INTRODUCTION TO C LANGUAGE

UNIT-I

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. C is a computer programming language. That means that you can use C to create lists of instructions for a computer to follow. C is one of thousands of programming languages currently in use. C has been around for several decades and has won widespread acceptance because it gives programmers maximum control and efficiency. C is an easy language to learn. It is a bit more cryptic in its style than some other languages, but you get beyond that fairly quickly. Possibly why C seems so popular is because it is reliable, simple and easy to use. Moreover, in an industry where newer languages, tools and technologies emerge and vanish day in and day out, a language that has survived for more than 3 decades has to be really good.

Problem Analysis in C : Problem analysis in Computer programming is the process where we break down problems into its components so that the problems can easily be understood. The way to do this is to write an ALGORITHM of the problem. Problem Analysis in Computer programming is a way where problems are discussed for giving the solution.

What is algorithm?

An algorithm is a procedure or step-by-step instruction for solving a problem. They form the foundation of writing a program.

For writing any programs, the following has to be known:

- Input
- Tasks to be preformed
- Output expected

Write an algorithm that perform addition of two numbers and display the result of the operation.

Algorithm :

1. START

2. INPUT A,B

3. C=A+B

4. PRINT C

5. STOP

There are 5 important characteristics or properties for a well defined algorithm. These are :-

Input – Zero or more values externally supplied to algorithm

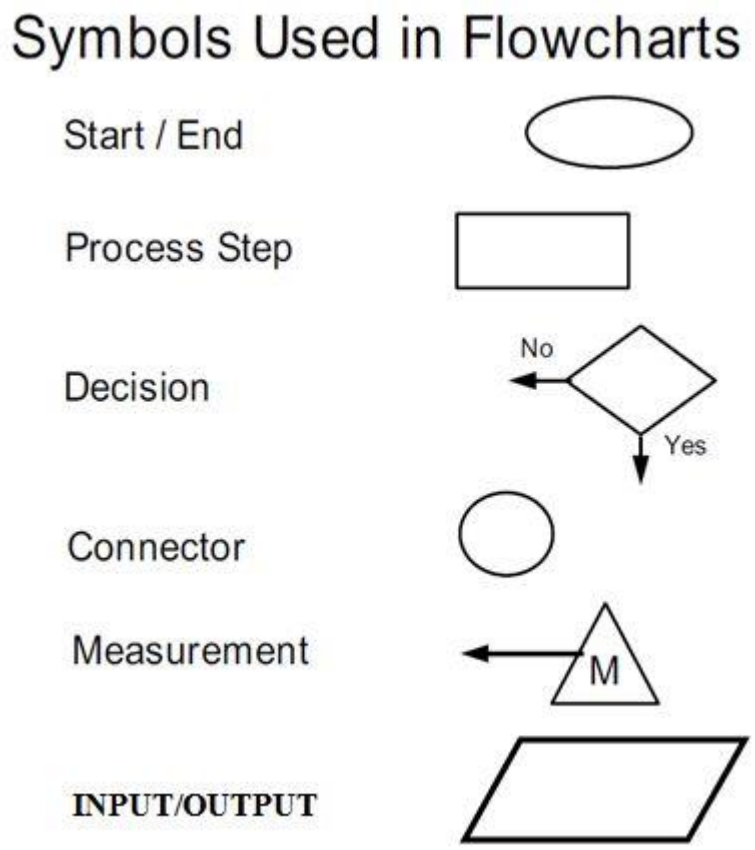
Output – Zero or more values are produced by algorithm.

Definiteness – Each instruction must be clear and unambiguous.

Finiteness – Algorithm must terminate after a finite number of steps.

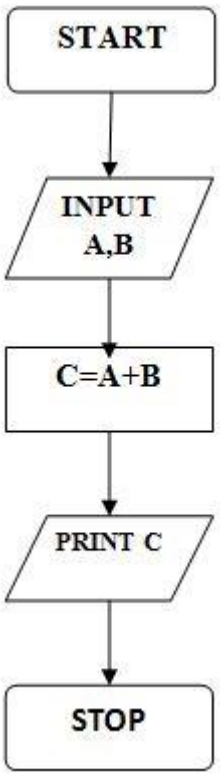
Effectiveness – Each instruction must be efficient and feasible.

Flowchart : Flow chart is very important tool by which we can understand the flow of algorithm and program very easily . It is pictorial representation of step by step solution of a problem. Programmer often uses it as a program planning tool for visually organizing step necessary to solve a problem. It uses boxes of different shapes that denotes different type of instruction. The set of rules that define how a particular problem can be solved in finite number of steps is known as algorithm. A good algorithm help us to create a good program A Flowchart is a diagram that uses graphic symbols to depict the nature and flow of the steps in a process. Another name for this tool is "flow diagram."



Q1.) WRITE AN ALGORITHM AND DRAW A FLOWCHART FOR THE ADDITION OF TWO NUMBERS WHICH ARE INPUT BY YOU AND DISPLAY THEIR SUM.

1. Starts
2. INPUT A,B
3. C=A+B
4. PRINT C
5. STOP



Control Structures in C :

Control Structures is used to control the flow of a program it includes decision control structure, loop control structure, and case control control structure. To send the control out of sequence of statements or from one part of program to another part we use control structure.

Decision Control Structure: There are several programming situation where we need to take decisions, C language too must be able to perform different sets of actions depending on the circumstances. C has three major decision making instructions the if statement, the if-else statement, and the switch statement.

The If Statement: C uses the keyword if to implement the decision control instruction. The general form of if statement or its syntax looks like this:

if (this condition is true)

execute this statement ;

or

if (this condition is true)

{

statement ;

statement ;

statement ;

The If-else Statement: The if statement by itself will execute a single statement, or a group of statements, when the expression following if evaluates to true. It does nothing when the expression evaluates to false. We can execute a statement or a group of statements if the expression evaluates to evaluates to false. The general form of if-else statement or its syntax looks like this:

if (condition)

{

do this ;

and this ;

}

else

{

do this ;

and this ;

}

Or

if (condition)

do this ;

else

and this;

Why is modular programming important?

As programs grow in size, it becomes important to break them into separate parts (modules) that communicate with rest of the program through a few well defined interfaces.

If we decompose the program into modules well is we can code each module independently. Also if change happens we can localize changes to a particular module without impacting the rest of the programs.

Character set of C :

character:- It denotes any alphabet, digit or special symbol used to represent information.

Use:- These characters can be combined to form variables. C uses constants, variables, operators, keywords and expressions as building blocks to form a basic C program.

Character set:- The character set is the fundamental raw material of any language and they are used to represent information. Like natural languages, computer language will also have well defined character set, which is useful to build the programs.

The characters in C are grouped into the following **two** categories:

1. **Source character set**
   a. Alphabets
   b Digits
   c. Special Character
   d. White Space

2. **Execution character set**
   a. Escape Sequence

ALPHABETS

Uppercase letters	A-Z
Lowercase letters	a-z

DIGITS

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

C Identifiers :

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program. For example:

```
int money;
```

```
double accountBalance;
```

Here, money and accountBalance are identifiers.

Identifier names must be different from keywords.We cannot use int as an identifier because int is a keyword.

Rules for naming identifiers:

1.A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.

2.The first letter of an identifier should be either a letter or an underscore.

3.We cannot use keywords as identifiers.

Keywords in C :

A keyword is a reserved word**.** You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below:

auto	Break	case	char	const	continue	default
double	Else	enum	extern	float	for	goto
int	Long	register	return	short	signed	sizeof
struct	Switch	typedef	union	unsigned	void	volatile

Data Types in C :

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

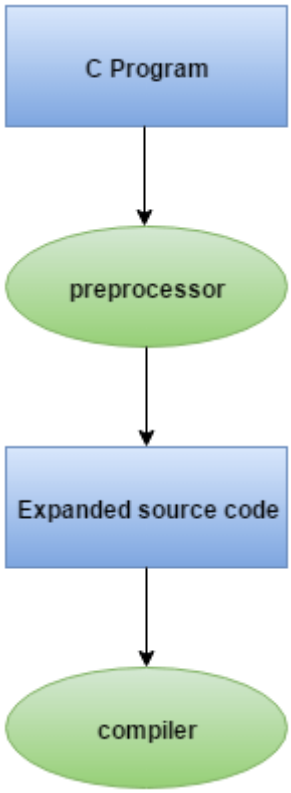
There are the following data types in C language.

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

C Preprocessor Directives :

The C preprocessor is a micro processor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros.

All preprocessor directives starts with hash # symbol.



C Macros :

A macro is a segment of code which is replaced by the value of macro. Macro is defined by #define directive. There are two types of macros:

1. Object-like Macros
2. Function-like Macros

Object-like Macros :

The object-like macro is an identifier that is replaced by value. It is widely used to represent numeric constants. For example:

#define PI 3.14

Here, PI is the macro name which will be replaced by the value 3.14.

Function-like Macros :

The function-like macro looks like function call. For example:

#define MIN(a,b) ((a)<(b)?(a):(b))

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. The function is also known as *procedure* or *subroutine* in other programming languages.

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.
- However, Function calling is always a overhead in a C program.

The syntax of creating function in c language is given below:

```
1. return_type function_name (data_type parameter)
2. {
3. //code to be executed
4. }
```

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.

C gets() function :

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

char[] gets(char[]);

C puts() function

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

int puts(**char**[])

fprintf() function

The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

int fprintf(FILE *stream, const char *format [, argument,])

UNIT-II

C Operators :

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Ternary or Conditional Operators
- Assignment Operator

Category	Operator	Associativity
Postfix	() [] -> . ++ -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left

Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

do-while loop in C :

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once

The syntax of do-while loop in c language is given below:

```
1. do{
2. //code to be executed
3. }while(condition);
```

while loop in C :

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

The syntax of while loop in c language is given below:

```
1. while(condition){
2. //code to be executed
3. }
```

for loop in C

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

The syntax of for loop in c language is given below:

```
1. for(initialization;condition;incr/decr){
2. //code to be executed
3. }
```

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.

UNIT-III

C Array :

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

Advantage of C Array :

1) Code Optimization: Less code to the access the data.

2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.

3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.

4) Random Access: We can access any element randomly using the array.

Declaration of C Array :

We can declare an array in the c language in the following way.

data_type array_name[array_size];

eg. **int** marks[5];

Two Dimensional Array in C :

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C :

The syntax to declare the 2D array is given below.

data_type array_name[rows][columns];

int twodimen[4][3];

Here, 4 is the number of rows, and 3 is the number of columns.

2-D array : int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

Passing Array to Function in C :

In C, there are various general problems which requires passing more than one variable of the same type to a function. For example, consider a function which sorts the 10 elements in ascending order. Such a function requires 10 numbers to be passed as the actual parameters from the main function. Here, instead of declaring 10 different numbers and then passing into the function, we can declare and initialize an array and pass that into the function. This will resolve all the complexity since the function will now work for any number of values.

Syntax :

Function name(arrayname);

Methods to declare a function :

1. return_type function(type arrayname[])
2. return_type function(type arrayname[SIZE])

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

Syntax :

char ch[10]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};

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

char ch[]="IITM";

POINTERS

C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

```
1. int n = 10;
2. int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type i
   nteger.
```

Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
1. int *a;//pointer to int
2. char *c;//pointer to char
```

EXAMPLE :

```
1. #include<stdio.h>
2. int main(){
3. int number=50;
4. int *p;
5. p=&number;//stores the address of number variable
6. printf("Address of p variable is %x \n",p); // p contains the address of the number therefore
   printing p gives the address of number.
7. printf("Value of p variable is %d \n",*p); // As we know that * is used to dereference a point
   er therefore if we print *p, we will get the value stored at the address contained by p.
8. return 0;
9. }
```

Pointer to array :

```
1. int arr[10];
2. int *p[10]=&arr; // Variable p of type pointer is pointing to the address of an integer array ar
   r.
```

Pointer to a function

```
1. void show (int);
2. void(*p)(int) = &display; // Pointer p is pointing to the address of a function
```

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.

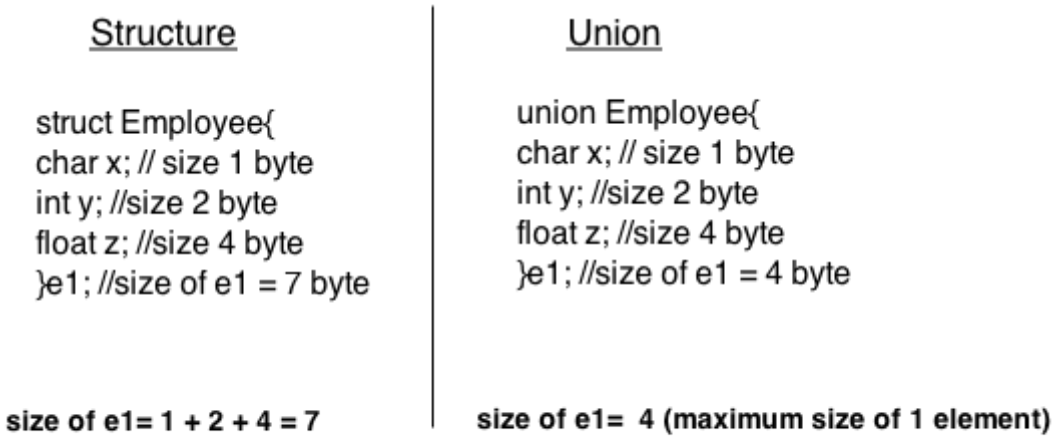
2) We can return multiple values from a function using the pointer.

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

C Union

Like structure, Union in c language is *a user-defined data type* that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.



Defining union

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

```
1. union employee
2. {   int id;
3.     char name[50];
4.     float salary;
5. };
```

C STRUCTURE :

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity Student may have its name (string),

roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

- Construct individual arrays for storing names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types.
- Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information
- The struct keyword is used to define the structure.

syntax to define the structure in c.

```
1. struct structure_name
2. {
3.    data_type member1;
4.    data_type member2;
5.    .
6.    .
7.    data_type memeberN;
8. };
```

Declaring structure variable :

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

Example :

```
1. struct employee
2. {   int id;
3.    char name[50];
4.    float salary;
5. };
```

Accessing members of the structure

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

The code to access the *id* member of *p1* variable by. (member) operator.

p1.id

Nested Structure in C

Structure can be nested in two ways :

1. By separate structure
2. By Embedded structure

C Nested Structure example

```
1. #include <stdio.h>
2. #include <string.h>
3. struct Employee
4. {
5.   int id;
6.   char name[20];
7.   struct Date
8.    {
9.     int dd;
10.    int mm;
11.    int yyyy;
12.   }doj;
13. }e1;
14. int main( )
15. {
16.  //storing employee information
17.  e1.id=101;
18.  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
19.  e1.doj.dd=10;
20.  e1.doj.mm=11;
21.  e1.doj.yyyy=2014;
22.
23.  //printing first employee information
24.  printf( "employee id : %d\n", e1.id);
25.  printf( "employee name : %s\n", e1.name);
26.  printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n", e1.doj.dd,e1.doj.mm,e1.d
     oj.yyyy);
27.  return 0;
28. }
```

UNIT-IV

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.

Example :

```
1. #include <stdio.h>
2. int main()
3. {
4. int a; //auto
```

```
5. char b;
6. float c;
7. printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.
8. return 0;
9. }
```

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.

EXAMPLE :

```
1. #include<stdio.h>
2. static char c;
3. static int i;
4. static float f;
5. static char s[100];
6. void main ()
7. {
8. printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.
9. }
```

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.
- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- 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.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

EXAMPLE:

```
#include <stdio.h>
int main()
{
register int a; // variable a is allocated memory in the CPU register. The initial default value of a is 0.
printf("%d",a);
}
```

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 variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

EXAMPLE :

```
#include <stdio.h>
int main()
{
extern int a;
printf("%d",a);
}
```

File Handling in C :

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

Functions in C :

Opening File: fopen()
FILE *fopen(const char * filename, const char * mode);

Closing File: fclose()

The fclose() function is used to close a file. The file must be closed after performing all the operations on it. The syntax of fclose() function is given below:

int fclose(FILE *fp);

Reading File : fscanf() function

The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.
Syntax :

int fscanf(FILE *stream, const char *format [, argument,])

Writing File : fputc() function

The fputc() function is used to write a single character into file. It outputs a character to a stream. Syntax :

int fputc(int c, FILE *stream)

Reading File : fgetc() function

The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

int fgetc(FILE *stream)

C PROGRAMMING APPLICATIONS :

Bubble Sort :

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. Bubble sort in C to arrange numbers in ascending order, we can modify it for descending order and can also sort strings. The bubble sort algorithm isn't efficient as its average-case complexity is $O(n^2)$ and worst-case complexity is $O(n^2)$.

Bubble sort program in C

```
#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);

  return 0;
}
```

SEARCHING :

LINEAR SEARCH :

Linear search in C to find whether a number is present in an array. If it's present, then at what location it occurs. It is also known as a sequential search. It is straightforward and works as follows: we compare each element with the element to search until we find it or the list ends.

Linear search program in C

```
#include <stdio.h>

int main()
{
  int array[100], search, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```

BINARY SEARCH :

Binary search in C language to find an element in a sorted array. If the array isn't sorted, you must sort it using a sorting technique such as merge sort. If the element to search is present in the list, then we print its location. The program assumes that the input numbers are in ascending order.s

```
#include <stdio.h>
int main()
{
  int c, first, last, middle, n, search, array[100];

  printf("Enter number of elements\n");
  scanf("%d", &n);
```

```
  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter value to find\n");
  scanf("%d", &search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);

  return 0;
}
```