**Data Structure**

**Lab 1-7 Practice Tasks**

**1. Arrays**

**Task 1: Initialize an Array and Print All Elements**

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[] = {10, 20, 30, 40, 50};

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

cout << "Array elements: ";

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

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

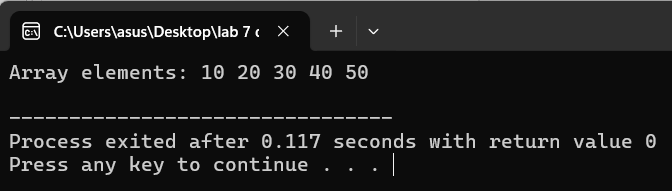
}

cout << endl;

return 0;

}

**Output:**



**Task 2: Find Maximum and Minimum Values**

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[] = {10, 20, 5, 25, 15};

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

int max = arr[0];

int min = arr[0];

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

if (arr[i] > max) max = arr[i];

if (arr[i] < min) min = arr[i];

}

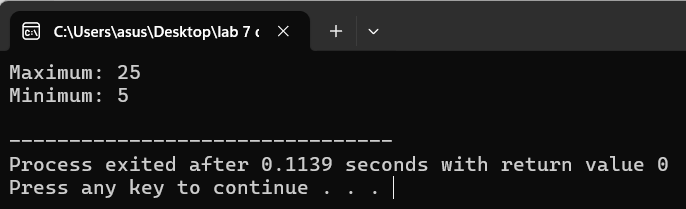
cout << "Maximum: " << max << endl;

cout << "Minimum: " << min << endl;

return 0;

}

**Output:**



**Task 3: Reverse the Elements of an Array**

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[] = {10, 20, 30, 40, 50};

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

cout << "Reversed array: ";

for (int i = size - 1; i >= 0; i--) {

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

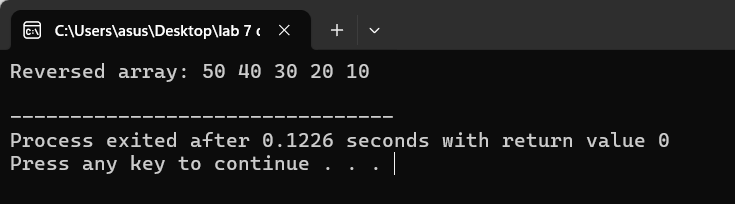
}

cout << endl;

return 0;

}

**Output:**



**2. Lists (Linked List)**

**Task 1: Add, Remove, and Display Elements in a Linked List**

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void add(Node\*& head, int data) {

Node\* newNode = new Node{data, nullptr};

if (!head) head = newNode;

else {

Node\* temp = head;

while (temp->next) temp = temp->next;

temp->next = newNode;

}

}

void remove(Node\*& head, int data) {

if (!head) return;

if (head->data == data) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* temp = head;

while (temp->next && temp->next->data != data) temp = temp->next;

if (temp->next) {

Node\* toDelete = temp->next;

temp->next = temp->next->next;

delete toDelete;

}

}

void display(Node\* head) {

while (head) {

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

int main() {

Node\* head = nullptr;

add(head, 10);

add(head, 20);

add(head, 30);

cout << "Linked list: ";

display(head);

remove(head, 20);

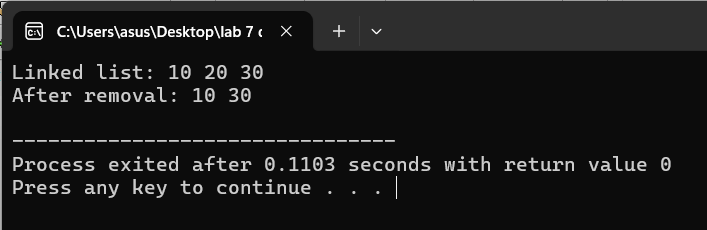
cout << "After removal: ";

display(head);

return 0;

}

**Output:**



**Task 2: Search for an Element in a Linked List**

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

bool search(Node\* head, int key) {

while (head) {

if (head->data == key) return true;

head = head->next;

}

return false;

}

int main() {

Node\* head = new Node{10, new Node{20, new Node{30, nullptr}}};

int key = 20;

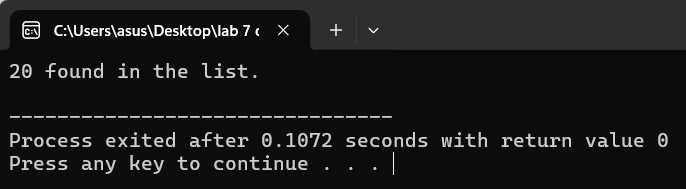
if (search(head, key)) cout << key << " found in the list." << endl;

else cout << key << " not found in the list." << endl;

return 0;

}

**Output:**



**Task 3: Insert an Element at the Beginning, Middle, and End**

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

void insertBeginning(Node\*& head, int data) {

head = new Node{data, head};

}

void insertEnd(Node\*& head, int data) {

Node\* newNode = new Node{data, nullptr};

if (!head) head = newNode;

else {

Node\* temp = head;

while (temp->next) temp = temp->next;

temp->next = newNode;

}

}

void insertMiddle(Node\* head, int data, int position) {

Node\* temp = head;

for (int i = 1; i < position - 1 && temp; i++) temp = temp->next;

if (temp) temp->next = new Node{data, temp->next};

}

void display(Node\* head) {

while (head) {

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

int main() {

Node\* head = nullptr;

insertBeginning(head, 10);

insertEnd(head, 30);

insertMiddle(head, 20, 2);

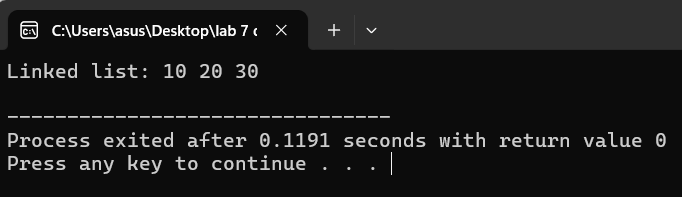
cout << "Linked list: ";

display(head);

return 0;

}

**Output:**



**3. Stacks**

**Task 1: Implement a Stack Using an Array**

**Code:**

#include <iostream>

#define MAX 100

using namespace std;

class Stack {

int arr[MAX];

int top;

public:

Stack() : top(-1) {}

void push(int data) {

if (top >= MAX - 1) {

cout << "Stack overflow\n";

} else {

arr[++top] = data;

}

}

void pop() {

if (top < 0) {

cout << "Stack underflow\n";

} else {

cout << "Popped: " << arr[top--] << endl;

}

}

void display() {

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

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

}

cout << endl;

}

};

int main() {

Stack stack;

stack.push(10);

stack.push(20);

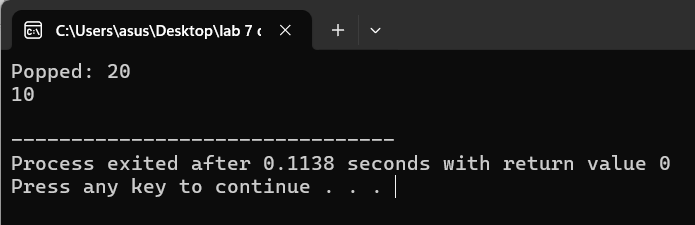
stack.pop();

stack.display();

return 0;

}

**Output:**



**Task 2: Check if a Given String of Parentheses is Balanced**

**Code:**

#include <iostream>

#include <stack>

using namespace std;

bool isBalanced(string str) {

stack<char> s;

for (char ch : str) {

if (ch == '(') s.push(ch);

else if (ch == ')') {

if (s.empty()) return false;

s.pop();

}

}

return s.empty();

}

int main() {

string str = "(())";

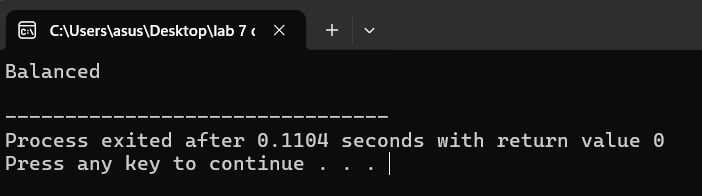
if (isBalanced(str)) cout << "Balanced\n";

else cout << "Not Balanced\n";

return 0;

}

**Output:**



**Task 3: Reverse a String Using a Stack**

**Code:**

#include <iostream>

#include <stack>

using namespace std;

void reverseString(string str) {

stack<char> s;

for (char ch : str) s.push(ch);

while (!s.empty()) {

cout << s.top();

s.pop();

}

cout << endl;

}

int main() {

string str = "hello";

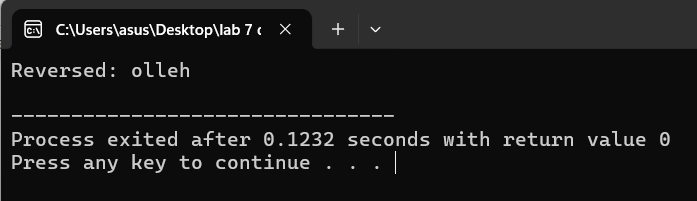
cout << "Reversed: ";

reverseString(str);

return 0;

}

**Output:**



**4. Queues**

**Task 1: Check if Queue Forms a Palindrome**

**Code:**

#include <iostream>

#include <queue>

#include <stack>

using namespace std;

bool isPalindrome(queue<int> q) {

stack<int> s;

int n = q.size();

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

int front = q.front();

s.push(front);

q.pop();

q.push(front);

}

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

int front = q.front();

if (front != s.top()) return false;

q.pop();

s.pop();

}

return true;

}

int main() {

queue<int> q;

q.push(1); q.push(2); q.push(3); q.push(2); q.push(1);

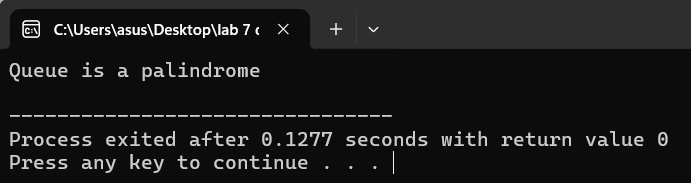
if (isPalindrome(q)) cout << "Queue is a palindrome\n";

else cout << "Queue is not a palindrome\n";

return 0;

}

**Output:**



**Task 2: Count Total Number of Elements in a Queue Without Modifying the Order**

**Code:**

#include <iostream>

#include <queue>

using namespace std;

int countQueue(queue<int> q) {

return q.size();

}

int main() {

queue<int> q;

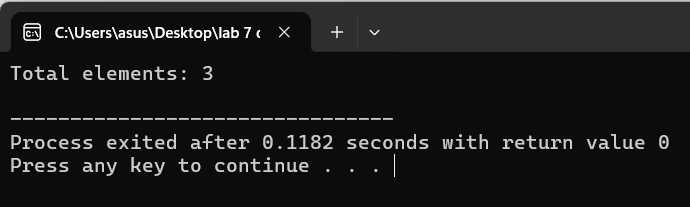
q.push(1); q.push(2); q.push(3);

cout << "Total elements: " << countQueue(q) << endl;

return 0;

}

**Output:**



**Task 3: Simulate a Ticket Queue**

**Code:**

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<string> ticketQueue;

ticketQueue.push("Alice");

ticketQueue.push("Bob");

ticketQueue.push("Charlie");

cout << "Processing queue:\n";

while (!ticketQueue

.empty()) {

cout << ticketQueue.front() << " is served.\n";

ticketQueue.pop();

}

return 0;

}

**Output:**

**5. Vectors**

**Task 1: Add Elements and Display Size and Capacity**

**Code:**

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<int> v;

for (int i = 1; i <= 5; i++) {

v.push\_back(i \* 10);

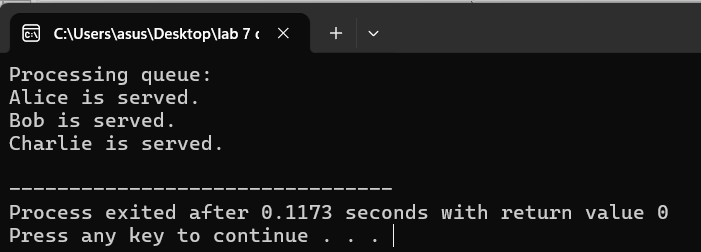
cout << "Size: " << v.size() << ", Capacity: " << v.capacity() << endl;

}

return 0;

}

**Output:**



**Task 2: Remove Duplicate Values**

**Code:**

#include <iostream>

#include <vector>

#include <unordered\_set>

using namespace std;

void removeDuplicates(vector<int>& v) {

unordered\_set<int> seen;

auto it = v.begin();

while (it != v.end()) {

if (seen.find(\*it) != seen.end()) it = v.erase(it);

else {

seen.insert(\*it);

++it;

}

}

}

int main() {

vector<int> v = {10, 20, 10, 30, 20, 40};

removeDuplicates(v);

cout << "Vector without duplicates: ";

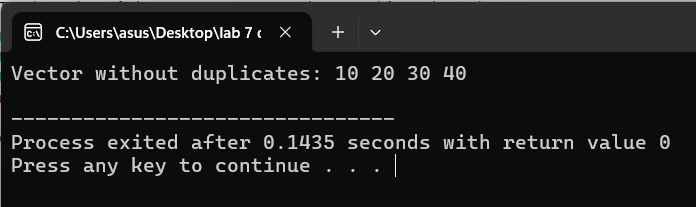
for (int num : v) cout << num << " ";

cout << endl;

return 0;

}

**Output:**



**Task 3: Sort Vector in Ascending Order**

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

vector<int> v = {50, 10, 40, 20, 30};

sort(v.begin(), v.end());

cout << "Sorted vector: ";

for (int num : v) cout << num << " ";

cout << endl;

return 0;

}

**Output:**

