PROGRAMMING FOR PROBLEM SOLVING USING C-LANGUAGE

Prepared by:

Dr. Biplab Kumar Paul

University of Engineering and Management, Kolkata-700156, India

Email ID: biplabkumar.paul@uem.edu.in

SYLLABUS

Module 1: Introduction to C Programming: Operators, Expressions, Program structures, Header files, Fundamental examples [2L]

Module 2: Conditional Branching and Loops: Writing and evaluation of conditionals and consequent branching, If, else, if-elseif statements, Switch-case, Iteration and loops [3L]

Module 3: Arrays: Arrays (1-D, 2-D), Character arrays and Strings [3L]

Module 4: Basic Algorithms: Notion of order of complexity through example programs, Searching, Basic Sorting Algorithms (Bubble, Insertion). [4L]

Module 5: Function: Functions (including using built in libraries), Parameter passing in functions, call by value, Passing arrays to functions: Call by reference, Return by reference [3L]

Module 6: Recursion: Recursion, as a different way of solving problems. Example programs, such as Finding Factorial, Fibonacci series, Ackerman function etc, Tail recursion concept [4L]

Module 7: Structure and union: Structures, Defining structures and Array of Structures, Union concept [3L]

Module 8: Pointers: Idea of pointers, defining pointers, Use of Pointers in self-referential structures, Basic of linked list creation, Linked list insert and delete operation [4L]

Module 9: File handling: Create, open, close files, opening modes, File copy using command line argument [4L]

Suggested Text Books

(i) Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice, Hall of India.

(ii) Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill

(iii) E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill

Suggested Reference Books

(i) Reema Thareja, Computer Fundamentals and programming in C, Oxford University Press, (ii) Yashavant Kanetkar, Let Us C, BPB Publications

Introduction to C

General purpose programming language

Can create software, graphic user interface application etc.

System program language

languages are designed for writing system software, which usually requires different development approaches when compared with application software

Portable program language

Transfer to another computer, hardware and os independent,

Mother of all program language

Concepts are same for other language like Python , java, java script, c++, c sharp etc

Features of C

Simple and Efficient

Basic syntax very simple and easy to learn.

Fast

Unlike interpreter-based Java and Python, which are C is a compiler-based program. Availability of only the essential and required features.

Portability

Machine and operating system independent.

E**xtensibility**

Easily and quickly extendable, edit new features, functionalities, and operations

Function-Rich Libraries

Extensive set of libraries with several built-in functions, easy to create user-defined functions and add them to C libraries.

Features of C

Dynamic Memory Management

Can utilize and manage the size of the data structure in C during runtime.

Modularity With Structured Language

Can break a code into different parts using functions

Mid-Level Programming Language

Initially developed as low-level programming.

Pointers

Using the C pointers, we can operate with memory, arrays, functions, and structures.

Recursion

Function that can call itself multiple times until a given condition is true, just like the loops. Provides the functionality of code reusability and backtracking.

Compiler	Interpreter
• A compiler takes the entire program in one go.	• An interpreter takes a single line of code at a time.
• The compiler generates an intermediate machine code.	• The interpreter never produces any intermediate machine code.
• The compiler is best suited for the production environment.	• An interpreter is best suited for a software development environment.
• The compiler is used by programming languages such as C, C ++, C #, Scala, Java, etc.	• An interpreter is used by programming languages such as Python, PHP, Perl, Ruby, etc.

Operators

To perform different type of operations like mathematical, logical, relational , computational etc operators are required

Operands

Operands are the values to perform the operations

Different operators

Arithmetic operators	perform mathematical calculations (+,-,*,/,%)
Assignment operators	assign the values for the variables in C programs. (=), +=, -=, *=, /=, %= (assignment addition/subtraction/ etc
Relational operators	compare the value of two variables. (>,>=,<,<=)
Logical operators	perform logical operations on the given two variables. (AND(&&), OR (||))
Bit wise operators	perform bit operations on given two variables. (AND (&), XOR (^), OR (|) ,Shift left (<<), Shift right (>>),one's complement (~)
Conditional (ternary) operators	return one value if the condition is true and returns another value if the condition is false. (? , :)
Increment/decrement	increase or decrease the value of the variable by one. (++,--)
Special operators	&, *, sizeof( ) and ternary operators.
Equality operators	Check equal or not (= =,!=)
Unary negation operator	convert its operand to a negative number(!)

Operator precedence:

Determines which operation is performed first in an expression with more than one operators with different precedence.

Operators Associativity

Used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left.

Operators	Description	Precedence level	Associativity
()	function call	1	left to right
[]	array subscript		
→	arrow operator		
.	dot operator		
+	unary plus	2	right to left
-	unary minus		
++	increment		
--	decrement		
!	logical not		
~	1's complement		
*	indirection		
&	address		
(data type)	type cast		
sizeof	size in byte		
*	multiplication	3	left to right
/	division		
%	modulus		
+	addition	4	left to right

<<	left shift	5	left to right
>>	right shift		
<=	less than equal to	6	left to right
>=	greater than equal to		
<	less than		
>	greater than		
==	equal to	7	left to right
!=	not equal to		
&	bitwise AND	8	left to right
^	bitwise XOR	9	left to right
|	bitwise OR	10	left to right
&&	logical AND	11	
||	logical OR	12	
?:	conditional operator	13	
=, *=, /=, %= &=, ^=, <<= >>=	assignment operator	14	right to left
	comma operator	15	

Components of a C Program:

1. Header Files Inclusion

Components of a C Program:

	1	`#include <stdio.h>`	Header
	2	`int main(void)`	Main
BODY	3	`{`	
BODY	4	`printf("Hello World");`	Statement
BODY	5	`return 0;`	Return
BODY	6	`}`	

Documentation Section

Details of the program like

Name:

Creator:

Date:

Descriptions:

Many other information you want to provide

Header Files Inclusion

A header file is a file with extension.h which contains C function declarations and macro definitions to be shared between several source files. All lines that start with **#** are processed by a preprocessor which is a program invoked by the compiler.

#include<stdio.h>: It is used to perform input and output operations using functions **scanf()** and **printf()**.

#include<string.h>: It is used to perform various functionalities related to string manipulation like strlen(), strcmp(), strcpy(), size(), etc.

#include<math.h>: It is used to perform mathematical operations like sqrt(), log2(), pow(), etc.

Main Method Declaration - int main()

It is the entry point of a C program and the execution typically begins with the first line of the main().

The empty brackets indicate that the main doesn't take any parameter.

The int that was written before the main indicates the return type of main().

The value returned by the main indicates the status of program termination.

Body of the program - Main (enclosed in {})

The body of a function in the C program refers to statements that are a part of that function.

It can be anything like manipulations, searching, sorting, printing, etc.

A pair of curly brackets define the body of a function. All functions must start and end with curly brackets.

Statement - (printf(“Hello World”);)

Statements are the instructions given to the compiler.

In C, a statement is always terminated by a **semicolon (;).**

In this particular case, we use printf () function to instruct the compiler to display “Hello World” text on the screen.

Return Statement – return 0;

The return statement refers to the return values which depend upon the return type of the function.

The return statement in our program returns the value from main().

The returned value may be used by an operating system to know the termination status of your program.

The value 0 typically means successful termination.

/*First c program with return statement*/

```
#include <stdio.h>
int main (void)
{
printf ("welcome to c Programming language.\n");
return 0;
}
```

Output: welcome to c programming language.

/* Simple program to add two numbers……*/

```
#include <stdio.h>
int main (void)
{
int v1, v2, sum;
//v1,v2,sum are variables and int is data type declared
v1 = 150;
v2 = 25;
sum = v1 + v2;
printf ("The sum of %i and %i is= %i\n", v1, v2, sum);
return 0;
}
```

Output: The sum of 150 and 25 is=175

Variables

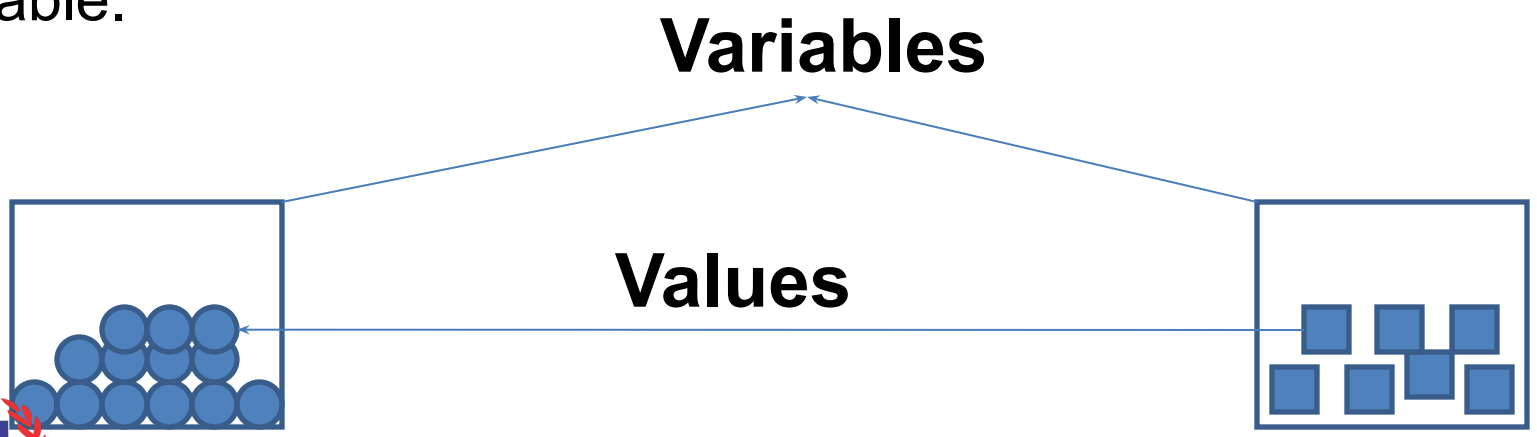
It is a named storage where we can store any kind of data

We consider the variables as a container or basket

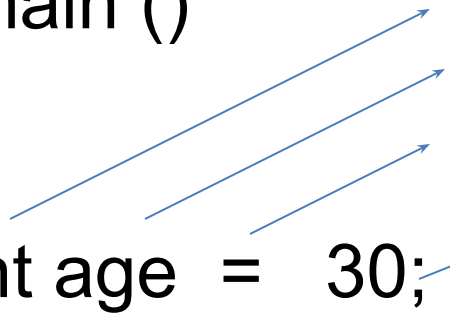
It helps to store the data in the computer memory location

We can store number, string, array,

Then we can access the data from the memory location through variable.



```
# include <stdio.h>
int main ()
{
    int age  =   30;

    return 0;
}
```



Declaration of datatype

Variable name

Assignment operator

Value of variable

Right approach	Wrong approach
myAge	1age
my_age	@age
age1	My age

Data type

Which type of data we could store in the variable is called data type.

Datatype define, how much memory is required to store data

int/float/char/void/array/pointer etc

```
# include <stdio.h>
int main ()
{
    int age  =   30;
    printf (“%d\n”, age);

    float age1 = 30.2;
    printf (“%f\n”, age1);

    char character = “a”;
    printf (“%c\n”, character);

    char my_name [] = “biplab”;
    printf (“%s\n”, my_name);

    return 0;
}
```

%d is for format specifier of integer type data

%f is for format specifier of float type data

%c is for format specifier of character type data

%s is for format specifier of string type data

User input

Using user input print your name:

```
# include <stdio.h>
int main ()
{
 char name[xx]; //value of character you want to give
 printf("Enter your name: ");
 scanf("%s",&name);
 printf("Your name is : %s", name);
}
```

& symbol is used to read a data

Add two user input number:

```
#include <stdio.h>
int main()
{
    int a;
    int b;
    printf("Enter the value of a :");
    scanf ("%d", &a);
    printf("Enter the value of b :");
    scanf ("%d", &b);
    int result=a+b;
    printf ("The result is : %d", result);
    return 0;
}
```

%d is for format specifier of integer type data

& symbol is used to read the number

```c
#include<stdio.h>

int main()
{
    //arithmetic operator
    int a = 10;
    int b = 5;
    int result = a+b;

    printf("the result is: %d",result);

    return 0;
}
```

```c

#include<stdio.h>

int main()
{
    //assignment operator
    int a = 10;
    a += 5; //a = a+5 , a = 10+5 = 15

    printf("the result is: %d",a);

    return 0;
}
```

Relational operation

```
# include <stdio.h>
int main ()
{
    int a=10;
    int b=10;
    int result=a==b;
    printf ("%d", result);
    return 0;
}
```

Logical operation

```
# include <stdio.h>
int main ()
{
    int a=10;
    int b=20;
    int result=a>b && a<b;
    printf ("%d", result);
    return 0;
}
```

Control statement

```
#include<stdio.h>
Int main ()
{
If (condition/expression) if true / if false
     {
     Body of the program
     }                                   /*control statement
else                                     If statement
     {                                   if else statement
     }                                   if else- if ladder
     Return o;                           */
}
```

Whether a person eligible to vote or not

```
#include<stdio.h>
int main ()
{
int age;
printf("Enter your age:");
scanf("%d", &age);
if (age>=18){
    printf ("You are eligible to vote.");
}
else{
    printf ("You are not eligible to vote");
}
return o;
}
```

To check if an input number is divisible by 2 or 3:

%d is for format specifier of integer type data

& symbol is used to read the number

Greatest among three number

```
#include <stdio.h>
int main( ){
int a, b, c;
printf("Please enter three numbers: ");
scanf("%d%d%d",&a, &b, &c);
if(a > b){
if(a > c){
printf("a is the greatest among the three"); }
else{
printf("c is the greatest among the three");}
}
else{
if(b > c){
printf("b is the greatest among the three");}
else{
printf("c is the greatest among the three");}
}
}
```

%d is for format specifier of integer type data

& symbol is used to read the number

To calculate your grade

```
#include<stdio.h>
Int main ()
{
int marks;
printf("Enter your marks:");
pcanf("%d", &marks);
If (marks>85 && marks <= 100)
{
        printf ("Your grade is O.");
}
else if (marks>60 && marks <= 85)
{
        printf ("Your grade is A");
}
else if (marks>40 && marks <= 60)
{
        printf ("Your grade is B");
}else{
        printf ("You are Fail");
}
return 0;
}
```

Switch statement

```
#include<stdio.h>
int main ()
{
int marks;
printf("Enter your marks:");
scanf("%d", &marks);
switch (condition)
{
case value 1;
code execute
printf("……..")
case value 2;
code execute
printf("……..")
default :
printf("……..")
```

```
#include<stdio.h>
int main ()
{
int marks=0;
printf("Enter your marks:");
pcanf("%d", &marks);
switch (marks/10)
{
case 10:
case 9:
        printf ("Your grade is O.");
        break;
case 8:
        printf ("Your grade is A.");
        break;
case 7:
        printf ("Your grade is B.");
        break;
……..
default:
printf("You are fail")
}
```

Loops

```
#include<stdio.h>
Int main ()
{
for loop/ while loop/ do while loop
printf(“hello world”)
printf(“hello world”)
printf(“hello world”)
printf(“hello world”)
return o;
}
```

While loop

```
#include<stdio.h>
int main ()
{
    int I =1;                    //loop initialization
    while(i<=10)                 //loop condition
    {
        printf("hello world\n");
        i++;                     // loop iteration
    }
return o;
}
```

Fahrenheit-Celsius table

```
#include<stdio.h>
int main()
{
    int fahr, cels;
    int lower, upper, step;
    lower =0;
    upper = 300;
    step = 20;
    fahr=lower;
    while (fahr<=upper)
    {
        cels=5*(fahr-32)/9;
        printf ("%d\t %d\n", fahr,
cels);
        fahr=fahr+step;
    }
```

For loop

```
#include<stdio.h>
int main ()
{
    int I =0;                    //loop initialization
    for(i=1; i<=10; i++)    //loop condition
    {
        printf("hello world\n");
    }
return o;
}
```

Fahrenheit-Celsius table

```
#include<stdio.h>
int main()
{
    int fahr;
    for (fahr=0; fahr<=300; fahr=fahr+20)
        printf ("%d  %.1f \n", fahr, (5.0/9.0)*(fahr-32));
}
```

Create a calculator

```
#include<stdio.h>
int main ()
{
    int num1, num2;
    float result=0;
    char operation;
    printf ("Enter the first number : \n");
    scanf ("%d", &num1);
    printf ("Enter the second number : \n");
    scanf ("%d", &num2);
    printf ("Choose operation: \n");
    scanf (" %c", &operation);
    switch(operation)
    {
        case '+':
            result=num1+num2;
            break;
```

```
        case '-':
            result=num1-num2;
            break;
        case '*':
            result=num1*num2;
            break;
        case '/':
            result=(float)num1/(float)num2;
            break;
        case '%':
            result=num1%num2;
            break;
        default:
            printf("Invalid operation. \n");
    }
    printf ("The result is : %d %c %d = %f\n", num1, operation, num2,
result);
    return 0;
}
```

Count the digits of a number

```
#include <stdio.h>
int main()
{
    int n;  // variable declaration
    int count=0;   // variable declaration
    printf("Enter a number");
    scanf("%d",&n);
    while(n!=0)
    {
        n=n/10;
            count++;
    }       printf("\nThe number of digits are : %d", count);
    return 0;
}
```

Do while loop

- The do while loop is a post tested loop.
- Using the do-while loop, we can repeat the execution of several 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);
```

Real-Life Example:

Real-life examples of the do-while loop could be anything by a combination of calling itself for a specified number or infinite times.

- Test driving for a new car uses a do-while loop approach, as we have to drive the car once to know if it fits us or not.
- Cleaning of the house until the dust is being washed/ removed.
- Whenever we fill out a form for a job we have to at least fill in the details for once to check if we are eligible or not.
- Whenever we go to a new restaurant we at least order once to check if the food suits us or not.

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

```
#include<stdio.h>
int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
do{
printf("%d \n",(number*i));
i++;
}while(i<=10);
return 0;
}
```

// C program to demonstrate an infinite loop using do while loop

```
#include <stdio.h>

int main()
{
    do {
        printf(“I love C.\n");
    } while (1);
    return 0;
}
```

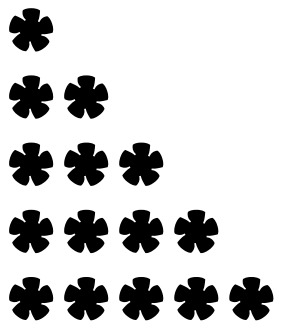
```
#include <stdio.h>
int main()
{
int n;
do
{
    printf("Enter a no.: \n");
    scanf("%d", &n);
}
while(n!=1);
printf("You are out of the loop");
}
```

Nested loops

```
for(j=1; j<=6;j++)
{
    for(i=1, i<=5; i++)
    {
        printf (“*”);
       }
printf(“\n”);
}
```

J=1, *****
J=2 ,*****
*****
*****
*****
j=6, *****

```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   for (i = 1; i <= rows; ++i) {
      for (j = 1; j <= i; ++j) {
         printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```



```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
   for (i = 1; i <= rows; ++i) {
      for (j = 1; j <= i; ++j) {
         printf("%d ", j);
      }
      printf("\n");
   }
   return 0;
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
#include <stdio.h>
int main() {
   int i, j, rows;
   printf("Enter the number of
rows: ");
   scanf("%d", &rows);
   for (i = rows; i >= 1; --i) {
      for (j = 1; j <= i; ++j) {
         printf("* ");
      }
      printf("\n");
   }
   return 0;
}
```

```
* * * * *
* * * *
* * *
* *
*
```

Array

Array is the collection of similar data types or collection of similar entities stored in contiguous memory locations.

Array of characters is a string.

Each data item of an array is called an element.

Each element is unique and located in a separated memory location.

Each element of an array shares a variable but each element has a different index no. known as subscript.

An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension.

The number of subscript is always starts with zero.

One dimensional array is known as vector and two dimensional arrays are known as matrix.

25	32	54	56	54	correct
d	j	u	t	h	correct
23	5.6	g	“h”	Biplab	incorrect

Advantages:

Array variable can store more than one value at a time where other variable can store one value at a time.

Declaration of an array:

Data_type array_name [size];

int arr[100]={10,20,30,100,5……}
int arr[]={10,20,30,100,5……}

We can represent individual array as:

int arr[5];

arr[0], arr[1], arr[2], arr[3], arr[4];

Create an Array and access the value

```
# include<stdio.h>
int main ()
{
     int marks [5]={20, 30, 40, 50, 60};
                          0     1    2   3    4    (indexing)
     printf("The first element is : %d", marks [index no]);
}
```

Replace a value

```
# include<stdio.h>
int main ()
{
     int marks[5]={20,30,40,50,60};
     marks [3]=70;
     printf ("The fourth element is : %d\n" , marks[3]);
}
```

Total size in byte for 1D array is:

Int a[]={1,2,3,4,5,6,7,8,9,10}
Total bytes=size of (data type) * size of array.

Example : if an array declared is:

int [10];

Total byte= 2 * 10 =20 byte.

sizeof(name_of_arr)/sizeof(name_of_arr[0])

Total size in byte for 1D array is:

Int a[]={1,2,3,4,5,6,7,8,9,10}
Total bytes=size of (data type) * size of array.

Example : if an array declared is:

int [10];

Total byte= 2 * 10 =20 byte.

sizeof(name_of_arr)/sizeof(name_of_arr[0])

Total size in byte for 1D array is:

Int a[]={1,2,3,4,5,6,7,8,9,10}
Total bytes=size of (data type) * size of array.

Example : if an array declared is:

int [10];

Total byte= 2 * 10 =20 byte.

sizeof(name_of_arr)/sizeof(name_of_arr[0])

Wap to find the number of elements present in the array

```
#include<stdio.h>
int main()
{
    int a[]={1,2,3,4,5,6,7,8,9,10};
    printf("%d", sizeof (a) / sizeof (a[0]));
}
```

/* Write a program to add 10 array elements */

```
#include<stdio.h>
void main()
{
    int i;
    int arr [10];
    int sum =0;
    for(i=0;i<=9;i++)
    {
        printf ("Enter the %d elements :", i+1 );
        scanf ("%d", &arr[i]);
    }
        for (i=0; i<=9; i++){
        sum = sum + arr[i];}
        printf ("The sum of the array is : %d\n", sum);
}
```

OUTPUT:

```
Enter a value for arr[0] =5
Enter a value for arr[1] =10
Enter a value for arr[2] =15
Enter a value for arr[3] =20
Enter a value for arr[4] =25
Enter a value for arr[5] =30
Enter a value for arr[6] =35
Enter a value for arr[7] =40
Enter a value for arr[8] =45
Enter a value for arr[9] =50
Sum = 275
```

WAP to print the array in reverse order

```
#include<stdio.h>
int main()
{
      int arr[5]={34,54,56,76,54};
      int i;
      for (i=0; i<5; i++)
      {
            printf (" %d ", arr[i]);
      }
printf("\n");
for (i=4; i>=0; i--)
{
      printf(" %d ", arr[i]);
}
return 0;
}
```

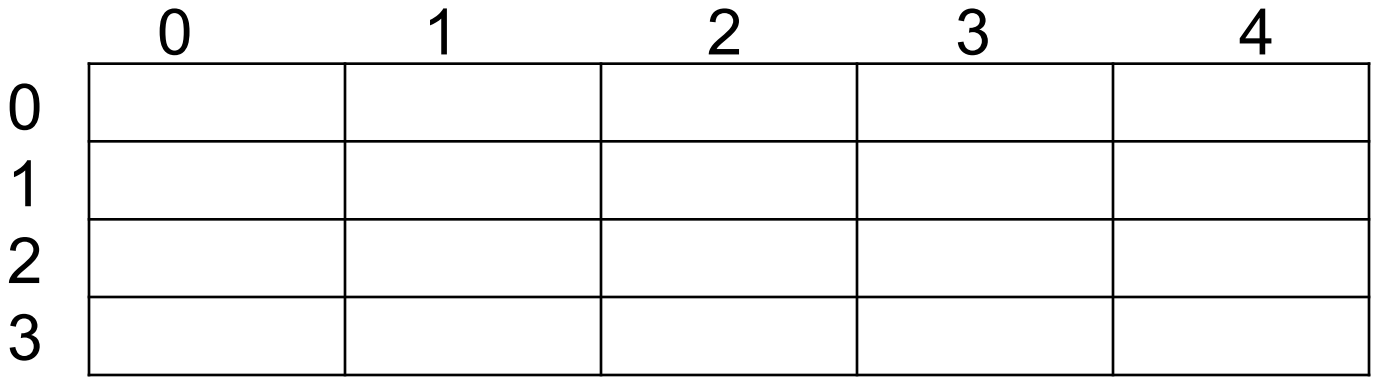
WAP to find out the repeated no in a number

```
#include<stdio.h>
int main(){
	int seen[10]={0};
	int N;
	printf ("Enter the no: ");
	scanf ("%d",&N);
	int rem;
	while(N>0){
		rem=N%10;
		if(seen[rem]==1)
			break;
		seen[rem]=1;
		N=N/10;}
	if (N>0)
		printf("Yes");
	else
		printf("No");
	return 0;
```

2 D array

The two dimensional array can be defined as an array of arrays
Data_type array_name [rows][columns];

Int arr [4][5]
Size of arr[4][5]= 4x5 =20 elements



WAP to access a particular value

```
#include<stdio.h>
int main()
{
int i=0;
int j=0;
int array[4][3] ={1,2,3,4,5,6,7,8,9,10,11,12};
for (i=0; i<4; i++)
    for (j=0;j<3; j++)
    {
        printf("%d\n", array [i][j]);
    }
return 0;
}
```

WAP to access a particular value

```
#include<stdio.h>
int main()
{
    int array[4][3] =
    {
        {1, 2, 3},
        {2, 3, 4},
        {3, 4, 5},
        {4, 5, 6},
    };
    printf ("%d", array [3][2]);
    return 0;
}
```

If we initialize an array as

int mat[4][3]={{11},{12,13},{14,15,16},{17}};

Then the compiler will assume its all rest value as 0,which are not defined.

Mat[0][0]=11, Mat[1][0]=12, Mat[2][0]=14, Mat[3][0]=17

Mat[0][1]=0, Mat[1][1]=13, Mat[2][1]=15 Mat[3][1]=0

Mat[0][2]=0, Mat[1][2]=0, Mat[2][2]=16, Mat[3][2]=0

write a c program to add all the user defined elements of a 2-D matrix

```c
#include <stdio.h>
int main() {
    int rows, cols, i, j;
    int matrix[100][100];
    int sum = 0;
    printf("Enter the number of rows and columns of the matrix: ");
    scanf("%d %d", &rows, &cols);
    printf("Enter the elements of the matrix:\n");
        // read in matrix elements
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
        // calculate sum of matrix elements
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            sum += matrix[i][j];
        }
    }
        // display result
    printf("Sum of all elements in the matrix = %d\n", sum);
    return 0;
}
```

```
Enter the number of rows
and columns of the matrix: 3
3
Enter the elements of the
matrix:
1
2
3
4
5
6
7
8
9
Sum of all elements in the
matrix = 45
```

write a c program to multiply the user defined elements of two 2-D array

```
#include <stdio.h>

int main() {
    int rows1, cols1, rows2, cols2, i, j, k;
    int matrix1[100][100], matrix2[100][100], result[100][100];

    // get dimensions of matrix 1
    printf("Enter no. of rows and columns of matrix 1: ");
    scanf("%d %d", &rows1, &cols1);

    // get elements of matrix 1
    printf("Enter elements of matrix 1:\n");
    for (i = 0; i < rows1; i++) {
        for (j = 0; j < cols1; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
```

//See next page

```
// get dimensions of matrix 2
  printf("Enter no. of rows and columns of matrix 2: ");
  scanf("%d %d", &rows2, &cols2);

  // get elements of matrix 2
  printf("Enter the elements of matrix 2:\n");
  for (i = 0; i < rows2; i++) {
      for (j = 0; j < cols2; j++) {
          scanf("%d", &matrix2[i][j]);
      }
  }

  // check if dimensions are valid for multiplication
  if (cols1 != rows2) {
      printf("Matrices cannot be multiplied");
      return 1;
  }                          //See next page
```

```
 // multiply matrices and store result in result matrix
    for (i = 0; i < rows1; i++) {
        for (j = 0; j < cols2; j++) {
            result[i][j] = 0;
            for (k = 0; k < cols1; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }

    // print result matrix
    printf("Result of matrix multiplication:\n");
    for (i = 0; i < rows1; i++) {
        for (j = 0; j < cols2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
Enter no. of rows and
columns of matrix 1: 2
2
Enter the elements of
matrix 1:
1
2
3
4
Enter no. of rows and
columns of matrix 2: 2
2
Enter the elements of
matrix 2:
1
2
3
4
Result of matrix
multiplication:
7 10
15 22
```

String

Array of character is called a string.

It is always terminated by the NULL character.

String is a one dimensional array of character.

We can initialize the string as

char name[]={'j','o','h','n','\o'};

From the above we can represent as;

Base Address

String can also be initialized as:

char name[]=”John”;

A string constant is a set of character that enclosed within the double quotes and is also called a literal.

String library function

There are several string library functions used to manipulate string and the prototypes for these functions are in header file “string.h”. Several string functions are

strlen(), strcmp(), strcpy(), strcat()

To get the no. of character in a string using strlen(str)

Example:-

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[50];
print(”Enter a string:”);
gets(str);
printf(“Length of the string is: %d\n”,strlen(str));
}
```

Enter a string: biplab
Length of the string is : 6

strcmp()

This function is used to compare two strings.

If the two string match, strcmp() return a value 0 otherwise it return a non-zero value.

It compare the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two string are not same.

strcmp(s1,s2) return a value:

<0 when s1<s2
=0 when s1=s2
>0 when s1>s2

```
/*String comparison…   */
#include<stdio.h>
#include<string.h>
void main(){
     char str1[10],str2[10];
     printf("Enter two strings:");
     gets(str1);
     gets(str2);
     if(strcmp(str1,str2)==0)
{
     printf("String are same\n");
}
else
{
     printf("String are not same\n");}}
```

strcpy()

This function is used to copying one string to another string.

The function strcpy(str1,str2) copies str2 to str1 including the NULL character.

Here str2 is the source string and str1 is the destination string.

The old content of the destination string str1 are lost.

The function returns a pointer to destination string str1.

```
#include<stdio.h>
#include<string.h>
void main()
{
char str1[10],str2[10];
printf(“Enter the second strings:”);
scanf("%s",str2);
strcpy(str1,str2);
printf("The first string value is : %s\n",str1);
}
```

Output:

```
Enter the second strings:delhi
The first string value is : delhi
```

Example:-

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str1[10],str2[10];
    printf("Enter a string:");
    scanf("%s",str2);
    strcpy(str1,str2);
    printf("First string:%s\t\tSecond string:%s\n",str1,str2);
    strcpy(str1,"Delhi");
    strcpy(str2,"Bangalore");

printf("First string :%s\t\tSecond string:%s",str1,str2);
}
```

strcat()

This function is used to concatenate or append a copy of a string at the end of the other string.

If the first string is “Purva” and second string is “Belmont” then after using this function the string becomes “PusvaBelmont”.

The NULL character from str1 is moved and str2 is added at the end of str1.

The 2nd string str2 remains unaffected.

A pointer to the first string str1 is returned by the function.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
    char str1[20],str2[20];
    strcpy(str1, "My name is ");
    strcpy(str2, "Biplab");
    strcat(str1,str2);
    printf("%s",str1);
}
```

Output:
My name is Biplab

Example:-

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str1[20],str2[20];
    printf("Enter two strings:");
    gets(str1);
    gets(str2);
    strcat(str1,str2);
    printf("First string:%s\t second string:%s\n",str1,str2);
    strcat(str1,"-one");
    printf("Now first string is %s\n",str1);
}
```

Arrays of strings

A string is a 1-D array of characters, so an array of strings is a 2-D array of characters. Just like we can create a 2-D array of int, float etc; we can also create a 2-D array of character or array of strings.

Here is how we can declare a 2-D array of characters.

```
char ch_arr[3][10]={ {„s“,“p“,“i“,“k“,“e“,“\0“},
                     { „t“,“o“,“m“,“\0“},
                     {„j“,“e“,“r“,“r“,“y“,“\0“}};
```

It is important to end each 1-D array by the null character otherwise, it's just an array of characters. We can't use them as strings.

```
char ch_arr[3][10]={‖spike‖,‖tom‖,‖jerry‖};
```

The following program demonstrates how to print an array of strings.

```
#include<stdio.h>
int main(){
int i;
char ch_arr[3][10] = {
"spike",
"tom",
"jerry"};
for(i = 0; i < 3; i++){
printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i);}
return 0;}
```

Output:

```
string = spike    address = 2981643408
string = tom      address = 2981643418
string = jerry    address = 2981643428
```

ALGORITHM

Algorithm is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time.

No matter what the input values may be, an algorithm terminates after executing a finite number of instructions.

We represent an algorithm using a pseudo language that is a combination of the constructs of a programming language together with informal English statements.

The ordered set of instructions required to solve a problem is known as an algorithm.

The characteristics of a good algorithm are:

- Precision – the steps are precisely stated (defined).
- Uniqueness – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- Finiteness – the algorithm stops after a finite number of instructions are executed.
- Input – the algorithm receives input.
- Output – the algorithm produces output.
- Generality – the algorithm applies to a set of inputs.

Write a algorithm to find out number is odd or even?

step 1 : start
step 2 : input number
step 3 : rem=number mod 2
step 4 : if rem=0 then
print "number even"
else
print "number odd"
endif
step 5 :stop

Order of algorithm complexity through example programs

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)

Searching

Searching is one of the most common problems that arise in computing.

Searching is the algorithmic process of finding a particular item in a collection of items.

A search typically answers either True or False as to whether the item is present.

On occasion it may be modified to return where the item is found.

Search operations are usually carried out on a keyfield. Well, to search an element in a given array, there are two popular algorithms available:

1. LinearSearch
2. BinarySearch

Linear Search

Very basic and simple search algorithm.

Search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.

It compares the element to be searched with all the elements present in the array and when the element is matched successfully,

It returns the index of the element in the array, else it return -1.

Linear Search is applied on unsorted or unordered lists, when there are fewer elements in a list.

Features of Linear Search Algorithm

1. It is used for unsorted and unordered small list of elements.
2. It has a time complexity of O(n), which means the time is linearly dependent on the number of elements, which is not bad, but not that good too.

Binary Search

1. Used with sorted array or list.
2. We start by comparing the element to be searched with the element in the middle of the list/array.
3. If we get a match, we return the index of the middle element.
4. If we do not get a match, we check whether the element to be searched is less or greater than in value than the middle element.
5. If the element/number to be searched is greater in value than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number),and start again from the step1.

6. If the element/number to be searched is lesser in value than the middle number, then we pick the elements on the left side of the middle element, and start again from the step1.

Binary Search is useful when there are large number of elements in an array and they are sorted. So a necessary condition for Binary search to work is that the list/array should be sorted.

Features of Binary Search

1. It is great to search through large sorted arrays.
2. It has a time complexity of O(log n) which is a very good time complexity
3. It has a simple implementation.

Sorting

- Sorting is the basic operation in computer science.
- Sorting is the process of arranging data in some given sequence or order (in increasing or decreasing order).

For example you have an array which contain 10 elements as follow; 10, 3 ,6 12, 4, 17, 5, 9

After shorting value must be; 3, 4, 5, 6, 9, 10, 12, 17

C language have following technique to sort values;

BubbleSort

SelectionSort

InsertionSort

Bubble Sort

Bubble sort is a simple sorting algorithm in which each element is compared with adjacent element and swapped if their position is incorrect.

It is named as bubble sort because same as like bubbles the lighter elements come up and heavier elements settle down. Both worst case and average case complexity is O (n^2)

```
#include<stdio.h>
int main()
{  int a[50],n,i,j,temp;
   printf("Enter the size of array: ");
   scanf("%d",&n);
   printf("Enter the array elements: ");
   for(i=0;i<n;++i)
      scanf("%d",&a[i]);
for(i=1;i<n;++i)
    for(j=0;j<(n-i);++j)
      if(a[j]>a[j+1])
      {  temp=a[j];
         a[j]=a[j+1];
         a[j+1]=temp;}          }
printf("\nArray after sorting: ");
for(i=0;i<n;++i)
   printf("%d ",a[i]);
return 0;
}
```

Output Enter the size of array: 4
Enter the array elements: 3 7 9 2
Array after sorting: 2 3 7 9

Selection Sort in C

Selection sort is the selection of an element and keeping it in sorted order.

The strategy is to find the smallest number in the array and exchange it with the value in first position of array.

Now, find the second smallest element in the remainder of array and exchange it with a value in the second position,

Carry on till you have reached the end of array.

Now all the elements have been sorted in ascending order step by Step Process.

The selection sort algorithm is performed using following steps...

Step 1: Select the first element of the list (i.e., Element at first position in the list).

Step 2: Compare the selected element with all other elements in the list.

Step 3: For every comparison, if any element is smaller than selected element(for Ascending order), then these two are swapped.

Step 4: Repeat the same procedure with next position in the list till the entire list issorted.

```
#include<stdio.h>
#include<conio.h>
void main(){
   int size,i,j,temp,list[100];
   clrscr();
   printf("Enter the size of the List: ");
   scanf("%d",&size);
   printf("Enter %d integer values: ",size);
   for(i=0; i<size; i++)
      scanf("%d",&list[i]);
//Selection sort logic
   for(i=0; i<size; i++){
      for(j=i+1; j<size; j++){
         if(list[i] >list[j])
{
            temp=list[i];
            list[i]=list[j];
            list[j]=temp;
            }
      }
   }
   printf("List after sorting is:
");
   for(i=0; i<size; i++)
      printf(" %d",list[i]);
   getch();
   }
```

Insertion Sort in C

The insertion sort inserts each element in proper place.

The strategy behind the insertion sort is similar to the process of sorting a pack of cards. You can take a card, move it to its location in sequence and move the remaining cards left or right as needed.

In insertion sort, we assume that first element A[0] in pass 1 is already sorted.

In pass 2 the next second element A[1] is compared with the first one and inserted into its proper place either before or after the first element.

In pass 3 the third element A[2] is inserted into its proper place and so on.

```
#include<stdio.h>
int main()
{
int i,j,n,temp,a[30];
printf("Enter the number of elements:");
scanf("%d",&n);
printf("\nEnter the elements\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=1;i<=n-1;i++)
{
    temp=a[i];
    j=i-1;
while((temp<a[j])&&(j>=0))
{
    a[j+1]=a[j]; //moves element forward
    j=j-1;
}
a[j+1]=temp; //insert element in properplace
}
printf("\nSorted list is as follows\n");
for(i=0;i<n;i++)
{printf("%d ",a[i]);
}return 0;
}
```

Introduction to functions

- A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.
- A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.
- The C standard library provides numerous built-in functions that your program can call.
- For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location, and many more functions.
- A function can also be referred as a method or a sub-routine or a procedure, etc.

Function definition

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function −

Return Type − A function may return a value. The return_type is the data type of the value the function returns.

Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Function Name − This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter.

This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function.

Parameters are optional; that is, a function may contain no parameters.

Function Body – The function body contains a collection of statements that define what the function does.

Category of functions:

A function depending on whether the arguments are present or not and whether a value is returned or not, may belong to one of following categories

1. Function with no return values, no arguments
2. Functions with arguments, no return values
3. Functions with arguments and return values
4. Functions with no arguments and return values.

Function with no return values, no arguments

In this category, the function has no arguments. It does not receive any data from the calling function. Similarly, it doesn't return any value. The calling function doesn‘t receive any data from the called function. So, there is no communication between calling and called functions.

```
void sum ()      \\function declaration
main(){
    sum()        \\func calling
}
void sum (){     \\func definition
    int a=7, b=5, sum=0;
    sum=a+b
    printf (“sum=%d”, sum);
}
```

Functions with arguments, no return values

In this category, function has some arguments. It receives data from the calling function, but it doesn‟t return a value to the calling function. The calling function doesn‘t receive any data from the called function. So, it is one way data communication between called and calling functions.

Note:

In the main() function, n value is passed to the nat() function. The n value is now stored in the formal argument n, declared in the function definition and subsequently, the natural numbers up to n are obtained.

Eg: Printing n Natural no.

```
#include<stdio.h>
#include<conio.h>
void nat( int);
void main(){
int n;
printf("\n Enter n value:");
scanf("%d",&n);
nat(n);}
void nat(int n){
int i;
for(i=1;i<=n;i++)
printf("%d\t",i);}
```

Functions with arguments and return values

In this category, functions have some arguments and it receives data from the calling function. Similarly, it returns a value to the calling function. The calling function receives data from the called function. So, it is two-way data communication between calling and called functions.

Eg. Factorial of a Number

```
#include<stdio.h>
#include<conio.h>
int fact(int);
voidmain()
{
int n;
printf("\n Enter n:");
scanf("%d",&n);
printf("\n Factorial of the number : %d", fact(n));
}
int fact(int n)
{
int i,f;
for(i=1,f=1;i<=n;i++)
f=f*i;
return(f);
}
```

Functions with no arguments and return values.

In this category, the functions have no arguments and it doesn‘t receive any data from the calling function, but it returns a value to the calling function. The calling function receives data from the called function. So, it is one way data communication between calling and called functions.

```
#include<stdio.h>
#include<conio.h>
int sum();
void main()
{
int s;
printf("\n Enter no. of elements to be added :");
s=sum();
printf("\n Sum of the elements :%d",s);
}
int sum()
{
int a[20], i, s=0,n;
scanf("%d",&n);
printf("\n Enter theelements:");
for(i=0;i< n; i++)
scanf("%d",&a[i]);
for(i=0;i< n; i++)
s=s+a[i];
return s;
}
```

Actual parameter – This is the argument which is used in function call.

Formal parameter – This is the argument which is used in function definition

```
add (m, n)
(Actual)
```

```
int add (int m, int n)
return (m+n)
```

Parameter Passing Mechanisms:

Two Ways of Passing Argument to Function in C Language :

A. Call by Reference

B. Call by Value

Function Calls: This calls the actual function

Syntax: function_name (arguments list);

Call by Value:

In call by value method, the value of the variable is passed to the function as parameter. The value of the actual parameter cannot be modified by formal parameter. Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

```
int x=10, y=20;
fun (x, y);
```

10	20

```
int fun (int x, int y)
{
    x=20;
    y=10;
)
```

20	10

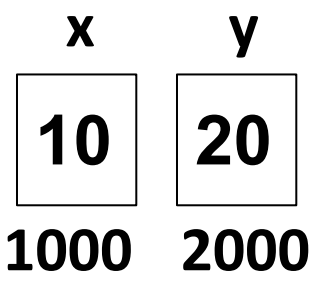
Call by Reference:

In call by reference method, the address of the variable is passed to the function as parameter.

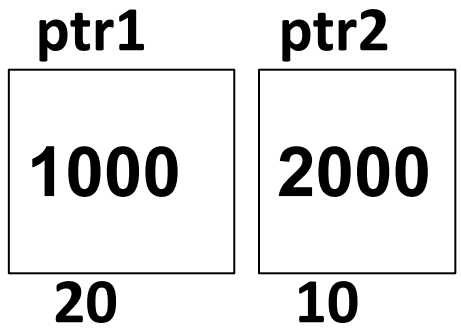
The value of the actual parameter can be modified by formal parameter.

Same memory is used for both actual and formal parameters since only address is used by both parameters.

```
int x=10, y=20;
fun (&x, &y);
```




```
int fun (int *ptr1, int *ptr2)
{
      *ptr1=20;
      *ptr2=10;
)
```



A.Call by Value:

```
#include<stdio.h>
void interchange(int number1,int number2)
{
int temp;
temp = number1;
number1 = number2;
number2 = temp;
}
int main() {
int num1=50,num2=70;
interchange(num1,num2);
printf("\nNumber 1 : %d",num1);
printf("\nNumber 2 : %d",num2);
return(0);
}
```

Output:

Number 1 :50

Number 2 :70

Explanation: Call by Value

While Passing Parameters using call by value , Xerox copy of original parameter is created and passed to the called function.

Any update made inside, method will not affect the original value of variable in calling function.

In the above example num1 and num2 are the original values and Xerox copy of these values is passed to the function and these values are copied into number1,number2 variable of the function respectively.

As their scope is limited to only function so they cannot alter the values inside main function.

B.Call by Reference/Pointer/Address:

```
#include<stdio.h>
void interchange(int *num1, int *num2)
{
int temp;
temp = *num1;
*num1 = *num2;
*num2 = temp;
}
int main() {
int num1=50,num2=70;
interchange(&num1, &num2);
printf("\nNumber 1 : %d",num1);
printf("\nNumber 2 : %d",num2);
return(0);
}
```

Output :

Number 1 :70

Number 2 :50

Call by Address

While passing parameter using call by address scheme, we are passing the actual address of the variable to the called function.

Any updates made inside the called function will modify the original copy since we are directly modifying the content of the exact memory location.

Summary of Call By Value and Call By Reference :

Point	Call by Value	Call by Reference
Copy	Duplicate Copy of Original Parameter is Passed	Actual Copy of Original Parameter is Passed
Modification	No effect on Original Parameter after modifying parameter in function	Original Parameter gets affected if value of parameter changed inside function

Passing Array to a Function:

Whenever we need to pass a list of elements as argument to any function in C language, it is preferred to do so using an array.

Declaring Function with array as a parameter

There are two possible ways to do so, one by using call by value and other by using call by reference.

We can either have an array as a Parameter

int sum (int arr[]);

Or, we can have a pointer in the parameter list, to hold the base address of our

array. int sum (int* ptr);

Returning an Array from a function

We don't return an array from functions, rather we return a pointer holding the base address of the array to be returned.

```
int* sum (int x[])
{
// statements return x ;
}
```

Passing a single array element to a function(Call by value)

In this type of function call, the actual parameter is copied to the formal parameters.

```
#include<stdio.h>
void giveMeArray(int a);
int main()
{
int myArray[] = { 2, 3, 4 };
giveMeArray(myArray[2]);
return 0;
}
void giveMeArray(int a)
{
printf("%d", a);
}
```

Output: 4

```
#include <stdio.h>
void disp( char ch)
{
printf("%c ", ch);
}
int main()
{
char arr[] = {'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j'};
for (int x=0; x<10; x++)
{
disp (arr[x]);
}
return 0;
}
```

Passing array to function using call by reference

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointer as a parameter to receive the passed address.

```
#include <stdio.h>
void disp( int *num){
printf("%d ", *num);}
int main(){
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8,9, 0};
for (int i=0; i<10;i++){
/* Passing addresses of array elements*/
disp (&arr[i]);}
return 0;
}
```

OUTPUT:

1 2 3 4 5 6 7 8 9 0

Passing a complete One-dimensional array to a function

We will only send in the name of the array as argument, which is nothing but the address of the starting element of the array, or we can say the starting memory address.

```
#include<stdio.h>
float findAverage(int marks[]);
int main(){
float avg;
int marks[] = {99, 90, 96, 93, 95};
avg = findAverage(marks);
printf("Average marks = %.1f", avg);
return 0;}
float findAverage(int marks[]){
int i, sum = 0;
float avg;
for (i = 0; i <= 4; i++)
{
sum += marks[i];
}
avg = (sum / 5);
return avg;
}
```

Output: 94.6

Passing a Multidimensional array to a function

For two dimensional array, we will only pass the name of the array as argument.

```
#include<stdio.h>
void displayArray(int arr[3][3]);
int main(){
int arr[3][3], i, j;
printf("Enter 9 no for the array: \n");
for (i = 0; i < 3;++i){
for (j = 0; j < 3;++j){
scanf("%d", &arr[i][j]);}}
// passing the array as argument
displayArray(arr)//
return 0;
}
```

```
void displayArray(int arr[3][3])
{
int i, j;
printf("The complete array is:\n");
for (i = 0; i < 3;++i)
{
// getting cursor to new
lineprintf("\n");
for (j = 0; j < 3;++j)
{
// \t is used to provide tab space
printf("%4d", arr[i][j]);}}}
```

Output:

Enter 9 no for the array: 1 2 3 4 5 6 7 8 9

The complete array is: 1 2 3 4 5 6 7 8 9

```
#include <stdio.h>
void displayString(char str[]);
int main()
{
char str[50]; printf("Enter string: ");
gets(str);
displayString(str);
// Passing string c to function.
return 0;
}
void displayString(char str[])
{
printf("String Output: ");
puts(str);
}
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void strreverse(char *string)
{
int i, len;
char c;
len=strlen(string);
charstring2[len+1];
for(i=0; i<len;i++)
{
c=string[i];
string2[len-i]=c;
}
string2[len+1]='\0';
string=string2;
//printf("%s\n", string);
}

int main(int argc, char *argv[])
{
char str[256];
printf("Type a String to reverse
it.[max. 255 chars]\n");
fgets(str, 255, stdin);
strreverse(&str[0]);
printf("%s", str);
return 0;
}
```

```
#include<stdio.h>
#include<math.h>
int main()
{
   printf("Enter a Number to Find Factorial: ");
   printf("\nFactorial of a Given Number is: %d ",fact());
   return 0;
}
int fact()
{
   int i,fact=1,n;
   scanf("%d",&n);
   for(i=1; i<=n; i++)
   {
      fact=fact*i;
   }
   return fact;
}
```

```
#include <stdio.h>
#include <string.h>
main()
{
    char s1[20], s2[20];
    printf("\nEnter first string: ");
    gets(s1);
    printf("\nEnter second string: ");
    gets(s2);
    strcat(s1, s2);
    printf("\nThe concatenated string is: %s", s1);
    getch();
}
```

```
#include<stdio.h>
int SumofNumbers(int a[], int Size);
int main()
{
 int i, Size, a[10];
 int Addition;
 printf("Please Enter the Size of an Array: ");
 scanf("%d", &Size);
 printf("\nPlease Enter Array Elements\n");
 for(i = 0; i < Size; i++)
  {
      scanf("%d", &a[i]);
  }
 Addition = SumofNumbers(a, Size);
 printf("Sum of All Elements in an Array = %d
", Addition);
 return 0;
}
```

```
int SumofNumbers(int a[], int
Size)
{
       int Addition = 0;
       int i;
       for(i = 0; i < Size; i++)
       {
      Addition = Addition + a[i];
       }
        return Addition;
  }
```

Inter Function communication

When a function gets executed in the program, the execution control is transferred from calling function to called function and executes function definition, and finally comes back to the calling function. In this process, both calling and called functions have to communicate each other to exchange information. The process of exchanging information between calling and called functions is called as inter function communication.

1. Downward Communication
2. Upward Communication
3. Bi-directional Communication

Downward Communication

In this type of communication, the data is transferred from calling function to called function but not from called function to calling function. The function with parameters and without return value is considered under Downward communication.

Example

```
#include <stdio.h>
#include<conio.h>
    void main(){
    int num1, num2 ;
    void addition(int, int) ; // fun dec
    clrscr() ;
    num1 = 10 ;
    num2 = 20 ;
    printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
    addition(num1, num2) ; // calling func
    getch() ;
}
void addition(int a, int b) // called function
{
    printf("SUM = %d", a+b) ;
}
```

Upward Communication

In this type of communication, the data is transferred from called function to calling function but not from calling function to called function. The function without parameters and with return value is considered under upward communication.

Example

```
#include <stdio.h>
#include<conio.h>
void main(){
    int result ;
    int addition() ; // func dec
    clrscr() ;
    result = addition() ; // calling func
    printf("SUM = %d", result) ;
    getch() ;
}
int addition() // called function
{
    int num1, num2 ;
    num1 =10;
    num2 =20;
    return (num1+num2) ;
}
```

Bi-Directional Communication

In this type of communication, the data is transferred from called function to calling function and also from calling function to called function. The function with parameters and with return value is considered under Bi-Directional communication.

```
#include <stdio.h>
void main(){
int num1, num2, result ;
int addition() ; // func declaration
num1 = 10 ;
num2 = 20 ;
result = addition(num1, num2) ; // calling function
printf("SUM = %d", result) ;
}
int addition(int num1, int num2) // called function
{
return (num1+num2) ;
}
```

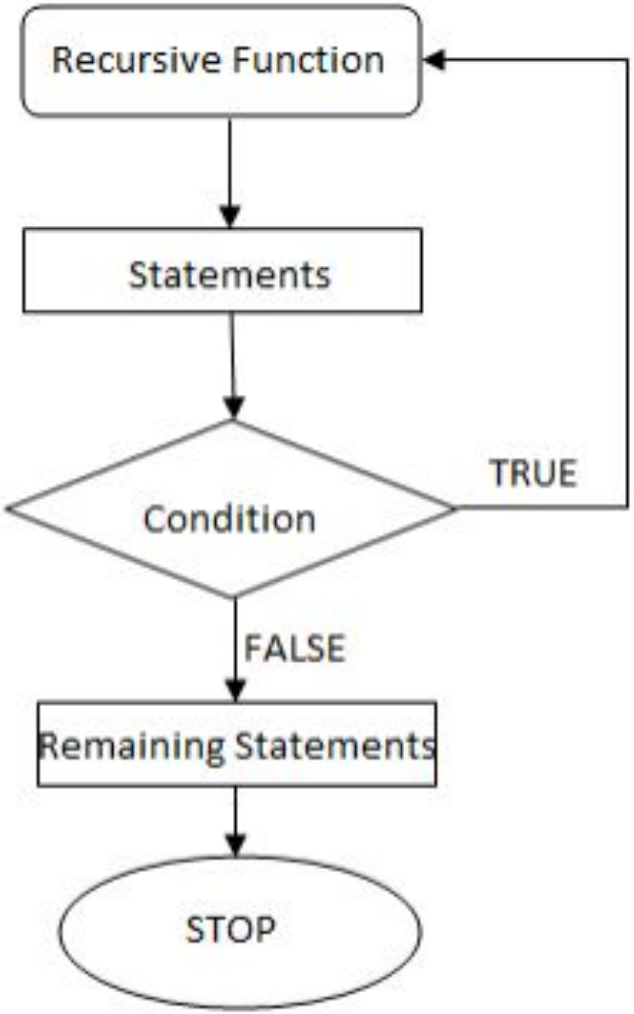
Addition of two numbers using pointers where bi-directional communication occurred.

```
#include <stdio.h>

void add(int *a, int *b, int *result);

int main() {
    int num1, num2, sum;
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    add(&num1, &num2, &sum);
    printf("The sum of %d and %d is %d\n",
num1, num2, sum);
    return 0;
}

void add(int *a, int *b, int *result) {
    *result = *a + *b;
}
```

- the add function takes three arguments
- pointers to a and b, which represent the two numbers to be added, and a pointer to result, which will store the sum of the two numbers.
- main function prompts the user to enter two integer numbers, reads them using scanf, and passes their addresses to add using the & operator.
- The add function dereferences the pointers to get the values of a and b, adds them together, and stores the result in the memory location pointed to by result.
- Finally, main dereferences the result pointer to print the sum of the two numbers to the console.

Recursion

The process of calling a function by itself is called recursion and the function which calls itself is called recursive function. Recursion is used to solve various mathematical problems by dividing it into smaller problems.



Syntax of Recursive Function

```
return_type recursive_func ([argument list])
{
statements;
        ... ... ...
        recursive_func ([actual argument]);
        ... ... ...
}
```

Example:

C Program to show infinite recursive function

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

In this program, we are calling main() from main() which is recursion. But we haven't defined any condition for the program to exit. Hence this code will print "Hello world" infinitely in the output screen.

Direct Recursion

A function is said to be direct recursive if it calls itself directly.

Example: C Program Function to show direct recursion

```
int fibo (int n)
{
    if (n==1 || n==2)
            return 1;
    else
            return (fibo(n-1)+fibo(n-2));
}
```

In this program, fibo() is a direct recursive function. This is because, inside fibo() function, there is a statement which calls fibo() function again directly.

Indirect Recursion

A function is said to be indirect recursive if it calls another function and this new function calls the first calling function again.

Example: C Program Function to show indirect recursion

```
int func1(int n)
{
    if (n<=1)
        return 1;
    else
        return func2(n);
}
int func2(int n)
{
    return func1(n);
}
```

Factorial of a number using direct recursion:

```
#include <stdio.h>
int factorial(int n);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    printf("Factorial of %d is %d", n,
factorial(n));
    return 0;
}

int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Using indirect recursion:

```
#include <stdio.h>
int factorial(int n);
int multiply(int a, int b);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    printf("Factorial of %d is %d", n,
factorial(n));
    return 0;
}

int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return multiply(n, factorial(n - 1));
    }
}

int multiply(int a, int b) {
    return a * b;
}
```

Sum of n natural number using direct recursion:

```
#include <stdio.h>
int sum(int n);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);

 printf("The sum of the first %d natural
numbers is %d", n, sum(n));

return 0;
}
int sum(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n + sum(n - 1);
    }
}
```

Using indirect recursion:

```
#include <stdio.h>
int sum(int n);
int add(int a, int b);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    printf("The sum of the first %d natural
numbers is %d", n, sum(n));
    return 0;
}
int sum(int n) {
    if (n == 1) {
        return 1;
    } else {
        return add(n, sum(n - 1));
    }
}
int add(int a, int b) {
    return a + b;
}
```

```
// convert binary to decimal

#include <stdio.h>
#include <math.h>

// function prototype
int convert(long long);

int main() {
  long long n;
  printf("Enter a binary number: ");
  scanf("%lld", &n);
  printf("%lld in binary = %d in decimal", n, convert(n));
  return 0;
}
```

```
// function definition
int convert(long long n) {
  int dec = 0, i = 0, rem;
  while (n!=0) {
    rem = n % 10;
    n =n/ 10;
    dec =dec+( rem * pow(2, i));
    ++i;
  }
  return dec;
}
```