. WAP using a nested for loop to find the prime numbers from 2 to 20.

```
#include <stdio.h>
int main()
{
int i, j;
for (i=2; i<20; i++)
 {
  for (j=2; j<=(i/j); j++)
  if !(i%j)
  break;
  if (j>(i/j)) printf ("%d is prime\n", i);
  }
  return 0;
 }
```

Output

2 is prime
3 is prime
5 " "
7 " "
11 " "
13 " "
17 " "
19 " "

Q. ~~Write~~ WAP to print the following:



```
#include <stdio.h>
int main()
{
int row, c, n, I, temp;
printf("Enter the number of rows in pyramid of
        stars you wish to see");
scanf ("%d", &n);
temp = n;
for (row=1; row<=n; row++)
{
for (i=1; i<temp; i++)
{
printf (" ");
temp--;
```

```
for (c=1; c<=2*row-1; c++)
{
 printf ("*") ;
 printf ("\n");
 }
 }
 }
 return 0;
 }
```

6. Program to print series from 10 to 1 using nested loops.

```
#include <stdio.h>
void main ()
{
 int a;
 a=10;
 for (k=1; k=10; k++)
{
 while (a>=1)
 {
 printf ("%d", a);
 a--;
 }
 printf ("\n");
 a=10;
 }
}
```

Output

10 9 8 76 54 321

1,
11
1,
1,
1,
1,
1,
1,
1,

Jump Statement

The break statement

→ The break statement provides an early exit from for, while and do just as switch.

→ When break is encountered inside any loop, control automatically passes to the first statements after the loop.

The continue statement

→ The continue statement is related to break, it causes the next iteration of the enclosing for, while, or do loop to begin

→ It is only applied to loops, not to switch.

```
main()
{
int i,j;
for(i=1; i<=2; i++)
 {
 for(j=1; j<=2; j++)
 {
  if(i==j)
   continue;
 printf("\n %d %d",i,j);
```

```
}
}
```

Output

```
17
21
```

The goto statements

→ The goto statement causes your program to jump to a different location, rather than execute the next statement in sequence. The format of the goto statement is;

Consider the following program fragment.

```
if (size > 12)
  goto a;
  goto b;
a: cost = cost * 1.05;
flag = 2;
b: bill = cost * flag;
```