AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 001

Introduction of C Language

What is a Language?

- Language is a collection of word and symbols.
- Language is generally used to build up a communication between persons, machines or computers.
- Generally human beings used different languages in different region for effective communication.
- Languages is used to give some instructions / information / commands / details to other person.
- Basically, computer is a machine and machine cannot do any thing without instruction/command.
- If we want to work with computer and with to do some work as per our requirement then we must give some instructions / command
- Generally computer understand some special languages only. These languages are known as computer languages.

There are mainly three different languages...

1. Low Level Languages
2. High Level Languages
3. Middle Level Languages

1. Low level programming language :

- Low level programming language is also known as Binary Language too.
- Binary Language consists only two characters **0 [Zero]** and **1 [One]**.
- Low level programming language does not require translator like compiler or interpreters.
- Machine can direct understand Low level programming with fast execution speed.

2. High level programming language :

- High level languages are user friendly language because their instructions are similar to human languages.
- High level languages have a set of grammar that makes it easy for a programmer to write programs and identify the correct errors in a program.
- All high level languages have its own compiler or interpreter to convert its code into machine code (Binary Codes).

For Examples,

Cobol, Fortran, Basic, Java etc.

4. Middle level programming language :

- Middle Level Programming is a language which has mixed power of Low Level Programming Language and High Level Programming Language.
- Middle Level Programming is Easy for Programming compare to Low Level Programming.
- Middle Level Programming is fast for execution compare to High Level Programming.

❖ What is POP?

POP is a procedure oriented programming language.

C, Basic and Fortran are example of POP.

What is translation?

Translation means to convert source-language by target-language.

Examples :

1. Gujarati **To** Hindi
2. Gujarati **To** English
3. English **To** Gujarati
4. Text **To** Binary
5. Binary **To** Text

❖ What is translator?

Translator is a special program. This translator accepts the user program and checks each statement and produces a set of Machine language instructions.

There are two types of Translators.

- **Complier**
- **Interpreters**

What is Compiler?

- A Compiler checks all the entire program written by user
- If program is free from error and mistakes then compiler produces this program as a complete program.
- If program founds some errors in the program then it does not execute a single statement of the program.
- So Complier translate whole program in Machine language before it starts execution.

What is Interpreters?

- Interpreters perform the similar job like complier but in different way.
- Interpreters translates one statement at a time and if it is error-free then executes that statement.
- This continues till the last statement in the program has been translated and executed.

Difference between Compiler & Interpreter?

Compiler	Interpreter
Compiler first checks all the statement for error and provide lists of all the errors in the program.	Error finding is much easier in Interpreter because it checks and executes each statement at a time.
Compiler take less time than interpreter because it translate and executes all statement at same time.	Interpreter take more time for the execution of a program compared to Compilers because it translates and executes each statement one by one.

Introduction to C : History Of C:

It was developed by step as given below.

Language	Year	Developed by
ALGOL (ALGOrithmic Language)	1960	International Committee
CPL (Combined Programming Language)	1963	Cambridge University
BPCL (Basic Combined Programming Language)	1967	Martin Richards & Cambridge University
B (B Language)	1970	Ken Thomson at AT&T Laboratory
Traditional C (C Language)	1972	Denis Ritchie at AT&T Laboratories
ANSI C (American National Standards Institute)	1989	ANSI Committee
C99	1999	Standardization Committee

What is C?

- ✓ C is programming language.
- ✓ It is also known as middle level programming language.
- ✓ Program execution speed is faster than High Level Programming Languages.
- ✓ C behaves as High Level Language through functions and reusability
- ✓ C programming is faster than other programming languages.
- ✓ C is a powerful and flexible language.
- ✓ C program has its own built in library functions
- ✓ Write C program with UDF (User Defined Function) makes program more simple and easy to understand.
- ✓ The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.
- ✓ C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

Shortcut Used in C editor :

What to DO ?	Shortcut
Save a program file	F2
Open a saved file	F3
To maximize editing window	F5
Undo last changes in Editor	Alt+BackSpace
Redo last changes by Undo	Shift+Alt+B.Space
Copy selected text	Ctrl + Insert Key
Cut selected text	Shift + Del
Paste copied/cut text	Shift + Insert Key
Clear Selected Text	Ctrl + Del
Compile a C Program	Alt + F9
Run a C Program	Ctrl + F9
Step by Step Debugging	F7

Structure of C Program :

Documentation Section

File include section (Link Section)

Define Symbolic Constant

Global declaration Section

Function Prototype

main( ) (Main Function Section)

{ **declaration part**

executable part

}

User defined Function or Sub program

First C Program :

// write a program to print Helllo Creative

```
#include <stdio.h>
int main()
{
        printf("Hello Creative");
        return 0;
}
```

#include <stdio.h> includes the standard input output library functions. The **printf()** function is defined in stdio.h .

int main() The main() function is the entry point of every program in c language.

return 0 The return 0 statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful execution.

Assignment Work:

1)Write a Program to Print your Bio-data using Printf Function

2) WAP to print Student Result Format using Printf function

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 002

Data Type & Variable

C Tokens :

- Identifiers
- Keywords
- Constants
- Operators
- Strings and special symbols, these are called as C tokens.

Identifiers :

Identifier is a name of variable, constant, structure, union, arrays or function given by the programmer.

Keywords :

- Keyword is a word which is having a specific meaning for a particular language.
- Keywords are reserved word by programming language.
- Keywords are not used as identifiers.
- There are 32 keywords available in C.

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

Variables :

- A variable is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.
- It is a way to represent memory location through symbol so that it can be easily identified.

- Let's see the syntax to declare a variable:

```
int a;
float b;
char c;
```

Rules for variable naming :

- The first character in identifier name must be an alphabet or underscore
- commas or blanks space are not allowed within an identifier name.
- special symbol other then an underscore can be not used as in identifier name.
- An identifier name is any combination of 1 to 31 alphabets, digits or underscores without space.
- Do not create unnecessarily long variable names as it adds to you typing efforts.
- C language is case sensitive. Upper and Lower case both are allowed in C programming. But they are not equivalent. Here **NUM** and **num** both are different.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Types of Variables in C :

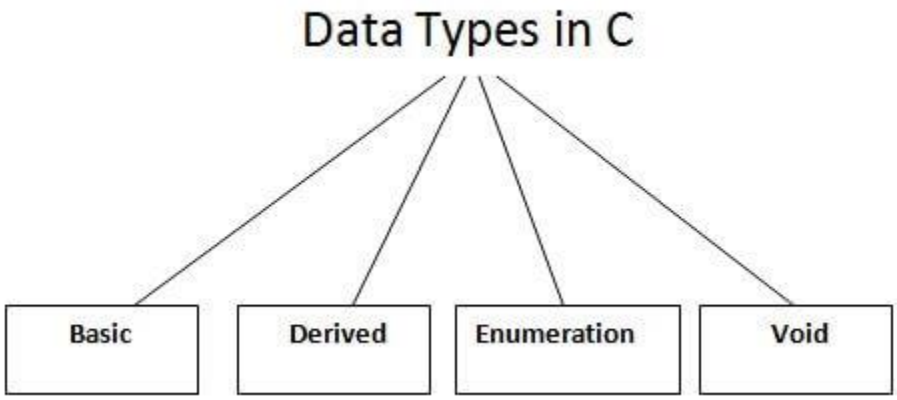
There are many types of variables in c:

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

All types of variable we will learn in ch - 21

Data Types in C :

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



There are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types :

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

Data Types	Memory Size	Format code	Range
char	1 byte	%c	−128 to 127
int	2 byte	%d	−32,768 to 32,767
unsigned int	2 byte	%u	0 to 65,535
long int	4 byte	%ld	-2,147,483,648 to 2,147,483,647
float	4 byte	%f	-2,147,483,648 to 2,147,483,647
double	8 byte	%lf	

Integer :

- ✓ This data type is used to store numbers without decimals.
- ✓ Range between -32768 to +32767
- ✓ Occupies 2 byte (16 bits)
- ✓ Format code is %d

Example,

```
int a,b,c;
a=10;
b=20;
c=a+b;
```

printf() and scanf() in C :

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function :

The printf() function is used for output. It prints the given statement to the console.

The syntax of printf() function is given below:

printf("format string",argument_list);
The format string can be %d (integer), %c (character), %s (string), %f (float) etc.

scanf() function :

The scanf() function is used for input. It reads the input data from the console.

scanf("format string",argument_list);
Program to print cube of given number
Let's see a simple example of c language that gets input from the user and prints the cube of the given number.

```
#include<stdio.h>
int main(){
int number;
printf("enter a number:");
scanf("%d",&number);
printf("cube of number is:%d ",number*number*number);
return 0;
}
```

Output :

```
enter a number:5
cube of number is:125
```

The scanf("%d",&number) statement reads integer number from the console and stores the given value in number variable.

The printf("cube of number is:%d ",number*number*number) statement prints the cube of number on the console.

Program to print sum of 2 numbers :
Let's see a simple example of input and output in C language that prints addition of 2 numbers.

```
#include<stdio.h>
int main(){
int x=0,y=0,result=0;

printf("enter first number:");
scanf("%d",&x);
printf("enter second number:");
scanf("%d",&y);

result=x+y;
printf("sum of 2 numbers:%d ",result);

return 0;
}
```

Output :

enter first number:9
enter second number:9
sum of 2 numbers:18

Assignment Work :

1) WAP to declare two variable and calculate (sum/sub/mul/div)using this variable

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 3

Unsigned |Long |Float |Char Data Type

Signed int:

- By default all the data types as signed. So it is not necessary to define any variable as signed.
  - Example :

```
signed int a=20;

int b=30;

int c=a+b;

clrscr();

printf("%d",c);
```

Unsigned int

- If we wants to store only positive values in variable, then it is very easy by using **unsigned**.
- If we define any variable as unsigned the positive range will be increased of that variable by negative value.

- Example :
  - **integer** value will store **-32768 to +32767** in signed
  - But if we will convert it in to **unsigned** it will store directly **0 to +65535 positive value.**
  - **Format code %u**

    Example :

```
Unsigned int a=25000,b=25000,c;
C=a+b;
Printf("C=%d",c);
Output :
C=50000
```

Long Int :

- A long int typically uses twice as many bits as a regular int, allowing it to hold much larger numbers.
- printf and scanf replace with %ld to indicate the use of a long int.
- long int allocate memory 4 bytes in c prog

Example :

```
long int a=200000,b=300000,c;
c=a+b;
printf("C=%ld",c);
```

output :

```
c=500000
```

Float :

- ✓ This data type is used to store numbers with decimals.
- ✓ Value without decimal can also be stored in it.
- ✓ Range between 3.4e - 38 to 3.4e +38
- ✓ Occupies 4 byte (32 bits)
- ✓ Format code is %f

Example,

```
float a,b,c;
a=10.50;
b=20.3;
c=a+b;
output:
c=30.8
```

Character :

This data type is used to store alphabets, number and special symbols in it.
Range between -128 to +127 (Range in ASCII)
Occupies 1 byte (8 bits)
Format code is %c
Example,

```
Char a='z';
```

Double :

- This data type is used to store numbers with decimals when float is insufficient.
- Range between 1.7e - 308 to 1.7e +308
- Occupies 8 byte (32 bits)
- Format code is %lf
  Example,

```
double a,b,c;
a=9990990.90;
b=20.50;
c=a+b;
```

Void :

- ❑ Actually there is no variable defined of this type but it is used do define function.
- ❑ Void is data type which specifies that the function does not return any value to the calling program.

■ Example,

```
void main( )
void printnum( )
```

Type Casting :

- ■ Type casting is a process which converts a value from one data type into another data type.
- ■ During the programming **we can convert integer value to float value** and **float value to integer value** as per requirement.
- ■ Example :

```
int a=20;
printf("%f",(float)a/3);   // It will convert data from integer to float.
```

Assignment Work :

1. WAP to Calculate Area and Circumference of circle (Area of Circle = **PI * $R^{2)}$**
   Circumference of **Circle = 2 * PI * R**)
2. WAP to Calculate Area of Square **(area = side * side)**
3. WAP to Calculate Area of Rectangle **(area = l * b)**

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 4

Introduction of Operator

(Arithmetic | Assignment)

What is Operator?

- Operators are the foundation of any programming language. Thus the functionality of C/C++ programming language is incomplete without the use of operators.
- We can define operators as symbols that help us to perform specific mathematical and logical computations on operands.
- In other words, we can say that an operator operates the operands.
- Like, +, -, *, /, &&, ||, !=, <, >, etc.

What is Operand?

Operand means the variable or value which is used before and after the operator.

Like, A+B (Here, A and B are operand and + is operator)

Operator :

(1) Arithmetic Operator
(2) Assignment Operator
(3) Relational Operator
(4) Logical Operator
(5) Increment / Decrement Operator
(6) Conditional Operator /Ternary Operator
(7) Bitwise Operator
(8) Special Operator

1. Arithmetic Operator :

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Reminder after integer division

Example :

1. WAP to declare two variable and calculate sum , sub, mul, div.
2. WAP to declare two variable and enter value of the variable and calculate Sum, sub,mul,div .

```c
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

2. Assignment Operators:

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value. The value on the right side must be of the same data-type of variable on the left side otherwise the compiler will raise an error.

Different types of assignment operators are shown below:

"=": This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.

For example:

```
a = 10;
```

```
b = 20;
ch = 'y';
```

"+=": This operator is combination of '+' and '=' operators. This operator first adds the current value of the variable on left to the value on right and then assigns the result to the variable on the left.

Example:

(a += b) can be written as (a = a + b)

Assignment Work :

1. WAP to find the simple interest **(SI = (P*R*N)/100)**
2. WAP Program to Convert temperature from degree centigrade to Fahrenheit(Fahrenheit = (**Celsius * 9 / 5) + 32**)
3. WAP to swap 2 value without using third variable
4. WAP to swap 2 value with third variable

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 5

Relational Operator

Logic Development

Control Statement

3 Relational Operator :

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.
- Relational Operators are return Boolean value like true and false

Operator	Meaning
<	Less than
<=	Less than or Equal to
>	Greater than
>=	Greater than or Equal to
= =	Equal to
!=	Not Equal to

Example :

```
Int a=5,b=2,c;
C=a>b;
Printf("\nC=%d",c);
```

Introduction of Logic :

- Logic is must to perform any task.
- Using various types of logic we can perform many programs.
- There are three types of Logic structures are possible.
  - **Sequence Logic**
  - **Decision Logic**
  - **Looping Logic**

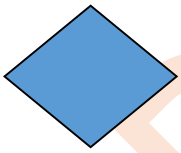
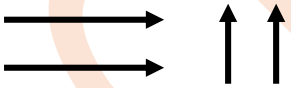
- **Sequence Logic.**
  - In this logic all the instructions are written in order to performed in a sequence.
- **Decision Logic**
  - When more then one option is available then we can use decision logic with.
    - **If… Then**
    - **If… Then…else**
    - **If… elseif… else… endif**
- **Looping Logic :**
  - When one or more instructions may be executed more then one time then we can use looping logic.

Instructions for Logic Development :

- To develop a program easily, we must have sufficient knowledge of Pre Programming Techniques.
- Pre Programming Techniques will help you to design a proper programming channel.
- If we are going to write a computer program using any programming language, we need to write a rough program first.
- There are two basic tools do develop basic or rough program.
  - Algorithm
  - Flowchart
- If we want to solve any error from any program the we must know the proper steps involved in solving problem.
- If the sequence of programming command is not proper then you will get wrong output.
- To solve these problems we can use Algorithm and Flowchart.

- What is Algorithm?
  - Write program solution step by step in textual form is known as algorithm.
- What is Flowchart?
  - A flowchart is a graphical representation of an algorithm.

Flowchart Symbols :

	**Terminal Symbol (Start / Stop)** This oval shape represents for Start and Stop.
	**Process Steps Symbol** The Rectangle represents the processing operation.
	**Input or Output Process Symbol** This shape represents input and output task.
	**Decision Making and** **Branching Symbol** **The diamond shape represent decision or branching statements.**
	**Flow Direction Symbol** **The arrows shows flow direction in which one has to proceed to solve the problem.**

Algorithm :

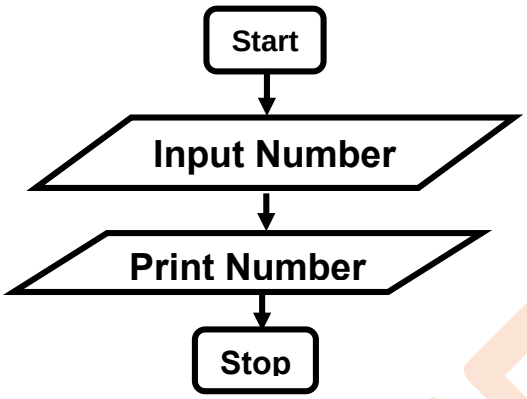
Write an algorithm and draw a flowchart to read and print value.

Step-1 Input the number

Step-2 Print the number

Step-3 Stop

Flowchart : **Draw a flowchart to read and print value.**



Algorithm :

2. Write an algorithm to make sum of three value.

Step-1 Input value1, value2, value3

Step-2 Calculate sum of three values Sum=value1+value2+value3

Step-3 Print sum of value

Step-4 Stop

Sequence Logic :

1. Algorithm Example :

Write an algorithm to make sum of three value.

Step-1 Input value1, value2, value3

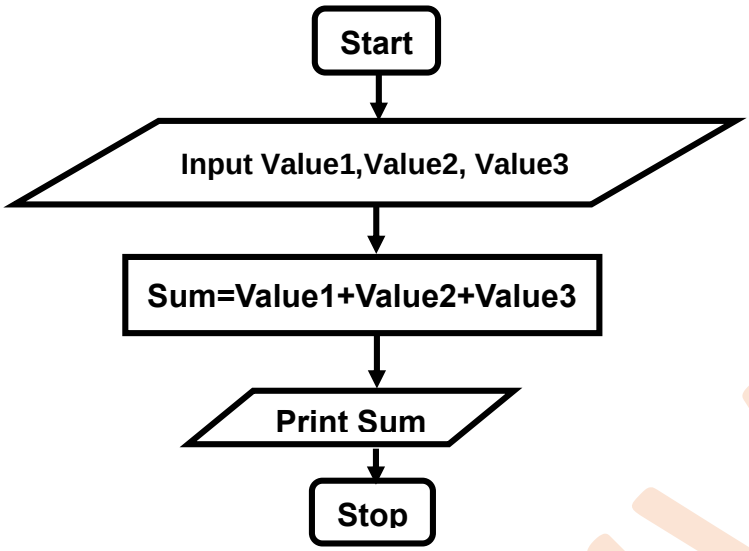
Step-2 Calculate sum of three values // Sum=value1+value2+value3

Step-3 Print sum of value

Step-4 Stop

Flowchart example:

Draw a flowchart to make sum of three value.



Control Statement :

- There are two type of Control structure
  - Conditional
  - Looping

✓ In conditional statement there are optional environment available.
✓ Example : to watch movie or to play cricket
✓ If Statement
✓ Switch Statement
✓ If Statement in Details

❑ We can write conditional statement IF in different ways...

- if (Wimple if Without Else Statement)
- if....else
- Nested if (if within if)
- if....else if ladder

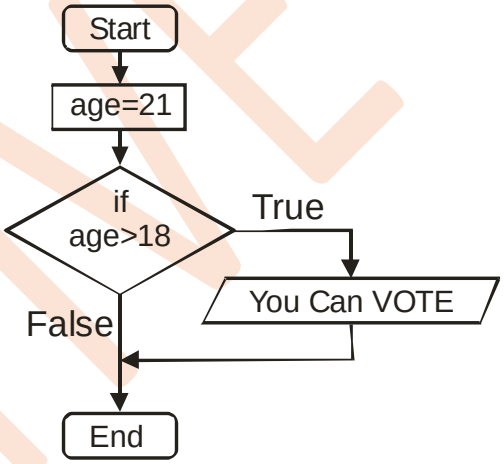
if (Simple if without else) :

Syntax : if(condition)
{
statements;
}

- Purpose:
  - If statement is the decision control statement. This statement will check the given condition. If condition is TURE then it execute given statements.

Example: To check that you can VOTE or Not

```
void main( )
{
    int age;
    printf("Enter your age :");
    scanf("%d",&age);
    if(age>18)
        printf("You can VOTE");
}
```



if....else :

Syntax :

```
if(condition)
{   statement Group1;
}
Else
{   statement Group2 ;
}
```

- Purpose:
  - If..else statement is a decision control statement.
  - If condition is TURE then it execute given statements group 1. If condition if FALSE then it executes statement group 2.

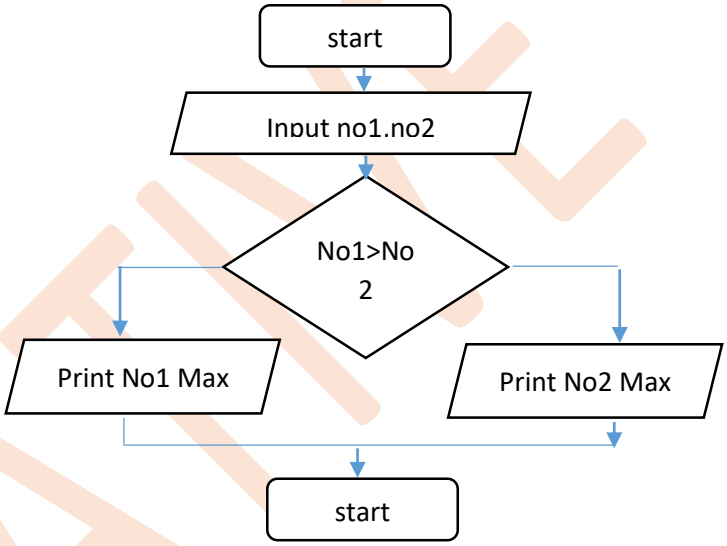
Example: To check that you can VOTE or Not

```
void main( )
{               int age;
                printf(“Enter your age :”);
                scanf(“%d”,&age);
                if(age>18)
                        printf(“You can VOTE”);
                else
                        printf("You Can Not VOTE");
        }
```

Draw a flowchart to find the maximum number from given two number.

Example :

```
void main( )
{               int no1,no2;
                Printf(“Enter No1);
                Scanf(“%d”,&no1);
                Printf(“Enter No2);
                Scanf(“%d”,&no1);
                if(no1>no1)
                        printf(“No1 max”);
                else
                        printf("No2 max");
        }
```



Assignment Work :

1. Draw a Flow chart to Find the Maximum/ Minimum number of two value
2. Draw a Flow chart to Find the Maximum/ Minimum number among the three number
3. Draw a Flow chart to Find the Maximum/ Minimum number among the four number
4. Draw a Flow chart to Find the Maximum/ Minimum number among the Five number
5. Draw a Flow char to check equal / min / max among the three value
6. WAP to Find the Maximum/ Minimum number of two value
7. WAP to Find the Maximum/ Minimum number among the three number
8. WAP to Find the Maximum/ Minimum number among the four number
9. WAP to Find the Maximum/ Minimum number among the Five number
10. WAP to check equal / min / max among the three value
11. WAP to check num and print **CREATIVE** if given num is greater than 10 or print **MULTIMEDIA** if given num is smallest 10 among three variable without logical operator
12. WAP to program to check whether the given number is positive ,negative or zero

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 6

Logical Operator

Ladder else if

4 Logical Operators:

They are used to combine two or more conditions

Logical AND operator: The '&&' operator returns true when both the conditions under consideration are satisfied. Otherwise it returns false. For example, a && b returns true when both a and b are true (i.e. non-zero).

Logical OR operator: The '||' operator returns true even if one (or both) of the conditions under consideration is satisfied. Otherwise it returns false. For example, a || b returns true if one of a or b or both are true (i.e. non-zero). Of course, it returns true when both a and b are true.

Logical NOT operator: The '!' operator returns true the condition in consideration is not satisfied. Otherwise it returns false. For example, !a returns true if a is false, i.e. when a=0.

Example :

```
Int a=6,b=3,c=3;

If(a>b && a>c)
{
      Printf("a is max");
}else if(b>c)
{
      Printf("b is max");
}else
{
      Printf("c is max");
}
```

The if-else-if Ladder :

Syntax :

```
if(condition1)
        statement block1;
else if(condition2)
        statement block2;
else if(condition3)
        statement block3;
else
        statement block n;
```

Purpose:

Here in this form, condition1 is evaluated first, if it is true then the statement block 1 will be executed. If it is false then condition2 will be evaluated, if it is true then the statement block 2 will be executed. This will happen untill one condition will be true. If all the condition are false then the else part will be executed.

Assignment Work :

1. WAP to Find the Maximum/ Minimum number among the Five number using Logical operator
2. WAP to print Full Student Result of FIVE sub marks and calculate Total , per , min, max, Result, Grade
3. WAP to input electricity unit charges and calculate total electricity bill according to the given condition:
   For first 50 units Rs. 0.50/unit /For next 100 units Rs. 0.75/unit
   /For next 100 units Rs. 1.20/unit / For unit above 250 Rs. 1.50/unit /An additional surcharge of 20% is added to the bill

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 7

Increment and Decrement Operator

Turnery Operator | Modulus Operator

5. Increment and Decrement Operator in C :

Increment Operators are used to increased the value of the variable by one and Decrement Operators are used to decrease the value of the variable by one in C programs.

Both increment and decrement operator are used on a single operand or variable, so it is called as a unary operator. Unary operators are having higher priority than the other operators it means unary operators are executed before other operators.

Syntax :

```
++  // increment operator
--  // decrement operator
```

Note: Increment and decrement operators are can not apply on constant.

Example

x= 4++; // gives error, because 4 is constant

Type of Increment Operator :

- pre-increment
- post-increment

pre-increment (++ variable) :

In pre-increment first increment the value of variable and then used inside the expression (initialize into another variable).

Syntax :

++ variable;

Example pre-increment :

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x,i;
        i=10;
        x=++i;
        printf("x: %d",x);
        printf("i: %d",i);
        getch();
}
```

Output :

```
x: 11
i: 11
```

In above program first increase the value of i and then used value of i into expression.

post-increment (variable ++)

In post-increment first value of variable is used in the expression (initialize into another variable) and then increment the value of variable.

Syntax :

variable ++;

Example post-increment

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

void main()
{
int x,i;
i=10;
x=i++;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

Output

```
x: 10
i: 11
```

In above program first used the value of i into expression then increase value of i by 1.

Type of Decrement Operator:

- pre-decrement
- post-decrement

Pre-decrement (-- variable) :

In pre-decrement first decrement the value of variable and then used inside the expression (initialize into another variable).

Syntax :

-- variable;

Example pre-decrement

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

void main()
{
        int x,i;
        i=10;
        x=--i;
        printf("x: %d",x);
        printf("i: %d",i);
        getch();
}
```

Output

```
x: 9
i: 9
```

In above program first decrease the value of i and then value of i used in expression.

post-decrement (variable --):

In Post-decrement first value of variable is used in the expression (initialize into another variable) and then decrement the value of variable.

Syntax :
variable --;

Example post-decrement

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x,i;
        i=10;
        x=i--;
        printf("x: %d",x);
        printf("i: %d",i);
        getch();
}
```

Output

x: 10
i: 9

In above program first used the value of x in expression then decrease value of i by 1.

Example of increment and decrement operator

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x,a,b,c;
        a = 2;
        b = 4;
        c = 5;
        x = a-- + b++ - ++c;
        printf("x: %d",x);
        getch();
}
```

Output
x: 0

6. Conditional or Ternary Operator (?:)

- C programming offers the conditional operator used in C language is (?) question mark and (:) column.
- It is used for checking the condition and gives output according to the condition.
- If condition is true then it will execute the first value or it will execute the second value given in the syntax.

Syntax: **<Condition>? <True Value> :< False Value>**

Purpose:

If condition will be true then it will return the true value else it will return the false value.

Example:

```
a=10;
b=12;
c= (a>b)? A: b        // It will store value of B
        printf("Value of C is %d",c);

        sub1=23, sub2=40;

        Printf ("Highest=%d", (sub1>sub2)? sub1:sub2)
```

Modulus operator in C :

The modulus operator is a symbol used in various programming languages. It is denoted by the percentage symbol (%). It is a modulus operator that is used in the arithmetic operator. It determines the remainder. In some cases, the remainder may be 0, it means the number is completely divisible by the divisor

Example :

```
Int a=13,b=2,c
C=a%b;
Printf("\nC=%d",c);
```

Output :

1

Assignment Work :

1. WAP to Find the Maximum/ Minimum number among the Three number Using Turnery Operator
2. WAP to check Odd / Even num of given Number

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 8

Control Statement (Switch case)

Switch Case :

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

Syntax :

```
switch(expression)
{
        case value1:
                statement block1;
                break;
        case value2:
                statement block2;
                break;
        case valueN:
                statement blockN;
                break;
        default:
                default statement block;
}
```

Purpose:

In switch statement, the expression gives one value which is compared with the values written with case statements. If one case is evaluated true then the statement block related to that case will be executed. Following portion of the program give the example of switch statement.

Break Keyword used for break the current case statement.

Example :

```
Void main() {
        Int n;
        Printf("Enter Day Value=");
        Scanf("%d",&n);
        Switch(n){
                Case 1:  printf("Sunday");
                Case2: printf("Monday");
                Case2: printf("Tuesday");
                Case2: printf("Wednesday");
                Case2: printf("Thursday");
                Case2: printf("Friday");
                Case2: printf("Saturday");
                Default: printf("plz input valid num");
        }
}
```

Assignment work :

1. WAP to input week number and print week day
2. WAP to input month number and print number of days in that month

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

Ch. 9

Bitwise Operator

7. Bitwise Operators in C :

Bitwise operators are used to perform bit manipulations.

Generally these operators used to perform hardware related task by c programming.

Operators	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift Left
>>	Shift Right

Example:

int a=7, b=5;

a & b : 111 & 101 = 101 = 5

a | b: 111 | 101 = 111 = 7

a ^ b: 111 | 101 = 010 = 2

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=23,d,b=10,c;        // here a= 10111   B=1010
    clrscr();
    c=a&b;                    // 10111&1010     = 00010
    printf("\nC=%d",c);       //   00010 = 2
    d=a|b;                    // 10111 | 1010 = 11111
    printf("\nD=%d",d);       // 11111=31
    getch();
}
```

8 C – Special Operators :

Operators	Description
&	This is used to get the address of the variable. Example : &a will give address of a.
*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

Size of operator :

Example :

```
Int a=20;
Printf("\nsize of A=%d",sizeof(a));
```

Output:

Size of A=2

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH – 10

Introduction Of Loop

While Loop

Loop Statements :

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language.

Why use loops in C language?

- ✓ The looping simplifies the complex problems into the easy ones.
- ✓ It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times.
- ✓ For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C :

1) It provides code reusability.
2) Using loops, we do not need to write the same code again and again.

Types of C Loops :

There are three types of loops in C language that is given below:

while
Do while
For

While Loop :

While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance

In condition you have to write a condition from relational operator then while loop execute till that condition will not false.

Syntax :

```
While(condition)
{
 statement 1;
 statement 2;
 increment/ decrement;
}
```

Assignment Work :

1. Write a program in C to display the first 10 natural numbers
2. Write a C program to print all natural numbers in reverse
3. WAP to find sum of first **N** natural numbers
4. WAP to print Range between two number
5. WAP to print Range of ODD num if first num is smallest and Print range of Even num if First num is largest
6. WAP to find factorial of a given number
7. WAP to print reverse given num
8. WAP program to count number of digits in a number

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH – 11

DO While Loop

Do while loop :

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of more parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

Syntax :

```
do{
//code to be executed
}while(condition);
```

Assignment Work :

1. WAP to Display The Multiplication Table of a Given Number
2. WAP to display the cube of the number upto given an number
3. WAP program to sum of digits in a number
4. WAP to find the number and sum of all integer between two user input num which are divisible by given num
5. WAP to enter password and check password with entered password, if password is match then print “WELCOME” otherwise ask to give correct password.

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH – 10

Introduction Of Loop

For Loop

For loop:

The for loop in C language is used to the statements or a part of the program more times. It is frequently used to traverse the data structures like the array and linked list.

Syntax:

```
for(Expression 1; Expression 2; Expression 3){
//code to be executed
}
```

Assignment Work :

1. WAP to print Fibonacci series in a given range
2. WAP to check given num is Armstrong or not
3. WAP to check given num is Palindrome or not
4. WAP to check given num is Prime or not
5. WAP to print Pattern given by your Faculty (15 pattern)

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH – 10

Introduction Of Loop

Continue | Break | Goto

BREAK STATEMENT:

- The break statement is used to terminate loops or to exit a switch.
- The break statement will break or terminate the inner-most loop.
- It can be used within a while, a do-while, a for or a switch statement.

Syntax : break;

CONTINUE STATEMENT :

- The continue statement is used to skip or to bypass some step or iteration of looping structure.
- It does not terminate the loop but just skip or bypass the particular sequence of the loop structure.

Syntax : continue ;

THE GOTO STATEMENT :

- The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program.
- In its general for the goto statement is written as goto label; Where label is an identifier used to label the target statement to which control will be transferred.

Example :

```
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
}
```

Assignment Work :

1. Program to calculate the sum and average of positive numbers If the user enters a negative number, the sum and average are displayed.
2. WAP to calculate sum of number if user enter negative num loop terminates (10 num maximum)
3. WAP to calculate sum of number is user enter negative it is not added to result (user enter maximum 10 num)

Extra Practical:

1. WAP to input basic salary of an employee and calculate its Incentive and Gross salary and net salary of given your faculty
2. WAP to create BANK Management System.

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH 14

ARRAY

ARRAY :

- An array is a collection of **similar elements** of the **same data type**.
- These similar elements could be all integers, or all floats, or all characters, etc.
- Individual values of array are called as elements.
- Array can be initialized at a place where it is declared.
- Arrays are helpful to store and access a list of values under a single variable name.

Benefits of Array:

- If you want to specify more then one variable then you have to specify as follow.

  int sub1, sub2, sub3, sub4, sub5;

- In place of normal variable declaration we can declare an array variable as given below.
- It means that array will be helpful to make you work easy to process.

  int sub[5];

Use of Array :

- Using a single identifier name, it is possible to access many values of same type.
- Sorting & arranging multiple data related to numeric & character.
- To manage multiple values using less number of program statements.
- To store data in sequential order in memory.
- To access its element using index which is always start from 0
- It is easy to use with other programming concept like looping structure.

Types of Array :

(1) One-Dimensional Array or Single Dimensional Array

(2) Two-Dimensional Array

(3) Multi-Dimensional Array

One-Dimensional Array :

- A list of items can be given a variable name using only one subscript and such a variable is called a ***single dimensional array***.

Declaration of Array:

- Like any other variables, arrays must be declared before they are used.
- Array declaration specifies the data type of array elements like, int, float etc. and the size of the array.

■ **Syntax :**

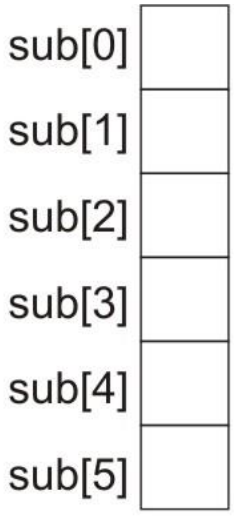
DataType VariableName[size];

■ **Purpose :**

- To declare single dimensional array we can use above syntax.

■ Example :

int sub[6];



Assignment Work :

1. WAP to get the array and display it
2. WAP to display the array with the position.
3. WAP to get the array and display it in the reverse order
4. WAP to get the array and copy it into the another array
5. WAP to get the array and copy the reverse of the array into the another array
6. WAP to get two array and merge it into the third array
7. WAP Sum of the elements of the array
8. WAP Find the max value from the array
9. WAP Find the min value from the array
10. WAP to get the array and search the element from it
11. WAP to replace the particular element from the array
12. WAP to insert the element in array
13. WAP to delete the element from the array

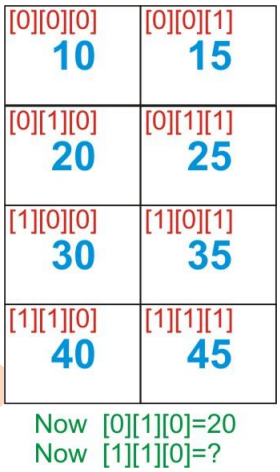
AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH 15

2D ARRAY



Two-Dimensional Array

- A list of items can be store in table format by **row** and **column** is known as Two-Dimensional Array.

- **Syntax** :

  DataType VariableName[rows][columns];

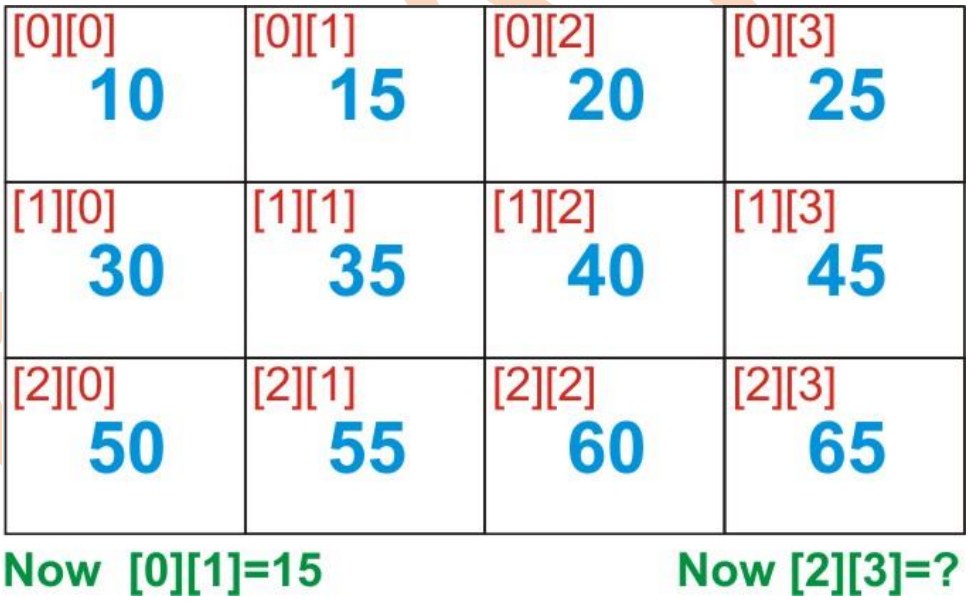
- **Purpose** :

  - To declare tow dimensional array we can use above syntax.

- Example :

  int std[3][4]; or

  int std[3][4]={10,15,20,25,30,35,40,45,50,55,60,65};



Multi-Dimensional Array :

- A list of items can be store in table format by **row** and **column** is with more then two dimension is known as **Multi-Dimensional Array.**

- **Syntax :**

DataType VariableName[rows][col1][Col2]...;

- **Purpose :**
  - To declare multi dimensional array we can use above syntax.
- Example :

  int std[2][2][2]; or

  int std[2][2][2]={10,15,20,25,30,35,40,45};

Assignment work :

1. WAP to get the 2-d array [matrix] and display it
2. WAP to Addition of two matrices
3. WAP to Sum of elements of the matrix
4. WAP to Find the max from the matrix
5. WAP to Find the min from the matrix
6. WAP to Row-wise sum
7. WAP to Column-wise sum

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH 16

STRING

C Strings :

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

1. **By char array**
2. **By string literal**

Let's see the example of declaring string by char array in C language.

```
char ch[10]={'C', 'R', 'E', 'A', 'T', 'I', 'V', 'E', '\0'};
```

We can also define the string by the string literal in C language. For example:

```
char ch[]="CREATIVE";
```

C gets() and puts() functions :

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

Gets() - > get the string   same as a scanf();

Puts() -> print the string same as a printf();

Assignment Work :

1. WAP to Enter char and display it.
2. WAP to Enter Char and display ASCII code
3. WAP to Enter decimal and display char.
4. WAP to enter name and display it.
5. WAP to enter name and count following format.
   - (i) Length of given string.
   - (ii) Total Num of alphabet.
   - (iii) Total Num of digit.
   - (iv) Total Num of Upper character.
   - (v) Total Num of Lower character.
   - (vi) Total Num of space.
   - (vii) Total Num of other character.
6. WAP to given string and display following format.
   - (i) Display Sentence case.
   - (ii) Display lowercase.
   - (iii) Display upper case.
   - (iv) Display title case.
   - (v) Display toggle case.
7. WAP to given string and find out the num of times ‘a’ is there in the string.
8. WAP to given string and Replace character in the string.
9. WAP to given string and Delete character in the string.
10. WAP to given string and display reverse order.

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH 17

C In Built Library Function

C String Functions :

There are many important string functions defined in "string.h" library.

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string,Second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string,second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

Maths Library Function :

The **math.h** header defines various mathematical functions and one macro. All the functions available in this library take **double** as an argument and return **double** as the result. Let us discuss some important functions one by one.

#include<math.h>

1) ceil( )

Syntax : ceil(double value);

Purpose:

To round a value with higher integer value. We can use **ceil** function.

Example :

```
double a=2.5;
printf("Ceil value of %lf is %0.0lf", a, ceil(a));
```

Output :

Ceil value of **2.5** is **3**

2) floor( )

Syntax : floor(double value);

Purpose:

To round a value with lower integer value. We can use **floor** function.

Example :

```
double a=2.5;
printf("Floor value of %lf is %0.0lf", a, floor(a));
```

Output :

Ceil value of **2.5** is **2**

3) pow( )

Syntax : double pow(double, double);

Purpose:

To find power of value we can use **pow** function.

Example :

```
double x = 2.0, y = 3.0;
printf("Ans =2lf",pow(x, y));
```

Output :

Ans =8.00

4) sqrt( )

Syntax : double sqrt(double);

Purpose:

To find square root value. We can use **sqrt** function.

Example :

```
double x = 4.0;
printf(“Square root of %lf is %lf", x, sqrt(x));
```

Output :

Square root of **4.00** is **2.00**

Extra Functions : <dos.h>

1) delay( )

Syntax : delay(milliseconds);

Purpose:

To hold the process in milliseconds. We can use **delay** function.

Example :

```
printf("Hi! Friends");
```

```
delay(5000);
printf("\nWelCome");
```

2) isalpha( )

Syntax : isalpha(char);

Purpose:

To check that given characters are **Alphabet or Not**. We can use **isalpha** function.

Example :

```
char val[]="CREATIVE";
if(isalpha(val))
    printf("Given String is ALPHA");
else
    printf("Given String is NOT ALPHA");
```

3) isalnum( )

Syntax : isalnum(char);

Purpose:

To check that given characters are **Alpha numeric or Not**. We can use **isalnum** function.

Example :

```
char val[]="123ABC";
if(isalnum(val))
    printf("Given String is ALNUM");
else
    printf("Given String is NOT ALNUM");
```

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH - 18

USER DEFINED FUNCTION

C Functions

- In c, we can divide a large program into the basic building blocks known as function.
- The function contains the set of programming statements enclosed by {}.
- A function can be called multiple times to provide reusability and modularity to the C program.
- In other words, we can say that the collection of functions creates a program.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.

Types of Functions

There are two types of functions in C programming:

1. **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

User Defined Functions (UDF) :

- Function which are defined by USER.
- Function which are define as per user requirement are known as **User Defined Function.**
- UDF are also used to stop repetition and duplication of codes.
- While working with user defined function, first of all we have to declare the function and its return type.
- If your function will not return any value then declare function with VOID type.
- If your data will be returned in the form of integer then we have to declare UDF with int data type.
- If your data will be returned in the form of float then we have to declare UDF float data type.

How to define UDF ?

Syntax:

ReturnType FunctionName(argument list);

Purpose:

To declare UDF with required return type we can use above syntax.

Example :

```
void starline( );
int sum(int,int);
float interest(float,float,float);
```

Types of User Defined Functions :

The function can categories in four types. As per its argument and return type.

[1] No Argument and No Return Value.
[2] With Argument and No Return Value.
[3] With argument and With Return Value.
[4] No Argument and With Return Value.

1) No Argument and No Return Value :

- ✓ When function doesn't have any argument and doesn't have any return value at that time it is called as *Function with no Argument and no Return Value.*
- ✓ Means the called function will not receive any argument form the calling function and it will not return any value to calling function.
- ✓ We can use any UDF at any number of times.

2) With Argument and No Return Value :

- ✓ When function have some argument to send called function and have not any return value at that time it is called as *Function with Argument and No Return Value.*
- ✓ Means the called function will receive some argument form the calling function but it will no return value to calling function.

3) With argument and With Return Value :

- ✓ When function have some argument to send called function and have return value at that time it is called as *Function with Argument and Return Value.*
- ✓ Means the called function will receive some argument form the calling function and it will return value to calling function.

4) No Argument and With Return Value :

- ✓ When function doesn't have any argument and have any return value at that time it is called as *Function with no Argument and Return Value.*
- ✓ Means the called function will not receive any argument form the calling function but it will return value to calling function.

Assignment:

1. WAP to enter two value and that values of sum,sub,mul and div using UDF.
2. WAP to area of triangle, circle, rectangle and square with switch case.
3. WAP to find out factorial of a given num using function.
4. WAP to interchange values of two variable using function and global Variable
5. WAP to find out maximum out of three num using function.
6. WAP to reverse a string using function.
7. WAP to sort a list of names in alphabetical order using function.

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH - 19

Recursion

Recursion:

- ✓ Any Function can also call to itself.
- ✓ When any function call it self it is called as recursive function (recursion).
- ✓ It will work till the specified condition will satisfied.
- ✓ **Some points to remember about Recursive Function are as follow.**
  1. Recursive function can be used as a loop.
  2. The function is terminated when any particular condition is not satisfied.
  3. If a recursive function returns some value, it starts returning a value at last when the condition is not satisfied.

Example :

In the following example, recursion is used to calculate the factorial of a number.

```
#include <stdio.h>
int fact (int);
int main()
{
   int n,f;
   printf("Enter the number whose factorial you want to calculate?");
   scanf("%d",&n);
   f = fact(n);
   printf("factorial = %d",f);
}
int fact(int n)  {
   if (n==0)  {
      return 0;
   } else if ( n == 1)  {
      return 1;
   } else {
      return n*fact(n-1);
   }
}
```

Output

```
Enter the number whose factorial you want to calculate?5
factorial = 120
```

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH - 20

Structure

Union

Structure in C :

- ✓ **Structure in c language** is *a user defined datatype* that allows you to hold different type of elements.
- ✓ Each element of a structure is called a member.
- ✓ It works like a template in C++ and class in Java. You can have different type of elements in it.
- ✓ It is widely used to store student information, employee information, product information, book information etc.

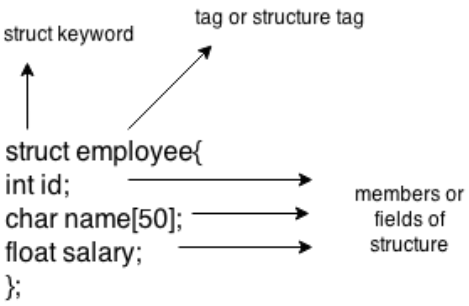
Defining structure :

The **struct** keyword is used to define structure. Let's see the syntax to define structure in c.

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Let's see the example to define structure for employee in c.

```
struct employee
{  int id;
   char name[50];
   float salary;
};
```



Here, **struct** is the keyword, **employee** is the tag name of structure; **id**, **name** and **salary** are the members or fields of the structure. Let's understand it by the diagram.

Example : example of structure in C language.

```
#include <stdio.h>
#include <string.h>
struct employee
{  int id;
   char name[50];
}e1;  //declaring e1 variable for structure
int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, "keyur patel ");//copying string into char array
  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
  return 0;
}
```

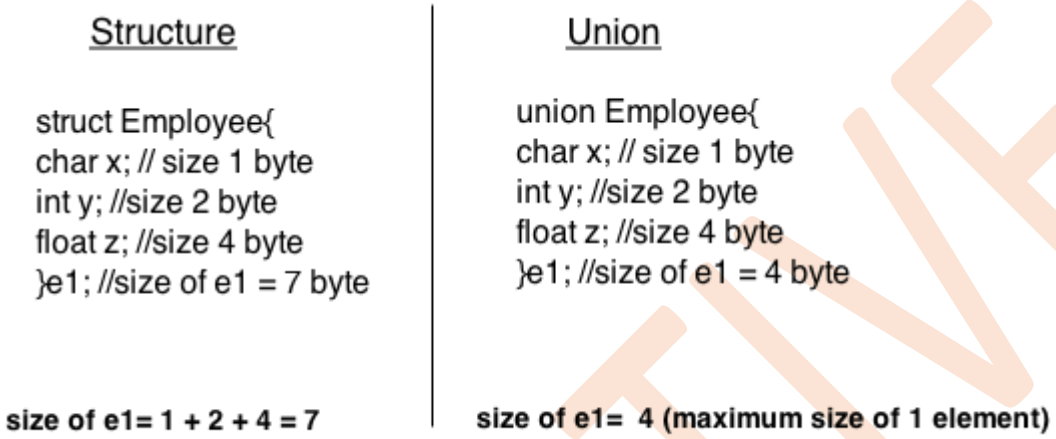
Output:

```
employee 1 id : 101
employee 1 name : keyur patel
```

C Union :

Like structure, **Union in c language** is *a user defined datatype* that is used to hold different type of elements.

But it doesn't occupy sum of all members size. It occupies the memory of largest member only. It shares memory of largest member.



Advantage of union over structure :

It **occupies less memory** because it occupies the memory of largest member only.

Defining union :

The **union** keyword is used to define union. Let's see the syntax to define union in c.

```
union union_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Example :

```
#include <stdio.h>
#include <string.h>
union employee
{  int id;
   char name[50];
}e1;  //declaring e1 variable for union
int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, " keyur patel ");//copying string into char array
  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
  return 0;
}
```

Output:

```
employee 1 id : 1869508435
employee 1 name : keyur patel
```

As you can see, id gets garbage value because name has large memory size. So only name will have actual value.

AN ISO 9001:2015 CERTIFIED INSTITUTE

CREATIVE®

DESIGN & MULTIMEDIA INSTITUTE

CH - 21

Type OF Variable

Storage Class

Pre-Processor

Variable :

What is Variable :

A variable is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Rules for defining variables

- ✓ A variable can have alphabets, digits, and underscore.
- ✓ A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- ✓ No whitespace is allowed within the variable name.
- ✓ A variable name must not be any reserved word or keyword, e.g. int, goto , etc.

Types of Variables :

1) Local Variable :

A variable that is declared and used inside the function or block is called local variable.
It's scope is limited to function or block. It cannot be used outside the block.Local variables need
to be initialized before use.

```
void function1(){
int x=10;//local variable
}
```

2) Global Variable :

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

```
int value=20;//global variable
void function1(){
int x=10;//local variable
}
```

3) Automatic Variable :

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using auto keyword. Automatic variables are similar as local variables.

```
void main(){
int x=10;//local variable (also automatic)
auto int y=20;//automatic variable
}
```

4) Static Variable :

- ✓ A variable that is declared with the static keyword is called static variable.
- ✓ It retains its value between multiple function calls.
- ✓ A normal or auto variable value is destroyed when a function call where the variable was declared is over.
- ✓ a static variable is initialized only the first time it is assigned. This means the code which declares and initializes the code will not overwrite the existing value each time the enclosing block is encountered.
- ✓ Default value is zero

```
void function1(){
int x=10;//local variable
static int y=10;//static variable
x=x+1;
```

```
y=y+1;
printf("%d,%d",x,y);
}
```

If you call this function many times, the local variable will print the same value for each function call, e.g, 11,11,11 and so on. But the static variable will print the incremented value in each function call, e.g. 11, 12, 13 and so on.

5 External Variable :

External variables are also known as global variables. These variables are defined outside the function. These variables are available globally throughout the function execution. The value of global variables can be modified by the functions. “extern” keyword is used to declare and define the external variables.

Scope − They are not bound by any function. They are everywhere in the program i.e. global.

myfile.h

```
extern int x=10;//external variable (also global)
```

program1.c

```
#include "myfile.h"
#include <stdio.h>
void printValue(){
   printf("Global variable: %d", global_variable);
}
```

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic
- External
- Static
- Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

Automatic :

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined.
- The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

Static :

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static.

Register :

The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.

The access time of the register variables is faster than the automatic variables.

The initial default value of the register local variables is 0.

The register keyword is used for the variable which should be stored in the CPU register.

We can store pointers into the register, i.e., a register can store the address of a variable.

External :

The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.

The default initial value of external integral type is 0 otherwise null.

An external variable can be declared many times but can be initialized at only once.

C Pre-Processor :

As the name suggests Preprocessors are programs that process our source code before compilation. There are a number of steps involved between writing a program and executing a program in C / C++.

Preprocessor programs provide preprocessors directives which tell the compiler to preprocess the source code before compiling. All of these preprocessor directives begin with a '#' (hash) symbol. The '#' symbol indicates that, whatever statement starts with #, is going to the preprocessor program, and preprocessor program will execute this statement. Examples of some preprocessor directives are: #include, #define, #ifndef etc. Remember that # symbol only provides a path that it will go to the preprocessor, and command such as include is processed by preprocessor program. For example, include will include extra code to your program. We can place these preprocessor directives anywhere in our program.

There are 4 main types of preprocessor directives:

1. Macros
2. File Inclusion
3. Conditional Compilation
4. Other directives

1 Macros: Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The '#define' directive is used to define a macro. Let us now understand the macro definition with the help of a program:

```
#include <iostream>

// macro definition
#define LIMIT 5
int main()
{
    for (int i = 0; i < LIMIT; i++) {
        std::cout << i << "\n";
```

```
    }

    return 0;
}
```

2 File Inclusion: This type of preprocessor directive tells the compiler to include a file in the source code program. There are two types of files which can be included by the user in the program:

Header File or Standard files: These files contains definition of pre-defined functions like printf(), scanf() etc. These files must be included for working with these functions. Different function are declared in different header files. For example standard I/O functions are in 'stdio.h file

Syntax:

#include< file_name >

where file_name is the name of file to be included. The '<' and '>' brackets tells the compiler to look for the file in standard directory.

user defined files: When a program becomes very large, it is good practice to divide it into smaller files and include whenever needed. These types of files are user defined files. These files can be included as:

#include"filename"

3 Conditional Compilation: Conditional Compilation directives are type of directives which helps to compile a specific portion of the program or to skip compilation of some specific part of the program based on some conditions. This can be done with the help of two preprocessing commands 'ifdef' and 'endif'.

4 Other directives: Apart from the above directives there are two more directives which are not commonly used. These are:

#undef Directive: The #undef directive is used to undefine an existing macro. This directive works as:

#undef LIMIT

Example of Pre-processor :

No	Directive	Use
1	#include	Specifies the files to be included
2	#define	Defines a macro substitution
3	#undef	Undefines a macro
4	#ifdef	Test for a macro definition
5	#endif	Specifies the end of #if
6	#if	Test a compile-time condition
7	#else	Specifies alternatives when #if test fails

C #include :

The #include pre-processor directive is used to paste code of given file into current file. It is used include system-defined and user-defined header files. If included file is not found, compiler renders error.

By the use of #include directive, we provide information to the pre-processor where to look for the header files. There are two variants to use #include directive.

1. #include <filename>
2. #include "filename"

The **#include <filename>** tells the compiler to look for the directory where system header files are held..

The **#include "filename"** tells the compiler to look in the current directory from where program is running.

#define :

The #define pre-processor directive is used to define constant or micro substitution. It can use any basic data type.

Syntax : #define token value

Let's see an example of #define to define a constant.

```
#include <stdio.h>
#define PI 3.14
main() {
  printf("%f",PI);
}
```

Output: 3.140000

C #undef :

The #undef preprocessor directive is used to undefine the constant or macro defined by #define.

Syntax: **#undef token :**

Let's see a simple example to define and undefine a constant.

```
#include <stdio.h>
#define PI 3.14
main() {
   printf("%f",PI);
  #undef PI
  printf("%f",PI);
}
```

Output: Compile Time Error: 'PI' undeclared

The #undef directive is used to define the preprocessor constant to a limited scope so that you can declare constant again.

Let's see an example where we are defining and undefining number variable. But before being undefined, it was used by square variable.

```
#include <stdio.h>
#define number 15
int square=number*number;
#undef number
main() {
  printf("%d",square);
}
```

Output: 225

C #ifdef :

The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

Syntax:

```
#ifdef MACRO
    //code
#endif
```

simple example to use #ifdef preprocessor directive.

```
#include <stdio.h>
#include <conio.h>
#define NOINPUT
void main()  {
    int a=0;
    #ifdef NOINPUT
        a=2;
    #else
        printf("Enter a:");
        scanf("%d", &a);
    #endif
        printf("Value of a: %d\n", a);
}
```

Output: `Value of a: 2`

But, if you don't define NOINPUT, it will ask user to enter a number.

Example 2:

```
void main() {
int a=0;
#ifdef NOINPUT
    a=2;
#else
    printf("Enter a:");
    scanf("%d", &a);
#endif
    printf("Value of a: %d\n", a);
}
```

Output:

```
Enter a:5
Value of a: 5
```

C #if

The #if preprocessor directive evaluates the expression or condition. If condition is true, it executes the code otherwise #elseif or #else or #endif code is executed.

Syntax:

```
#if expression
        //code
#endif
```

Let's see a simple example to use #if #else preprocessor directive.

```
#include <stdio.h>
#include <conio.h>
#define NUMBER 1
void main()
{
        #if NUMBER==0
                printf("Value of Number is: %d",NUMBER);
        #else
                print("Value of Number is non-zero");
        #endif
        getch();
}
```

CH – 22 POINTER

POINTER :

The **pointer in C language** is a variable, it is also known as locator or indicator that points to an address of a value.

Advantage of pointer :

1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.

2) We can return multiple values from function using pointer.

3) It makes you able to access any memory location in the computer's memory.

Usage of pointer :

There are many usage of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

2) Arrays, Functions and Structures

Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

Symbols used in pointer :

Symbol	Name	Description
& (ampersand sign)	address of operator	determines the address of a variable.
* (asterisk sign)	indirection operator	accesses the value at the address.

Address Of Operator :

The address of operator '&' returns the address of a variable. But, we need to use %u to display the address of a variable.

Example:

```
#include<stdio.h>
int main()
{
        int number=50;
        printf("value of number is %d, address of number is %u",number,&number);
        return 0;
}
```

Output : value of number is 50, address of number is fff4

Declaring a pointer :

The pointer in c language can be declared using * (asterisk symbol).

```
int *a;  //pointer to int

char *c;  //pointer to char
```

Example:

```
#include<stdio.h>
int main(){
        int number=50;
        int *p;
        p=&number;   //stores the address of number variable
        printf("Address of p variable is %x \n",p);
        printf("Value of p variable is %d \n",*p);
        return 0;
}
```

Output :

Address of number variable is fff4

Address of p variable is fff4
Value of p variable is 50

NULL Pointer :

A pointer that is not assigned any value but NULL is known as NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value. It will a better approach.

```
int *p=NULL;
```

In most the libraries, the value of pointer is 0 (zero).

Program 1:

Pointer Program to swap 2 numbers without using 3rd variable

```
#include<stdio.h>
int main()
{
        int a=10,b=20,*p1,*p2;
          *p1=&a;
        *p2=&b;
        printf("Before swap: *p1=%d *p2=%d",*p1,*p2);
        *p1=*p1+*p2;
        *p2=*p1-*p2;
        *p1=*p1-*p2;
        printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);
        return 0;
}
```

Output
Before swap: *p1=10 *p2=20
After swap: *p1=20 *p2=10

POINTER WITH ARRAY

Example:

Write a program using pointer to read an array if integer and print element in reverse order

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

```
void main()
{
	int n,i,arr[10];
	int *ptr;
	clrscr();
	ptr=&arr[0];
	printf("Enter n=");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter a[%d]=",i);
		scanf("%d",ptr);
		ptr++;
	}
	ptr--;
	for(i=n-1;i>=0;i--)
	{
		printf("\nArr[%d]=%d",i,*ptr);
		ptr--;
	}
	getch();
}
```

Dynamic memory allocation in C :

The concept of dynamic memory allocation in c language *enables the C programmer to allocate memory at runtime*. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

malloc()

calloc()

realloc()

free()

Before learning above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

static memory allocation	dynamic memory allocation
memory is allocated at compile time.	memory is allocated at run time.

memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Now let's have a quick look at the methods used for dynamic memory allocation.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

malloc() function in C :

- ✓ The malloc() function allocates single block of requested memory.
- ✓ It doesn't initialize memory at execution time, so it has garbage value initially.
- ✓ It returns NULL if memory is not sufficient.

The syntax of malloc() function is given below:

ptr=(cast-type*)malloc(byte-size)

Example:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
        int n,i,*ptr,sum=0;
        printf("Enter n=");
        scanf("%d",&n);
        ptr=(int*)malloc(n*sizeof(int));
```

```
        if(ptr==NULL)
        {
                printf("sorry! unable to allocate mamory");
        }else
        {
                for(i=0;i<n;i++)
                {
                        printf("Enter a[%d]=",i);
                        scanf("%d",ptr+i);
                        sum+=*(ptr+i);
                }
                printf("\nsum=%d",sum);
                free(ptr);
        }
}
```

calloc() function in C :

- ✓ The calloc() function allocates multiple block of requested memory.
- ✓ It initially initialize all bytes to zero.
- ✓ It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

```
ptr=(cast-type*)calloc(number, byte-size)
```

Example :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
        int n,i,*ptr,sum=0;
        printf("Enter n=");
        scanf("%d",&n);
        ptr=(int*)calloc(n,sizeof(int));
        if(ptr==NULL)
        {
                printf("sorry! unable to allocate mamory");
        }else
        {
                for(i=0;i<n;i++)
                {
                        printf("Enter a[%d]=",i);
```

```
                scanf("%d",ptr+i);
                sum+=*(ptr+i);
        }
        printf("\nsum=%d",sum);
        free(ptr);
    }
}
```

realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

syntax of realloc() function.

ptr=realloc(ptr, **new**-size)

Example :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
    int n,i,*ptr,sum=0;
    printf("Enter n=");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));
    if(ptr==NULL)
    {
        printf("sorry! unable to allocate mamory");
    }else
    {
        ptr=(int*)realloc(ptr,(n*2)*sizeof(int)));
        for(i=0;i<n*2;i++)
        {
            printf("Enter a[%d]=",i);
            scanf("%d",ptr+i);
            sum+=*(ptr+i);
        }
        printf("\nsum=%d",sum);
        free(ptr);
    }
}
```

free() function in C

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

Syntax of free() function.

```
free(ptr)
```

CH – 23

File Handling

File Handling in C

- While we are working with any program for data entry, at that time we must store some kind of data inside the computer.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

Creation of the new file

Opening an existing file

Reading from the file

Writing to the file

Deleting the file

Opening File: fopen()

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

```
FILE *fopen( const char * filename, const char * mode );
```

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode

Example : Write File

```
void main()
{
        FILE *fptr;
        char *s;
        clrscr();
        printf("Enter string=");
        gets(s);
        printf("\nstring=%s",s);
        fptr=fopen("cdmi1.txt","w");
        fputs(s,fptr);
        fclose(fptr);
        getch();
}
```

Example : Read File

```
void main() {
        FILE *fptr;
        char ch;
        clrscr();
        fptr=fopen("cdmi1.txt","r");
        while((ch=fgetc(fptr))!=EOF)
        {   printf("%c",ch);
```

```
        }
        fclose(fptr);
        getch();
}
```