

Name of the student	SHASHIDHAR KUMBAR
Roll No	133
USN	01FE22BEV039
Div	VLSI

Part- A

1.Variables and Data Types

1.What is the difference between a variable and a data type in C programming? Provide examples to illustrate.

Ans. Data type: Specifies the kind of information (characters, integers, etc.) that a variable can store.

A variable is a designated storage place that, depending on the kind of data, has a particular value (e.g., an integer to record age).

2.What is the difference between a variable and a data type in C programming? Provide examples to illustrate.

Ans. Size: Allotted memory, such as an int, which is normally 4 bytes.

Value range: Permitted values (integer numbers within a given range, for example, int).

Operations: Allowed activities (mathematical operations for numeric types, for example).

Categories:

Fundamental:

int :whole numbers (like -5 and 10).

float: Decimal numbers (such as 3.14 and -1.25).

char: A single character, such as "a" or "Z."

3. How are variables declared and initialized in C programming? Provide examples of variable declarations with different data types.

Ans. Declaration:

Specifies the data type and creates a named space in memory.

Syntax: data type variable name;

Initialization :

Assigns an initial value during declaration.

Syntax: data type variable name = value;

Examples:

int age; (integer variable)

float pi = 3.14; (float variable with initial value)

char initial = 'A'; (character variable)

4. Discuss the scope and lifetime of variables in C programming. What are global and local variables?

Ans. Scope defines the section of the code that contains a variable's accessibility.

There are two primary types:

Local: Excluding a block (code enclosed in curly brackets {}).

Global: Available at all times during the program.

Duration: Relates to the duration of memory allotted to the variable.

Local variables: Only present when a block is being executed.

Global variables: Last till the end of the program.

Local: Used to avoid name conflicts within particular functions.

Global: Due to possible unforeseen negative effects throughout the program, it should only be utilized seldom.

5. Explain the concept of type casting in C programming. When is type casting necessary, and how is it performed?

Ans. Casting a value explicitly from one data type to another is known as type casting.

Required at:

transferring values between incompatible types (such as float and integer).

using routines designed to handle particular data types.

Cast operator (data type) was used before the value.

age = 25 int; (float)age = float pi;

2. OPERATORS:

1. Describe the purpose and usage of the ternary conditional operator (?:) in C programming. Provide an example demonstrating its usage.

Ans. When evaluating a condition in C, the ternary conditional operator?: returns one of two values according to whether the condition is true or false. It is frequently employed as a condensed version of an if-else statement.

Eg: int x = 10;

int y = (x > 5) ? 100 : 200

printf("Value of y: %d\n", y);

2. Discuss the bitwise operators available in C programming. Explain their usage with suitable examples.

Ans.

i]Bitwise operators :These operators work with specific bits that are contained in a variable. The bitwise operators that C offers are as follows:

ii]Bitwise AND (&):

Syntax: operand1 & operand2

Performs a bitwise AND operation on corresponding bits of operand1 and operand2.

Example: int result = 5 & 3;

iii]Bitwise OR (|):

Syntax: operand1 | operand2

Performs a bitwise OR operation on corresponding bits of operand1 and operand2.

Example: int result = 5 | 3;

iv]Bitwise XOR (^):

Syntax: operand1 ^ operand2

Performs a bitwise XOR (exclusive OR) operation on corresponding bits of operand1 and operand2.

Example: int result = 5 ^ 3;

v]Bitwise NOT (~):

Syntax: ~operand

Performs a bitwise NOT (complement) operation, which flips all the bits of operand.

Example: int result = ~5;

3. Explain the difference between the postfix and prefix increment operators (++) in C programming. Provide examples to illustrate.

Ans.

i]Operator for Postfix Increment (x++):

The variable's current value is returned first in the postfix increment operator, and the value is subsequently increased.

For instance:

```
int x = 5;
```

```
int y = x++;
```

```
printf("x: %d, y: %d\n", x, y);
```

ii] Operator for Prefix Increment (++x):

The variable's value is increased first via the prefix increment operator, and the increased value is then returned.

For instance:

```
int x = 5;  
int y = ++x;  
printf("x: %d, y: %d\n", x, y);
```

4. What is the significance of the logical AND (&&) and logical OR (||) operators in C programming? How are they used in conditional expressions?

Ans.

i] Logical AND (&&):

The logical AND operator returns true (1) if both of its operands are true, otherwise it returns false (0).

```
int x = 5;  
int y = 10;  
if (x > 0 && y > 0)  
    printf("Both x and y are positive\n");
```

ii] Logical OR (||):

The logical OR operator returns true (1) if at least one of its operands is true, otherwise it returns false (0).

```
int x = 5;  
int y = 10;  
if (x == 0 || y == 0)  
    printf("Either x or y is zero\n");
```

5. Discuss the concept of operator precedence and associativity in C programming. Provide examples to demonstrate how they affect expression evaluation

Ans.

i] Operator Order of Priority:

The order in which operators are evaluated in an expression is specified by operator precedence. Higher precedence operators are evaluated before lower precedence operators.

For instance, multiplication (*) is evaluated as $2 + (3 * 4)$ since it has a greater precedence than addition (+). This yields a value of 14.

To force evaluation in a specific sequence and override the default

precedence, use parentheses ().
For instance: `int result = 2 + 3 * 4;`

li]Associativity of Operators:

The sequence in which operators with the same precedence are evaluated is specified by operator associativity. It can be associative from right to left or left to right, or left to right or left to left.

For instance, because the addition operator (+) is associative from left to right, `2 + 3 + 4` is evaluated as `(2 + 3) + 4`, which yields 9.

The assignment (=) operator is associative from right to left, though, thus `a = b = 5` is evaluated as `a = (b = 5)`, which assigns the value 5 to both a and b.

For instance: `int result1 = 2 + 3 + 4;`

`int a, b;`

`a = b = 5;`

3.CONTROL STRUCTURES:

1.Describe the purpose and usage of the switch statement in C programming. How does it differ from the if-else statement?

Ans The switch statement is used to regulate the flow of execution according to a variable or expression's value. It offers a different approach to writing several if-else statements that examine the same variable's value.

When several conditional branches are created based on the value of a single variable, the switch statement makes the code clearer and more concise.

It offers a more effective substitute for several if-else statements, particularly in situations when there are a lot of potential values.

Switch statement differs from if else in manner:

The switch statement compares the value of a single expression against multiple constant values, while the if-else statement evaluates different expressions or conditions independently.

2. Explain the concept of nested control structures in C programming. Provide an example demonstrating nested if-else statements.

Ans: The term "nested control structures" describes the use of control structures (like if-else, switch, while, for, etc.) inside other control

structures. This makes it possible to develop looping and decision-making structures that are more intricate.

Eg:

```
{  
    int x = 10;  
    int y = 20;  
    if (x > 0) {  
        if (y > 0) {  
            printf("Both x and y are positive.\n");  
        } else {  
            printf("x is positive, but y is non-positive.\n");  
        }  
    } else {  
        printf("x is non-positive.\n");  
    }  
}
```

3. Discuss the role of the break and continue statements in loop control in C programming. Provide examples to illustrate their usage.

Ans:] Statement of Break:

An early break in a loop's execution can be achieved with the break statement.

A loop is instantly broken when a break statement is met, and control is moved to the sentence that comes after the loop.

Eg:

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        break; // Exit the loop when i equals 5  
    }  
    printf("%d ", i);  
}
```

ii] Statement Continued:

To move on to the following iteration of a loop and skip the remainder of the current iteration, use the continue statement.

Control returns to the loop's condition or increment/decrement statement upon encountering a continue statement, skipping the remaining statements in the loop body.

Eg:

```
for (int i = 0; i < 10; i++) {  
    if (i % 2 == 0) {
```

```
        continue; // Skip even numbers
    }
    printf("%d ", i);
}
```

4. What are the advantages of using the for loop over the while loop in C programming? Provide examples comparing the two.

Ans: One Location for Initialization, Condition, and Increment:

The for loop makes the loop structure more clear-cut and understandable by allowing you to declare the increment/decrement, the loop condition, and the loop variables all on one line.

In particular, for straightforward loops with distinct setup, condition, and increment/decrement logic, this might result in clearer, more understandable code.

More Suitable for Looping Through Ranges:

When iterating over a range of data, like iterating over an array or completing a predetermined amount of iterations, the for loop is frequently utilized.

Because of its design, it works well in situations where there is a need for the loop control variable to be initialized, tested against a condition, and updated in a predictable way.

Eg:

```
// Example using for loop
printf("Using for loop:\n");
for (int i = 0; i < 5; i++) {
    printf("%d ", i);
}
printf("\n");
```

```
// Example using while loop
printf("Using while loop:\n");
int j = 0;
while (j < 5) {
    printf("%d ", j);
    j++;
}
```

5. Explain the concept of short-circuit evaluation in C programming. How does it affect the evaluation of logical expressions in if statements?

Ans: The idea of short-circuit evaluation states that a logical expression's evaluation should end as soon as its result can be ascertained, without considering every component of the expression. This implies that the remaining portions of the expression are not examined if the result of the expression can be ascertained by evaluating only a piece of it.

Eg:

If the left operand of && is evaluated to false, the entire expression is false, and the right operand is not evaluated because the result is already determined. This is because for the entire expression to be true, both operands must be true.

Ex: if (x > 0 && y > 0)

4.FUNCTIONS

1. Describe the purpose and structure of a function prototype in C programming. Why is it necessary to declare function prototypes?

Ans: A function prototype (also known as a function declaration) serves as a forward declaration of a function before its actual implementation. It provides essential information about the function to the compiler, including its name, return type, and the types of parameters it accepts. The function prototype has the following structure:

```
:: return_type function_name(parameter1_type, parameter2_type, ...);
```

Purpose of Function Prototypes:

Compiler Communication: Function prototypes inform the compiler about the existence and signature (return type and parameter types) of functions before they are called. This allows the compiler to perform type checking and generate appropriate code for function calls.

Preventing Implicit Declarations: Without function prototypes, if a function is called before its definition or without its signature being known, the compiler assumes an implicit declaration, which defaults to returning an integer. This can lead to compilation errors or unexpected behavior if the actual function definition differs from the implicit declaration.

2. Explain the difference between call by value and call by reference in C programming. Provide examples to illustrate both concepts.

Ans: Call by Value:

In call by value, the value of the actual argument (or variable) is copied into the formal parameter of the function. Modifications made to the formal parameter inside the function do not affect the original argument outside the function.


```
#include <stdio.h>
void increment(int *x) {
    printf("Inside function: %d\n", *x);
}

int main() {
    int num = 10;

    printf("Before function call: %d\n", num);
    increment(&num);
    printf("After function call: %d\n", num);
}
```

Call by Reference:

In call by reference, instead of passing the value of the variable, the memory address (pointer) of the variable is passed to the function. This allows the function to directly access and modify the original variable in memory.

```
#include <stdio.h>
void increment(int *x) {
    (*x)++;
    printf("Inside function: %d\n", *x);
}

int main() {
    int num = 10;
    printf("Before function call: %d\n", num);
    increment(&num);
    printf("After function call: %d\n", num);
}
```

3. Discuss the concept of recursion in C programming. Provide an example of a recursive function and explain how it works.

Ans: Recursion in C programming is a technique where a function calls itself directly or indirectly to solve a problem. It is a powerful and elegant way of solving problems that can be naturally expressed in terms of smaller instances of the same problem. Recursion consists of two main components: a base case (or termination condition) and a recursive case.

Base Case: The base case is a condition that specifies when the recursion should stop. It provides a termination point for the recursive calls, preventing infinite recursion. Without a base case, the recursive function would continue calling itself indefinitely, leading to a stack overflow.

Recursive Case: The recursive case is where the function calls itself with modified arguments to solve a smaller instance of the same problem. Each recursive call brings the problem closer to the base case until the base case is reached and the recursion stops.

CODE

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    // Recursive case: n! = n * (n-1)!
    else {
        return n * factorial(n - 1);
    }
}
int main() {
    int n = 5;
    int result = factorial(n);

    printf("Factorial of %d is: %d\n", n, result); // Output: Factorial of 5 is: 120

    return 0;
}
```

4. What is the significance of the return statement in C programming? How are values returned from functions?

Ans: Passing Results: Functions often perform computations or operations, and the return statement allows them to pass the result of these operations back to the calling code. Function Termination: When a return statement is encountered, the function execution is immediately terminated, and control is transferred back to the caller.

Error Handling: Functions can use the return statement to indicate errors or exceptional conditions by returning special error codes or values. Exit Status: In the case of main() function, the `

5. Describe the role of function parameters and arguments in C programming. How are function arguments passed to parameters?

Ans: Role of Function Parameters:

Customization: Parameters allow functions to accept input values, enabling them to perform tasks with different data values.

Flexibility: By accepting parameters, functions can be reused with various inputs, enhancing code modularity and maintainability.

Abstraction: Parameters abstract the details of data manipulation within a function, providing a higher level of abstraction to the calling code.

Role of Function Arguments:

Data Input: Function arguments are the actual values provided to a function when it is called. They represent the data that the function will operate on.

Customization: By passing different arguments to a function, you can customize its behavior and achieve different outcomes.

Execution: Arguments drive the execution of the function, influencing the actions performed by the function's code.

Pass by Value

Pass by Reference

5.ARRAYS:

1.Explain the concept of arrays in C programming. How are arrays declared and initialized?

Ans. Declaration: To declare an array, we specify the data type of the elements it will hold and the size of the array. It follows the syntax: data type array name [array size];

Initialization: Arrays can be initialized during declaration or later in the program. Initialization assigns initial values to the elements of the array. There are two ways to initialize arrays:

```
int numbers[5] = {1, 2, 3, 4, 5}
```

2.Discuss the difference between a one-dimensional array and a multi dimensional array in C programming. Provide examples of both.

Ans. One-Dimensional Array (1D): A one-dimensional array is a collection of elements stored in a contiguous memory block.

It's like a list of items where each item is accessed using a single index.

Declared with a single pair of square brackets [].

Declaration and initialization of a 1D array

```
int numbers[5] = {1, 2, 3, 4, 5}
```

```
char vowels[5] = {'a', 'e', 'i', 'o', 'u'}
```

Multi-Dimensional Array: A multi-dimensional array is an array of arrays, where each element of the array is itself an array.

It's like a table or grid, where data is arranged in rows and columns.

Declared with multiple pairs of square brackets [], indicating the dimensions.

Declaration and initialization of a 2D array (3x3)

```
int matrix[3][3] = {1, 2, 3},{4, 5, 6},{7, 8, 9}
```

3. Describe the process of accessing array elements in C programming.

How are array indices used to access elements?

Ans. Arrays are often accessed using loops, such as for or while, to iterate through each element sequentially.

Normally we use for Loop

```
for (int i = 0; i < 5; i++)
```

```
    printf("%d ", numbers[i]);
```

4. What is the significance of the null character ('\0') in C strings? How is it used to determine the end of a string?

Ans. Strings are represented as arrays of characters terminated by a null character '\0'.

The null character marks the end of the string, indicating where the string's contents conclude.

It's essential for string manipulation functions to determine the string's length and to prevent buffer overflows when working with strings.

5. Explain the concept of dynamic memory allocation for arrays in C programming. How are dynamic arrays allocated and deallocated?

Ans. Dynamic memory allocation in C programming refers to the process of allocating memory during program execution rather than at compile time. This allows for flexibility in memory usage, especially when the size of data structures like arrays cannot be determined beforehand.

Allocation using malloc(), calloc(), realloc()

Deallocation using free()

6. POINTERS:

1. Describe the purpose and usage of pointers in C programming. How are pointers declared and initialized?

Ans. Pointers in C programming are variables that store memory addresses. They provide a way to indirectly access the memory location of a variable, allowing for dynamic memory allocation, efficient manipulation of data structures, and interaction with hardware.

Dynamic Memory Allocation

Efficient data structure.

2. Explain the concept of pointer arithmetic in C programming. Provide examples to illustrate addition and subtraction operations on pointers.

Ans. Pointer arithmetic in C programming allows you to perform arithmetic operations directly on pointers. This is particularly useful when working with arrays, as it provides a convenient way to navigate through elements in memory.

Addition Operation

Subtraction Operation

Increment and Decrement Operation

3. Discuss the difference between pass by value and pass by reference in function arguments using pointers in C programming. Provide examples to illustrate both approaches.

Pass by value: In pass by value, a copy of the variable's value is passed to the function. Any changes made to the parameter within the function do not affect the original variable.

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

```
void increment(int num)
```

```
    num++;
```

```
int main()
```

```
{
```

```
    int num = 10;
```

```
    increment(num);
```

```
    printf("Original value of num: %d\n", num);
```

```
    return 0;
```

```
}
```

Pass by Reference using Pointers: In pass by reference, the memory address of the variable is passed to the function using a pointer. Any changes made to the parameter within the function affect the original variable.

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

```
void incrementByRef(int *ptr)
```

```
    (*ptr)++;
```

```
int main()
```

```
{
```

```
    int num = 10;
```

```
    incrementByRef(&num);
```

```
    printf("Modified value of num: %d\n", num);
```

```
    return 0;
```

}

4. Describe the concept of NULL pointers in C programming. How are NULL pointers used and checked for in programs?

Ans. NULL pointer is a pointer that does not point to any memory location. It's a special value assigned to pointers to indicate that they are not currently pointing to a valid object or memory address. The concept of NULL pointers is essential for error handling and indicating the absence of a valid reference.

5. Explain the role of pointers in dynamic memory allocation in C programming. How are pointers used to allocate and deallocate memory dynamically?

Ans. Pointers play a crucial role in dynamic memory allocation in C programming. They are used to allocate and deallocate memory dynamically, allowing programs to request memory from the heap at runtime.

Allocation of memory

Deallocation of memory

Reallocation of memory

7.STRINGS:

1. Discuss the concept of strings in C programming. How are strings represented and manipulated in C?

Ans. Strings are sequences of characters terminated by a null character '\0'. C does not have a built-in string data type like some other programming languages, so strings are typically represented as arrays of characters.

1. String Representation:

Strings are represented as arrays of characters, where each character is stored in a consecutive memory location. The last character of the string is always the null character '\0', which marks the end of the string.

2. String Manipulation: C provides a library of string manipulation functions declared in the <string.h> header, such as strcpy(), strcat(), strlen(), strcmp(), etc. These functions allow you to perform various operations on strings like copying, concatenating, finding the length, and comparing strings.

2. Explain the difference between character arrays and string literals in C programming. Provide examples to illustrate both concepts.

Ans.

1. Character Arrays:

Character arrays are arrays of characters that can be explicitly declared and manipulated by the programmer.

They are mutable, meaning their contents can be modified after declaration.

Character arrays are terminated by a null character '\0' to mark the end of the string.

2. String Literals:

String literals are sequences of characters enclosed in double quotes. They are automatically null-terminated by the compiler, so you don't need to include the null character explicitly. String literals are immutable, meaning their contents cannot be modified after compilation.

3. Describe common string manipulation functions available in the C standard library. Provide examples of functions like strlen, strcpy, strcat, and strcmp.

Ans. The C standard library provides several string manipulation functions declared in the <string.h> header.

strlen(): size_t strlen(const char *str): Returns the length of the null-terminated string str.

strcpy(): char *strcpy(char *dest, const char *src): Copies the null-terminated string src to dest.

strcat(): char *strcat(char *dest, const char *src): Concatenates the null-terminated string src to the end of dest.

strcmp(): int strcmp(const char *str1, const char *str2): Compares the null-terminated strings str1 and str2.

8.STRUCTURE AND UNIONS:

1. Describe the purpose and usage of structures in C programming. How are structures declared and accessed?

Purpose and Usage of Structures:

*Organizing Data: Structures allow you to organize related data elements into a single unit, making it easier to manage and manipulate data.

*Creating Custom Data Types: Structures enable you to define custom data types that represent real-world entities with multiple properties.

*Passing Complex Data to Functions: Structures are often used to pass complex data to functions by encapsulating related variables into a single parameter.

*Memory Allocation: Structures facilitate memory allocation by providing a contiguous block of memory for all its members.

Declaration:

Structs are declared using the struct keyword followed by the name of the structure and a list of member variables enclosed in curly braces {}.

```
struct Student {  
    int id;  
    char name[50];  
    float gpa;  
};
```

Accessing Members:

Individual members of a structure can be accessed using the dot (.) operator.

```
printf("ID: %d\n", s1.id);  
printf("Name: %s\n", s1.name);  
printf("GPA: %.2f\n", s1.gpa);
```

2. Discuss the concept of structure members in C programming. How are individual members of a structure accessed and modified?

A. Concept of Structure Members:

Data Type: Each member of a structure can have a different data type, including basic data types (int, float, char, etc.), arrays, pointers, or even other structures.

Name: Structure members are identified by their unique names, which are specified when the structure is defined.

Memory Layout: Structure members are stored in contiguous memory locations, and their order in memory is the same as their declaration order within the structure.

Accessing and Modifying Structure Members:

Accessing Members:

Individual members of a structure are accessed using the dot (.) operator followed by the member name.

```
struct Point {  
    int x;  
    int y;  
};  
struct Point p1 = {3, 5};
```



```
printf("X coordinate: %d\n", p1.x);  
printf("Y coordinate: %d\n", p1.y);
```

3.Explain the difference between structures and unions in C programming.
When would you choose one over the other?

Structures:

Memory Allocation:

Each member of a structure is allocated its own memory space.

The total memory allocated for a structure is the sum of the memory allocated for each member.

Usage:

Structures are used when you need to store and manage multiple related data elements simultaneously.

Each member of a structure holds different types of data, and all members are accessible simultaneously.

Choosing Between Structures and Unions:

Choose structures when you need to store and access multiple related data elements simultaneously.

Choose unions when you need to conserve memory and only one type of data is needed at any given time.

4.Discuss the concept of typedef in C programming. How is typedef used to define custom data types, including structures and unions?

Ans. typedef is a keyword used to create aliases for existing data types. It allows programmers to create their own names for data types that already exist, making the code more readable and understandable. The typedef declaration does not create a new type; rather, it creates a new name for an existing type.

Syntax:

```
typedef existing_data_type new_type_name;
```

Custom Datatypes:

```
typedef int myInt;
```

```
typedef float myFloat;
```

Custom structure:

```
typedef struct {  
    int age;  
    char name[20];  
}
```

Custom Unions:

```
typedef union {
```

```
int intValue;  
float floatValue;  
}
```

9.FILE HANDLING

1.Explain the concept of file handling in C programming. How are files opened, read from, and written to using standard file handling functions?

Ans: It allows the programmer to perform various operations such as opening, reading from, writing to, and closing files. In C, file handling is accomplished using standard file handling functions provided by the C standard library (stdio.h).

1. Opening a File:

To open a file, you typically use the `fopen()` function, which takes two parameters: the file name (including the path if necessary) and the mode in which you want to open the file (read, write, append, etc.).

2. Reading from a File:

To read from a file, you can use functions like `fscanf()` or `fgets()` to read formatted data or lines of text respectively. Alternatively, you can use `fread()` to read binary data.

3. Writing to a File:

To write to a file, you can use functions like `fprintf()` for writing formatted data, or `fputs()` for writing strings, or `fwrite()` for writing binary data.

4. 4. Closing a File:

It's important to close a file after you finish working with it. This frees up system resources and ensures that any buffered data is written to the file.

3.Describe the role of file pointers in C programming. How are file pointers used to navigate and manipulate files?

Ans: file pointers play a crucial role in file handling operations. A file pointer is a special type of pointer variable used to keep track of the current position within a file during input/output operations. It points to the location of the next byte to be read from or written to in the file. File pointers are essential for navigating and manipulating files in C.

Opening File

Moving the File pointer

Reading from File

Writing To File

Closing the File.

3. Discuss the difference between text files and binary files in C programming. How are they opened and processed differently?

Ans: 1. Text Files:

Human Readable: Text files contain data that is human-readable, typically consisting of characters and lines of text.

Character Encoding: Text files are encoded using a character encoding scheme like ASCII or UTF-8, where each character corresponds to a specific byte value.

Newline Characters: Text files often use newline characters (\n) to represent the end of a line.

2. Binary Files:

Machine Readable: Binary files contain data in a format that is not directly human-readable. They can store any type of data, including numbers, structures, and raw binary data.

No Character Encoding: Binary files do not rely on character encoding schemes and can contain any sequence of bytes, including null characters.

No Newline Characters: Binary files may not have newline characters, and the data within them may not be organized into lines.

Opening and Processing Differences:

Opening: Text files are usually opened in text mode ("r", "w", "a", etc.) using functions like fopen(). Binary files can be opened in either text mode or binary mode ("rb", "wb", "ab", etc.). It's important to specify binary mode when working with binary files to prevent any special treatment of newline characters.

Processing: When reading from or writing to a text file, the data is typically processed using functions like fgets(), fscanf(), fprintf(), etc., which handle text input/output. For binary files, functions like fread() and fwrite() are commonly used to read and write blocks of binary data directly.

4. Explain the purpose of file modes in C programming. Provide examples of different file modes like "r", "w", "a", etc.

Ans: "r" (Read Mode):

Opens the file for reading only.

fopen("Text.txt","r");

"w" (Write Mode):

Opens the file for writing only.

If the file does not exist, it creates a new file. If the file exists, it truncates the file to zero length.

```
fopen("Text.txt","w");
```

"a" (Append Mode):

Opens the file for appending data to the end of the file.

If the file does not exist, it creates a new file.

The file pointer is positioned at the end of the file, allowing new data to be appended.

```
fopen("Text.txt","a");
```

"r+" (Read/Write Mode):

Opens the file for both reading and writing.

```
fopen("Text.txt","r+");
```

"w+" (Read/Write Mode):

Opens the file for reading and writing.

If the file exists, it truncates the file to zero length; otherwise, it creates a new file.

```
fopen("Text.txt","w+");
```

"a+" (Append/Read Mode):

Opens the file for reading and appending

```
fopen("Text.txt","a+");
```

5. Describe error handling techniques in file operations in C programming. How are errors detected and handled when working with files?

Ans: Always check if the file opening operation was successful by comparing the file pointer returned by fopen() to NULL. If it's NULL, it means the file couldn't be opened for some reason (e.g., file not found, insufficient permissions, etc.), and appropriate error handling should be done.

ERROR check:

```
if (fptr == NULL)
{
    printf("File not available.");
    exit(0);
}
```

Part- B

1.PRINT “HELLO WORLD”

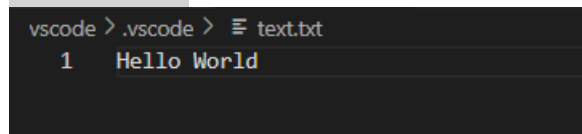
```
#include<stdio.h>
#include<stdlib.h>

int main()
{
    FILE *fptr;

    fptr=fopen("Text.txt","w");           -----O(1)

    fputs("Hello World" ,fptr);           -----O(1)
    fclose(fptr);                         -----O(1)
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1  Hello World
```

TIME COMPLEXITY

For this particular code the total time complexity is given by $O(1)$.
($O(1)$, $O(1)$, $O(1)$)

2. FACTORIAL OF THE NUMBER

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

int main()
{
    FILE *fptr;
    int num;
    long long int mul=1;

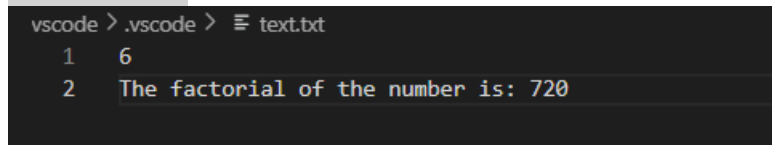
    fptr=fopen("text.txt","w");           -----O(1)
    fprintf(fptr,"%d",rand()%12+1);
    fclose(fptr);
    fptr=fopen("text.txt","r");           -----O(1)

    while(fscanf(fptr,"%d",&num)!=EOF);
    fclose(fptr);

    for(int i=num;i>0;i--)                -----O(num)
    {
        mul=mul*i;
    }

    fptr=fopen("text.txt","a");           -----O(1)
    fputs("\nThe factorial of the number is: ",fptr);
    fprintf(fptr,"%lld",mul);
    fclose(fptr);                         -----O(1)
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 6
2 The factorial of the number is: 720
```

TIME COMPLEXITY

For this particular code the total time complexity is given by $O(\text{num})$.
($O(1)$, $O(1)$, $O(\text{num})$, $O(1)$, $O(1)$)

3.PRIME NUMBERS

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

int prime1(int n)                                -----O(n*sqrt(n))
{
    for(int i=2;i<sqrt(n);i++)
    {
        if(n%i==0)
            return 0;
    }
    return 1;
}

int main()
{
    int arr[10],k=0, num;
    FILE *fptr;

    randonGenrator(20);                          -----O(n)
    fptr = fopen("text.txt", "r");                -----O(1)

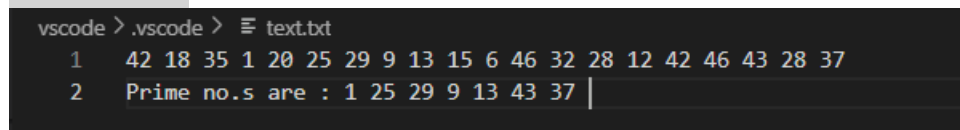
    if (fptr == NULL)
    {
        printf("File not available.");
        exit(0);
    }
    while(fscanf(fptr, "%d", &num) != -1)         -----O(n)
    {
        if(prime1(num)){
            arr[k]=num;
            k++;
        }
    }
    fclose(fptr);

    fptr=fopen("text.txt","a");                   -----O(1)
```

```
fprintf(fp, "\n");  
fputs("Prime no.s are : ", fp);  
for(int i=0;i<k;i++)  
{  
    fprintf(fp, "%d ",arr[i]);  
}  
fclose(fp);  
return 0;  
}
```

-----O(n)

OUTPUT



```
vscode > .vscode > ≡ text.txt  
1 42 18 35 1 20 25 29 9 13 15 6 46 32 28 12 42 46 43 28 37  
2 Prime no.s are : 1 25 29 9 13 43 37 |
```

TIME COMPLEXITY

For this particular code the total time complexity is given by $O(n \cdot \sqrt{n})$.
($O(n \cdot \sqrt{n})$, $O(1)$, $O(n)$, $O(1)$, $O(n)$, $O(n)$)

4.FIBONACCI SERIES

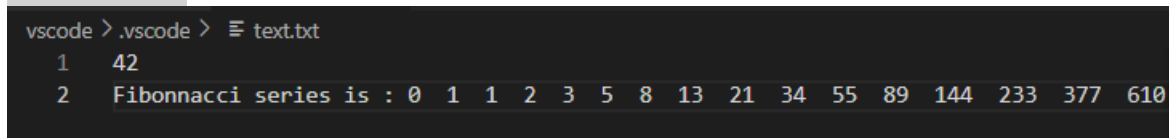
```
#include<stdio.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

int main()
{
    int i;
    int a=-1;
    int b=1;
    int c,n;
    FILE *fptr;

    randonGenrator(1);
    fptr = fopen("text.txt", "r");           -----O(1)
    fscanf(fptr, "%d", &n);

    fclose(fptr);
    fptr = fopen("text.txt", "a");           -----O(1)
    fputs("\nFibonnacci series is : ", fptr);
    for(i=1;i<=n;i++)                       -----O(n)
    {
        c=a+b;
        fprintf(fptr,"%d ",c);
        a=b;
        b=c;
    }
}
```

OUTPUT



```
vscode > .vscode > text.txt
1 42
2 Fibonnacci series is : 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

TIME COMPLEXITY

For this particular code the total time complexity is given by $O(n)$.
($O(1)$, $O(n)$, $O(1)$)

5.SUM OF DIGITS

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"

int main()
{
    randonGenrator(1);
    int sum = 0, n, r;
    FILE *fptr = NULL;

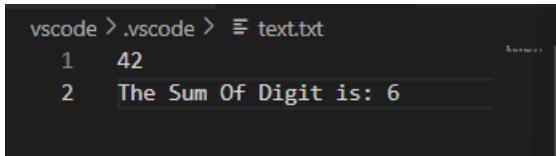
    fptr = fopen("text.txt", "r");
    fscanf(fptr, "%d", &n);
    fclose(fptr);

    while (n)
    {
        r = n % 10;
        sum = sum + r;
        n = n / 10;
    }

    fptr = fopen("text.txt", "a");
    if (fptr == NULL)
    {
        printf("Unable to open file.\n");
        exit(1);
    }

    fputs("\nThe Sum Of Digit is: ",fptr);
    fprintf(fptr, "%d", sum);
    fclose(fptr);
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 42
2 The Sum Of Digit is: 6
```

6.REVERSE THE DIGIT

```
#include<stdio.h>
#include<stdlib.h>
int reverse(int );

int main()
{
    FILE *fptr;
    int num,p;

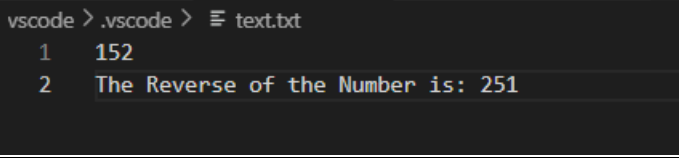
    fptr=fopen("text.txt","w");
    fprintf(fptr,"%d",152); //Enter the no. here:
    fclose(fptr);
    fptr=fopen("text.txt","r");
    fscanf(fptr,"%d",&num);
    fclose(fptr);

    p=reverse(num);
    fptr=fopen("text.txt","a");
    fputs("\nThe Reverse of the Number is: ",fptr);
    fprintf(fptr,"%d",p);
}

int reverse(int n)
{
    int temp=n;
    int r, sum=0;

    while(n)
    {
        r=n%10;
        sum=sum*10+r;
        n=n/10;
    }
    return sum;
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 152
2 The Reverse of the Number is: 251
```

7.PALINDROME NUMBER

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

int ispalindrome(int);

int main()
{
    FILE *fptr;
    int num,p;

    fptr=fopen("text.txt","w");                -----O(1)
        fprintf(fptr,"%d",151); //Enter the no. here:
    fclose(fptr);

    fptr=fopen("text.txt","r");                -----O(1)
    fscanf(fptr,"%d",&num);
    p=ispalindrome(num);
    fclose(fptr);

    fptr=fopen("text.txt","a");                -----O(1)
    if(p)
        fputs("\nThe Given No. is Palindrome:",fptr);
    else
        fputs("\nThe Given No. is Not Palindrome:",fptr);
    fclose(fptr);
}

int ispalindrome(int n)
{
    int temp=n;
    int r, sum=0;

    while(n)                -----O(n)
    {
        r=n%10;
        sum=sum*10+r;
        n=n/10;
    }
    if(temp==sum)
```

```
    return 1;  
    return 0;  
}
```

OUTPUT

```
vscode > .vscode > ≡ text.txt  
1    151  
2    The Given No. is Palindrome:
```

```
vscode > .vscode > ≡ text.txt  
1    155  
2    The Given No. is Not Palindrome:
```

TIME COMPLEXITY

For this particular code the total time complexity is given by $O(n)$.
($O(1)$, $O(1)$, $O(n)$, $O(1)$)

8.AREA OF THE SHAPES

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

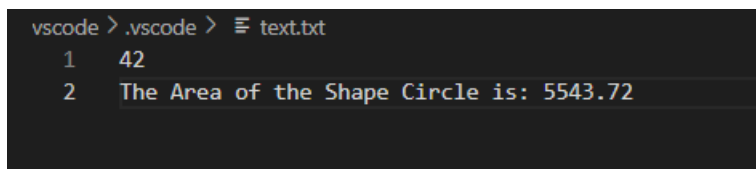
int main()
{
    randonGenrator(1);
    float area,n;
    FILE *fptr = NULL;

    fptr = fopen("text.txt", "r");
    fscanf(fptr, "%f", &n);
    fclose(fptr);

    area = 3.1427*(n*n);
    fptr = fopen("text.txt", "a");
    if (fptr == NULL)
    {
        printf("Unable to open file.\n");
        exit(1);
    }

    fputs("\nThe Area of the Shape Circle is: ",fptr);
    fprintf(fptr, "%0.2f", area);
    fclose(fptr);
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 42
2 The Area of the Shape Circle is: 5543.72
```

9. SIMPLE CALCULATOR

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

int main()
{
    FILE *fptr;
    int a[2];
    int k=0,num1,num2;

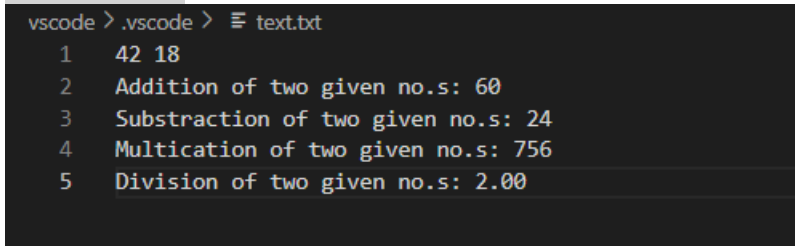
    randonGenrator(2);
    fptr = fopen("text.txt", "r");

    fscanf(fptr,"%d",&num1);
    fscanf(fptr,"%d",&num2);

    fptr=fopen("text.txt","a");
    fputs("\nAddition of two given no.s: ",fptr);
        fprintf(fptr,"%d", num1+num2);
    fputs("\nSubstraction of two given no.s: ",fptr);
        fprintf(fptr,"%d", num1-num2);
    fputs("\nMultication of two given no.s: ",fptr);
        fprintf(fptr,"%d", num1*num2);
    double d=num1/num2;
    fputs("\nDivision of two given no.s: ",fptr);
        fprintf(fptr,"%0.2lf", d);

    fclose(fptr);
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1  42 18
2  Addition of two given no.s: 60
3  Substraction of two given no.s: 24
4  Multication of two given no.s: 756
5  Division of two given no.s: 2.00
```

10.ARRAY OPERATIONS

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"

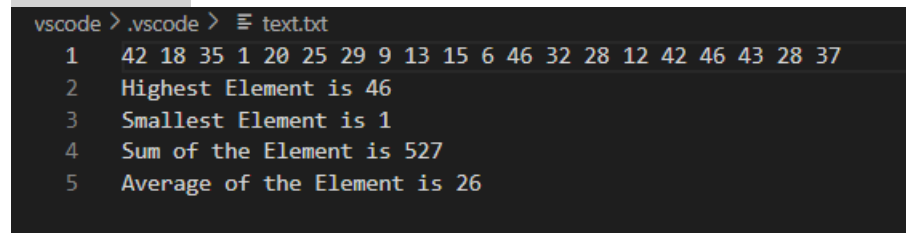
int main()
{
    int arr[100];
    FILE *fptr;
    int sum=0, h=0,k=0,num,s=99999;

    randonGenrator(20);
    fptr = fopen("text.txt", "r");
    if (fptr == NULL)
    {
        printf("File not available.");
        exit(0);
    }
    while(fscanf(fptr, "%d", &num) != -1)
    {
        arr[k]=num;
        k++;
    }
    fclose(fptr);
    for(int i=0;i<k;i++)
    {
        if(arr[i]>h)
            h=arr[i];
    }
    for(int i=0;i<k;i++)
    {
        if(arr[i]<s)
            s=arr[i];
    }
    for(int i=0;i<k;i++)
        sum=sum+arr[i];
    fptr=fopen("text.txt","a");
    fputs("\nHighest Element is ",fptr);
```



```
    fprintf(fp, "%d ",h);  
    fputs("\nSmallest Element is ",fp);  
    fprintf(fp, "%d ",s);  
    fputs("\nSum of the Element is ",fp);  
    fprintf(fp, "%d ",sum);  
    fputs("\nAverage of the Element is ",fp);  
    fprintf(fp, "%d ",sum/k);  
}
```

OUTPUT



The screenshot shows a VS Code terminal window with the following output:

```
vscode > .vscode > ≡ text.txt  
1 42 18 35 1 20 25 29 9 13 15 6 46 32 28 12 42 46 43 28 37  
2 Highest Element is 46  
3 Smallest Element is 1  
4 Sum of the Element is 527  
5 Average of the Element is 26
```

11. STRING OPERATION

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LENGTH 100

int main()
{
    char str1[MAX_LENGTH];
    char str2[MAX_LENGTH];

    printf("Enter string 1: ");
    fgets(str1, MAX_LENGTH, stdin);
    str1[strcspn(str1, "\n")] = '\0';

    printf("Enter string 2: ");
    fgets(str2, MAX_LENGTH, stdin);
    str2[strcspn(str2, "\n")] = '\0';

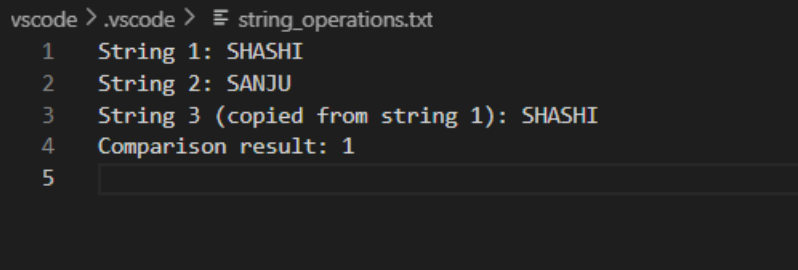
    char str3[MAX_LENGTH];
    strcpy(str3, str1);
    printf("String 3 (copied from string 1): %s\n", str3);

    int cmp_result = strcmp(str1, str2);
    if (cmp_result == 0)
    {
        printf("String 1 and String 2 are equal.\n");
    }
    else if (cmp_result < 0)
    {
        printf("String 1 is lexicographically smaller than String 2.\n");
    }
    else
    {
        printf("String 1 is lexicographically greater than String 2.\n");
    }

    FILE *file = fopen("string_operations.txt", "w");
```

```
if (file == NULL)
{
    printf("Error opening file.\n");
    return 1;
}
fprintf(file, "String 1: %s\n", str1);
fprintf(file, "String 2: %s\n", str2);
fprintf(file, "String 3 (copied from string 1): %s\n", str3);
fprintf(file, "Comparison result: %d\n", cmp_result);
fclose(file);
}
```

OUTPUT



```
vscode > .vscode > ≡ string_operations.txt
1 String 1: SHASHI
2 String 2: SANJU
3 String 3 (copied from string 1): SHASHI
4 Comparison result: 1
5
```

12. LINEAR SEARCH

```
#include <stdio.h>
#include <stdlib.h>
#include
"C:\\Users\\shash\\OneDrive\\Desktop\\vscode\\vscode\\randG.h"

int main()
{
    FILE *fptr;
    int num, s, k = 0;
    int a[100];

    randonGenrator(10);

    fptr = fopen("text.txt", "r");
    if (fptr == NULL)
    {
        printf("Unable to open file.\n");
        exit(1);
    } else
    {
        while (fscanf(fptr, "%d", &num) != EOF)
        {
            a[k] = num;
            k++;
        }
    }

    fclose(fptr);

    fptr=fopen("text.txt","a");
    printf("Enter the element to search: ");
    scanf("%d", &s);

    for (int i = 0; i < 10; i++)
    {
        if (s == a[i])
        {
            fputs("\nThe Element is Found.",fptr);
        }
    }
}
```

```
        return 0;
    }
}
fputs("\nElement not Found.\n",fptr);
fclose(fptr);
return 0;
}
```

OUTPUT

```
vscode > .vscode > ≡ text.txt
1  42 18 35 1 20 25 29 9 13 15
2  The Element is Found.
```

```
vscode > .vscode > ≡ text.txt
1  42 18 35 1 20 25 29 9 13 15
2  Element not Found.
3
```

13. BINARY SEARCH

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

int binarysearch(int [], int, int);
void bubblesort1(int [], int);

int main() {
    int arr[100];
    int n = 15, key, num;
    int p, k = 0;
    FILE *fptr;

    randonGenrator(n);
    fptr = fopen("text.txt", "r");

    if (fptr == NULL) {
        printf("File not available.");
        exit(1);
    }

    while (fscanf(fptr, "%d", &num) != EOF)
    {
        arr[k] = num;
        k++;
    }
    fclose(fptr);

    printf("Enter the key to search: ");
    scanf("%d", &key);

    bubblesort1(arr, n);

    fptr=fopen("text.txt","a");
    fputs("\nStored Elements are : ", fptr);
    for(int i=0;i<n;i++)
    {
        fprintf(fptr, "%d ",arr[i]);
```

```
}

p = binarysearch(arr, key, n);

if (p == 1)
{
    fputs("\nThe key is found\n",fptr);
}
else {
    fputs("\nThe key is not found",fptr);
}
fclose(fptr);
return 0;
}

void bubblesort1(int a[], int n) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++)
        {
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}

int binarysearch(int a[], int key, int n) {
    int l = 0, u = n - 1;
    int mid;

    while (l <= u) {
        mid = (l + u) / 2;

        if (key == a[mid])
            return 1;

        else if (key < a[mid])
```

```
        u = mid - 1;

    else
        l = mid + 1;
    }

    return -1;
}
```

OUTPUT

Key =15

```
vscode > .vscode > ≡ text.txt
1  42 18 35 1 20 25 29 9 13 15 6 46 32 28 12
2  Stored Elements are : 1 6 9 12 13 15 18 20 25 28 29 32 35 42 46
3  The key is found
4  
```


14.SELECTION SORT

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

void readarray(int [], int);
void displayarray(int [], int);
void selectionsort(int [], int);

int main()
{
    int n=10;

    int arr[10],k=0, num;
    FILE *fptr;

    randonGenrator(n);
    fptr = fopen("text.txt", "r");

    if (fptr == NULL)
    {
        printf("File not available.");
        exit(0);
    }

    while(fscanf(fptr, "%d", &num) != -1)
    {
        arr[k]=num;
        k++;
    }
    fclose(fptr);

    selectionsort(arr, n);

    fptr=fopen("text.txt","a");
    fputs("\nSorted Array is : ", fptr);
    for(int i=0;i<k;i++)
    {
```

```
        fprintf(fp, "%d ",arr[i]);
    }
    fclose(fp);

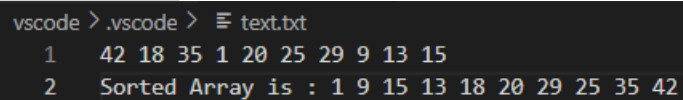
}

void selectionsort(int a[], int n)
{
    int min;
    int temp;

    for(int i=0;i<n;i++)
    {
        min=i;
        for(int j=i+1;j<n;j++)
        {
            if(a[j]<a[min])
                min=j;

            temp=a[i];
            a[i]=a[min];
            a[min]=temp;
        }
    }
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1  42 18 35 1 20 25 29 9 13 15
2  Sorted Array is : 1 9 15 13 18 20 29 25 35 42
```

15.BUBBLE SORT

```
#include<stdio.h>
#include<stdlib.h>
#include"C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

void bubblesort1(int *, int);
void bubblesort2(int *, int);

int main()
{
    FILE *fptr;
    int num;
    int arr[100];
    int k = 0;
    char a[100]="Ascending Order: ";
    char d[100]="Descending Order: ";
    int z=20;

    randonGenrator(z);
    fptr = fopen("text.txt", "r");

    if (fptr == NULL)
    {
        printf("Unable to open file for reading.\n");
        exit(0);
    }

    //rewind(fptr);

    while (fscanf(fptr, "%d", &num) == 1)
    {
        arr[k] = num;
        k++;
    }

    fclose(fptr);

    bubblesort1(arr,z);
    fptr=fopen("text.txt","a");
```

```
fprintf(fp, "\n");
fprintf(fp, "%s", a);

for(int i=0;i<z;i++)
{
    fprintf(fp, "%d ",arr[i]);
}

fclose(fp);

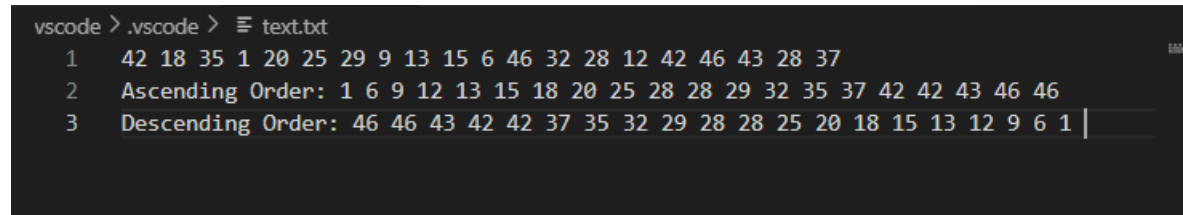
bubblesort2(arr,z);
fp=fopen("text.txt","a");
fprintf(fp, "\n");
fprintf(fp, "%s", d); //fputs("Ascending Order : ", fp);
for(int i=0;i<z;i++)
{
    fprintf(fp, "%d ",arr[i]);
}

printf("The Elements are perfectly Sorted n again Stored in File
Successfully");
return 0;
}

void bubblesort1(int a[], int n)
{
    int temp=0;
    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-1-i;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
```

```
void bubblesort2(int a[], int n)
{
    int temp=0;
    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-1-i;j++)
        {
            if(a[j]<a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1  42 18 35 1 20 25 29 9 13 15 6 46 32 28 12 42 46 43 28 37
2  Ascending Order: 1 6 9 12 13 15 18 20 25 28 28 29 32 35 37 42 42 43 46 46
3  Descending Order: 46 46 43 42 42 37 35 32 29 28 28 25 20 18 15 13 12 9 6 1 |
```

16.INSERTION SORT

```
#include <stdio.h>
#include <stdlib.h>
#include "C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

void insertionSort(int [], int);

int main()
{
    int arr[10], k = 0, num;
    FILE *fptr;

    randonGenrator(10);
    fptr = fopen("text.txt", "r");

    if (fptr == NULL)
    {
        printf("File not available.");
        exit(0);
    }

    while (fscanf(fptr, "%d", &num) != EOF)
    {
        arr[k] = num;
        k++;
    }
    fclose(fptr);

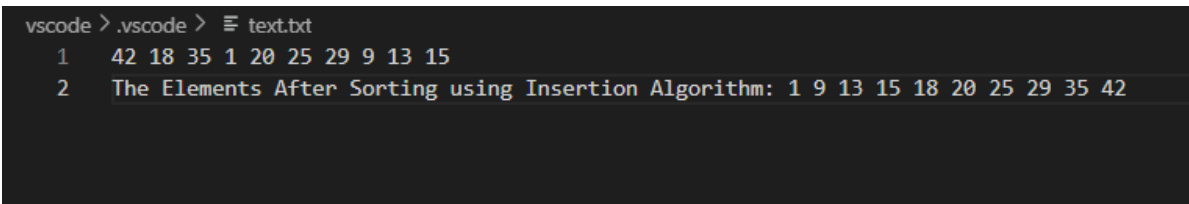
    insertionSort(arr, k);

    fptr = fopen("text.txt", "a");
    fputs("\n\nThe Elements After Sorting using Insertion Algorithm: ", fptr);
    for (int i = 0; i < k; i++)
        fprintf(fptr, "%d ", arr[i]);
    fclose(fptr);

    return 0;
}
```

```
void insertionSort(int a[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = a[i];
        j = i - 1;
        while (j >= 0 && a[j] > key)
        {
            a[j + 1] = a[j];
            j = j - 1;
        }
        a[j + 1] = key;
    }
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 42 18 35 1 20 25 29 9 13 15
2 The Elements After Sorting using Insertion Algorithm: 1 9 13 15 18 20 25 29 35 42
```

17. MATRIX OPERATIONS (ADDITION)

```
#include<stdio.h>
#include<stdlib.h>
#define ROW 10
#define COL 10

void readmatrix(int a[ROW][COL], int r, int c);
void displaymatrix(int a[ROW][COL], int r, int c);
void addmat(int a[ROW][COL], int b[ROW][COL], int y[ROW][COL], int r, int c);

int main()
{
    int a[ROW][COL];
    int b[ROW][COL];
    int y[ROW][COL];
    int r, c;

    printf("Enter the order: ");
    scanf("%d%d",&r,&c);
    printf("Enter the matrix A elements: \n");
    readmatrix(a, r, c);

    printf("Enter the matrix B elements: \n");
    readmatrix(b, r, c);
    printf("The matrix A elements are: \n");
    displaymatrix(a, r, c);

    printf("The matrix B elements are: \n");
    displaymatrix(b, r, c);

    addmat(a,b,y,r,c);

    printf("The Sum of matrix A & B is: \n");
    displaymatrix(y, r, c);
}

void addmat(int a[ROW][COL], int b[ROW][COL],int y[ROW][COL], int r,int c)
{
    for(int i=0;i<r;i++)
```



```
{
    for(int j=0;j<c;j++)
    {
        y[i][j]=a[i][j]+b[i][j];
    }
}
}

void displaymatrix(int a[][COL], int r, int c)
{
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
}

void readmatrix(int a[][COL], int r, int c)
{
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}
```

Output

```
Enter the order: 2 3
Enter the matrix A elements:
1 2 3 4 5 6
Enter the matrix B elements:
5 6 7 8 9 2
The matrix A elements are:
1 2 3
4 5 6
The matrix B elements are:
5 6 7
8 9 2
The Sum of matrix A & B is:
6 8 10
12 14 8
```

MATRIX OPERATIONS(MULTIPLICATION)

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

#define ROW 10
#define COL 10

void readmatrix(int[][COL], int, int);
void displaymatrix(int[][COL], int, int);
void mulmat(int[][COL], int[][COL], int[][COL], int, int, int);

int main()
{
    int a[ROW][COL];
    int b[ROW][COL];
    int y[ROW][COL];
    int r1, c1, r2, c2;

    printf("Enter the order of matrix A: ");
    scanf("%d%d",&r1,&c1);

    printf("Enter the matrix A elements: \n");
    readmatrix(a, r1, c1);

    printf("Enter the order of matrix B: ");
    scanf("%d%d",&r2,&c2);

    printf("Enter the matrix B elements: \n");
    readmatrix(b, r2, c2);

    printf("The matrix A elements are: \n");
    displaymatrix(a, r1, c1);

    printf("The matrix B elements are: \n");
    displaymatrix(b, r2, c2);

    if(c1!=r2)
    {
        printf("The Multiplication cannot be performed..!");
    }
}
```

```
        exit(0);
    }

    mulmat(a,b,y,r1,c1,c2);

    printf("The Multiplication of matrix A & B is: \n");
    displaymatrix(y, r2, c2);
}

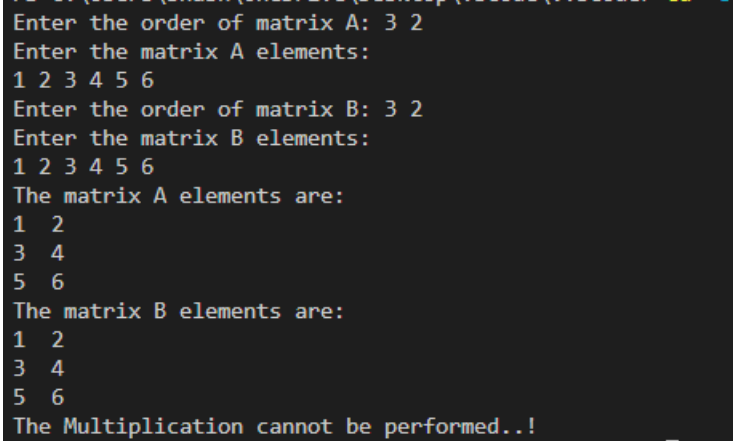
void mulmat(int a[][COL], int b[][COL], int y[][COL], int r1, int c1, int c2)
{
    for(int i=0;i<r1;i++)
    {
        for(int j=0;j<c1;j++)
        {
            y[i][j]=0;
            for (int k=0;k<c1;k++)
            {
                y[i][j]= y[i][j]+ a[i][k]*b[k][j];
            }
        }
    }
}

void displaymatrix(int a[][COL], int r, int c)
{
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
}

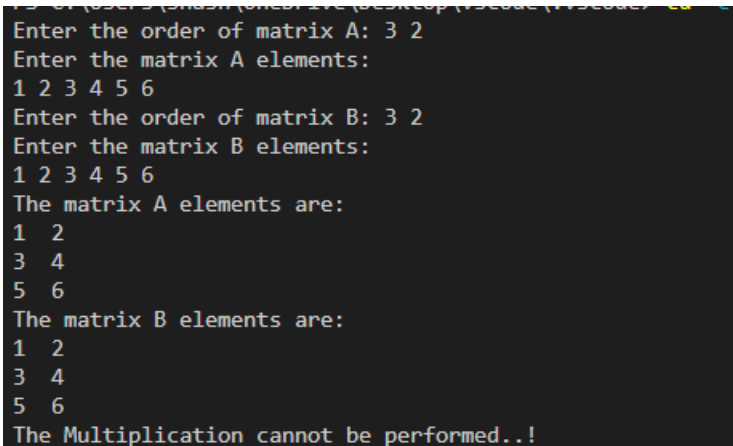
void readmatrix(int a[][COL], int r, int c)
{
    for(int i=0;i<r;i++)
    {
        for(int j=0;j<c;j++)
```

```
{  
    scanf("%d",&a[i][j]);  
}  
}  
}
```

OUTPUT



```
Enter the order of matrix A: 3 2  
Enter the matrix A elements:  
1 2 3 4 5 6  
Enter the order of matrix B: 3 2  
Enter the matrix B elements:  
1 2 3 4 5 6  
The matrix A elements are:  
1 2  
3 4  
5 6  
The matrix B elements are:  
1 2  
3 4  
5 6  
The Multiplication cannot be performed..!
```



```
Enter the order of matrix A: 3 2  
Enter the matrix A elements:  
1 2 3 4 5 6  
Enter the order of matrix B: 3 2  
Enter the matrix B elements:  
1 2 3 4 5 6  
The matrix A elements are:  
1 2  
3 4  
5 6  
The matrix B elements are:  
1 2  
3 4  
5 6  
The Multiplication cannot be performed..!
```

Part- C

1.LINKED LIST(15)

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

struct node
{
    int data;
    struct node *next;
};

void addatbeg(struct node **, int);
void displaydata(struct node **);
int deleteatbeg(struct node **);
void addatend(struct node **, int);
int deleteatend(struct node **);
int lenghtoflist(struct node *);
int searchelement(struct node *, int );
int frequency(struct node *, int);
void addatanypos(struct node **, int, int);
void reverselist(struct node **);
void deletetestocc(struct node **, int);
void sortlist(struct node *);
void r_Duplicate(struct node *);
void reverseKthNode(struct node **, int );
void deleteAtPos(struct node **, int);

int main()
{
    struct node *head=NULL;
    int n, e, f, pos, key;

    while(1)
    {
        printf("\n1.Add at beginning\n");
        printf("2.Delete at beginning\n");
        printf("3.display list\n");
```

```
printf("4.Add at end\n");
printf("5.Delete at end\n");
printf("6.Length of list\n");
printf("7.Search an element\n");
printf("8.Frequency of element:\n");
printf("9.Add node at any given position\n");
printf("10.Reverse List\n");
printf("11.Delete the First occurrence\n");
printf("12.Sort_list\n");
printf("13.Remove Duplicate\n");
printf("14.Reverse Every kth node\n");
printf("15.Delete at position\n");

printf("Enter n: ");
scanf("%d",&n);

switch(n)
{
    case 1: printf("Enter the date: ");
            scanf("%d",&e);
            addatbeg(&head, e);
            break;

    case 2: if(head==NULL)
            printf("List empty");
            else
            {
                e=deleteatbeg(&head);
            }
            break;

    case 3: displaydata(&head);
            break;

    case 4: printf("Enter the data: ");
            scanf("%d",&e);
            addatend(&head, e);
            break;

    case 5: if(head==NULL)
```

```
        printf("List empty");
    else
    {
        e=deleteatend(&head);
    }
    break;

case 6: e=lengthoflist(head);
    printf("List length=%d",e);
    break;

case 7: if(head==NULL)
    {
        printf("list empty");
    }
    else
    {
        printf("Enter the key: ");
        scanf("%d",&e);
        f=searchelement(head, e);
        if(f)
            printf("Key is found:");
        else
            printf("Key not found");
    }
    break;

case 8: printf("Enter the key for frequency: ");
    scanf("%d",&e);

    f=frequency(head, e);

    printf("The frequency of %d is %d", e, f);

case 9: printf("Enter the data: ");
    scanf("%d",&e);
    printf("Enter position: ");
    scanf("%d",&pos);
    addatanypos(&head, e, pos);
    break;
```

```
        case 10: reverselist(&head);
                break;

        case 11: printf("Enter the key: ");
                scanf("%d", &key);
                deletetoccc(&head, key);
                break;

        case 12: sortlist(head);
                break;

        case 13: r_Duplicate(head);
                break;

        case 14: printf("Enter k: ");
                scanf("%d", &e);
                reverseKthNode(&head, e);
                break;

        case 15: printf("Enter pos: ");
                scanf("%d", &e);
                deleteAtPos(&head, e);
                break;

        default: exit(0);
    }
}

void addatbeg(struct node **head, int e)
{
    struct node *p;

    p = (struct node *) malloc(sizeof(struct node));

    if(p == NULL)
    {
        perror(" ");
        return;
    }
}
```



```
}

p->data=e;
p->next=*head;
*head=p;
}

void displaydata(struct node **head)
{
    struct node *cur=*head;

    if(*head==NULL)
    {
        printf("-----List Empty-----\n-----Enter data-----\n");
        return;
    }
    while(cur)
    {
        printf("%d\t",cur->data);
        cur=cur->next;
    }
}

int deleteatbeg(struct node **head)
{
    struct node *p=*head;

    *head=(*head)->next;
    int e=p->data;
    free(p);
    return e;
}

void addatend(struct node **head, int e)
{
    struct node *p;
    struct node *cur=*head;

    p=(struct node*)malloc(sizeof(struct node));
```

```
if(p==NULL)
{
    perror(" ");
    return;
}

p->data=e;
p->next=NULL;

if(*head==NULL)
    *head=p;
else
{
    while(cur->next)
        cur=cur->next;

    cur->next=p;
}
}

int deleteatend(struct node **head)
{
    struct node *cur=*head;
    struct node *prev=NULL;
    int e;

    while(cur->next)
    {
        prev=cur;
        cur=cur->next;
    }

    e=cur->data;
    if(prev)
        prev->next=NULL;
    else
        *head=NULL;

    free(cur);
}
```

```
    return e;
}

int lenghtoflist(struct node *head)
{
    struct node *p=head;
    int count=0;

    while(p)
    {
        count++;
        p=p->next;
    }
    return count;
}

int searchelement(struct node *head, int key)
{
    struct node *cur=head;

    while(cur)
    {
        if(key==cur->data)
            return 1;
        cur=cur->next;
    }
    return 0;
}

int frequency(struct node *head, int key)
{
    struct node *cur=head;
    int c=0;

    while(cur)
    {
        if(key==cur->data)
        {
            c++;
            cur=cur->next;
        }
    }
}
```

```
    }  
  }  
  return c;  
}  
  
void addatanypos(struct node **head, int e, int pos)  
{  
    struct node *p;  
    struct node *cur=*head;  
    struct node *prev=NULL;  
  
    p=(struct node *)malloc(sizeof(struct node));  
  
    if(p==NULL)  
    {  
        perror("");  
        return ;  
    }  
  
    p->data=e;  
    p->next=NULL;  
  
    if(pos<=0 || *head==NULL)  
    {  
        p=*head;  
        *head=p;  
        return;  
    }  
  
    for(int i=1;i<=pos && cur ;i++)  
    {  
        prev=cur;  
        cur=cur->next;  
    }  
    prev->next=p;  
    p->next=cur;  
}  
  
void deleteAtPos(struct node **head, int pos)
```

```
{
    struct node *prev=NULL;
    struct node *cur=*head;

    if(pos<=0)
    {
        *head=(*head)->next;
        free(cur);
        return;
    }
    else if(pos>lenghtoflist(*head))
    {
        while(cur->next)
        {
            prev=cur;
            cur=cur->next;
        }
        prev->next=NULL;
        free(cur);
    }
    else
    {
        for(int i=0;i<pos-1;i++)
        {
            prev=cur;
            cur=cur->next;
        }
        prev->next=cur->next;
        free(cur);
    }
}

void reverselist(struct node **head)
{
    struct node *p=*head;
    struct node *q, *r=NULL;

    if(p)
        q=p->next;
    if(q)
```

```
    r=q->next;

while(q)
{
    q->next=p;
    p=q;
    q=r;

    if(r)
        r=r->next;

}

if(*head)
    (*head)->next=NULL;

*head=p;
}

void deletestocc(struct node** head, int key)
{
    struct node* cur = *head;
    struct node* prev = NULL;

    if (cur == NULL)
        return;

    while (cur != NULL && cur->data != key)
    {
        prev = cur;
        cur = cur->next;
    }

    if (cur == NULL)
        return;

    if (cur == *head)
    {
        *head = cur->next;
        free(cur);
    }
}
```

```
        return;
    }

    prev->next = cur->next;
    free(cur);
}

void sortlist(struct node *head)
{
    struct node *min;
    int temp;
    struct node *i, *j;

    for(i=head;i&& i->next;i=i->next)
    {
        min = i;
        for (j=i->next;j!=NULL;j=j->next)
        {
            if (j->data < min->data)
                min = j;
        }
        if (min!=i)
        {
            temp = i->data;
            i->data = min->data;
            min->data = temp;
        }
    }
}

void reverseKthNode(struct node **head, int k)
{
    if (*head == NULL || k <= 1)
    {
        return;
    }

    int n = lengthoflist(*head) / k;

    struct node *cur = *head;
```

```
struct node *prev = NULL;

for (int i = 0; i < n; i++)
{
    struct node *p = cur;
    struct node *q = NULL;
    struct node *r = NULL;

    for (int c = 0; c < k; c++)
    {
        struct node *temp = cur->next;
        cur->next = q;
        q = cur;
        cur = temp;
    }

    if (i == 0)
        *head = q;
    else
        prev->next = q;

    p->next = cur;
    prev = p;
    r = cur;
}
}

void r_Duplicate(struct node* head)
{
    if (head == NULL)
        return;

    struct node* current = head;

    while (current != NULL)
    {
        struct node* runner = current;

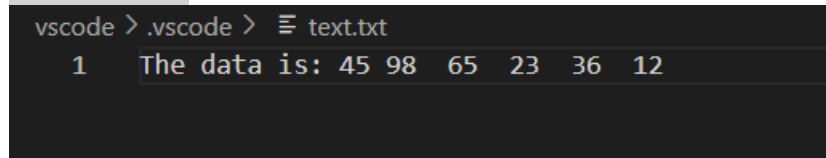
        while (runner->next != NULL)
        {
```



```
    if (runner->next->data == current->data)
    {
        struct node* temp = runner->next;
        runner->next = runner->next->next;
        free(temp);
    }
    else
    {
        runner = runner->next;
    }
}

current = current->next;
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 The data is: 45 98 65 23 36 12
```

DOUBLY LINKED LIST(7)

Wynk online music player

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct song
{
    char name[20];
    char singer[20];
    double time;
};

struct node
{
    struct song s;
    struct node *next;
    struct node *prev;
};

void addatbeg(struct node **head, struct song s);
void addatendl(struct node **head, struct song s);
void Displaylist(struct node *head);
void readdata(struct song *s);
void deletename(struct node **head, char song[]);
void displaytime(struct node *head);
void deletesinger(struct node **head, char singer[]);
void reverselist(struct node **head);

int main()
{
    struct song s;
    struct node *head = NULL;
    int ch;
    char name[20];

    while (1)
    {
        printf("1. Add a song\n");
```

```
printf("2. Delete song by name\n");
printf("3. Delete songs by singer\n");
printf("4. Display songs by duration\n");
printf("5. Display all songs\n");
printf("6. Reverse the list\n");
printf("7. Delete the song\n");
printf("8. Exit\n");

printf("Enter choice: ");
scanf("%d", &ch);

switch (ch)
{
case 1:
    readdata(&s);
    addatendll(&head, s);
    break;

case 2:
    printf("Enter the song name to be deleted: ");
    scanf("%s", name);
    deletename(&head, name);
    break;

case 3:
    printf("Enter the singer name to delete all songs: ");
    scanf("%s", name);
    deletesinger(&head, name);
    break;

case 4:
    displaytime(head);
    break;

case 5:
    Displaylist(head);
    break;

case 6:
    reverselist(&head);
```

```
        break;

    case 7:
        deletesongprev(&head);
        break;

    default:
        printf("Invalid choice\n");
        exit(0);
    }
}

void readdata(struct song *s)
{
    printf("Enter song name: ");
    scanf("%s", s->name);

    printf("Enter singer name: ");
    scanf("%s", s->singer);

    printf("Enter duration: ");
    scanf("%lf", &s->time);
}

void addatbeg(struct node **head, struct song s)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL)
    {
        printf("Memory allocation failed\n");
        return;
    }

    newNode->s = s;
    newNode->next = NULL;
    newNode->prev = NULL;

    if (*head == NULL)
    {
```

```
*head = newNode;
(*head)->next = *head;
(*head)->prev = *head;
}
else
{
    newNode->next = *head;
    newNode->prev = (*head)->prev;
    (*head)->prev->next = newNode;
    (*head)->prev = newNode;
    *head = newNode;
}
}

void Displaylist(struct node *head)
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }

    struct node *cur = head;
    do
    {
        printf("Song name: %s\n", cur->s.name);
        printf("Singer name: %s\n", cur->s.singer);
        printf("Duration: %.2lf\n\n", cur->s.time);
        cur = cur->next;
    } while (cur != head);
}

void deletename(struct node **head, char song[])
{
    if (*head == NULL)
    {
        printf("List is empty\n");
        return;
    }
}
```

```
struct node *cur = *head;
struct node *prevNode = NULL;

do
{
    if (strcmp(cur->s.name, song) == 0)
    {
        if (cur == *head)
        {
            if (cur->next == *head)
            {
                free(cur);
                *head = NULL;
                return;
            }
            else
            {
                (*head)->next->prev = (*head)->prev;
                (*head)->prev->next = (*head)->next;
                *head = (*head)->next;
                free(cur);
                return;
            }
        }
        else
        {
            cur->prev->next = cur->next;
            cur->next->prev = cur->prev;
            free(cur);
            return;
        }
    }

    prevNode = cur;
    cur = cur->next;
} while (cur != *head);

printf("Song not found\n");
}
```

```
void deletesinger(struct node **head, char singer[])
{
    if (*head == NULL)
    {
        printf("List is empty\n");
        return;
    }

    struct node *cur = *head;
    struct node *prevNode = NULL;
    int isDeleted = 0;

    do
    {
        if (strcmp(cur->s.singer, singer) == 0)
        {
            isDeleted = 1;

            if (cur == *head)
            {
                if (cur->next == *head)
                {
                    free(cur);
                    *head = NULL;
                    return;
                }
                else
                {
                    (*head)->next->prev = (*head)->prev;
                    (*head)->prev->next = (*head)->next;
                    *head = (*head)->next;
                    free(cur);
                    cur = *head;
                }
            }
            else
            {
                cur->prev->next = cur->next;
                cur->next->prev = cur->prev;
                struct node *temp = cur;
```

```
        cur = cur->next;
        free(temp);
    }
}
else
{
    prevNode = cur;
    cur = cur->next;
}

} while (cur != *head);

if (!isDeleted)
    printf("No songs found for the given singer\n");
}

void displaytime(struct node *head)
{
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }

    struct node *cur = head;
    struct node *shortest = cur;
    cur = cur->next;

    while (cur != head)
    {
        if (cur->s.time < shortest->s.time)
            shortest = cur;

        cur = cur->next;
    }

    printf("Shortest Song:\n");
    printf("Song name: %s\n", shortest->s.name);
    printf("Singer name: %s\n", shortest->s.singer);
    printf("Duration: %.2lf\n\n", shortest->s.time);
}
```



```
}

void addatendll(struct node **head, struct song s)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL)
    {
        printf("Memory allocation failed\n");
        return;
    }

    newNode->s = s;
    newNode->next = NULL;
    newNode->prev = NULL;

    if (*head == NULL)
    {
        *head = newNode;
        (*head)->next = *head;
        (*head)->prev = *head;
    }
    else
    {
        struct node *lastNode = (*head)->prev;
        newNode->next = *head;
        newNode->prev = lastNode;
        lastNode->next = newNode;
        (*head)->prev = newNode;
    }
}

void reverselist(struct node **head)
{
    if (*head == NULL)
    {
        printf("List is empty\n");
        return;
    }

    struct node *current = *head;
```

```
struct node *temp = NULL;

do
{
    temp = current->prev;
    current->prev = current->next;
    current->next = temp;
    current = current->prev;
} while (current != *head);

*head = temp->prev;
}

void deletesongprev(struct node **head)
{
    struct node *cur = *head;
    struct node *temp = cur->prev;

    if (cur->next == cur)
    {
        free(cur);
        *head = NULL;
        return;
    }

    do
    {
        if (cur->s.time > temp->s.time)
        {
            temp->next = cur->next;
            cur->next->prev = temp;
            if (cur == *head)
                *head = cur->next;
            struct node *temp2 = cur;
            cur = cur->next;
            free(temp2);
        }
        else
        {
            cur = cur->next;
        }
    }
}
```

```
        temp = temp->next;  
    }  
} while (cur != *head);  
  
}
```

OUTPUT

```
Song name: aa  
Singer name: bb  
Duration: 12.00
```

```
Song name: cc  
Singer name: dd  
Duration: 13.00
```

```
Song name: ee  
Singer name: ff  
Duration: 13.00
```

LINEAR QUEUES(3)

Chocolate distribution on some basis

```
#include <stdio.h>
#include "C:\Users\shash\OneDrive\Desktop\vscode\vscode\randG.h"

#define MAX_QUEUE_SIZE 100

typedef struct
{
    int front;
    int rear;
    int data[MAX_QUEUE_SIZE];
} Queue;

void initQueue(Queue* queue)
{
    queue->front = -1;
    queue->rear = -1;
}

int isEmpty(Queue* queue)
{
    return queue->front == -1;
}

int isFull(Queue* queue)
{
    return (queue->rear + 1) % MAX_QUEUE_SIZE == queue->front;
}

void enqueue(Queue* queue, int data)
{
    if (isFull(queue)) {
        printf("Queue is full.\n");
        return;
    }

    if (isEmpty(queue)) {
        queue->front = queue->rear = 0;
```

```
} else {
    queue->rear = (queue->rear + 1) % MAX_QUEUE_SIZE;
}

queue->data[queue->rear] = data;
}

int dequeue(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    }

    int data = queue->data[queue->front];

    if (queue->front == queue->rear)
        queue->front = queue->rear = -1;
    else
        queue->front = (queue->front + 1) % MAX_QUEUE_SIZE;

    return data;
}

void distributeCandies(int candies[], int n)
{
    Queue queue;
    initQueue(&queue);

    for (int i = 0; i < n; ++i)
    {
        if (candies[i] > 0)
        {
            enqueue(&queue, i);
        }
    }
    int candiesToDistribute = n;
    int currentChild = dequeue(&queue);

    while (candiesToDistribute > 0)
    {
```

```
candies[currentChild]++;

if (candies[currentChild] > 0)
    enqueue(&queue, currentChild);

currentChild = dequeue(&queue);
candiesToDistribute--;
}
}

int main()
{
    int arr[10], k=0, num;
    FILE *fptr;

    randonGenrator(5);
    fptr = fopen("text.txt", "r");

    if (fptr == NULL)
    {
        printf("File not available.");
        exit(0);
    }

    while(fscanf(fptr, "%d", &num) != -1)
    {
        arr[k]=num;
        k++;
    }
    fclose(fptr);

    //int numChildren = sizeof(arr) / sizeof(arr[0]);

    distributeCandies(arr, k);

    fptr=fopen("text.txt","a");

    fputs("\nFinal distribution of candies:\n",fptr);
    for (int i = 0; i < k; ++i)
    {
```

```
fputs("\nChild ",fptr);  
fprintf(fptr, "%d ",i+1);  
fputs("has ",fptr);  
fprintf(fptr, "%d ",arr[i]);  
fputs("Candies",fptr);  
}  
fclose(fptr);  
return 0;  
}
```

OUTPUT

```
vscode > .vscode > ≡ text.txt  
1 41 67 34 0 69  
2 Final distribution of candies:  
3  
4 Child 1 has 43 Candies  
5 Child 2 has 68 Candies  
6 Child 3 has 35 Candies  
7 Child 4 has 0 Candies  
8 Child 5 has 70 Candies
```

CIRCULAR QUEUE(2)

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

#define MAX_SIZE 5

struct CircularQueue
{
    int items[MAX_SIZE];
    int front;
    int rear;
};

int is_empty(struct CircularQueue* queue)
{
    return queue->front == -1;
}

int is_full(struct CircularQueue* queue)
{
    return (queue->rear + 1) % MAX_SIZE == queue->front;
}

void enqueue(struct CircularQueue* queue, int item)
{
    if (is_full(queue))
    {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }

    if (is_empty(queue))
    {
        queue->front = 0;
        queue->rear = 0;
    }
    else
    {
        queue->rear = (queue->rear + 1) % MAX_SIZE;
```



```
}

queue->items[queue->rear] = item;
printf("Enqueued: %d\n", item);
}

int dequeue(struct CircularQueue* queue)
{
    if (is_empty(queue))
    {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }

    int item = queue->items[queue->front];

    if (queue->front == queue->rear)
    {
        queue->front = -1;
        queue->rear = -1;
    }
    else
    {
        queue->front = (queue->front + 1) % MAX_SIZE;
    }

    printf("Dequeued: %d\n", item);
    return item;
}

void display(struct CircularQueue* queue)
{
    if (is_empty(queue))
    {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue contents: ");
    int i = queue->front;
```

```
do
{
    printf("%d ", queue->items[i]);
    i = (i + 1) % MAX_SIZE;

} while (i != (queue->rear + 1) % MAX_SIZE);
printf("\n");
}
int main()
{
    struct CircularQueue queue;
    queue.front = -1;
    queue.rear = -1;

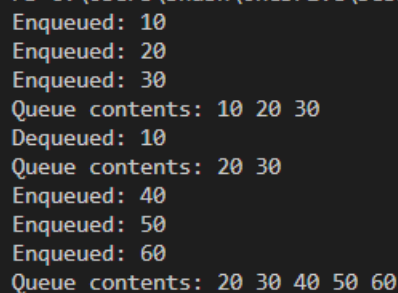
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);
    display(&queue);

    dequeue(&queue);
    display(&queue);

    enqueue(&queue, 40);
    enqueue(&queue, 50);
    enqueue(&queue, 60);
    display(&queue);

    return 0;
}
```

OUTPUT



```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue contents: 10 20 30
Dequeued: 10
Queue contents: 20 30
Enqueued: 40
Enqueued: 50
Enqueued: 60
Queue contents: 20 30 40 50 60
```

STACKS(6)

Check for the balanced parenthesis

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_SIZE 100

struct stack
{
    char items[MAX_SIZE];
    int top;
};

void initialize(struct stack *s)
{
    s->top = -1;
}

int isFull(struct stack *s)
{
    return s->top == MAX_SIZE - 1;
}

int isEmpty(struct stack *s)
{
    return s->top == -1;
}

void push(struct stack *s, char c)
{
    if (isFull(s))
    {
        printf("Stack Overflow\n");
        exit(EXIT_FAILURE);
    }
    s->items[++s->top] = c;
}
```

```
char pop(struct stack *s)
{
    if (isEmpty(s))
    {
        printf("Stack Underflow\n");
        exit(EXIT_FAILURE);
    }
    return s->items[s->top--];
}

char peek(struct stack *s)
{
    if (isEmpty(s))
    {
        printf("Stack is empty\n");
        exit(EXIT_FAILURE);
    }
    return s->items[s->top];
}

int isBalanced(char equation[])
{
    struct stack s;
    initialize(&s);

    for (int i = 0; equation[i] != '\0'; i++)
    {
        if (equation[i] == '(' || equation[i] == '[' || equation[i] == '{')
        {
            push(&s, equation[i]);
        }
        else if (equation[i] == ')' || equation[i] == ']' || equation[i] == '}')
        {
            if (isEmpty(&s))
            {
                return false;
            }

            char top = pop(&s);
```

```
        if ((equation[i] == ')' && top != '(') || (equation[i] == ']' && top != '[') ||
(equation[i] == '}' && top != '{'))
        {
            return false;
        }
    }
}

return isEmpty(&s);
}

int main()
{
    FILE *fptr;
    fptr = fopen("text.txt", "w");

    if (fptr == NULL)
    {
        printf("File not available.");
        exit(EXIT_FAILURE);
    }

    fputs("Check for Paranthesis\nThe Equation is :", fptr);
    fputs("((x+y)z)", fptr);
    fclose(fptr);

    char equation[MAX_SIZE];
    fptr = fopen("text.txt", "r");

    if (fptr == NULL)
    {
        printf("File not available.");
        exit(EXIT_FAILURE);
    }

    fgets(equation, MAX_SIZE, fptr);
    fclose(fptr);

    fptr=fopen("text.txt","a");
    if (isBalanced(equation))
```

```
{  
    fputs("\nThe equation has balanced parentheses.\n",fptr);  
}  
else  
{  
    fputs("\nThe equation does not have balanced parentheses.\n",fptr);  
}  
fclose(fptr);  
return 0;  
}
```

OUTPUT

```
vscode > .vscode > ≡ text.txt  
1 Check for Paranthesis  
2 The Equation is :((x+y)z)  
3 The equation has balanced parentheses.  
4
```

```
vscode > .vscode > ≡ text.txt  
1 Check for Paranthesis  
2 The Equation is :((x+y)(z)  
3 The equation does not have balanced parentheses.  
4
```

INFIX TO POSTFIX

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 30

struct stack
{
    char items[MAX_SIZE];
    int top;
};

void initialize(struct stack *);
int isFull(struct stack *);
int isEmpty(struct stack *);
void push(struct stack *, char);
char pop(struct stack *);
char peek(struct stack *);
int isOperand(char);
int isOperator(char);
int getPrecedence(char);
void infixToPostfix(char [], char []);

int main()
{
    FILE *fptr=fopen("text.txt","w");

    fputs("+*ab*cd",fptr);
    fclose(fptr);

    char infix[30];
    char postfix[30];

    fptr=fopen("text.txt","r");
    fgets(infix, MAX_SIZE, fptr);
    fclose(fptr);

    infixToPostfix(infix, postfix);
```

```
fptr=fopen("text.txt","w");
fputs("The infix Expression is: ",fptr);
fprintf(fptr,"%s",infix);

fputs("\nThe postfix expression is: ",fptr);
fprintf(fptr,"%s",postfix);
fclose(fptr);
return 0;
}

void initialize(struct stack *s)
{
    s->top = -1;
}

int isFull(struct stack *s)
{
    return s->top == MAX_SIZE - 1;
}

int isEmpty(struct stack *s)
{
    return s->top == -1;
}

void push(struct stack *s, char c)
{
    if (isFull(s))
    {
        printf("Stack Overflow\n");
        return;
    }
    s->items[++s->top] = c;
}

char pop(struct stack *s)
{
    if (isEmpty(s))
    {
```



```
        printf("Stack Underflow\n");
        return '\0';
    }
    return s->items[s->top--];
}

char peek(struct stack *s)
{
    if (isEmpty(s))
    {
        printf("Stack is empty\n");
        return '\0';
    }
    return s->items[s->top];
}

int isOperand(char c)
{
    return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
}

int isOperator(char c)
{
    return c == '+' || c == '-' || c == '*' || c == '/';
}

int getPrecedence(char c)
{
    {
        switch (c) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
        }
        return -1;
    }
}

void infixToPostfix(char infix[], char postfix[])
```

```
{
    struct stack s;
    initialize(&s);
    int i, j = 0;

    for (i = 0; infix[i]; i++)
    {
        char c = infix[i];

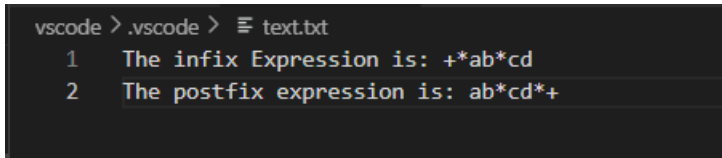
        if (isOperand(c))
        {
            postfix[j++] = c;
        }
        else if (c == '(')
        {
            push(&s, c);
        }
        else if (c == ')')
        {
            while (!isEmpty(&s) && peek(&s) != '(')
            {
                postfix[j++] = pop(&s);
            }
            if (!isEmpty(&s) && peek(&s) != '(')
            {
                printf("Invalid expression\n");
                return;
            }
            else
            {
                pop(&s);
            }
        }
        else if (isOperator(c))
        {
            while (!isEmpty(&s) && getPrecedence(c) <=
getPrecedence(peek(&s)))
            {
                postfix[j++] = pop(&s);
            }
        }
    }
}
```

```
        push(&s, c);
    }
}

while (!isEmpty(&s))
{
    postfix[j++] = pop(&s);
}

postfix[j] = '\0';
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1 The infix Expression is: +*ab*cd
2 The postfix expression is: ab*cd*+
```

PRIORITY QUEUE(3)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "C:\Users\shash\OneDrive\Desktop\vscode\.vscode\randG.h"

#define MAX_SIZE 10

struct PQElement
{
    int data;
    int priority;
};

struct PQueue
{
    struct PQElement items[MAX_SIZE];
    int size;
};

void displayQueue(struct PQueue *, FILE *);
void initializeQueue(struct PQueue *);
int isEmpty(struct PQueue *);
void push(struct PQueue *, int);

int main()
{
    int arr[100];
    int k=0,num;

    srand(time(NULL));

    struct PQueue queue;
    initializeQueue(&queue);

    randonGenrator(10);
    FILE *fptr;

    fptr=fopen("text.txt", "r");
```

```
if (fptr == NULL)
{
    printf("Error opening file.\n");
    return 1;
}
while(fscanf(fptr, "%d", &num) != -1)
{
    arr[k]=num;
    k++;
}
fclose(fptr);

fptr=fopen("text.txt","a");

printf("Pushing elements into the priority queue:\n");
for (int i = 0; i < 10; i++)
    push(&queue, arr[i]);

displayQueue(&queue, fptr);

fclose(fptr);

return 0;
}
void initializeQueue(struct PQueue *queue)
{
    queue->size = 0;
}

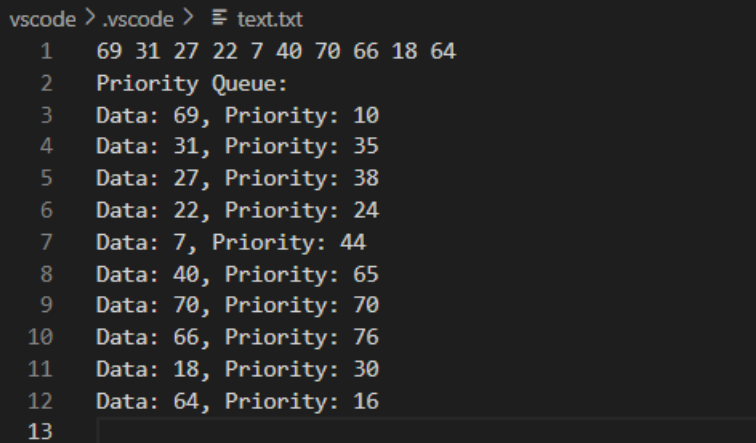
int isEmpty(struct PQueue *queue)
{
    return queue->size == 0;
}

void push(struct PQueue *queue, int data)
{
    if (queue->size==MAX_SIZE)
    {
        printf("Queue is full.\n");
        return 0;
    }
}
```

```
}
struct PQElement element;
element.data = data;
element.priority = rand() % 100;
queue->items[queue->size++] = element;
}

void displayQueue(struct PQueue *queue, FILE *file)
{
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    fprintf(file, "\nPriority Queue:\n");
    for (int i = 0; i < queue->size; i++) {
        fprintf(file, "Data: %d, Priority: %d\n", queue->items[i].data, queue->items[i].priority);
    }
}
```

OUTPUT



```
vscode > .vscode > ≡ text.txt
1  69 31 27 22 7 40 70 66 18 64
2  Priority Queue:
3  Data: 69, Priority: 10
4  Data: 31, Priority: 35
5  Data: 27, Priority: 38
6  Data: 22, Priority: 24
7  Data: 7, Priority: 44
8  Data: 40, Priority: 65
9  Data: 70, Priority: 70
10 Data: 66, Priority: 76
11 Data: 18, Priority: 30
12 Data: 64, Priority: 16
13
```