The C Language Programming

Objective

At the end of this session you will understand :

- Internal Architecture of the computer.
- Need a translator(Compiler and Interpreter).
- Need for programming.

INDEX

19. Dynamic Memory Allocation.
20. Macros
21. Recursion.
22. Structure and typedef in C .
    a. Creating the structure variable
    b. Accessing the structure variable.
    c. pointer to structure.
    d. typedef with Structure.
23. Union in C and Difference between Structure and union
24. File Handling.
25. Data Structure.

1.Introduction to C Language

- C language was developed by Dennis Ritchie in 1972 at At and T bell laboratories.
- C language is called as the mother language
    - C language is considered as the mother language of all the modern programming languages because most of the compilers, JVMs, Kernels, etc. are written in C language, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.
- C language is a middle level programming language.
    - C is considered as a middle-level language because it supports the feature of both low-level and high-level languages

2.History of C Language

- C programming language was developed in 1972 by Dennis Ritchie at Bell Laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.
- C was developed using BCPL and B by Dennis Ritchie at the Bell lab in 1972. So, in terms of the history of C language, it was used in mainly academic environments, but at long last with the release of many C compilers for commercial use and the increasing popularity of UNIX, it began to gain extensive support among professionals.

3.Features of the C Language

C is a widely used programming Language. it provides many features.

- Easy to learn : Syntaxes are easy to understand.
- Middle level programming language : supports the both high and low level programming language features.
- Structured programming language : we can break the problem into small parts using functions.
- Rich library : Provides many library files to develop a robust application.
- Pointer : we can directly interact with the memory using pointers.

4.Installation for the C language

- Vscode (software).
- MingW Compiler.

5.First C Program

```c
#include <stdio.h>

int main(){

printf("Hello C Language");

return 0;

}
```

- #include <stdio.h> : Standard input / output Header file
- **int** main(): Starting point of the Program and curly brackets ({ }) indicates the block of the code.
- printf("Hello C Language"); : printf() is used to print the formatted output to console and "hello C language" is the output.
- **return** 0; : Determines the state of the program which indicates the program has successfully executed.

6.printf() and scanf()

- printf() and scanf() are used for input and output in c language.
- Both are inbuilt library functions defined in the stdio.h header file.

6.1 printf() :

- printf() is used for output.
- syntax : printf("Message to print"); :this will print the given message to console.
- syntax : printf("format string",argument_list);
- format strings can be
  - %d : for integer
  - %f : for float
  - %c : char
- Arguments list : name of the variable to print on a console.
- Example : printf("a = %d ",a);
- Output : a = 10 (Suppose the a variables value is 10)

6.2 scanf() :

- scanf() is used for taking the values from the user.
- syntax : scanf("formatstring",&variableName);
- format strings can be
  - %d : for integer
  - %f : for float
  - %c : char etc.
- Arguments list : name of the variable with address of operator to scan the values from the user.
- Example : scanf("%d ",&a);

7.Variable in C Language

- Name given to the location in a memory which is used to hold the value.
- The value of the C variable may change in the program.
- C variables can hold values like integer, float point, character etc.
- E.g. Age, height etc.
- Syntax : Datatype Var_name;
- Example : int a ;

7.2 Rules to Declare the variable

- Variable names must start with a letter (Uppercase or lowercase or underscore).
- Space is not allowed.
- After the first character, variable names can contain letters, digits, or underscores.
- Variable names are case-sensitive.

- Certain words are reserved and cannot be used as variable names, as they are part of the C language syntax (e.g., "int", "for", "while", "if", etc.).

8.Data types

- Data type specifies the type of data hold by variable
- There are two types of data types in the C language.
    - Predefined Data type : int , char , float etc..
    - User defined : array structures etc.

9.Keywords

- Keywords are predefined words.

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

10.operators

- Arithmetic Operators : + , - , * , / , %
- Relational Operators : < , > , <= , >= , != , ==
- Logical operators : && , || , !
- Ternary operator : ?:

- Increment operator : ++
- Decrement operator : - -

11.Control Statements

- control statements are used to perform the operation based on specific conditions.
- control statement are :
  - if - else statement
  - if - else ladder
- if - else statement :
- Syntax :
  - **if (condition) {**
  - **executes if the condition is true ....**
  - **}**
  - **else {**
  - **executes if the condition is false ....**
  - **}**
- if the condition is true then it will execute the if block and if it is false then it will execute the false part.
- if - else - ladder
- syntax :
- if - else ladder can be used to check the multiple conditions
  - **if (condition) {**
  - **executes if the condition is true ....**
  - **}**
  - **else if (condition) {**
  - **executes if the condition is true....**
  - **}**

- **else if (condition) {**
- **executes if the condition is true....**
- **}.....**

12.Switch case Statements

- The switch statement in C is an alternative to if-else-if ladder statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable.
- In switch...case, expression is either an integer or a character.
- If the value of switch expression matches any of the constants in case, the relevant codes are executed and control moves out of the switch case statement.
- If the expression doesn't match any of the constants in case, then the default statement is executed.
- Syntax :
  - **switch**(expression){
  - **case** value1:
  - //code to be executed;
  - **break**; //optional
  - **case** value2:
  - //code to be executed;
  - **break**; //optional
  - .....
  - **default**:code to be executed **if** all cases are not matched;
  - }

13.Loops in C languages

- The looping can be defined as repeating the same process multiple times until the specific condition is satisfied.
- There are three types loops in C language
    - for loop
    - while loop
    - do - while loop
- for loop
    - **for (initialization ; condition ; inc / dec){**
    - **statements ……..**
    - **}**
    - continue till the specific condition is valid also called as entry control loop.
- While loop
    - **initialization ;**
    - **while(condition){**
    - **statements ……..**
    - **}**
    - continue till the specific condition is valid also called as entry control loop.
- do while loop
    - **initialization ;**
    - **do {**
    - **statements ……..**
    - **}while(condition);**
    - continue till the specific condition is valid, also called exit controlled loop.
- break : break is the keyword used to bring program control out of the loop.

- continue : continue is the keyword used to bring the program control to the beginning of the loop or skip the current step.

13.1 Function and call by value in C languages

- Function is the block of the code used to execute the task multiple times.
- functions provide reusability and modularity to the program.
- creating a function is called as call by value.
- Types of the function :
  - library function: printf() and scanf()
  - user defined function : created by user.
- Functions Aspects :
  - Function declaration : also called as function prototype
    - eg :**returnType funName(args1 ,args2 …… n);**
  - function call : calling the function
    - **functionName(args1 , args2…..n);**
  - function definition :
    - **returnType funName(args1 ,args2 …… n){**
    - **statements ……**
    - **}**
- Categories of the function based on return type and parameters list
  - Function with return type with parameter values
  - Function with return type without parameter values
  - Function without return type with parameter values

- Function without return type without parameter values.

14.1 Pointers in C languages

- pointers is the special variable in c which can store the address of another variable.
- we can create the pointer of any data type
- Syntax :
  - **int n = 10;**
  - **int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer**
- pointers are declared with asterisk mark (*).
- to store the address of the variable into the pointer use address of operator (&) such as **int* p = &n;**
- if we want to print the value of the variable pointed by pointer we need to use asterisk mark with pointer name such as *Pointer_name this is called dereferencing the pointer.

14.2 Pointer Arithmetic

- We can perform only few operations with pointers such as
- + , - , ++ , - - .
- Suppose the array is of type integer and starts with address 1000 then if we perform the addition then it will increase by 4 bytes which leads to the ans as 1004.

14.3 Double Pointers

- Double pointers is the pointer which can store the address of another pointer.
- Syntax :

- **int a = 10 ;**
- **int *ptr = &a ;// pointer**
- **int **dptr = &ptr; // double pointer**

15.Call by reference

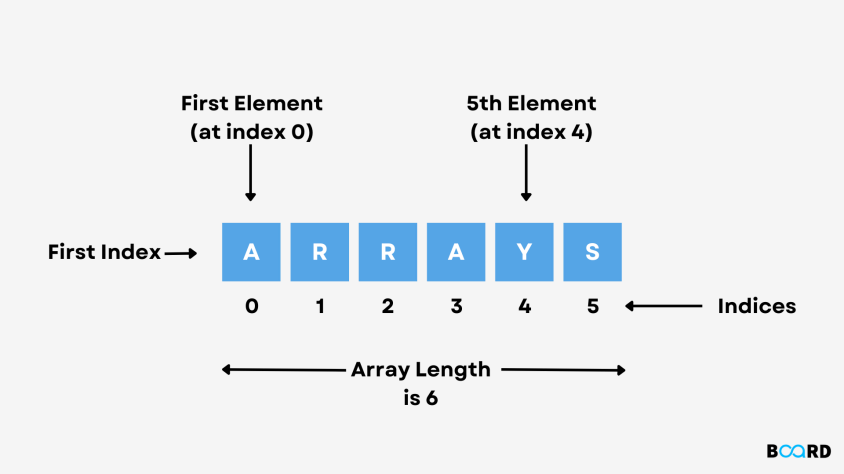
- We can return only one value from the function , to return more than one value from the function we need to pass the variable with address to function definition which is called call by reference.
- function declaration :
    - **returnType funName(daratype * a);**
- function call :
    - **funName(&VarName);**
- function definition :
    - **returnType funName(dataType *a){**
    - **statement ……**
    - **}**

16.Array in C languages

- Array is the collection of the same data type stored in a contiguous memory location.
- Array is a user defined data type.
- Array index starts with 0 to size - 1.
- values in the array called array elements.
- Advantages :
    - Code optimization
    - Ease of traversing
    - Ease of sorting

- Random access
- Disadvantage :
  - array has a fixed size.
- Array types
  - One Dimensional array
  - Multi Dimensional array

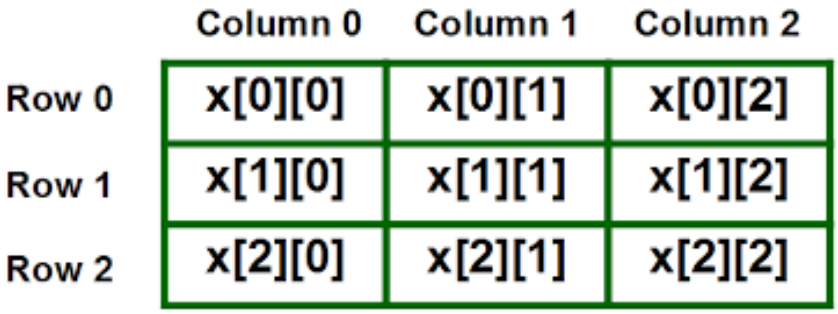
16.1.One Dimensional Array or 1D array

- Declaration : int arrayName[size] ;
- Initialization : int arrayName[size] = { 11 , 22 , 33 };
- array index starts from 0 till size - 1.
- 

16.2.Two Dimensional Array or 2D array

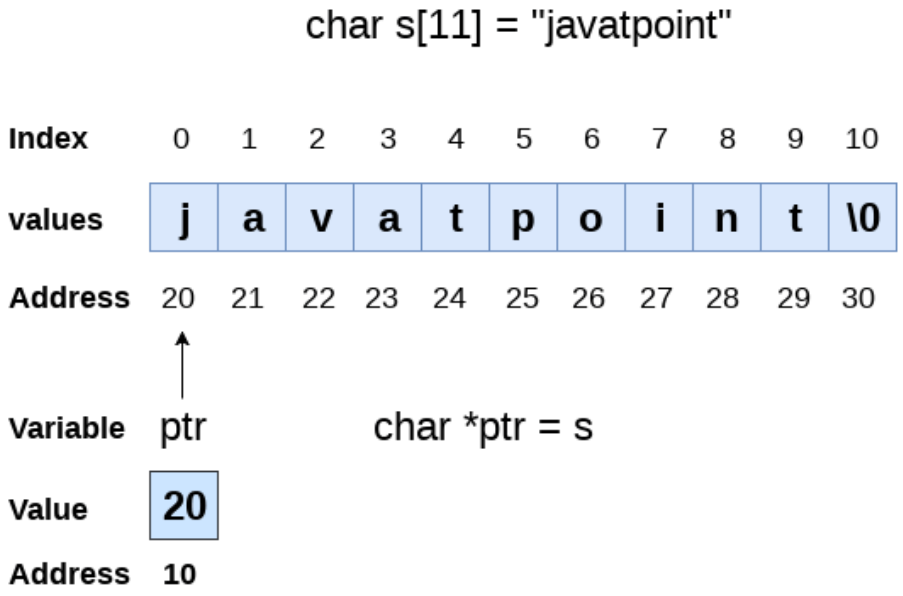
- 2D array is the collection of 1D arrays.
- a 2D array is organized as matrices which can be represented as rows and columns.
- Declaration :
  - **int arrayName[rowSize][colSize];**
- Initialization :
  - **int rr[3][3]={**

```
        {1,2,3},
        {2,3,4},
        {3,4,5}
    };
```



17.1.String in C languages

- Character array is nothing but a string terminated by null character (‘ \0 ‘).
- ‘\0’ is important to indicate the end of the string.
- Its index also starts with 0.
- two ways to declare the string.
    - BY character array
        - **char** ch[]={‘h’,’e’,’l’,’l’,’o’, '\0'};
    - by string literal
        - **char** ch[]= ‘hello’;



17.2.String functions

- C language provides some inbuilt string function.
- strlen(str) : returns the length of the string
- strcpy(str1 , str2) : str2 will copy copied into str1;
- strcat(str1 , str2) : to concat str2 to str1.
- strcmp(str1 , str2) : returns true if str1 and str2 are the same.

18.1 Dynamic Memory Allocation

- Memory can be allocated in two ways :
  - Compile time or static memory allocation
  - Run time or dynamic memory allocation

18.2.Limitation of static memory allocation

- Insufficient memory storage.
- No facility to release the memory once it has been allocated.

- Allocate memory on stack.

18.2.Dynamic Memory allocation

- The static memory reserves the memory at compile time , which cannot be altered during execution.
- letter on it allows us to adjust memory dynamically using four inbuilt functions as malloc() , calloc() , realloc() , free() defined in the stdlib.h header file.
- Static allocation uses the stack space while dynamic memory allocation uses head area to store.

18.3.malloc()

- it initializes the created block to garbage values.
- Syntax : **int *ptr = (int *)malloc(sizeof(int));**
- The program dynamically allocates 1 byte of memory sufficient for a char type and assigns it to ptr.

18.4.calloc()

- it initializes the created block to zero values.
- Syntax :**int *ptr =(int*)calloc(num, size-in-bytes);**

18.5.free()

- free function will deallocate the dynamically allocated memory.
- Syntax :**free(pointer_name);**

18.5.realloc()

- realloc will resize the memory block previously allocated with malloc or calloc.
- Syntax :**int *ptr =(int*)realloc(arr , num * size-in-bytes);**

19 Macros

- Macros in C programming are known as pieces of code which can be replaced by macro value.
- Suppose there is a piece of code which requires multiple times then we can use macro.
- defined with the help of #define preprocessor directive.
- Syntax :
  - #define MacroName ExpandableCode;
  - here : #define is the preprocessor directive
  - MacroName : Name of the macro
  - ExpandableCode : Expandable code
  - Example : #define PI 3.14
- The Expandable code is replaced with macro name before program compilation which is known as preprocessing.

20 Recursion

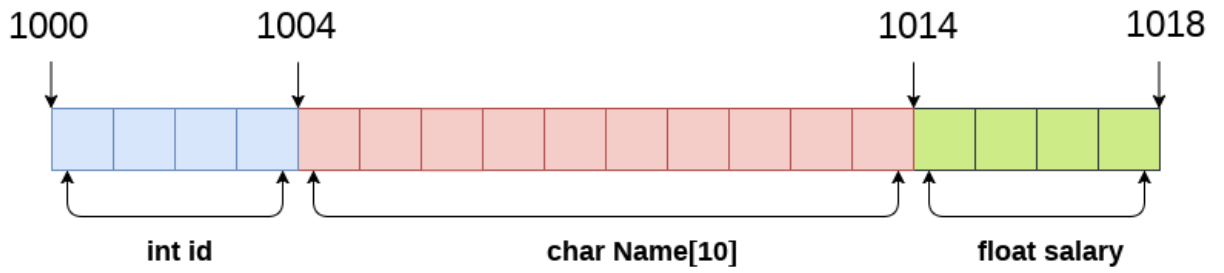
- Recursion in C is a programming technique where a function calls itself to solve a problem. It's based on the concept of breaking down a problem into smaller subproblems that can be solved more easily.
- Example :

```
#include <stdio.h>

// Function declaration
int recursiveFunction(int n);

int main() {
    int result;
    int num = 5; // Example input

    // Function call
    result = recursiveFunction(num);

    // Output the result
    printf("Result: %d\n", result);

    return 0;
}
// Recursive function definition
int recursiveFunction(int n) {
    // Base case
    if (n == 0 || n == 1) {
        return 1;
    }
    // Recursive case
```

```
    else {
        return n * recursiveFunction(n - 1);
    }
}
```

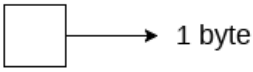
21 Structure in C Language

- Structure is the user defined data type which allows us to group the data of different types.
- To create the structure in C we use the struct keyword.
- Each element of the structure is called a member.
- Syntax :
    - **struct structure_name**
    - **{**
    - **data_type member1;**
    - **data_type member2;**
    - **.**
    - **.**
    - **data_type memeberName;**
    - **};**
- Memory allocation for the structure :
- Here
    - struct : keyword
    - Employee : Structure name
    - id , name and salary are the members of the structure.




```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

```
sizeof (emp)  =  4 + 10 + 4 = 18 bytes

where;
 sizeof (int) = 4 byte
 sizeof (char) = 1 byte
 sizeof (float) = 4 byte
```



- The Size of the structure variable is the addition of the total memory size occupied by each member of the structure.
- In the given image the size of the Employee is 18 bytes but it is going to give you the result as 20 bytes instead of 18 bytes , it will give you the total block size in a byte.
- Example :
    - if the structure size is 18 then result is 20 byte
    - if the structure size is 22 then result is 24 byte
    - if the structure size is 25 then result is 28 byte

21.2 Creating the Structure variable :

- Declaring the structure means creating the data type of the structure to access it we need to create the variable for the structure.
- We can create the structure variable in two ways
- Syntax to create the Structure variable :
- Way 1 :
    - **struct StructureName;**
    - Example:

- **struct Employee**
- **{ int id;**
- ** char name[50];**
- ** float salary;**
- **}; // Employee Structure**
- **struct Employee e1;// variable of a structure**

- The variable e1 is the can be used to access the value stored in the structure.
- Way 2 :
  - Example:
  - **struct Employee**
  - **{ int id;**
  - ** char name[50];**
  - ** float salary;**
  - **}e1; // Employee Structure with variable e1**

21.3 Accessing Structure variable :

- To access structure variables we need to use dot(.) operator with structure variable and member name.
- Example : for Employee Structure
  - **printf("Employee id : = %d \n",e1.id);**
  - **printf("Employee Name : = %s \n",e1.name);**
  - **printf("Employee Salary : = %d \n",e1.salary);**

21.4 typedef with Structure :

- We can rename any data type using typedef in C language.
- Syntax : **typedef <existing_name> <alias_name>**
- Example : **typedef struct Employee**
- **{ int id;**
- **char name[50];**
- **float salary;**
- **}emp;**
- After typedef we don't need to write the struct keyword along with the Structure variable , from here an out emp will be the another name for the structure.
- We can just write new name and variable name as above : **emp employee1;**

21.5 Pointer to Structure :

- Pointer variable that can store the address of a structure is called a pointer to structure.
- Syntax :
  - **struct Employee e1 ;**
  - **struct Employee *ptr = &e1 ;**
  - way to access members of structure using pointer
  - **pointerName->MemberName.**
- Arrow operator (->) is used with a pointer only.

22.1 Union in C Language

- union in C is the same as structure it is also used to store the collection of the data of different data type.
- union keyword is used for the declaration of the structure.
- Syntax :
    - **union unionName**
    - **{**
    - **data_type member1;**
    - **data_type member2;**
    - **.**
    - **data_type memeberName;**
    - **};**
- Variable creation syntax of the union is the same as structure.
- Syntax
    - **union Employee e1;// variable of a union**

22.2 The Difference between Structure and the Union :

Struct	Union
The struct keyword is used to define a structure.	The union keyword is used to define union.
When the variables are declared in a structure, the compiler allocates memory to each member. The size of a structure is equal or greater to the	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is

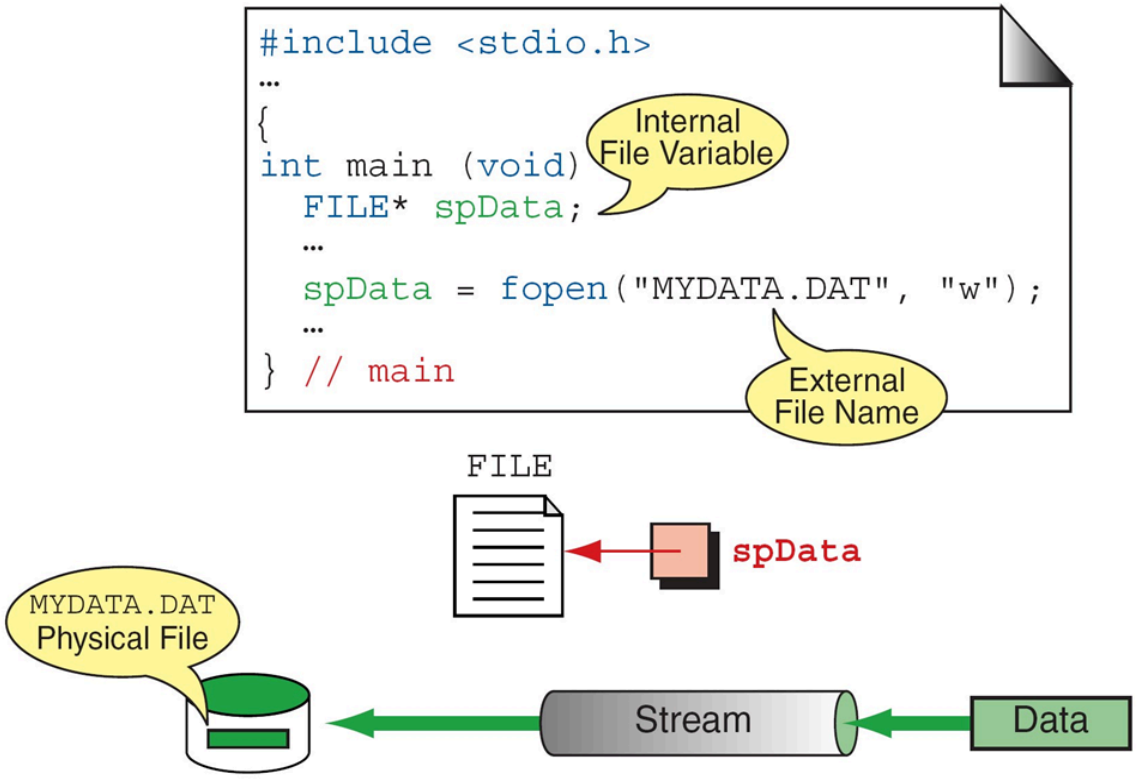
sum of the sizes of each data member.	equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variables members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables.
Each variable member will be assessed at a time.	Only one variable member will be assessed at a time.
We can initialize multiple variables of a structure at a time.	In union, only the first data member can be initialized.
All variable members store some value at any point in the program.	Exactly only one data member stores a value at any particular instance in the program.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It is used to store different data type values.	It is used for storing one at a time from different data type values.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time.

23.1 File Handling

- file is a sequence of bytes.
- C programming language provides access to high level functions as well as low level (OS level) calls to handle files on your storage devices.
- When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device.
- C File I/O provides following goals-
    - How to open a file to write to it.
    - How to open a file to read from it.
    - How to open a file to append data to it.
- A program requires several pieces of information about a file, including the name the OS uses for it, the position of the current character etc.
- C uses a structure called FILE to store the attributes of a file.
- FILE structure is defined in the stdio.h header file.
- To perform operations on files, first we have to create a pointer of FILE structure type.
- Steps :
    - Open the file using fopen() library function.
    - perform the operation.
    - Close the file.

23.2 FOPEN()

- The file open function (fopen) serves following purposes:
- ⁻ You can use the fopen( ) function to create a new file or to open an existing file.
- This call will initialize an object of the type FILE
- Syntax:**fopen("filename","mode");**



23.3 File Opening Modes

Mode	Description
r	• opens a text file in read mode • result is error if file is not existing
w	• opens a text file in write mode • file will be created if file does not exist. • file opened in blank mode if file is already existing
a	• opens a text file in append mode

23.4 fopen()

- Syntax : fopen("fileName.extension","fileOpeningMode");
- it will return null if the file cannot be opened.

23.5 fclose()

- Syntax : fclose(filePointerName);
- it will close the file.

23.5 fputc()

- Syntax : **int** fputc(**char** c, **FILE** *stream)
- prints one character into the file stream.

23.5 FGETC()

- Syntax : int fgetc(char c, FILE *stream)
- return one character from the file stream.

23.5 FGETS()

- Syntax :char* fgets(char *s, int n, FILE *stream)
- reads a line of character from a file.

23.5 FGETS()

- Syntax :char* fgets(char *s, int n, FILE *stream)
- reads a line of character from a file.

23.6 FPRINTF()

- Syntax :int fprintf(FILE *stream, const char *format [, argument, ...])
- The fprintf() function is used to write a set of characters into a file. It sends formatted output to a stream.

23.7 FSCANF()

- Syntax :int fscanf(FILE *stream, const char *format [, argument, ...])
- The fscanf() function is used to read a set of characters from a file. It reads a word from the file and returns EOF at the end of the file.

23.1 Data Structure

23.2 Stack

1. Definition:

- A stack is a linear data structure that follows the Last In, First Out (LIFO) principle.
- It allows access to only one data item at a time, which is the most recently added item.
- Operations on a stack include push (to add an item), pop (to remove the top item), and peek (to view the top item without removing it).

2. Implementation:

- Stacks can be implemented using arrays or linked lists.
- In array implementation, a fixed-size array is used to store the stack elements, and a pointer (often called "top") keeps track of the top element.
- In linked list implementation, each element of the stack is a node containing the data and a reference to the next node.

3. Operations:

- Push: Adds an item to the top of the stack.
- Pop: Removes the top item from the stack.
- Peek: Returns the top item without removing it.
- isEmpty: Checks if the stack is empty.
- isFull: Checks if the stack is full (for array implementation).

23.2 Queue

1. FIFO (First In, First Out): Queue follows the FIFO principle, meaning the element inserted first will be the one to be removed first.
2. Operations:
    - Enqueue: Adding an element to the rear of the queue.
    - Dequeue: Removing an element from the front of the queue.
    - Peek/Top: Viewing the element at the front without removing it.
    - isEmpty: Checking if the queue is empty.
    - Size: Determining the number of elements in the queue.
3. Implementation: Queue can be implemented using arrays, linked lists, or other data structures. Arrays might have limitations on dynamic resizing, while linked lists offer flexibility but consume more memory.
4. Applications:
    - Used in breadth-first search (BFS) algorithms.
    - Managing processes in operating systems.
    - Print queue management in computer systems.
    - Task scheduling in computer systems.
    - Simulation of real-world queues, like lines in stores or banks.

23.3 Link List

1. Data Structure: A linked list is a linear data structure where elements are stored in nodes. Each node contains a data field and a reference (or pointer) to the next node in the sequence.
2. Types of Linked Lists:
    - Singly Linked List: Each node points to the next node in the sequence.

- Doubly Linked List: Each node has pointers to both the next and the previous nodes.
- Circular Linked List: Last node's pointer points back to the first node, forming a circular structure.

3. Dynamic Memory Allocation: Linked lists use dynamic memory allocation, allowing them to grow or shrink in size during program execution.
4. Operations:
    - Insertion: Adding a new node to the list.
    - Deletion: Removing a node from the list.
    - Traversal: Accessing each node in the list sequentially.
    - Search: Finding a specific element in the list.
    - Update: Modifying the value of a node.
5. Advantages:
    - Dynamic size: Linked lists can grow or shrink in size dynamically, unlike arrays, which have fixed sizes.
    - Efficient Insertions and Deletions: Insertion and deletion operations can be performed in O(1) time complexity if the position is known.
    - Memory Utilization: Memory is allocated as needed, reducing wastage compared to fixed-size arrays.
6. Disadvantages:
    - No Random Access: Unlike arrays, elements in a linked list cannot be accessed randomly in constant time. Traversal is required to reach a specific element.
    - Extra Memory Usage: Each node in a linked list requires additional memory to store the pointer/reference to the next node.
    - Overhead: Linked lists have some overhead due to the pointers/references.
7. Applications:
    - Dynamic memory allocation, e.g., in memory allocation algorithms.
    - Implementation of stacks, queues, and other data structures.
    - Representation of sparse matrices.
    - Implementation of adjacency lists in graphs.

23.3 Doubly Link List

1. Structure: A doubly linked list is a type of linked list where each node has pointers to both the next and the previous nodes, forming a bidirectional sequence.
2. Node Structure: Each node in a doubly linked list typically contains three fields:
    - Data: Holds the value or payload of the node.
    - Next Pointer: Points to the next node in the sequence.
    - Previous Pointer: Points to the previous node in the sequence.
3. Traversal: In addition to forward traversal (from the head to the tail), doubly linked lists support backward traversal (from the tail to the head) due to the presence of previous pointers.
4. Insertion and Deletion:
    - Insertion: New nodes can be inserted at the beginning, end, or at any position within the list. Insertions can be performed efficiently in O(1) time complexity.
    - Deletion: Nodes can be removed from any position in the list. Deletions also have O(1) time complexity if the position is known.
5. Advantages:
    - Bidirectional traversal: Doubly linked lists allow efficient backward traversal, which is not possible in singly linked lists without additional effort.
    - Easy deletion of nodes: Deletion of a node from a doubly linked list is easier compared to a singly linked list because the previous node's pointer can be directly updated.
    - Implementation of data structures: Doubly linked lists are often used to implement other data structures like stacks, queues, deques, and associative arrays.

6. Disadvantages:
    - Additional memory overhead: Each node in a doubly linked list requires extra memory to store both next and previous pointers, increasing memory usage compared to singly linked lists.
    - More complex implementation: Managing both forward and backward pointers adds complexity to the implementation and may require more careful handling, especially during insertions and deletions.
7. Applications:
    - Browser history: Doubly linked lists are used to implement backward and forward navigation in web browsers.
    - Undo functionality: They are utilized in implementing undo operations in text editors or graphic design software.
    - Cache eviction policies: Doubly linked lists are used in implementing cache eviction policies like LRU (Least Recently Used) and LFU (Least Frequently Used).