**Tasks for practicing arrays, stacks, and lists**

**Array Tasks**

**1. Sum and Average:** Create an array of integers with 10 elements. Write a program to calculate and display the sum and average of the array elements.

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[10] = {12, 45, 23, 67, 34, 89, 56, 10, 78, 90};

int sum = 0;

float average;

// Calculate sum

for (int i = 0; i < 10; i++) {

sum += arr[i];

}

// Calculate average

average = sum / 10.0;

// Output sum and average

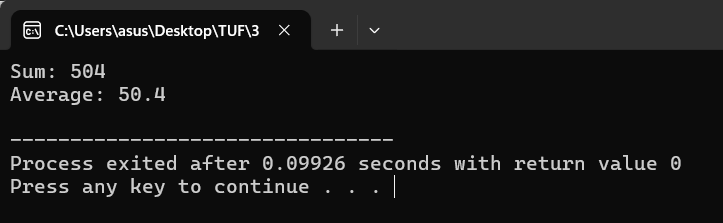
cout << "Sum: " << sum << endl;

cout << "Average: " << average << endl;

return 0;

}

**Output:**

**2. Max and Min:** Given an array of 5 integers, write a program to find the maximum and minimum values in the array.

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[5] = {9, 5, 2, 11, 3};

int maxVal = arr[0];

int minVal = arr[0];

// Find max and min

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

if (arr[i] > maxVal) {

maxVal = arr[i];

}

if (arr[i] < minVal) {

minVal = arr[i];

}

}

// Output max and min

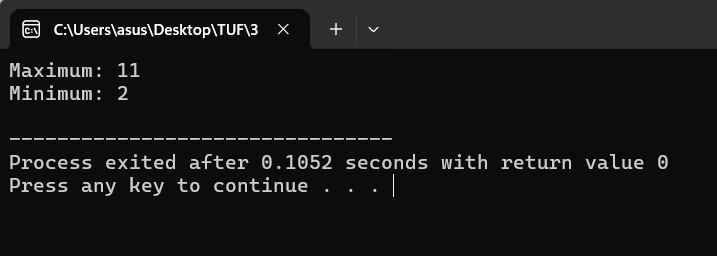
cout << "Maximum: " << maxVal << endl;

cout << "Minimum: " << minVal << endl;

return 0;

}

**Output:**

****

**3. Reverse an Array:** Write a program to reverse the elements of an array with 8 elements and display the reversed array.

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[8] = {1, 2, 3, 4, 5, 6, 7, 8};

cout << "Original Array: ";

for (int i = 0; i < 8; i++) {

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

}

cout << endl;

// Reverse array

cout << "Reversed Array: ";

for (int i = 7; i >= 0; i--) {

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

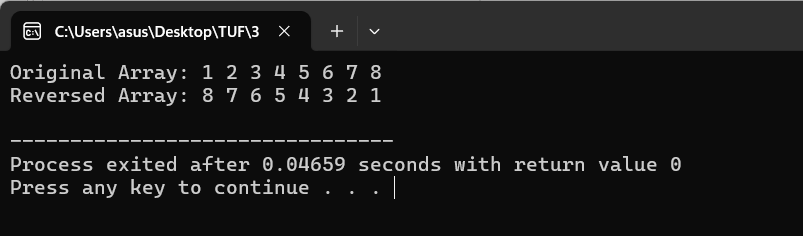
}

cout << endl;

return 0;

}

**Output:**

****

**4. Count Occurrences:** Create an array of characters. Write a program to count how many times a specific character (input by the user) appears in the array.

**Code:**

#include <iostream>

using namespace std;

int main() {

char arr[] = {'a', 'b', 'c', 'a', 'd', 'e', 'f', 'a', 'g', 'h'};

char searchChar;

int count = 0;

cout << "Enter the character to search: ";

cin >> searchChar;

// Count occurrences

for (int i = 0; i < 10; i++) {

if (arr[i] == searchChar) {

count++;

}

}

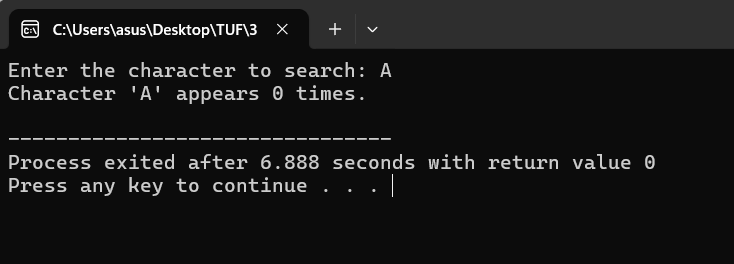
// Output count

cout << "Character '" << searchChar << "' appears " << count << " times." << endl;

return 0;

}

**Output:**

****

**5. Remove Duplicates:** Write a program that removes duplicate elements from an array of integers and prints the unique values.

**Code:**

#include <iostream>

using namespace std;

int main() {

int arr[10] = {1, 2, 2, 3, 4, 5, 5, 6, 7, 7};

int n = 10;

// Remove duplicates

cout << "Array after removing duplicates: ";

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

bool isDuplicate = false;

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

if (arr[i] == arr[j]) {

isDuplicate = true;

break;

}

}

if (!isDuplicate) {

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

}

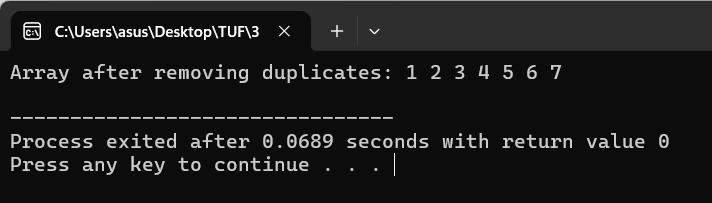
}

cout << endl;

return 0;

}

**Output:**



**Stack Tasks**

**1. Palindrome Check:** Use a stack to check if a word entered by the user is a palindrome (a word that reads the same backward and forward).

**Code:**

#include <iostream>

#include <stack>

using namespace std;

bool isPalindrome(string str) {

stack<char> s;

for (char c : str) {

s.push(c); // Push each character onto the stack

}

for (char c : str) {

if (c != s.top()) { // Compare each character with the stack's top element

return false;

}

s.pop();

}

return true;

}

int main() {

string word;

cout << "Enter a word: ";

cin >> word;

if (isPalindrome(word)) {

cout << word << " is a palindrome!" << endl;

} else {

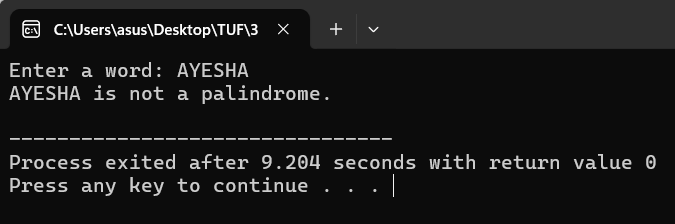
cout << word << " is not a palindrome." << endl;

}

return 0;

}

**Output:**

****

**2. Balanced Parentheses:** Write a program using a stack to check if a string of parentheses (`(), {}, []`) is balanced. For example, `(())` is balanced, but `(()` is not.

**Code:**

#include <iostream>

#include <stack>

using namespace std;

bool isBalanced(string str) {

stack<char> s;

for (char c : str) {

if (c == '(' || c == '{' || c == '[') {

s.push(c);

} else if (c == ')' && !s.empty() && s.top() == '(') {

s.pop();

} else if (c == '}' && !s.empty() && s.top() == '{') {

s.pop();

} else if (c == ']' && !s.empty() && s.top() == '[') {

s.pop();

} else {

return false;

}

}

return s.empty();

}

int main() {

string expression;

cout << "Enter an expression: ";

cin >> expression;

if (isBalanced(expression)) {

cout << "The parentheses are balanced." << endl;

} else {

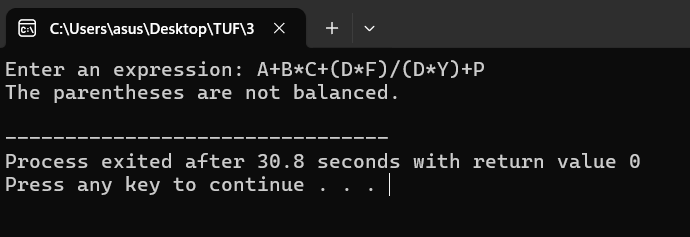
cout << "The parentheses are not balanced." << endl;

}

return 0;

}

**Output:**

****

**3. Browser History:** Simulate a basic browser history using a stack. Allow the user to "visit" a new page (push) and "go back" to the previous page (pop).

**Code:**

#include <iostream>

#include <stack>

using namespace std;

int main() {

stack<string> history;

string currentPage;

int choice;

do {

cout << "1. Visit new page\n2. Go back\n3. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

cout << "Enter URL: ";

cin >> currentPage;

history.push(currentPage);

cout << "You are now on: " << currentPage << endl;

} else if (choice == 2) {

if (!history.empty()) {

history.pop();

if (!history.empty()) {

cout << "You are now on: " << history.top() << endl;

} else {

cout << "No more history to go back to." << endl;

}

} else {

cout << "No history available." << endl;

}

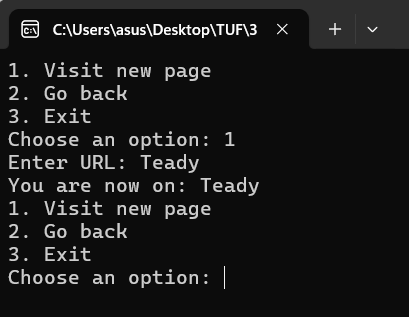
}

} while (choice != 3);

return 0;

}

**Output:**

****

**4. Evaluate Postfix Expression:** Given a postfix expression (e.g., "5 1 2 + 4 \* + 3 -"), write a program to evaluate it using a stack.

**Code:**

#include <iostream>

#include <stack>

#include <sstream>

using namespace std;

int evaluatePostfix(string expression) {

stack<int> s;

stringstream ss(expression);

string token;

while (ss >> token) {

if (isdigit(token[0])) {

s.push(stoi(token));

} else {

int val2 = s.top(); s.pop();

int val1 = s.top(); s.pop();

if (token == "+") s.push(val1 + val2);

else if (token == "-") s.push(val1 - val2);

else if (token == "\*") s.push(val1 \* val2);

else if (token == "/") s.push(val1 / val2);

}

}

return s.top();

}

int main() {

string expression = "5 1 2 + 4 \* + 3 -";

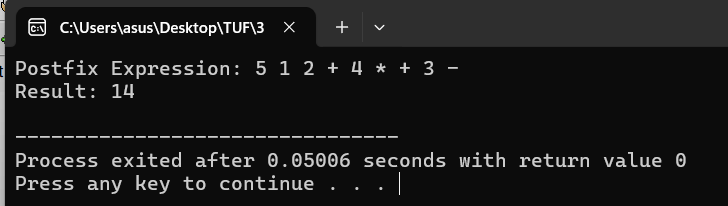
cout << "Postfix Expression: " << expression << endl;

cout << "Result: " << evaluatePostfix(expression) << endl;

return 0;

}

**Output:**

****

**5. Undo Function:** Implement an "Undo" function for a text editor. Each action (input by the user) is pushed to the stack, and when the user chooses "Undo," the last action is popped.

**Code:**

#include <iostream>

#include <stack>

using namespace std;

int main() {

stack<string> actions;

string action;

int choice;

do {

cout << "1. Perform Action\n2. Undo Last Action\n3. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

cout << "Enter action: ";

cin >> action;

actions.push(action);

cout << "Performed action: " << action << endl;

} else if (choice == 2) {

if (!actions.empty()) {

cout << "Undoing action: " << actions.top() << endl;

actions.pop();

} else {

cout << "No actions to undo." << endl;

}

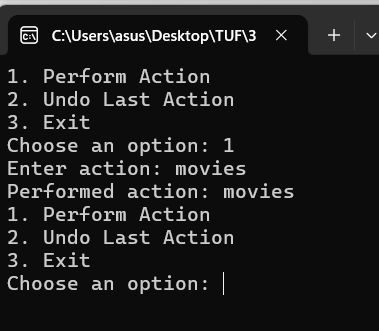
}

} while (choice != 3);

return 0;

}

**Output:**



**List Tasks**

**1. Shopping List:** Create a program that allows the user to add items to a shopping list (using a linked list), remove items, and display the current list.

**Code:**

#include <iostream>

#include <list>

using namespace std;

int main() {

list<string> shoppingList;

string item;

int choice;

do {

cout << "1. Add Item\n2. Remove Item\n3. Display List\n4. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

cout << "Enter item to add: ";

cin >> item;

shoppingList.push\_back(item);

} else if (choice == 2) {

cout << "Enter item to remove: ";

cin >> item;

shoppingList.remove(item);

} else if (choice == 3) {

cout << "Shopping List: ";

for (string &i : shoppingList) {

cout << i << " ";

}

cout << endl;

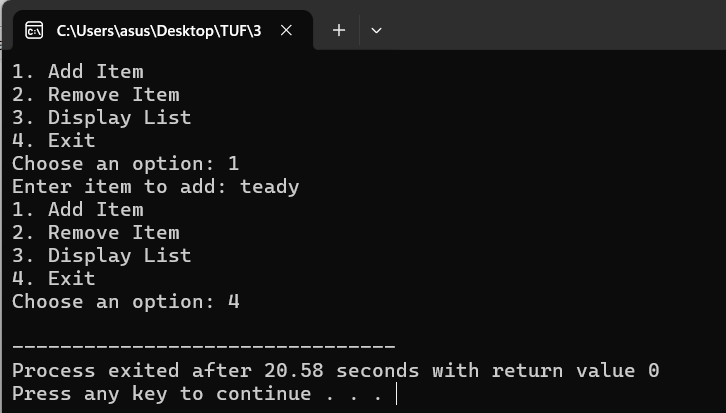
}

} while (choice != 4);

return 0;

}

**Output:**

****

**2. Task Manager:** Write a program to manage tasks using a list. Each task has a priority (e.g., high, medium, low). The user can add, remove, and display tasks in priority order.

**Code:**

#include <iostream>

#include <list>

using namespace std;

struct Task {

string name;

string priority;

};

bool compareTasks(const Task &a, const Task &b) {

if (a.priority == "high" && b.priority != "high") return true;

if (a.priority == "medium" && b.priority == "low") return true;

return false;

}

int main() {

list<Task> taskList;

Task task;

int choice;

do {

cout << "1. Add Task\n2. Remove Task\n3. Display Tasks\n4. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

cout << "Enter task name: ";

cin >> task.name;

cout << "Enter priority (high, medium, low): ";

cin >> task.priority;

taskList.push\_back(task);

taskList.sort(compareTasks);

} else if (choice == 2) {

cout << "Enter task name to remove: ";

cin >> task.name;

taskList.remove\_if([task](const Task &t) { return t.name == task.name; });

} else if (choice == 3) {

cout << "Task List:\n";

for (const Task &t : taskList) {

cout << t.name << " (" << t.priority << ")" << endl;

}

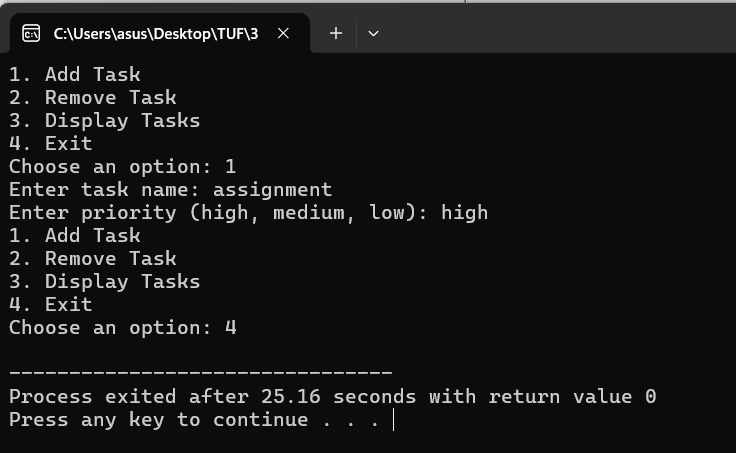
}

} while (choice != 4);

return 0;

}

**Output:**

****

**3. Student Roll Call:** Create a list of student names. Allow the user to add or remove names, and print the list in alphabetical order.

**Code:**

#include <iostream>

#include <list>

#include <algorithm>

using namespace std;

int main() {

list<string> studentList;

string name;

int choice;

do {

cout << "1. Add Student\n2. Remove Student\n3. Display Students\n4. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

cout << "Enter student name: ";

cin >> name;

studentList.push\_back(name);

studentList.sort(); // Sort the list alphabetically

} else if (choice == 2) {

cout << "Enter student name to remove: ";

cin >> name;

studentList.remove(name);

} else if (choice == 3) {

cout << "Student List:\n";

for (const string &student : studentList) {

cout << student << endl;

}

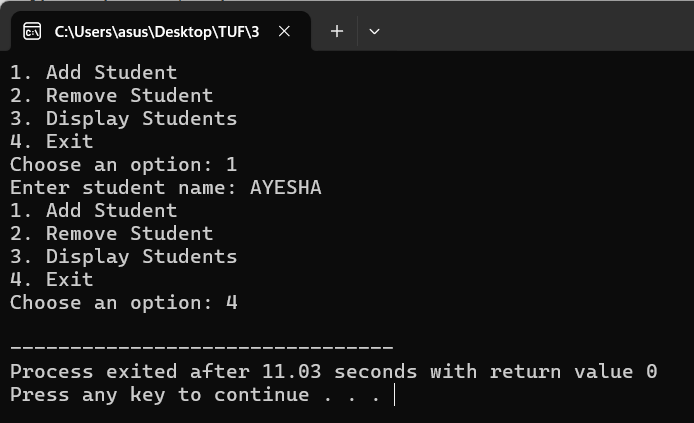
}

} while (choice != 4);

return 0;

}

**Output:**

****

**4. FIFO Queue Simulation**: Using a list, simulate a first-in-first-out (FIFO) queue for a ticketing system. Allow users to "take a ticket" (add to the end) and "serve a ticket" (remove from the front).

**Code:**

#include <iostream>

#include <list>

using namespace std;

int main() {

list<int> ticketQueue;

int ticketNumber = 1; // Start ticket number from 1

int choice;

do {

cout << "1. Take Ticket\n2. Serve Ticket\n3. Display Queue\n4. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

ticketQueue.push\_back(ticketNumber);

cout << "Ticket " << ticketNumber << " taken." << endl;

ticketNumber++; // Increment ticket number for the next person

} else if (choice == 2) {

if (!ticketQueue.empty()) {

cout << "Serving ticket: " << ticketQueue.front() << endl;

ticketQueue.pop\_front(); // Serve the first ticket in the queue

} else {

cout << "No tickets to serve." << endl;

}

} else if (choice == 3) {

cout << "Current Queue: ";

for (int ticket : ticketQueue) {

cout << ticket << " ";

}

cout << endl;

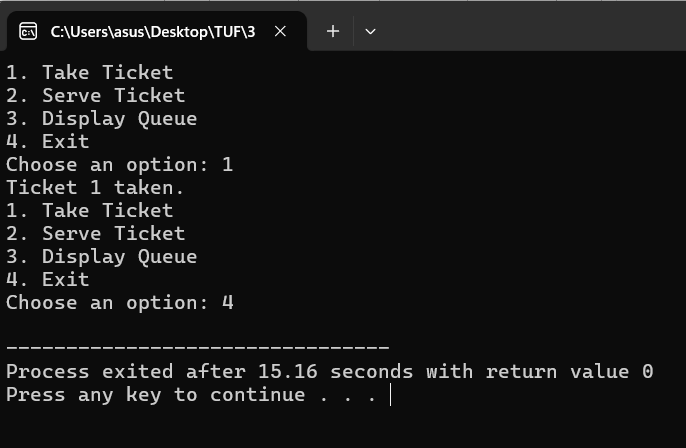
}

} while (choice != 4);

return 0;

}

**Output:**

****

**5. Playlist Manager:** Create a music playlist manager where the user can add songs at the beginning, end, or specific position, remove songs, and display the playlist.

**Code:**

#include <iostream>

#include <list>

using namespace std;

int main() {

list<string> playlist;

string song;

int choice, position;

do {

cout << "1. Add Song to Beginning\n2. Add Song to End\n3. Add Song at Position\n";

cout << "4. Remove Song\n5. Display Playlist\n6. Exit\n";

cout << "Choose an option: ";

cin >> choice;

if (choice == 1) {

cout << "Enter song name: ";

cin >> song;

playlist.push\_front(song); // Add song to the beginning

} else if (choice == 2) {

cout << "Enter song name: ";

cin >> song;

playlist.push\_back(song); // Add song to the end

} else if (choice == 3) {

cout << "Enter song name: ";

cin >> song;

cout << "Enter position: ";

cin >> position;

auto it = playlist.begin();

advance(it, position - 1); // Move iterator to the specific position

playlist.insert(it, song);

} else if (choice == 4) {

cout << "Enter song name to remove: ";

cin >> song;

playlist.remove(song); // Remove the specified song

} else if (choice == 5) {

cout << "Playlist:\n";

for (const string &s : playlist) {

cout << s << endl;

}

}

} while (choice != 6);

return 0;

}

**Output:**

