C is middle Level Language

→ Developed in At & T bell Laboratory by Dennis Ritche.

C Program Structure

• A 'c' program has following form ;

→ Preprocessor Commands
→ Type defination
→ Function proto types → declare function types & variables passed to function.
→ Variables
→ Functions .

~~Ace~~ Pre Processor

```
# include < file name .h>

# include < stdio .h >
```

Preprocessor directives are instuctions for the compiler.
Prepoocessor directives are prefixed by # character
In 'c' two different processor commands are needed.

① # include <file name .h> → This include directive used to link c source file objectives files and library files together.
② # define Name value. The define directive is used to set definations # define PI 3.14

Scanned by CamScanner

Character set

The character set in c language can be grouped into following catagories.

(i) letters → (Alphabet from A to z and a to z)
(ii) Digits → (Numbers from 0 to 9)
(iii) special ~~o~~ character.
(iv) white spaces.

Special Character

, Comma
. Period
; semicolon
: colon
? question mark
' Apostrope
~~" question ma~~
" Quotation mark
! exclamation
| vertical Bar
/ slass
\ Back slass
~ Tilde
~~[illegible]~~ _ Underscore
$ Dollar sign
~~[illegible]~~ & Ampersand
^ Carret
* Asterisk
- minus sign
+ Plus sign
< operation angel.
> closing angle
(Left parnthesis
) Right ,,
% percentage sign
[~~R~~ Left bracket
] Right bracket
Number sign.

White space

(1) Blank space
(2) Horrizontal Tab
(3) ~~Char~~ Carriage Return
(4) New Line
(5) Form feed.

Key word in "C"

auto	else	register	Union	break
enum	return	unsigned	Case	extenson
short	Void	Char	float	signed
Volatile	Consist	for	size of	while
Continues	go to	static	default	if
struct	do	int	switch	double
Long	typedef			

Constant

A constant can be define as a quantity that doesn't change during excuting of a program.

C support for constant

1. Integer Constants.
2. Real constants.
3. Single character constants.
4. String constants.

Integer Constant

→ Must be have atleast one digit and should not have decimal point.
→ It could be negative or positive.
→ 3types – Decimal integer – octal integer – Hexadecimal integer.

Real Integer

→ It must have atleast one digit and one decimal point.
→ The consist of a fractional part in their representation 0.0026, 0.97, 435.29.

ſingle Character conſtant

A ſingle character conſtant repreſent a ſingle character which iſ encloſed in pair of ~~a~~ quotation ſymbole.

example:- ~~[illegible]~~ '5', "a", etc

Backſlaſh Character

(\a) Audible alert (Ball)
(\f) Form feed
(\r) carriage Return
(\v) vertical Tab
(\") Double Quote
(\\) Back ſlaſh
(\b) Back ſlaſh
(\n) New line
(\t) Horizontal tab
"\'" ſingle quote
(\?) Queſtion mark
(\0) Null

Variableſ

A variable iſ a name given to the ſpace in memory for holding data ſuch aſ integerſ, ~~[illegible]~~ characterſ, floating point number, ſtring etc.

→ A variable aſ value that can change any time

Ex → number, ſalary
Emp - name
Emp - Code

Scanned by CamScanner

User defind type declaration

→ Used to declare variables.
Syntax :- typed-~~ef~~ type indntlfier.
ex :- typed ~~ef~~ ef int salary.
typedef float average.

Managing input output string.

include <stdio . h> standard input output header file.

Single character input output

We use get char function for this. The getchar takes following form variable — name — getchar ( ) ;

Formatted input

It refers to input data arranged in a particular format. Input values are generally taken by using scanf function. scanf ("control string", arg.1, arg.2, arg.3, ... arg n")

Inputing Integer numbers

% wd w is integer number that specifies the field width of the no to be read and 'd' is data type character.

Input Real Number

scanf ("%f" & variable)

ex:- scanf (" %f, %f, &a &b, &c")

| & → ampersand |

with input data 321.76, 4.321, 678.
The value 321.76 is assigned to a 4.321 to b
& 678.0 to c.

Input character strings

Single character or strings can be input by using character specifier.
% wc or % ws. and represent field width.

%c - Print a character.
%d - Print a Integer.
%i - Print a Integer.
%e - Print float value in expontaial form.
%f - Print float value.
%g - print using %e or %f which over is smaller
%o - Print actual value.
%s - Print a string.
%u - Print a unsigned integer.
%p - Print a pointer value.
%ld - Long.

Scanned by CamScanner

Operators

An operator is a symbol which helps the user to command the computer to do a certain and mathematical or logical manipulations operators are used in "c" language. Program to operate on data & variable.

① Arithmetic Operators
② Relational operators
③ Logical operators
④ Assignments operators
⑤ Increasement & decreasement operators.
⑥ Conditional operators.
⑦ Betwise operator.
⑧ special operators.

(1) Aritmetic operators are used for addition, subtraction, multipication, division, etc.
(+, -, x, ÷)

② Relational operators

It is used to compare the relationship between operands and bring out decision and program.

Operators		Description
>	⟶	greaterthan
<	⟶	Less than
>=	⟶	greaterthan or equall to
<=	⟶	Lessthan or equall to
=	⟶	equall to
≠	⟶	not equall to

3. logical Operators

logical operators the word logical refers to the ways these relationship can be connected ~~by~~ together using the rules of format logic.

operators	Meaning
&&	Logical AND
||	Logical OR
!	Logical not

4. Assignment operation.

Used to evaluate an expression on the right of the expression and substraction it to value or variable on the left expression.

operators

a = a+1	a+= 1
a = a-1	a- = 1
a = a*(n+1)	a* =(n+1)
a = a/(n+1)	a/= (n+1)
a= a % b	a% = b

5. Increasement and decreasement operators are one of the unary operators which are very useful in Languages. They are used extensively used in for and while loop.

1. ++ variable name
2. variable name ++
3. -- variable name
4. variable neme --

Scanned by CamScanner

⑥ Betwise operator

A betwise operator operates on each bit of data. These bits are used for testing. Complementing or shifting bits to the right or left. Betwise operators may not be applied to float or double.

Operator	Meaning
&	betwise AND
|	Betwise or
^	Betwise Enclusive
<<	shift Left
>>	shift Right

⑦ Special operators

C support special operators such as comma operator, size of operator, pointer operator (~~&~~ & and *) and member selection. (. and →)

Comma operator

The comma operator canbe used to link,

(6) Conditional an Tenenary Operator

The conditional operator consists of 2 symbols the question mark(?) and the colon (:)
The syntax of terenary operator is as follows.

exp1 ? exp2 : exp3

(7) Bitwise operators.

A bitwise operator operates on each bit of data These bits are used for testing, complementing or shifting bits to the right ~~&~~ or left. Bitwise operators may not be applied to a float or double

operator	Meaning.
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive
<<	Shift left
>>	Shift right

(8) Special Operators.

C supports some special operators such as comma operator, size of operator, pointer operators (& and *) and member selection operator (. and ->)

The comma Operator

The comma operator can be used to link related expressions together. A comma-linked list of expression are evaluated left to right and value of right most expression is the value of the combined expression.

For example the statement:

value = (x=10, y=5, x+y);

first assign 10 to x and 5 to y and finally assigns 15 to value. Some comma has the lowest precedence in operators the paranthesis is necessary. Some examples of comma operator are

In for loops.

```
for (n=1, m=10, n<=m, n++, m++)
```

In while loops

```
while (c=getChar(), c!='10')
```

Exchanging values

```
t=x, x=y, y=t;
```

The size of operator

The 'size of' operator gives the size of the data type or variable in terms of bytes occupied in the memory. The operand may be a variable, a constant or a data type qualifier.

Example:

```
m= size of(sum);
n= size of(long int);
k= size of(235L);
```

The size of operator is generally used to determine the length of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during the execution of programs.

Precedence in Arithmetic Operators

An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C

High priority * / %

Low priority + –

Rules for evaluation of expression

- First parenthesized sub expression left to right are evaluated.
- If parenthesis are nested, the evaluation begins with the innermost sub expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
- The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When Parenthesis are used, the expressions within parenthesis assume highest priority.

Type conversions in expressions

Implicit type conversion

C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance. This automatic type conversion is know as implicit type conversion.

During evaluation it adheres to very strict rules and type conversion. If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.

The following rules apply during evaluating expressions

All short and char are automatically converted to int then

1. If one operand is long double, the other will be converted to long double and result .....will be long double.
2. If one operand is double, the other will be converted to double and result will be double.
3. If one operand is float, the other will be converted to float and result will be float.
4. If one of the operand is unsigned long int, the other will be converted into unsigned .....long int and result will be unsigned long int.
5. If one operand is long int and other is unsigned int then
   .....a. If unsigned int can be converted to long int, then unsigned int operand will be ....converted as such and the result will be long int.
   .....b. Else Both operands will be converted to unsigned long int and the result will be ....unsigned long int.
6. If one of the operand is long int, the other will be converted to long int and the result will be long int.
7. If one operand is unsigned int the other will be converted to unsigned int and the .....result will be unsigned int.

Explicit Conversion

Many times there may arise a situation where we want to force a type conversion in a way that is different from automatic conversion.

Consider for example the calculation of number of female and male students in a class

$$\text{Ratio} = \frac{\text{femalestudents}}{\text{male students}}$$

Since if female students and male students are declared as integers, the decimal part will be rounded off and its ratio will represent a wrong figure. This problem can be solved by converting locally one of the variables to the floating point as shown below.

Ratio = (float) female students / male students

The operator float converts the female students to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed by floating point mode, thus retaining the fractional part of the result. The process of such a local conversion is known as explicit conversion or casting a value. The general form is

(type name) expression

Operator precedence and associativity

Each operator in C has a precedence associated with it. The precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of these levels. The operators of higher precedence are evaluated first.

The operators of same precedence are evaluated from right to left or from left to right depending on the level. This is known as associativity property of an operator.

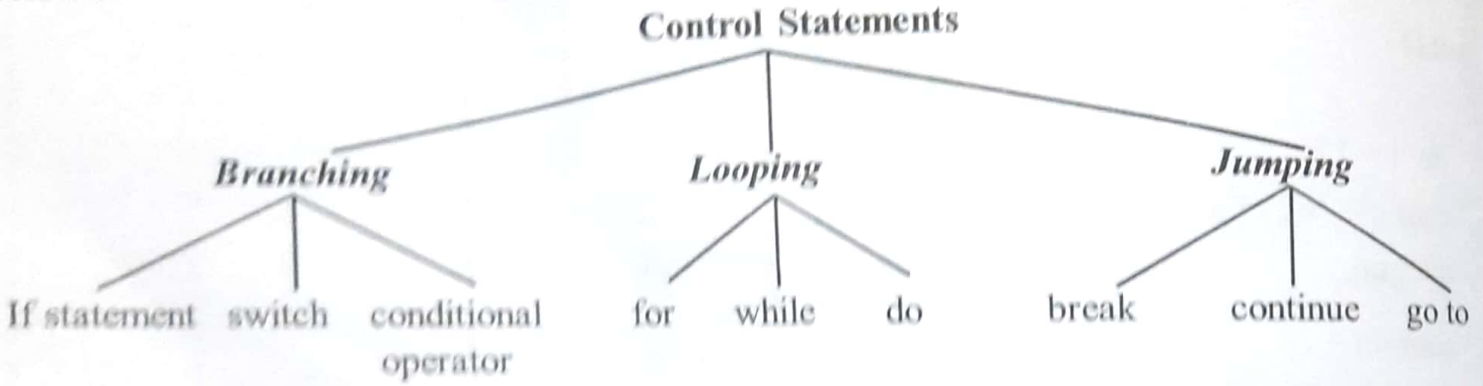
The table given below gives the precedence of each operator.

Order	Category	Operator	Operation	Associativity
1.	Highest precedence	( ) [ ] → :: .	Function call	L → R Left to Right
2.	Unary	! ~ + – ++ - - & * Size of	Logical negation (NOT) Bitwise 1's complement Unary plus Unary minus Pre or post increment Pre or post decrement Address Indirection Size of operant in bytes	R → L Right → Left

Order	Category	Operator	Operation	Associativity
3.	Member Access	.* →*	Dereference Dereference	L → R
4.	Multiplication	* / %	Multiply Divide Modulus	L → R
5.	Additive	+ –	Binary Plus Binary Minus	L → R
6.	Shift	<< >>	Shift Left Shift Right	L → R
7.	Relational	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	L → R
8.	Equality	= = ! =	Equal to Not Equal to	L → R
9.	Bitwise AAND	&	Bitwise AND	L → R
10.	Bitwise XOR	^	Bitwise XOR	L → R
11.	Bitwise OR	|	Bitwise OR	L → R
12.	Logical AND	&&	Logical AND	L → R
13.	Conditional	? :	Ternary Operator	R → L
14.	Assignment	= *= %= /= += – = &= ^= |= <<= >>=	Assignment Assign product Assign reminder Assign quotient Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	R → L

6.4 : Decision Control and Looping Statements

The control statements are used to control the cursor in a program according to the condition o according to the requirement in a loop. Further we can say, changing the order or flow controls, these ar required. There are mainly three types of control statement or flow controls. These are illustrated a below :



1. Branching Statement :

C supports many branching statement depending upon their flow of control and according the decision making policy. So it is also called decision-making statements. The various branching statemen used are as :

(a) if statement

(b) Switch tatement

(c) conditional operator statement

(a) **if statement :**

The if statement is a powerful decision making statement which can handle a single condition or group of statements. These have either true or false action.

There are mainly four types of if statements used in the C programming as :

(i) *Simple if statement*

(ii) *If-else statement*

(iii) *nested if statement*

(iv) *else-if or ladder if or multi-condition if statement*

(i) ***Simple if statement :***

When only one condition occurs in a statement, then simple if statement is used having one block. The general syntax and general flow symbol of simple if statement is :

if (condition)

{

true-statement block;

}

statement –x;

Here first of all condition is true, then the statement block will be executed and after execution of this block, statement –x will be executed. But if condition is false, then only statements-x will be executed. Note that both the times statment-x will be executed. For example, Suppose if you want to assign 'A' grade to those students who have percentage greated than 80 and give the result pass to all those student whether they have not 'A' grade. The procedure is as follows :

Program 6.1. : Program to compute the grade using simple if statement :

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
main( )
      {
      float per;
      char grade = " ";
      clrscr ();
      puts("\n Enter the percentage:");
      scanf ("%f', &per);
      if (per>=80)
         {
         grade = 'A'
         }
      printf ("\n Result is pass");
      print("n\Grade %c",grade);
sleep(10);
}
```

The output is :

```
Enter the percentage :90
Result is pass
Grade A
```

(ii) ***if-else statement :***

This statement also has a single condition with two different blocks. One is true block and other false block. The general syntax and general flow symbol used is as :

```
if (condition)
    {
        true statement block;
    }
else
    {
    flase statement block;
    }
statement-x;
```

In this statement block, first of all condition will be checked. If condition is true then true statement block will be exceuted and after execution of true block, statement x will be executed., but if condition is false then first false block statements will be exceuted and then statement-x will be exceuted. Note that in both the cases statement-x will be executed.

For example, suppose if you want to find the greatest number between to numbers, then the program procedure is written as belowl;

Program 6.2 : Program to find the greatest number between two numbers.

```
#include<stdio.h>
#include<conio.h>
main( )
    {
        int a,b,g;
        clrscr( );
        printf("\n Enter the value of two numbers:");
        scanf("%d%d",&a,&b);
        if(a>b)
        {
        g = a;
        }
else
{
g = b;
}
printf ("\n Greatest Number is : %d",g);
getche ( );
}
```

The output is :

```
Enter the value of two numbers : 40 36
Greatest Number is : 40
```

(iii) ***Nested if statement :***

When an if statement occurs within another if statement, then such type of is called **nested if statement.** This statement is some complex than the simple if or if-else statement, but is very useful to solve real problems.

Nested if statement can be used to solve the problems which have embedded type conditions(in other words conditions based on some more conditions). The general syntax of this statement is as:

```
if (condition -1)
    {
        if (condition -2)
        {
                st-1;
        }
    else
        {
                sf-1;
        }
    }
else
    {
    if(condition-3)
    {
        st-2;
    }
else
    {
        sf-2;
    }
} statement-x;
```

```
getche( );
}
```

The output is:

```
Enter the three numbers : 20 30 10
Greatest number is :30
```

(iv) ***Ladder if or else if statement(if-else-if) :***

When in a complex problem number of conditions arise in a sequence, then we can use ladder-if or else if statement to solve the problem in a simple manner.

In this statement first condition will be checked, if it is true then action will be taken, otherwise of further next condition will be checked and this process will continue till the end of the condition.

The general syntax is as :

```
if (condition-1)
    {
        st-1;
    }
    else if (conditoin-2)
    {
        st-2;
    }
    else if (condition-3)
    {
        st-3;
    }
    ...
    ...
    ...
    else if (condition-n)
        {
                st-n;
        }
    else
    {
    default-statement;
    }
statement-x;
```

For example, suppose if you want to print Red, Green, White, Yellow colors according to codes 1, 2, 3, then program is as :

(b) Switch statement :

When number of conditions (multiple conditions)occurs in a problem and it is very difficult to solve such type of complex problem with the help of **ladder if** statement, then there is need of such type of statement which should have different alternatives or different cases to solve the problem in simple and easy way. For this purpose **Switch Statement** is used. It is also called **Case statement** because it has different cases and different blocks. It is also called multi-decision statement having multiple blocks.

The general syntax of this statement is as :

```
switch(e or v)
    {
case values 1:
        block 1;
        break;
case value 2 :
        block 2;
        break;
case values3:
        block3;
        break;
...
...
case value n :
        block n;
        break;
default :
        block n+1;
}
statement-x;
```

Where is an arithmetic expression and v is the variable.

Program 6.5. : Program to print the color according to the code.

```
#include<stdio.h>
main( )
{
int code;
printf("\n MAIN MENU");
printf("\n1.For color Red");
printf("\n2.For color Green");
printf("\n3.For color White");
printf("\n4.For color Yellow");
printf("\n Enter the color code:");
scanf("%d",&code);
switch(code)
{
case 1:
        printf("\n color is RED");
        break;
case 2:
        printf("\n color is GREEN");
        break;
case 3:
        printf("\n color is WHITE");
        break;
case 4:
        printf("\n color is YELLOW");
        break;
default :
        printf("\n Color does not found");
}
        getche();
}
```

The output is :

```
MAIN MENU
1.  For color Red
2.  For color Green
3.  For color White
4.  For color Yellow
Enter the color code: 2
color is GREEN
```

(c) Conditional Control Statement :

This statement is based on conditional operator. This statement solves the problem's conditon single line and is a fast executable operation. For this purpose, we can take combination of ? and

The general syntax used for such type of statement is as:

exp1 ? exp2 : exp 3;

This is the substitute of if-else statement. This is used for a single condition. Similarly for nesting of condition, the conditional control statement has the general syntax as :

expl ? (exp2 ? exp3 : exp4): exp5 ;

This is used for two conditions. Here first of all exp1 will be executed and if it is true, then further exp2 will be checked. If exp2 is true, then exp3 will be executed, otherwise exp4 will be executed. But if exp1 is false, then only the exp5 will be executed. This is the substitute of **nested if statement**.

2. Looping Statements :

When a single statement or a group of statements will be executed again and again in a program (in an interative way), then such type processing is called loop. The looping statements used in C-language are :

(a). while statement or shile loop

(b). do statement or do loop

(c). for statement or for loop

(d). Nested for loop statement

(a) while statement :

While statement or while loop is an entry control loop. In this, first of all condition is checked and if it is true, then group of statements or body of loop is executed. It will execute again and again till condition becomes false.

The general syntax is :

```
while (test condition)
    {
    block of statements;
    }
statement -x;
```

For example, Suppose if you want to find the average of first n positive integer numbers, then the procedure applied to compute the average by using the **while loop** is :

Program 6.6. : Program to illustrate the concept of while loop.

```
#inclued<stdio.h>
#include<conio.h>
main( )
{
int i, sum, n;
float av;
clrscr( );
printf("\n Enter the value of n:");
scanf("%d",&n);
sum=0;
i=1;
while(i <=n)
{
        sum = sum +i;
        i = i + 1;

    }
    av=(float) sum/n;
    printf("\n sum is :%d", sum);
    printf("\n Average is: %f", av);
    getche();
    }
```

The output is :

```
Enter the value of n: 10
sum is: 55
Average is: 5.5
```

(b) do statement or do-loop or do-while( ) :

It is also called do-while statement. In this statement, first body of the loop is executed and the condition is checked. If condition is true, then the body of the loop is executed. When con becomes false, then it will exit from the loop. The syntax of do-loop is as follows :

```
do
    {
    block of statements;
    }
    while(condition);
statement-x;
```

Note that semicolon(;) must be at the end of the while condition. For example as int he above statment in the while loop, the average of n positive integer numbe by using the **do-while loop** ca computed as :

Program 6.7. : Program to find the average of N numbers using do-while()loop.

```
#include<stdio.h>
#include<condio.h>
main( )
    {
        int sum, n, i;
        float av;
        printf("\n Enter the value of n:");
        scanf("%d",&n);
            sum =0
            i = 1;
            do
```

```
                {
                sum = sum + i;
                    i = i +1;
                }  while(i < = n);
printf ("\n Sum is %d", sum);
av=(float)sum/n;
printf("\n Average is :%d",av);
getche();
```

The output is :

```
Enter the value of n:10
sum is: 55
Average is : 5.5
```

(c) for statement or for loop :

It is a looping statement, which repeat again and again till it satisfies the defined condition. It is one step loop, which initialize, check the condition and increment/ decrement the step in the loop in a single statement. The general syntax is as :

```
for (initial value; test condition; increment/ decrement)
    {
        body of the loop;
    }
    while(condition);
statement-x;
```

It is also entry controlled loop, where first condition is checked and the body of the loop be executed. In this, first we initalize the value, then in the loop we apply the condition and further we increment or decrement the loop according to requirement. After execution or completion of the body of the loop, when the condition becomes false, the statement-x will be executed. For example, to find average of first n positive integer numbers, the for loop procedure be applied as:

Program 6.8 : Program to find the average of Numbers by using the for loop.

```
#include<stdio.h>
#include<conio.h>
main()
    {
        int n, i, sum;
        float av;
        clrscr();
        printf("\n Enter the value of N:");
        scanf("%d",&n);
        for(i =1; i<=n; i ++)
```

x = y, y = t ;
2 0 0

```
        {
                sum=sum + i;

        }
    Printf("\n Sum is : %d", sum);
    av= ( float) sum/n;
    printf("\n Average is : %f",av);
    getche( );
}
```

The output is:

```
Enter the value of n:10
Sum is : 55
Average is : 5.5
```

(d) Nested for statement :

When a for statement is executed within another for statement, then it is called nested for stat

We can apply number of nested for statements in C-language.

The general syntax is;

```
for (initial value; test condition 1; increment1/ decrement 1)
    {
    for (initial value 2; test condition 2; increment2/ decrement2)
        {
            inner-body of the loop;
        }
    outer-body of the loop
    }  statement-x;
```

For one value of outer loop, inner looop will repeat upto test condition2. So inner loop completed first and then outer loop will be completed. After completion of inner and outer loop, stat x will be executed. For example, suppose if you want to print "Hello!" six times and "OK" two then this can be print by using for loop. The procedure is as follows :

Program 6.9 : Program to illustrate the concept of nested for loop.

```
main( )
    {
        int n=2, m=3, i, j;
        printf("\n OUTPUT is as :");
        for (i =1; i < n; i++)
            {
                for (j =1; j< =m; j++)
                    {
                        printf("\n Hello!");
            }
            printf("\n OK");
        }
    getche();
}
```

The result of the above program be as :

OUTPUT is as:

```
Hello!
Hello!
Hello!
OK
Hello!
Hello!
Hello!
OK
```

3. Jumping Statement :

There are three different controls used to jump from one C program statement to another and make the execution of the programming procedure fast. These three Jumping controls are:

(a). goto statement

(b). break statement

(c). continue statement

(a). goto statement :

The powerful Jumping Statement in the C language is **goto** statement. It is sometimes also called part of branching statement. The go to moves the controls on a specified address called label or label name. The goto is mainly of two types. One is conditional and the other is unconditional.

Also jump can be either in forward direction or in backward direction. The different types of goto statement is :

(i) forward goto

(ii) Backward goto

(i) Forward goto :

In this the control moves forward at a specified label either according to a condition or without condition. The general syntax is as :

```
sl;
s2;
goto label;
s3;
s4;
label :
s5;
s6;
```

Unconditional forward goto statement

```
sl;
s2;
if (condition)
goto label;
s3;
s4;
label:
s5;
s6;
```

Conditional forward goto statement

Note that here first statement s1 and s2 will be executed, then specified label(in C-languag label name) and execute the statement s5 and s6. It will skip a part of the program i.e. s3 and s4 state In C-language label name is either a single character or combination of characters.

This concept can be illustrated with the help of following C program as:

Program 6.10 : Program to add and subtract two numbers by using unconditional g to statement.

```
#include<stdio.h>
main()
{
    int a, b, c, d ;
    printf("\n Enter the value of a and b:");
    scanf("%d%d", &a,&b);
    c= a+b;
    d=a–b;
    printf("\n Addition result is :%d",c);
    goto mm;
    printf("\n Subtraction result is :%d",d);
    mm:
    printf("\n End of the program");
    getche();
}
```

The output is:

```
Enter the value of a and b: 5 2
Addition result is :7
End of the program
```

Backward goto :

The backward go to or backward jump moves the control back to the specified addresses and so creates a loop.

In the case of conditional backward statement, it creates finite looping. But in the case of unconditional backward go to or jump, it creates infinite looping.

The general syntax of backward jump statment is as follows :

sl;

label :

s2 ;

s3;

goto label;

s4;

Unconditional backward goto statement

sl;

label :

s2;

s3;

if(condition)

goto label;

s4;

Conditional backward goto statement

Here first of all statements s1, s2 and s3 will be executed. Then it will find a backward go to statement with a specified address and again s2, s3 statement will repeat either according to a condition or infinite times if there is no condition. Below written example is of conditional backward go to:

Program 6.11 : Program to compute a square value by using the conditional backward goto statement.

```
# include<stdio.h>
main()
      {
      int x, y;
start :
      printf("\n Enter the value of x:");
      scanf("%d", &x);
      if(x<0)
         {
         goto start;
```

```
        }
        y=sqrt(x);
        printf("\n Square root of %d is %d", x, y);
        getche();

}
```

The output is :

```
Enter the value of x : -5
Enter the value of x : 3
Square root of 3 is 9
```

(b) break Statement :

The statement always used with the decision-making statement like if and switch statements. Th statemnt will quit from the loop when the condition is true.

The general syntax for break statement is as :

break;

This statement is used within do-loop, while loop and for loop. The use of break statement wi the while loop structure is as :

```
while (conditional)
{
...
if (condition2)
      {
      break;
      }
}     statement-x;
```

By using structure of the **for loop,** the break statement is used as:

```
for (initial value; condition, increment/ decrement)
      {
         if(condition-1)
               {
               break;
               }
      }
      statement-x;
```

As in the above structure, when condition-1 becomes true, then it will break the loop and quit fr the for loop and move to the statement-x, i.e. after exiting from the for loop, statement-x will be execut For example, Suppose you are computing the sum of first n positive integer number (i=1 to n) and t computation will quit if the value of the I becomes 7.

continue statement :

c) Continue statemnt also comes with if statemnt. This statement is also used within any loop statement ke do loop, while loop and for statement. The general syntax of this statement is as :

continue ;

This statement will skip some part of iteration(loop) and comes to the next looping step i.e. it will increment/ decrement the loop value, when continue occurs. The general structure of the continue statement is as :

```
while(condition-1)
    {
       sl;
       s2;
    if(condition-2)
       {
       continue;
       }
    s3;
    s4;
    }
Statement-x;
```

In the above structure, First of all the statement s1 and s2 will be executed. When a condition met within the loop and if the condition becomes true, then the continue statement be executed and it will move to start the next looping iteration by skipping the statement s3 and s4. When condition becomes false, then the statement s3 and s4 will execute. After completion of the loop, the statement-x will be executed.