**PRAVEER RAJ**

**BATCH - MATHS AND COMPUTING**

**WEEK 1**

**ROLL NO. 1**

// QUESTION 1(LAB 0)

#include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int target) {

int left = 0, right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

}

if (arr[mid] > target) {

right = mid - 1;

}

else {

left = mid + 1;

}

}

return -1;

}

int main() {

int arr[] = {1, 3, 5, 7, 9, 11, 13};

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

int target = 7;

int result = binarySearch(arr, size, target);

if (result != -1) {

cout << "Element found at index " << result << endl;

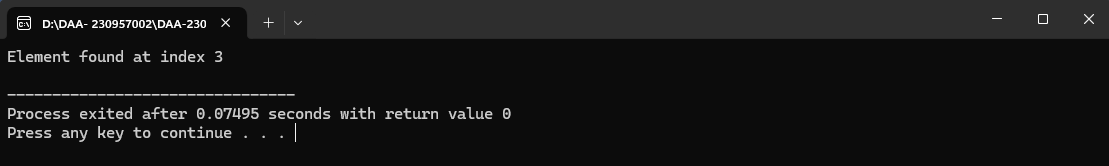
} else {

cout << "Element not found" << endl;

}

return 0;

}



// QUESTION 2(LAB 0)

#include <iostream>

using namespace std;

class Stack {

int arr[5];

int top;

public:

Stack() {

top = -1;

}

void push(int x) {

if (top >= 4) {

cout << "Stack Overflow" << endl;

} else {

arr[++top] = x;

}

}

void pop() {

if (top == -1) {

cout << "Stack Underflow" << endl;

} else {

top--;

}

}

void displayTop() {

if (top == -1) {

cout << "Stack is empty" << endl;

} else {

cout << "Top element: " << arr[top] << endl;

}

}

bool isEmpty() {

return top == -1;

}

};

int main() {

Stack s;

s.push(10);

s.push(20);

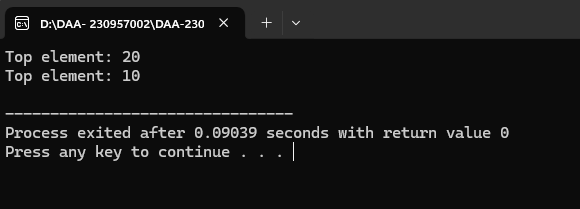
s.displayTop();

s.pop();

s.displayTop();

return 0;

}



// QUESTION 3(LAB 0)

#include <iostream>

using namespace std;

class Queue {

int arr[5];

int front, rear;

public:

Queue() {

front = -1;

rear = -1;

}

void enqueue(int x) {

if (rear >= 4) {

cout << "Queue Overflow" << endl;

} else {

if (front == -1) {

front = 0;

}

arr[++rear] = x;

}

}

void dequeue() {

if (front == -1 || front > rear) {

cout << "Queue Underflow" << endl;

} else {

front++;

}

}

void displayFront() {

if (front == -1 || front > rear) {

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

} else {

cout << "Front element: " << arr[front] << endl;

}

}

bool isEmpty() {

return front == -1 || front > rear;

}

};

int main() {

Queue q;

q.enqueue(10);

q.enqueue(20);

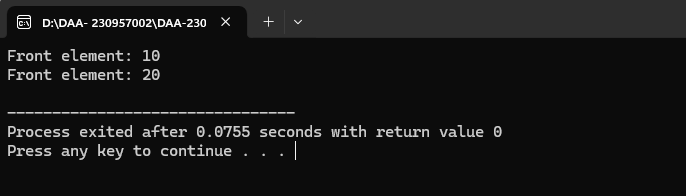
q.displayFront();

q.dequeue();

q.displayFront();

return 0;

}

***OUTPUT:-***

// QUESTION 4(LAB 0)

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int value) {

data = value;

next = prev = NULL;

}

};

class DoublyLinkedList {

Node\* head;

public:

DoublyLinkedList() {

head = NULL;

}

void insertAtBegin(int value) {

Node\* newNode = new Node(value);

if (head == NULL) {

head = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (head == NULL) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

void deleteFromBegin() {

if (head == NULL) {

cout << "List is empty" << endl;

} else {

Node\* temp = head;

head = head->next;

if (head != NULL) {

head->prev = NULL;

}

delete temp;

}

}

void deleteFromEnd() {

if (head == NULL) {

cout << "List is empty" << endl;

} else {

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

if (temp->prev != NULL) {

temp->prev->next = NULL;

} else {

head = NULL;

}

delete temp;

}

}

void traverseForward() {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

void traverseBackward() {

Node\* temp = head;

if (temp == NULL) return;

while (temp->next != NULL) {

temp = temp->next;

}

while (temp != NULL) {

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

temp = temp->prev;

}

cout << endl;

}

};

int main() {

DoublyLinkedList list;

list.insertAtBegin(10);

list.insertAtEnd(20);

list.traverseForward();

list.deleteFromBegin();

list.traverseForward();

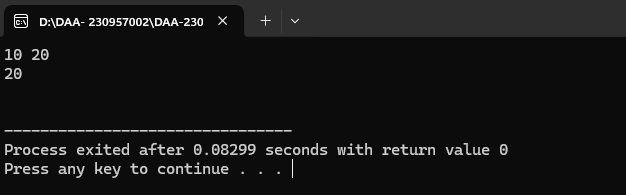
list.deleteFromEnd();

list.traverseForward();

return 0;

}

OUTPUT:-



// QUESTION 5(LAB 0)

#include <iostream>

using namespace std;

class Graph {

int adjMatrix[5][5];

int vertices;

public:

Graph(int v) {

vertices = v;

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

for (int j = 0; j < vertices; j++) {

adjMatrix[i][j] = 0;

}

}

}

void addEdge(int u, int v) {

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1;

}

void removeEdge(int u, int v) {

adjMatrix[u][v] = 0;

adjMatrix[v][u] = 0;

}

void display() {

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

for (int j = 0; j < vertices; j++) {

cout << adjMatrix[i][j] << " ";

}

cout << endl;

}

}

};

int main() {

Graph g(5);

g.addEdge(0, 1);

g.addEdge(1, 2);

g.display();

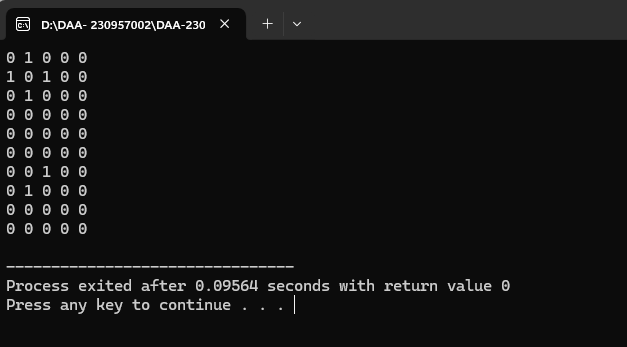
g.removeEdge(0, 1);

g.display();

return 0;

}

OUTPUT:-



// QUESTION 6(LAB 0)

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = NULL;

}

};

class BST {

private:

Node\* root; // Keep root private

public:

BST() {

root = NULL;

}

// Public method to access root safely

Node\* getRoot() {

return root;

}

// Public method to insert a value

void insert(int value) {

root = insertRec(root, value);

}

Node\* insertRec(Node\* root, int value) {

if (root == NULL) {

return new Node(value);

}

if (value < root->data) {

root->left = insertRec(root->left, value);

} else {

root->right = insertRec(root->right, value);

}

return root;

}

void inorder() {

inorderRec(root);

cout << endl;

}

void inorderRec(Node\* root) {

if (root != NULL) {

inorderRec(root->left);

cout << root->data << " ";

inorderRec(root->right);

}

}

// Modify search method to return Node\* instead of bool

Node\* search(int value) {

return searchRec(root, value);

}

Node\* searchRec(Node\* root, int value) {

if (root == NULL) return NULL; // Return NULL if node is not found

if (root->data == value) return root; // Return node if found

if (value < root->data) return searchRec(root->left, value); // Search left

return searchRec(root->right, value); // Search right

}

Node\* deleteNode(Node\* root, int value) {

if (root == NULL) return root;

if (value < root->data) {

root->left = deleteNode(root->left, value);

} else if (value > root->data) {

root->right = deleteNode(root->right, value);

} else {

if (root->left == NULL) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = minValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current && current->left != NULL) {

current = current->left;

}

return current;

}

};

int main() {

BST tree;

tree.insert(50);

tree.insert(30);

tree.insert(70);

tree.inorder(); // Print inorder traversal before deletion

// Search for node and delete it

Node\* node = tree.search(30);

if (node != NULL) {

// Use the public method getRoot() to get the root of the tree and update it after deletion

tree.deleteNode(tree.getRoot(), 30); // Delete node with value 30

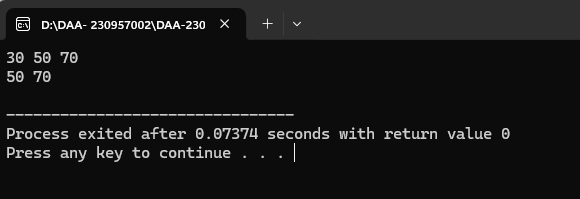
}

tree.inorder(); // Print inorder traversal after deletion

return 0;

}

OUTPUT:-



// QUESTION 1(LAB 1)

#include <iostream>

using namespace std;

int gcdEuclid(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int main() {

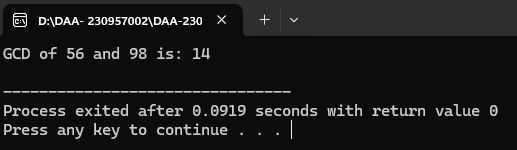
int a = 56, b = 98;

cout << "GCD of " << a << " and " << b << " is: " << gcdEuclid(a, b) << endl;

return 0;

}

OUTPUT:-



// QUESTION 2(LAB 2)

#include <iostream>

using namespace std;

int gcdConsecutive(int a, int b) {

int gcd = 1;

int minVal = (a < b) ? a : b;

for (int i = 1; i <= minVal; i++) {

if (a % i == 0 && b % i == 0) {

gcd = i;

}

}

return gcd;

}

int main() {

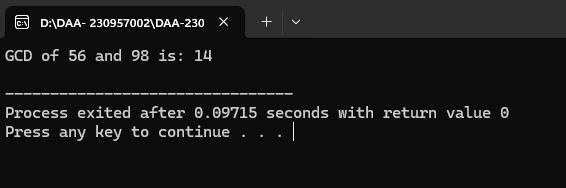
int a = 56, b = 98;

cout << "GCD of " << a << " and " << b << " is: " << gcdConsecutive(a, b) << endl;

return 0;

}

OUTPUT:-



// QUESTION 3(LAB 2)

#include <iostream>

using namespace std;

void selectionSort(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

int minIndex = i;

for (int j = i+1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

swap(arr[i], arr[minIndex]);

}

}

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > arr[j+1]) {

swap(arr[j], arr[j+1]);

}

}

}

}

void printArray(int arr[], int n) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int arr1[] = {64, 25, 12, 22, 11};

int arr2[] = {64, 25, 12, 22, 11};

int n = sizeof(arr1)/sizeof(arr1[0]);

selectionSort(arr1, n);

cout << "Sorted array using Selection Sort: ";

printArray(arr1, n);

bubbleSort(arr2, n);

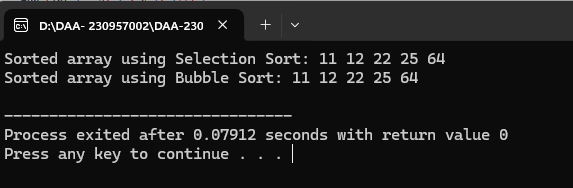
cout << "Sorted array using Bubble Sort: ";

printArray(arr2, n);

return 0;

}

OUTPUT:-



// QUESTION 4(LAB 2)

#include <iostream>

using namespace std;

int bruteForceStringMatch(string text, string pattern) {

int m = pattern.length();

int n = text.length();

for (int i = 0; i <= n - m; i++) {

int j;

for (j = 0; j < m; j++) {

if (text[i + j] != pattern[j]) {

break;

}

}

if (j == m) {

return i; // Match found

}

}

return -1; // No match found

}

int main() {

string text = "ABABABCABABABCABAB";

string pattern = "ABABC";

int result = bruteForceStringMatch(text, pattern);

if (result != -1) {

cout << "Pattern found at index " << result << endl;

} else {

cout << "Pattern not found" << endl;

}

return 0;

}

OUTPUT:-

