**Lab Manual 03**

**DATA STRUCTURE**

Submitted by: **Tahira Inam**

Roll No: 105

Instructor: Azka Mir

3rd Semester (Blue)



Department of Software Engineering

University of Sialkot

Contents

[Objectives 3](#_Toc185619659)

[Exercises 3](#_Toc185619660)

[Exercise 3.1: Calculator with factorial, sum, Fibonacci and power functions. 3](#_Toc185619661)

[Exercise 3.2: Manage Playlist of Songs using singly Linked List with reverse and forward display functions. 3](#_Toc185619662)

[Source Code and Outputs 3](#_Toc185619663)

[Exercise 3.1 Code 3](#_Toc185619664)

[Output: 7](#_Toc185619665)

[Exercise 3.2 Code 8](#_Toc185619666)

[Output: 12](#_Toc185619667)

## Objectives

1. Understanding the concepts of recursion and its applications.
2. Implement recursion in two contexts: simple function operations and using linked lists.
3. Demonstrate recursive techniques for traversing and processing data.

## Exercises

### Exercise 3.1: Calculator with factorial, sum, Fibonacci and power functions.

Create a calculator with the following functions:

1. Factorial(): Calculate factorial of 30 and display it.

2. Sum(): Calculate power of a number and display it.

3. FibonacciSeries(): Calculate power of a number and display it.

4. Power(): Calculate power of a number and display it.

Note: A menu should be created and input should be taken by the user.

### Exercise 3.2: Manage Playlist of Songs using singly Linked List with reverse and forward display functions.

(a) Implement a singly linked list of Songs Playlist with dynamic memory allocation.

(b) Write a function to recursively traverse the linked list in forward order.

(c) Write another function to recursively traverse the linked list in reverse order.

## Source Code and Outputs

### Exercise 3.1 Code

#include <iostream>

using namespace std;

class Calculator {

public:

// Factorial Function

int factorial(int n) {

if (n <= 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

// Summation Function

int sum(int n) {

if (n <= 1) {

return n;

} else {

return n + sum(n - 1);

}

}

// Recursive Fibonacci Function

int fibonacci(int n) {

if (n <= 1) {

return n;

} else {

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

// Display Fibonacci Series Function

void fibonacciSeries(int terms) {

cout << "\n\tFibonacci Series: ";

for (int i = 0; i < terms; ++i) {

cout << fibonacci(i) << " ";

}

cout << endl;

}

// Power Function

int power(int base, int exp) {

if (exp == 0) {

return 1;

} else {

return base \* power(base, exp - 1);

}

}

};

int main() {

Calculator calc;

int choice;

while(true) {

cout << "\n\n\t=== Calculator Menu ===\n";

cout << "\t1. Factorial\n";

cout << "\t2. Summation\n";

cout << "\t3. Fibonacci Series\n";

cout << "\t4. Power\n";

cout << "\t5. Exit\n";

cout << "\tEnter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int num;

cout << "\n\tEnter a number to calculate factorial: ";

cin >> num;

cout << "\tFactorial of " << num << " = " << calc.factorial(num) << endl;

break;

}

case 2: {

int num;

cout << "\n\tEnter a number to calculate summation: ";

cin >> num;

cout << "\tSummation of numbers up to " << num << " = " << calc.sum(num) << endl;

break;

}

case 3: {

int terms;

cout << "\n\tEnter the number of terms for the Fibonacci series: ";

cin >> terms;

calc.fibonacciSeries(terms);

break;

}

case 4: {

int base, exp;

cout << "\n\tEnter the base number: ";

cin >> base;

cout << "\tEnter the exponent: ";

cin >> exp;

cout <<"\t"<< base << "^" << exp << " = " << calc.power(base, exp) << endl;

break;

}

case 5:;

cout << "\n\tExiting the program...\n";

return 0;

break;

default:

cout << "\n\tInvalid choice! Please try again.\n";

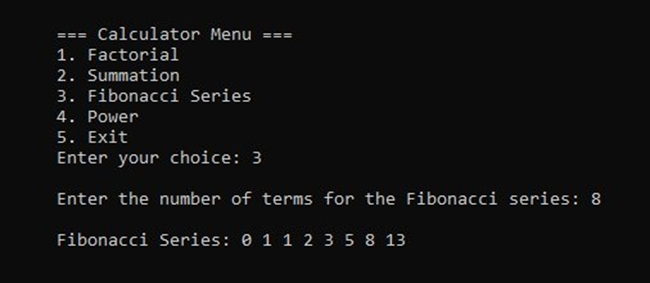
}

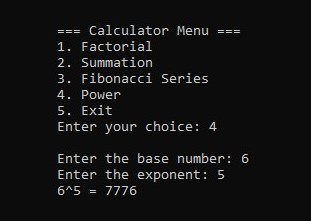
}

return 0;

}

### Output:





### Exercise 3.2 Code

#include <iostream>

#include <string>

using namespace std;

struct Node {

string songName;

Node\* next;

};

class Playlist {

public:

Node\* head;

Node\* tail;

Playlist() {

head = NULL;

tail = NULL;

}

// Function to add a song to the playlist

void insert() {

Node\* temp = new Node;

cout << "\n\tEnter Song Name: ";

cin.ignore();

getline(cin, temp->songName);

temp->next = NULL;

if (head == NULL) {

head = temp;

tail = temp;

} else {

tail->next = temp;

tail = temp;

}

cout << "\n\tSong added successfully!" << endl;

}

// Recursive function to traverse the playlist in forward order

void traverseForward(Node\* node) {

if (node == NULL) {

return;

}

cout << "\t" << node->songName << endl;

traverseForward(node->next);

}

// Recursive function to traverse the playlist in reverse order

void traverseReverse(Node\* node) {

if (node == NULL) {

return;

}

traverseReverse(node->next);

cout << "\t" << node->songName << endl;

}

// Function to print the playlist in forward order using recursion

void displayForward() {

cout << "\n\tPlaylist in Forward Order:" << endl;

traverseForward(head);

}

// Function to print the playlist in reverse order using recursion

void displayReverse() {

cout << "\n\tPlaylist in Reverse Order:" << endl;

traverseReverse(head);

}

};

// MAIN FUNCTION

int main() {

Playlist p;

while (true) {

cout << "\n\tMENU\n";

cout << "\n\t1. ADD SONG";

cout << "\n\t2. DISPLAY PLAYLIST IN FORWARD ORDER";

cout << "\n\t3. DISPLAY PLAYLIST IN REVERSE ORDER";

cout << "\n\t4. EXIT\n";

int choice;

cout << "\n\tEnter your choice: ";

cin >> choice;

switch (choice) {

case 1:

p.insert();

break;

case 2:

p.displayForward();

break;

case 3:

p.displayReverse();

break;

case 4:

return 0;

default:

cout << "\n\tINVALID CHOICE! Please try again." << endl;

}

}

return 0;

}

### Output: