Practical-1

Name : Jadhav Harshvardhan Vijaysinh

Reg. No.: 2020BIT065.

1 . Stack :

#include<iostream>

using namespace std;

const int MAX\_SIZE = 100;

class Stack {

    private:

        int data[MAX\_SIZE];

        int top;

    public:

        Stack() {

            top = -1;

        }

        void push(int x) {

            if (top == MAX\_SIZE - 1) {

                std::cout << "Stack overflow" << std::endl;

                return;

            }

            data[++top] = x;

        }

        void pop() {

            if (top == -1) {

                std::cout << "Stack underflow" << std::endl;

                return;

            }

            top--;

        }

        int peek() {

            return data[top];

        }

        bool isEmpty() {

            return top == -1;

        }

};

int main() {

    Stack s;

    s.push(1);

    s.push(2);

    s.push(3);

    std::cout << s.peek() << std::endl; // Output: 3

    s.pop();

    std::cout << s.peek() << std::endl; // Output: 2

    s.pop();

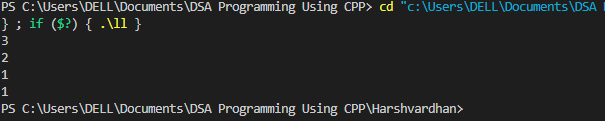
    std::cout << s.peek() << std::endl; // Output: 1

    s.pop();

    std::cout << s.isEmpty() << std::endl; // Output: 1

}

Output :



2 . Queue :

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

class Queue {

private:

    int front, rear;

    int arr[MAX\_SIZE];

public:

    Queue() {

        front = rear = -1;

    }

    bool isEmpty() {

        return front == -1 && rear == -1;

    }

    bool isFull() {

        return rear == MAX\_SIZE - 1;

    }

    void enqueue(int value) {

        if (isFull()) {

            cout << "Queue is full" << endl;

            return;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else {

            rear++;

        }

        arr[rear] = value;

    }

    void dequeue() {

        if (isEmpty()) {

            cout << "Queue is empty" << endl;

            return;

        }

        if (front == rear) {

            front = rear = -1;

        } else {

            front++;

        }

    }

    int peek() {

        if (isEmpty()) {

            cout << "Queue is empty" << endl;

            return -1;

        }

        return arr[front];

    }

};

int main() {

    Queue q;

    q.enqueue(1);

    q.enqueue(2);

    q.enqueue(3);

    cout << "Queue front: " << q.peek() << endl;

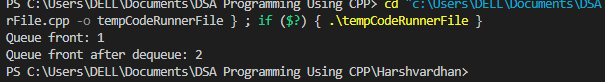
    q.dequeue();

    cout << "Queue front after dequeue: " << q.peek() << endl;

    return 0;

}

Output :



3 . Linked List :

#include <iostream>

struct Node {

    int data;

    Node\* next;

};

class LinkedList {

private:

    Node\* head;

public:

    LinkedList() {

        head = NULL;

    }

    void insert(int data) {

        Node\* newNode = new Node();

        newNode->data = data;

        newNode->next = head;

        head = newNode;

    }

    void display() {

        Node\* temp = head;

        while (temp != NULL) {

            std::cout << temp->data << " ";

            temp = temp->next;

        }

        std::cout << std::endl;

    }

};

int main() {

    LinkedList list;

    list.insert(5);

    list.insert(10);

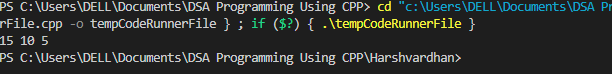
    list.insert(15);

    list.display();

    return 0;

}

Output:



4 . Trees :

#include <iostream>

class Node {

public:

    int data;

    Node\* left;

    Node\* right;

    Node(int data) {

        this->data = data;

        this->left = nullptr;

        this->right = nullptr;

    }

};

class BinaryTree {

private:

    Node\* root;

public:

    BinaryTree() {

        root = nullptr;

    }

    void insert(int data) {

        root = insertHelper(root, data);

    }

    Node\* insertHelper(Node\* node, int data) {

        if (node == nullptr) {

            node = new Node(data);

        } else if (data <= node->data) {

            node->left = insertHelper(node->left, data);

        } else {

            node->right = insertHelper(node->right, data);

        }

        return node;

    }

    bool search(int data) {

        return searchHelper(root, data);

    }

    bool searchHelper(Node\* node, int data) {

        if (node == nullptr) {

            return false;

        }

        if (node->data == data) {

            return true;

        }

        if (data <= node->data) {

            return searchHelper(node->left, data);

        } else {

            return searchHelper(node->right, data);

        }

    }

};

int main() {

    BinaryTree bt;

    bt.insert(5);

    bt.insert(3);

    bt.insert(7);

    bt.insert(2);

    bt.insert(4);

    bt.insert(6);

    bt.insert(8);

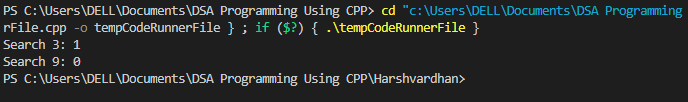
    std::cout << "Search 3: " << bt.search(3) << std::endl;

    std::cout << "Search 9: " << bt.search(9) << std::endl;

    return 0;

}

Output:



5 . Graphs :

#include <iostream>

#include <vector>

class Graph {

private:

    int V;

    std::vector<int> \*adj;

public:

    Graph(int V) {

        this->V = V;

        adj = new std::vector<int>[V];

    }

    void addEdge(int from, int to) {

        adj[from].push\_back(to);

    }

    void printGraph() {

        for (int i = 0; i < V; i++) {

            std::cout << "Adjacency list of vertex " << i << ": ";

            for (int j : adj[i]) {

                std::cout << j << " ";

            }

            std::cout << std::endl;

        }

    }

};

int main() {

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    g.printGraph();

    return 0;

}

Output :

