C PROGRAMMING LANGUAGE

For The Beginners

DP Publishing


C
PROGRA
LANGU
For The Beginners
01010101
01010101
01010101

• What is C?

C is a general-purpose computer programming language developed by **Dennis Ritchie in 1972** at the Bell Telephone Laboratories for use with the **UNIX** operating system. Today, C is running under a number of operating systems including **MS-DOS (Microsoft Disk Operating System).**
➢ The C compiler combines the compatibility of assembly language (low

level language) with the feature of a high-level language therefore C is a higher level programming language.
➢ It is well suited for writing both system software and business package.
➢ C is a most robust language whose rich set of built in functions; operators can be used to write a complex program.

➢ C is highly portable. This means that C program written for one computer can be run on another computer without any modification or little modification.

➢ C is a procedure oriented (procedural) and structural programming language.

• Low-Level and High-Level Language

Low-Level Language
➢ In low level language programming we directly deal with memory addresses and
CPU registers.

➢ These programs directly convert into source code to code which is understandable by CPU.
➢ The example of low level language is Assembly Language .

High-Level Language

➢ In high level language programming to refer the memory addresses we use variable and pointers and there is strict syntax which is to be followed.

➢ Source code firstly compiled and converts into machine code. These machine codes convert into the code which is understandable by CPU. ➢ High level languages are C, C++, Java etc…

• How to Install and Start C?

➢ First download the freeware C language compiler from internet or CD. Many compilers are available but most popular compiler is **Turbo C (TURBOC.exe)** .

➢ **Turbo C** was an Integrated Development Environment **(IDE)** and compiler for the C Programming language from **Borland** .

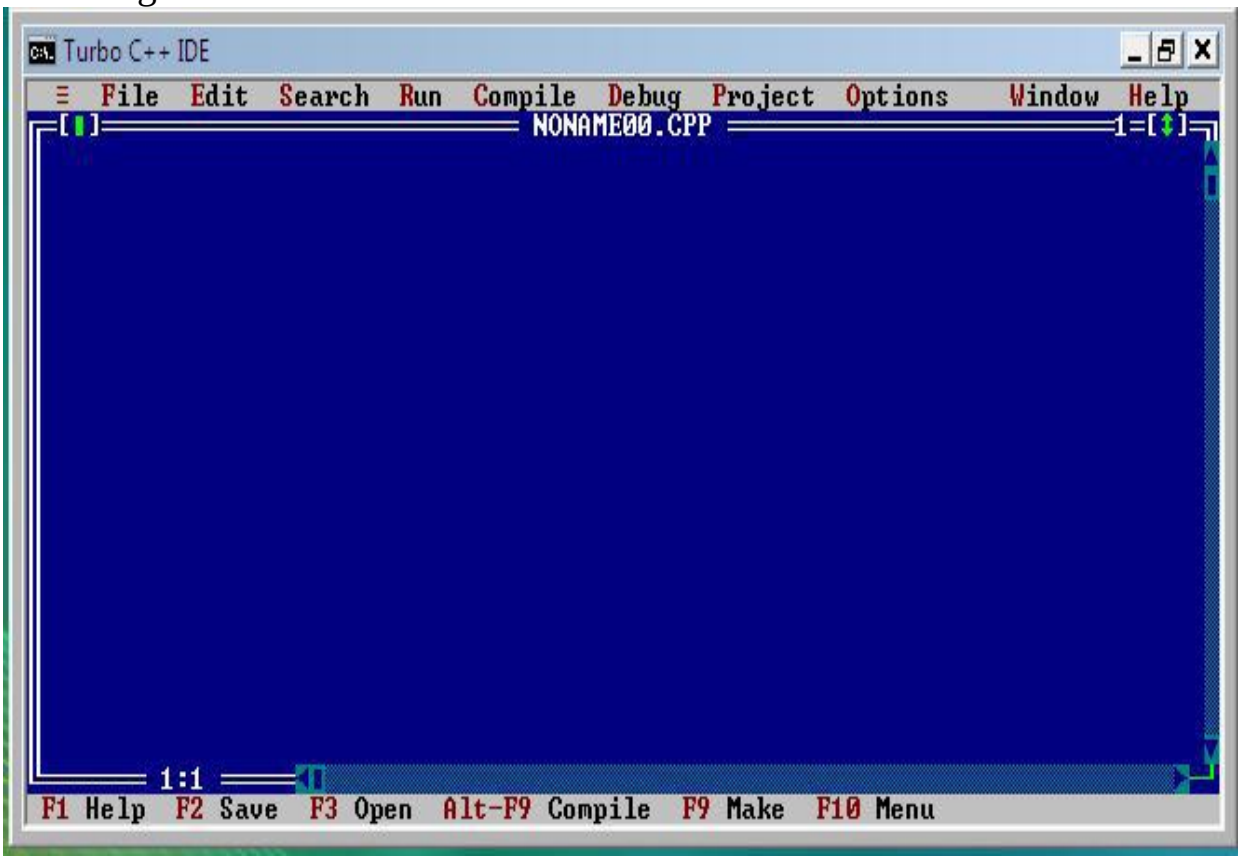
➢ Run TURBOC.exe file with following option

o Turboc.exe **C: -d-o**

➢ When run above command then create TC folder in C: drive.

➢ Now run the **TC.exe** available in C:\TC\BIN directory then display

following screen.



• Operate Turbo C++ IDE

➢ Some short cut keys are display in status bar such as (F1 for Help, F2 for Save etc…)
➢ For other operation you have to use the MENU options such as File, Edit, Search, Run, Compile etc...

Some Useful Operations

Create a New File
Save a File
Open a File
Close a File
Compile a File
Create a EXE file

Run a File
For User Screen (View Output)

FILE -> New
FILE -> Save **(F2)**
FILE -> Open **(F3)**
Alt + F3
COMPILE -> Compile **(Alt + F9)** COMPILE -> Make **(F9)**
RUN -> Run **(Ctrl + F9)**
WINDOW -> User Screen **(Alt + F5)**

• C PROGRAM STRUCTURE

C language program is consists of following sections.
Documentation Section
Link Section
Definition Section
Global Declaration Section
main( ) Function Section {
}
Subprogram Section
Function 1 (User Define Functions) Function 2

1. Document Section

The documentation section consists of a set of comment lines giving program information.

2. Link Section

The link section provides instruction to the compiler to link functions from the system library.

3. Definition Section

The definition section defines all symbolic constants.

4. Global Declaration Section

There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions.

5. main Function Section
Every C program must have one main( ) function section.
6. Subprogram Section
The subprogram section contains all the user defined functions that are called in the main function.

• C CHARACTER SET

C language character set denotes any valid character in C programming language. This character can be alphabet, digit, special symbols and white spaces. Following table shows the valid alphabets, numbers, special symbols and white spaces allowed in C language.

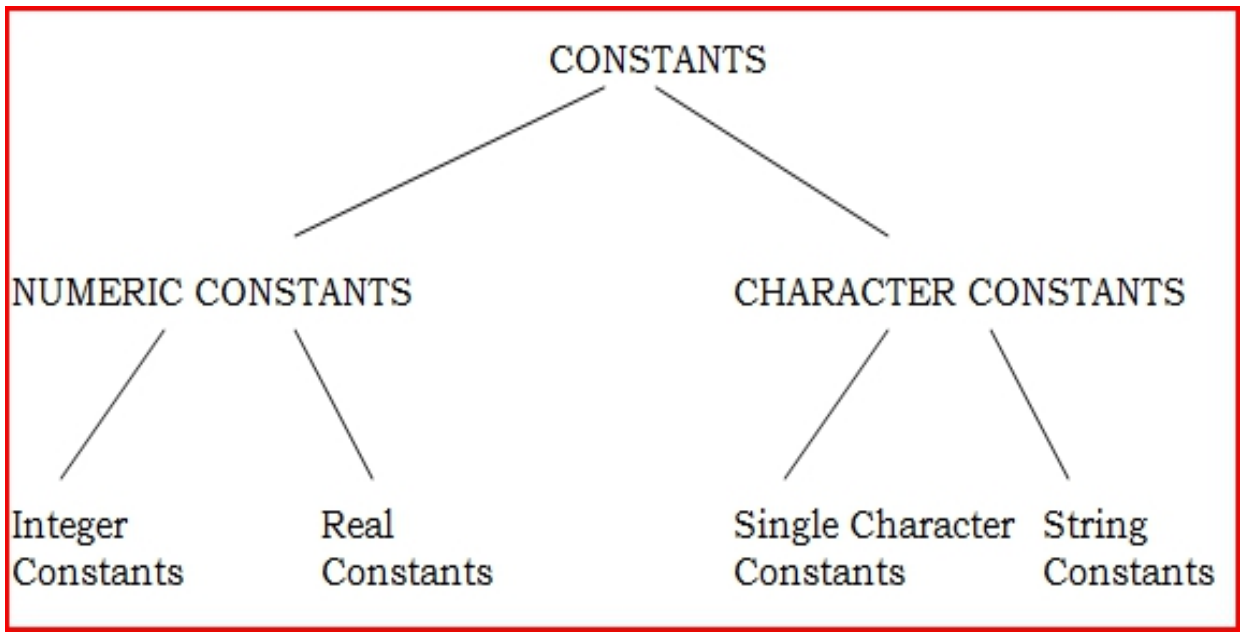
Alphabet Digits A, B, ..., Y, Z. a, b, ..., y, z. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Symbols
, comma ; semi colon . period / slash ~ tilde
$ dollar sign

White spaces
Blank space Horizontal tab Carriage return New line
Form feed

• C CONSTANTS

Constants in C refer to fixed values that do not change during the execution of a program. C language constants are also known as Literals. C supports several types of constants as under.



Integer constants

An integer constant refers to a sequence of digits. Decimal integers consist of a set of digits, 0 to 9, preceded by an optional – or + sign. Valid examples of decimal integer constants are 123, -321, 0, +78

Real constants
Real constants are numerical values with fractional part. Valid examples of real constants are 215.50, .98, +58.90, -0.80

Single Character Constants
A single character constant contains a single character enclosed within a pair of single quote marks. Examples of character constants are '5', 'A', 'x'

String Constants

A string constant is a sequence of characters enclosed in double quotes. The string may contain letters, numbers, special characters and blank space. Examples are "Hello!", "1987", "Wel Come", "? ….!", "5+3"

• C LANGUAGE VARIABLES (C LANGUAGE IDENTIFIERS)

A variable is a data name that may be used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution. A variable name can be chosen by the programmer in a meaningful way. A variable name must follow following rules.

1. The first character in the variable name must be Letter.
2. Variable name length is maximum 31 characters.
3. Keywords are not allowed as variable name.
4. No commas or blanks are allowed within a variable name.
5. No special symbol other than underscore(as in **gross_sal**) can be used in a variable name.

Some examples of valid variable names are *John, Value, T_raise, Delhi, x1, ph_value, mark, sum1, distance*
Some examples of invalid variable names are *123, (area), x-y, %, 25 th , do*

• C LANGUAGE KEYWORDS

In C language, keywords are special words with fixed meaning. These meanings cannot be changed.

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

• C LANGUAGE DATA TYPES

PRIMARY DATA TYPES
Integral Data Types Character
Signed Unsigned

int unsigned int char short int unsigned short int signed char long unsigned long unsigned char

Floating point Type void float double long double
Type Length Range

char 8 bits -128 to 127 signed char 8 bits -128 to 127 unsigned char 8 bits 0 to 255

int 16 bits -32768 to 32767 signed int 16 bits -32768 to 32767 unsigned int 16 bits 0 to 65535 signed short int 8 bits -128 to 127 unsigned short int 8 bits 0 to 255

long 32 bits -2147483648 to 2147483647 signed long 32 bits -2147483648 to 2147483647 unsigned long 32 bits 0 to 4294967295

float 32 bits $3.4 * 10^{-38}$ to $3.4 * 10^{+38}$ double 64 bits $1.7 * 10^{-308}$ to $1.7 * 10^{+308}$ long double 80 bits $3.4 * 10^{-4932}$ to $3.4 * 10^{+4932}$

• TYPE CONVERSION AND TYPE CASTING IN C LANGUAGE

Any mathematical operation requires same data type of operands. In mathematical expression, various data types of operands are used. In this situation expression cannot be evaluated. To avoid this, C language provides automatic type conversion.
Type conversion is a process to convert one data type to another data type. If lower data type is converted to higher data type, this is known as *type promotion* . If higher data type is converted to lower data type, this is known as *type demotion.*

int i,x;
float f;
double d; long l;

x = l / i + i * f d;
Type Casting

When type conversion process is handled by programmer, this is known as type casting or explicit type casting. In this process, programmer explicitly converts one data type to another data type using type operator as shown below.

Example :
sum = sum + (float) 1 / i;
Here, value 1 is converted in to float before performing operation with variable i.

• C LANGUAGE OPERATORS

Operator Description

( ) Function Call
[ ] Array element reference + Unary Plus
- Unary Minus
++ Increment
- Decrement
! Logical Negation
~ One's Complement * Pointer reference & Address
sizeof Size of an object
(type) Type case (conversion) * *Multiplication*
/ Division
% Modulus
+ Addition
- Substraction
<< Left shift
>> Right shift
< Less than
<= Less than or equal to > Greater than
>= Greater than or equal to *== Equality*
!= Inequality
& Bitwise AND
^ Bitwise XOR
| Bitwise OR

&& Logical AND
|| Logical OR
?: Conditional expression = *=
/= %=

+= -= Assignment Operators &= ^=

|= <<=
>>=
, Comma operator

Associativity Rank / Precedence *L To R 1*
R To L 2
L To R 3
L To R 4
L To R 5
L To R 6
L To R 7

L To R 8 L To R 9 L To R 10 L To R 11 L To R 12 R To L 13

R To L 14
L To R 15

Operator Precedence: When more than one operator are present in an expression, operator evaluated first. Operator priority.
precedence precedence decides which operator is is also known as operator

Operator Associativity : When more than one operator with same precedence are present , operator associativity decides operator is evaluated left to right or visa versa.

• **ARITHMETIC OPERATORS IN C**
There are five arithmetic operators available in C language.

Operator Description Example ***** Multiplication n1 * n2 / Division n1 / n2 % Modulus n1 % n2 + Addition n1 + n2
- Subtraction n1 – n2

Result
Multiplication of n1 and n2 Division of n1 and n2
Modulus of n1 and n2 Addition of n1 and n2
Subtraction of n1 and n2

Multiplication, Division and Modulus operator's priority is higher than Addition and Subtraction operator. So if in any arithmetic statement these operators are available then Multiplication, Division and Modulus operation is performed first. After that Addition and Subtraction is performed.
Example :

#include < stdio.h > #include < conio.h > void main( )
{

int i=3,j=4,k=5;
int a,b,c;
clrscr( );
a= **i * i** ;
printf(" %d ",a); // prints 9
b = **4 + 4 / 2 * 3** ;
printf("\n%d",b); // prints 10 because division and multiplication first. c = **12 % 5** ;
printf("\n%d",c); // prints 2

}

• **RELATIONAL OPERATORS IN C**

There are six relational operators available in C language.

Operator Description Example < Less than n1 < n2

<= Less than n1 <= n2 equal to
> Greater than n1 > n2

>= Greater than n1 >= n2
equal to
== Equality n1 == n2
!= Inequality n1 != n2

Result
true if value of n1 is less than n2
true if value of n1 is less than or equal to n2
true if value of n1 is greater than n2
true if value of n1 is greater than or equal to n2
true if n1 and n2 are equal. **true** if n1 and n2 are not equal. ➢ Relation operators are useful in relational conditions (i.e. decision

making and looping structures).
➢ Relation operators are binary operators and require two operands. ➢ Relation operator returns result of operation either true (1 or other

than zero) or false (zero).
Example :
#include < stdio.h > #include < conio.h >

void main( ) {
int a,b; clrscr( );

a=10; b=10;

if (**a==b**)
printf(“ Both are equal “);
if (**a!=b**)
printf(“ Both are inequal “);
if (**a>=b**)
printf(“ a is greater than and equal to b “);
if (**a>b**)
printf(“ a is greater than b “);
if (**a<=b**)
printf(“ a is less than and equal to b “);
if (**a<b**)
printf(“ a is less than b “);
}

• LOGICAL OPERATORS IN C

There are three logical operators in C Language. Logical operators are use to join multiple relation conditions.

Operator Description Example ! Logical NOT ! n1
&& Logical AND n1 > 0

&& n1 <= 100 | | Logical OR n1 == n2
|| n1 == n3

Result
returns **true** if n1 is **false** . returns **true** only if both conditions are **true**
returns **true** if any one condition is **true** .
Example :

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

void main( ) {
float per; clrscr( );

printf(“Enter Student’s Percentage of Marks : “); scanf(“%d”, &per);
if(per < 40 || per >100) printf(“ Fail “);
if (per >= 70)
printf(“ Distinction “);
if (per >= 60 **&&** per < 70) printf(“ First Class “);
if (per >= 48 **&&** per < 60) printf(“ Second Class”);
if(per >=40 **&&** per < 48) printf(“ Pass Class “); }

• **TERNARY OPERATORS IN C (CONDITIONAL OPERATOR)** The conditional operator ***? and :*** are sometimes called ternary operators because they take three arguments. Ternary operator can be use instead of if … else condition.

Syntax :
*expression1 **?** expression2 **:** expression3*
expression1 expression2

expression3 Relational condition
if Relational condition in expression1 is **true,** the value returned will be expression2.
if Relational condition in expression1 is **false** , the value returned will be expression3.

y=(x > 5 ? 3 : 4);
This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.

• UNARY INCREMENT AND DECREMENT OPERATOR (++ , - -)
Two types of Unary operator (1) increment operator ++ (2) decrement operator --. Those work in two flavors PRE and POST as per its position.
pre increment

Increment operator position is before the operand. In pre increment operator, the operand value is increased by one before assignment. *Ans= ++a;*
It is combination of two statements (1) *a=a+1; (2) ans=a;*
post increment
Increment operator position is after the operand. In post increment operator, the operand value is increased by one after any assignment. *Ans= a++;*
It is combination of two statements (1) *ans=a; (2) a=a+1;*
pre decrement
Decrement operator position is before the operand. In pre decrement operator, the operand value is decrease by one before any assignment. *Ans= --a;*
It is combination of two statements (1) *a=a-1; (2) ans=a;*
post decrement
Decrement operator position is after the operand. In post decrement operator, the operand value is decrease by one after any assignment. *Ans= a--;*
It is combination of two statements (1) *ans=a; (2) a=a-1;*
Example :

```
#include <stdio.h>

void main( ) {
int a,b,c,d;
int i=2,j=2,k=2,m=2;
a=++i;
b=j++;
c= --k;
d= m--;
printf("\n a=%d b=%d c=%d d=%d",a,b,c,d ) ; // Returns a=3 b=2 c=1 d=2
printf("\n i=%d j=%d k=%d m=%d",i, j, k, m ) ; // Returns i=3 j=3 k=1 m=1
```

• **C LANGUAGE TOKENS**

The language elements are called tokens. Identifiers, operators, keywords, Literals and special symbols are examples of tokens. ***Identifiers:***

The names of variables, functions or arrays in a program are an identifier.

Keywords

The keywords are reserved identifiers used by C language. C language has total 32 keywords in all. Some of them are *do, while, for, break etc.…*

Literals (Constants)

A literal denotes constant value.

integer : 200 0 -7
real : 3.14 -3.14 3e+5
character : ‘a’ ‘A’ ‘ * ‘ ‘ $ ’
string : “abba” “3.14” “is”

Comments

The compiler ignores everything written in comments. The C language supports two kinds of comments:

Multi Line comment /* *text* */
Line Comment // *text*

Operators

The operators are necessary to perform various operations. C Language supports following types of operators.

Arithmetic operators (+ , - , * , % , /)
Relational operators (< , > , >= , <=, == , !=)
Logical operators(! , && , ||)
Conditional operator (? :)

Special symbols

C language supports special symbols in the program. Some of them are as under.

Before preprocessor directives.
{ When program starts.
} When program ends.
; To terminate program statements.

• Defining Constants in C Program

There are two simple ways in C to define constants:
1. Using **#define** preprocessor.
2. Using **const** keyword.

The #define Preprocessor:
The #define macro definitions allow constant values to be declared for use throughout your code. Macro definitions are not variables and cannot be changed by your program code like variables. The syntax for creating a constant using #define in the C language is:

#define identifier value
For example: #define AGE 10
The const Keyword
We can use *const* prefix to declare constants with a specific data type. The syntax for creating a constant using *const* in the C language is:

const data-type variable = value;
For example: const int AGE=10;
PROGRAM:

#include <stdio.h>
#define LENGTH 10 #define NEWLINE '\n'
void main() {
const int WIDTH=5; int area;

area = LENGTH * WIDTH;
printf("value of area : %d", area); printf("%c", NEWLINE);

• DECISION STRUCTURES

1. if Statement
The if statement is a powerful decision making statement and is used to control the flow of execution of statements.

Syntax : **if** (*relational Condition)* { …

Statements-block; …
}
Statement-x;

If relational condition is **true** then **statements-block** is executed in the program otherwise skipped and the execution will jump to the **statement-x** .
Execution Flow of If Statement:
Entry
Condition^{True}
False Statements-block
Statement-x
Next Statements
Example :

#include<stdio.h> void main( )
{

int i;
clrscr( );
printf(“Enter Any Number : “); scanf(“%d”,&i);

if (i % 2 == 0)

{
printf(“ Number is divisible by 2 “);
}

2. if … else Statement
The **if … else** statement is an extension of the simple **if** statement.

Syntax : **if** (*relational Condition)* {

True block statements; }
else
{

False block statements; }
Statement –x;

If relational condition is **true** then **true block statements** part is executed. If relational condition is **false** then **false block statement part (else part)** is executed.

Execution Flow of If…Else Statement:
Entry
False Condition[True]
False block statements True block statements
Statement-x
Next Statements

Example : #include<stdio.h> void main( )
{

int i; printf(“Enter Any Number : “); scanf(“%d”,&i);
if (i % 2 == 0)
{

printf(“ Number is Even “); }
else
{

printf(“ Number is Odd “); }

3. Nested if … else Statement

When a series of decisions are involved, we may have to use more than one if … else statement in nested form as follows.
Syntax :

if (relational condition -1) {
if (relational condition -2) {
Statement-1; }
else
{
Statement-2; }
}

else
{
Statement-3;
}

Execution Flow of Nested If Statement:
Entry
False Condition-1 True
False Condition-2 Statement-3
True
Statement-2 Statement-1
Statement-x

Next Statements void main( ) {

int amt, discount; clrscr( );
if(amt > 5000) {
if (amt > 10000)
discount=amt*0.20;

else
discount=amt*0.10; }
else
{

discount=amt*0.05; }
}

4. if … else if Statement

There is another way of putting if conditions together when multi path decisions are involved. A multi path decision is a chain of if conditions in which the statement associated with each else is an if.

Syntax :
if (relational condition 1) statement – 1;

else if (relational condition 2) statements - 2;
else if(relational condition 3) statements - 3;

else
default – statement;

statement – x;

Example :
#include<stdio.h> void main( )
{

float per;
clrscr( );
printf(“Enter percentage : “); scanf(“%f”,&per);
if(per >= 70)

printf(“Distintion”); else if(per >= 60)
printf(“First class”); else if(per >= 48)
printf(“Second class”); else if(per >= 40)
printf(“Pass class”); else
printf(“Fail”); }

5. switch … case Statement

We know that when one of the many alternatives is to be selected, we can design a program using if statements to control the selection. However, if statements increase the complexity of the program. To solve this problem switch statement is useful.

Syntax : **switch** (*integer expression)* {

case constant 1 : do this;
case constant 2 : do this;
case constant 3 : do this;
…
default :
do this; }

Each case is compared with integer expression. If any case matches, statements following that case are executed. If any case does not match with integer expression, default case is executed.

```
#include<stdio.h> void main( )
{

int i=2;
clrscr( );
switch ( i ) {

case 1 :
printf( “I am in case 1 \n” ); break;

case 2 :
printf( “I am in case 2 \n” ); break;

case 3 :
printf( “I am in case 3 \n” ); break;

default :
printf( “I am in default \n” ); }
}
```

• LOOP STRUCTURES

During looping a set of statements are executed until some conditions for termination of the loop is encountered. A program loop therefore consists of two segments one known as **body of the loop** and other is **the control statement** . The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

Depending on the position of control statement in the loop, a control structure classified in two types: **entry-control loop** and **exit control loop** .

In entry-control loop, the control conditions are tested before the start of loop execution. If conditions are not satisfied then the body of the loop will not be executed.

In exit-control loop , the control conditions are tested at the end of loop body and therefore the body is **executed unconditionally for the first time** .

Flow of Entry Control Loop Flow of Exit Control Loop
Entry Entry
False Condition Body of the loop
True
Body of the loop Condition True
False
Statement-x Statement-x Next Statements Next Statements

In looping process in general would include the following three steps
1. Setting and initialization of a conditional variable.
2. Test for a specified conditions for the execution of the loop. If it is true then Execution of the statements in the loop else exit from the loop.
3. Change the value of conditional variable.

The test may be either to determine whether the loop has repeated the specified number of times or to determine whether the particular condition has been met.

The C language provides for three loop constructs for performing loop operations. They are:

1. The While statement
2. The Do statement
3. The For statement

1. The While Statement:
The simplest of all looping structure in C is the while statement. It is entry control loop. The general format of the while statement is:

Syntax : *initialization* ;
while (*relational condition*) {

….
….
Statements;
….
….
expression that change conditional variable value ;

}

Here the given test condition is evaluated and if the condition is true then the body of the loop is executed. After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statements immediately after the body of the loop. The body of the loop may have one or more statements. The braces are needed only if the body contained two are more statements

Example :

#include < stdio.h > void main ( )
{

int i;
clrscr( );
i=1;
while (i <= 10) {

printf("%d\n",i); **i=i + 1** ;
}

2. The Do while statement:

The do while loop is also a kind of loop, which is similar to the while loop in contrast to while loop, the do while loop tests at the bottom of the loop after executing the body of the loop. Since the body of the loop is executed first and then the loop condition is checked we can be assured that the body of the loop is executed at least once. It is exit control loop

Syntax : *initialization* ; **do**
{

….
….
Statements;
….

….
expression that change conditional variable value

} **while** (*relational condition*);

Here the statement is executed, then expression is evaluated. If the condition expression is true then the body is executed again and this process continues till the conditional expression becomes false. When the expression becomes false the loop terminates.

Example : #include < stdio.h > void main ( )
{

int i;
clrscr( ); **i=1;**
do
{

printf("%d\n",i); **i=i + 1** ;
} **while (i <= 10); 3. The For loop statement:**
The for loop provides a more concise loop control structure. It is entry control loop. The general form of the for loop is:

Syntax :
for (initialize; condition ; expression) {

….
statements; ….

}

When the control enters for loop the variables used in for loop is initialized with the starting value such as I=1. The value which was initialized is then checked with the given test condition. The test condition is a relational expression, such as I <=10 that checks whether the given condition is satisfied or not if the given condition is satisfied the control enters the body of the loop or else it will exit the loop. The body of the loop is entered only if the test condition is satisfied and after the completion of the execution of the loop the control is transferred back to the expression part of the loop.

The conditional variable is incremented using an assignment statement such as I++ and the new value of the control variable is again tested to check whether it satisfies the loop condition. If the value of the control variable satisfies then the body of the loop is again executed. The process goes on till the control variable fails to satisfy the condition.

Example :

```
#include < stdio.h > void main ( )
{

int i;
clrscr( );
for ( i=1;i<=10;i++) {

printf("%d\n",i); }
```

• BREAK, CONTINUE STATEMENTS IN C

1. break Statement

We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test. The keyword **break** allows us to do this. When the keyword **break** is encountered inside any C loop, control automatically passes to the first statement after the loop.

Example:

```
void main( )
{

clrscr( );
for(i=1 ; i <= 100 ; i++) {
printf( "%d",i );
if( i == 50 ) break; }
```

}

2. continue Statement

In some programming situation we want to take the control to the beginning of the loop, bypassing the statements inside the loop which have not yet been executed. The keywords **continue** allows us to do this. When the keyword **continue** is encountered inside any C loop, control automatically passes to the beginning of the loop.

Example:

```
void main( )
{

clrscr( );
for(i=1 ; i <= 100 ; i++) {
if( i % 5 == 0 ) continue; printf( “%d”,i ); }
```

• ARRAY

Definition :
An array is a collection of elements having same name, same data type and different index value.

Declaring Arrays
We can declare an array by specify its data type, name and the number of elements the array holds between square brackets immediately following the array name.
Here is the syntax:

data_type array_name[size];

For example, to declare an integer array which contains 100 elements we can do as follows:

```
int a[100];
```

There are some rules on array declaration.
• The data type can be any valid C data types.

• The array name has to follow the rule of variable name. • The size of array has to be a positive constant integer.

Initializing Arrays
To initialize an array, you provide initializing values which are enclosed within curly braces in the declaration and placed following an equals sign after the array name.
Here is an example of initializing an integer array.

int a[5] = { 10,20,30,40,50 };
This statement stores 10 to a[0], 20 to a[1], 30 to a[2], 40 to a[3] and 50 to a[4].

Accessing Arrays Elements
We can access array elements via indexes array_name[index]. Indexes of array starts from 0 not 1 so the highest elements of an array is array_name[size-1].
Here is an example of accessing an array

printf(“First Element of A array:=%d”,a[0]);
printf(“Fourth Element of A array:=%d”,a[3]);
Program for Store and Print Array Elements:
#include<stdio.h>
void main( ) {
int a[10],i;

// Store values in an array for(i=0;i<10;i++)
{

printf(“Enter element %d = “,i+1); scanf(“%d”,&a[i]);
}
// Print values in an array
for(i=0;i<10;i++)

{
printf(“Element a[%d] = %d “,i , a[i]);
}

}

Multidimensional Arrays:

An array with more than one index value is called a multidimensional array. The entire array above is called single-dimensional array. To declare a multidimensional array you can do follow syntax

data_type array_name[][][];

The number of square brackets specifies the dimension of the array. For example to declare two dimensions integer array we can do as follows:

int matrix[3][3];

Initializing Multidimensional Arrays

You can initialize an array as a single-dimension array. Here is an example of initialize an two dimensions integer array:

```
int matrix[3][3] = {
{11,12,13}, {21,22,23}, {32,31,33},

};
```

Program for Store and Print Two Dimension Array Elements:

```
#include<stdio.h>

void main( ) {
int a[3][3],i,j; clrscr( );

// Store values in an array

for(i=0;i<3;i++) {
for(j=0;j<3;j++) {
printf("Enter element a[%d][%d] = ",i,j); scanf("%d",&a[i][j]);
}
}
// Print values in an array

for(i=0;i<3;i++) {
for(j=0;j<3;j++) {
```

```
printf("%3d ",a[i][j] ); }
printf("\n");
}

}
```

CHARACTER ARRAY AND STRING IN C

C does not have a string type as other modern programming languages. C only has character type so a C string is defined as an array of characters. String is terminated by a special character which is called as null terminator or null parameter (\0). So when you define a string you should be sure to have sufficient space for the null terminator.

single dimensional array declaration
char a[15] Declares an array of 15 character elements. char b[25] Declares an array of 25 character elements.

single dimensional array initialization
char a[15] ="caet";
This statement stores 'c' to a[0], 'a' to a[1] , 'e' to a[2] , 't' to a[3] and ' \0 ' to a[4].
Character array must be terminated by ' \0 ' (Null character).

char b[15] ={ 'c','a', 'e', 't','\0' };
This statement stores 'c' to b[0], 'a' to b[1] , 'e' to b[2] , 't' to b[3] and ' \0 ' to b[4].

Two dimensional array declaration
char a[5][15] Declares an array of 75 character elements in 5 rows and

15 columns.
char b[5][25] Declares an array of 125 character elements in 5 rows and 25 columns.
char c[10][20] Declares an array of 200 character elements in 10 rows and 20 columns.

Two dimensional array initialization char a[5][15] = {
{ 'b','c','a','\0' },
{ 'b','b','a','\0' },

{ ‘b’,’c’,’o’,’m’,’ \0‘ },
{ ‘m’,’c’,’a’,’\0’ },
{ ‘m’,’b’,’a’,’\0’ } };

Program for read and display string Single Dimensional Array:

#include<stdio.h> #include<conio.h> void main( )
{

char nm[50];
clrscr( );
printf(“Enter Your name “);
gets(nm); // stores character values in an array

nm
printf(“Your name is ”);
puts(nm); // display inputed values on the screen

}
Program for read and display string Multi Dimensional Array:
#include<stdio.h>

void main( ) {
char nm[5][25]; int i,j;
clrscr( );

// input part for(i=0;i<5;i++) {

printf(“Enter Name %d = “,i+1); gets(nm[i]);
}

// output part for(i=0;i<5;i++) {

puts(nm[i]); }

• LIBRARY FUNCTIONS OF <STRING.H>

strlen( )
Description: counts no of characters present in the string excluding null character.

Example :
#include<stdio.h>
#include<string.h>
void main( )
{

char nm[25]= “computer”;
int result;
result=strlen(nm);
printf(“No of characters : %d “, result);

}
strcpy( )
Description: copy one string to another string.

Example :
#include<stdio.h>
#include<string.h>
void main( )
{

char nm[25]= “computer”;
char temp[25];
strcpy(temp,nm);
printf(“\nOriginal String = %s“, nm); printf(“\nCopy String = %s“, temp);

}
strcat( )
Description: Appends copy of one string to another string.

Syntax : **strcat(string1,string2)**
String2 is append to an end of String1.
Example :
#include<stdio.h>
#include<string.h>
void main( )
{
char s1[25]= **“Turbo”;**
char s2[10]=**”C++”;**
strcat(s1,s2);
printf(“\n String 1 = %s”,s1); // TurboC++ printf(“\n String 2 = %s”,s2); // C++

strcmp( )
Description: The string comparison starts with first character in each

string and continues with subsequent characters until the corresponding character differs or until the end of the strings is reached.
Return value :

Value 0
> 0
< 0

Description
Both strings are equal.
First string is large than second string. First string is small than second string.

Example :
#include<string.h> void main( )
{

int result; char nm[25]= **“computer”;**
char temp[25]=**”Computer”;**
result=strcmp(nm,temp);
if (result == 0)

printf(“Both strings are equal”); else if(result < 0)
printf(“String 1 is small than String 2”); else if (result > 0)
printf(“String 1 is large than String 2”); }

strlwr( )
Description : strlwr converts all uppercase (A to Z) in to lowercase (a to z). All other characters remain unchanged.
Syntax : **strlwr(string1)**
String1 is converted to lowercase.

Example :
#include<stdio.h>
#include<string.h>
void main( )
{

char s1[25]= **“James Bond 007”;**

printf(“\nOriginal String = %s”,s1); // James Bond 007
strlwr(s1);
printf(“\nString after strlwr = %s”,s1);// james bond 007

strupr( )
Description : strupr converts all lowercase (a to z) in to uppercase (A to Z). All other characters remain unchanged.
Syntax : strupr(string1)
String1 is converted to uppercase.
Example :
#include<stdio.h> #include<string.h>
void main( ) {
char s1[25]= **“James Bond 007”;**

printf(“\nOriginal String = %s”,s1); // James Bond 007 **strupr(s1);**
printf(“\nString after strupr = %s”,s1); // JAMES BOND 007

}
strrev( )
Description : strrev changes all characters in a string to reverse order, except

the terminating null character.
Syntax : **strrev(string1)**
String1 is reversed.

Example :

```
#include<stdio.h>
#include<string.h>
void main( )
{

char s1[25]= “COMPUTER”;

printf(“\nOriginal String = %s”,s1); // COMPUTER strrev(s1);
printf(“\nString after strrev = %s”,s1); // RETUPMOC
```

LIBRARY FUNCTIONS OF <CTYPE.H>
isalpha( )
Description : This function checks whether the given character is

alphabet(A to Z , a to z). Function returns non zero value on success and zero on failure.
Syntax : **isalpha(char c)**

Example :

```
#include<ctype.h>
void main( )
{

char ch;
int result;
printf(“Enter Any character : “);
scanf(“%c”,&ch);
result=isalpha(ch);
if ( result != 0 )

printf(“Your character is an alphabet”); else
printf(“Your character is not an alphabet”); }
```

isdigit( )
Description : This function checks whether the given character is

digit(0 to 9). Function returns non zero value on success and zero on failure.
Syntax : **isdigit(char c)**

Example :
#include<ctype.h>
void main( )
{

char ch;
int result;
clrscr( );
printf("Enter Any character : "); scanf("%c",&ch);

result=isdigit(ch);

if (result != 0)
printf("Your character is a digit");
else
printf("Your character is not a digit");

isalnum( )
Description : This function checks whether the given character is

alphabet (A to Z or a to z) or digit(0 to 9). Function returns non zero value on success and zero on failure.
Syntax : **isalnum(char c)**

Example :
#include<stdio.h>
#include<ctype.h>
void main( )
{

char ch;
int result;
clrscr( );

```
printf(“Enter Any character : “);
scanf(“%c”,&ch);
result=isalnum(ch);
if ( result != 0 )

printf(“Your character is an alpha numeric”); else
printf(“Your character is not an alpha numeric ”); }
```

isspace( )

Description : This function checks whether the given character is

space , new line, carriage return or tab. Function returns non zero value on success and zero on failure.

Syntax : **isspace(char c)**

Example :

```
#include<stdio.h>
#include<ctype.h>
void main( )
{

char ch;
int result;
clrscr( );
printf(“Enter Any character : “);
scanf(“%c”,&ch);
result=isspace(ch);
if ( result != 0 )

printf(“Your character is a space”); else
printf(“Your character is not a space”);
```

isupper( )

Description : This function checks whether the given character is an

uppercase letter (A to Z). Function returns non zero value on success and zero on failure.

Syntax : **isupper(char c)**

Example :

```
#include<stdio.h>
#include<ctype.h>
void main( )
{

char ch;
int result;
clrscr( );
printf("Enter Any character : ");
scanf("%c",&ch);
result=isupper(ch);
if ( result != 0 )

printf("Your character is an uppercase letter"); else
printf("Your character is not an uppercase letter"); }
```

islower( )

Description : This function checks whether the given character is

lowercase letter (a to z). Function returns non zero value on success and zero on failure.

Syntax : **islower(char c)**

Example :

```
#include<stdio.h>
#include<ctype.h>
void main( )
{

char ch;
int result;
clrscr( );
printf("Enter Any character : ");
scanf("%c",&ch);
result=islower(ch);
if ( result != 0 )
```

printf(“Your character is a lowercase letter”); else
printf(“Your character is not a lowercase letter”);

toupper( )
Description : This function converts character to uppercase if it is in

lowercase. All other values are remain unchanged.
Syntax : **toupper(char c)**

Example :
#include<stdio.h>
#include<ctype.h>
void main( )
{

char ch;
char result;
clrscr( );
printf(“Enter Any character : “);
scanf(“%c”,&ch);
printf(“Your character is : %c”,ch); **result=toupper(ch);**
printf(“After toupper function : %c”,result);

}
tolower( )
Description : This function converts character to lowercase if it is in

uppercase. All other values are remain unchanged.
Syntax : **tolower(char c)**

Example :
#include<stdio.h>
#include<ctype.h>
void main( )
{

char ch;
char result;
clrscr( );

```
printf(“Enter Any character : “);
scanf(“%c”,&ch);
printf(“Your character is : %c”,ch); result=tolower(ch);
printf(“After tolower function : %c”,result);
```

LIBRARY FUNCTIONS OF <STDIO.H>

printf ( )
Description : Sends formatted text output to standard output device(generally monitor).
Example :

```
#include<stdio.h>
void main( )
{

printf(“Hello World”);

}
```

scanf ( )
Description : Accepts formatted data from standard input device and stores it in memory location.
Example :

```
#include<stdio.h>
void main( )
{

int no;
printf(“Enter any number : ”);
scanf (“%d”,&no);

}
```

gets( )
Description : Accepts string from standard input device and stores it in memory locations.
Example :

```
#include<stdio.h>
void main( )
{

char nm[25];
printf(“\nEnter Your Name : ”);
gets(nm);
printf(“My name is : “);
puts(nm);

}
```

puts( )

Description : Prints string to standard output device (Screen). Example :

```
#include<stdio.h>
void main( )
{

char nm[25];
printf(“\nEnter Your Name : ”); gets(nm);
printf(“My name is : “);
puts(nm);
```

LIBRARY FUNCTIONS OF <CONIO.H>

getch( )

Description : getch reads a single character directly from the keyboard without echoing to the screen.

Example :

```
#include<conio.h>
void main( )
{

char ch;
printf(“\nEnter Any Character : ”);
ch=getch();
printf(“\n Your character is %c “, ch);
```

}

getche( )

Description : getch reads a single character directly from the keyboard and echoes it to the screen.

Example :

```
#include<stdio.h>
#include<conio.h>
void main( )
{

char ch;
printf("\nEnter Any Character : ");
ch=getche();
printf("\n Your character is %c ", ch);

}
```

getchar( )

Description : getchar reads a single character from standard input. The getchar is line buffered that means characters are available after enter key.

Example :

```
#include<stdio.h>
#include<conio.h>
void main( )
{

int ch;
printf("\nEnter Any Character String : "); while((ch=getchar( ))!='\n')
{

printf("%c",ch);
}
}
```

clrscr( )

Description : clrscr clears current text window and places the cursor in the upper left corner.

UDF (USER DEFINED FUNCTION)
A UDF stand for User-Defined Function. It is a function provided by the user of a program. UDF write for solving a specific task.

Benefit of using function
• UDF provide modularity to the software.
• UDF provide code reusability because it can be executed as many

times as necessary from different points in your program so it helps you avoid duplication of work.

• By using UDF, you can divide complex tasks into smaller manageable tasks and test them independently before using them together.

Structure of UDF
return type <function name> (<parameter list>) {
Statements; }

Function header consists of three parts:
• **Data type of return value of function** ; it can be any legal data type such as int, char, pointer… if the function does not return a value then return type is **void.**

• **Function name;** function name is a sequence of letters and digits, the first of which is a letter, and where an underscore counts as a letter. The name of a function should meaningful. for example result(),

• **And a parameter list;** parameter passed to function have to be separated by a comma, and each parameter has to indicate it data type and parameter name.

Function body
Function body is the place you write your own source code.
Return statement
Return statement returns a value to where it was called. A copy of return value being return is made automatically. If the void keyword used, the function don’t need a return statement.

Types of UDF:
Basically three types of UDF
1) No argument no return value 2) With argument no return value 3) With argument with return value

No argument no return value:
• This user defined function return data type is **void. • no parameter list** pass in function.

With argument no return value:
• This user defined function return data type is **void • parameter list** pass in function.

With argument with return value:
• This user defined function return data type is can be any legal data type such as int, char etc… and **parameter list** pass in function.

• Function body has a **return statement** .
• At a time of function call return value store in any variable or directly display using printf statement.

For example:
#include<stdio.h> #include<conio.h>

void sum1( );
void sum2(int, int); int sum3(int, int); **void main()**
{

int no1,no2,ans; clrscr();
sum1();
sum2(10,30);
ans=sum3(100,200);

Function Declaration
Function Calling

printf("\n With Arg. with Return Sum:=%d",ans); }
void sum1( )
{

```
int x=1, y=2,ans;
ans=x+y;
printf (“\n No Arg. No Return Sum :=%d”,ans);

}
void sum2(int x, int y)
{

int ans; Function Definition ans=x+y;
printf (“\n With Arg. No Return Sum :=%d”,ans);

}
int sum3(int, int)
{

return (x+y); }
```

OUTPUT:

No Arg. No Return Sum :=3 With Arg. No Return Sum :=40 With Arg. with Return Sum :=300