**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.

#include <iostream>

using namespace std;

int main(){

int size = 10;

int arr[size] = {4,6,12,64,82,4,65,42,12,10};

int sum = 0;

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

sum += arr[i];

}

float avg = float(sum)/float(size);

cout<<sum<<endl;

cout<<avg;

}

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

#include <iostream>

using namespace std;

int main(){

int size = 5;

int arr[size] = {6,82,4,64,12};

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<<max<<endl;

cout<<min<<endl;

}

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

#include <iostream>

using namespace std;

int main(){

int size = 8;

int arr[size] = {4,6,12,64,82,4,65,42};

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

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

}

}

**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.

#include <iostream>

using namespace std;

int main(){

int size = 5;

char arr[size] = {'A','B','C','A','B'};

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

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

}

cout<<endl;

char ch;

cout<<"Find Occurence: ";

cin>>ch;

int count = 0;

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

if(ch==arr[i]){

count++;

}

}

cout<<"Occurence: "<<count;

}

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

#include <iostream>

using namespace std;

int main() {

int size = 10;

int arr[size] = {4, 6, 12, 64, 82, 4, 65, 42, 12, 10};

int unique[size];

int uniqueCount = 0;

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

bool isDuplicate = false;

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

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

isDuplicate = true;

break;

}

}

if (!isDuplicate) {

unique[uniqueCount] = arr[i];

uniqueCount++;

}

}

cout << "Unique elements: ";

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

cout << unique[i] << " ";

}

return 0;

}

**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).

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

string word, revWord;

cout << "Enter a word: ";

cin >> word;

revWord = word;

reverse(revWord.begin(), revWord.end());

if (word == revWord) {

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

} else {

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

}

return 0;

}

**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.

#include <iostream>

#include <stack>

bool isBalanced(string expression) {

stack<char> s;

for (char c : expression) {

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

s.push(c);

}

else if (c == ')' || c == '}' || c == ']') {

if (s.empty()) return false;

char top = s.top();

if ((c == ')' && top != '(') ||

(c == '}' && top != '{') ||

(c == ']' && top != '[')) {

return false;

}

s.pop();

}

}

return s.empty();

}

int main() {

string expression;

cout << "Enter an expression with parentheses: ";

cin >> expression;

if (isBalanced(expression)) {

cout << "The expression is balanced." << endl;

} else {

cout << "The expression is not balanced." << endl;

}

return 0;

}

**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).

#include <iostream>

#include <stack>

#include <string>

int main() {

stack<string> history;

string currentPage;

while (true) {

cout << "\nCurrent Page: " << (currentPage.empty() ? "None" : currentPage) << endl;

cout << "Enter command (visit <page>, back, exit): ";

string command;

getline(cin, command);

if (command.rfind("visit ", 0) == 0) {

if (!currentPage.empty()) {

history.push(currentPage);

}

currentPage = command.substr(6);

cout << "Visiting: " << currentPage << endl;

} else if (command == "back") {

if (!history.empty()) {

currentPage = history.top();

history.pop();

cout << "Going back to: " << currentPage << endl;

} else {

cout << "No more pages in history." << endl;

}

} else if (command == "exit") {

break;

} else {

cout << "Invalid command." << endl;

}

}

return 0;

}

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

#include <iostream>

#include <stack>

#include <sstream>

#include <string>

using namespace std;

int evaluatePostfix(const string& expression) {

stack<int> s; // Stack to hold integers

stringstream ss(expression);

string token;

while (ss >> token) {

// If the token is a number, push it onto the stack

if (isdigit(token[0])) {

s.push(stoi(token));

} else {

// The token is an operator

int operand2 = s.top(); s.pop(); // Pop top element (second operand)

int operand1 = s.top(); s.pop(); // Pop next element (first operand)

switch (token[0]) {

case '+':

s.push(operand1 + operand2);

break;

case '-':

s.push(operand1 - operand2);

break;

case '\*':

s.push(operand1 \* operand2);

break;

case '/':

s.push(operand1 / operand2);

break;

default:

cerr << "Unknown operator: " << token << endl;

return 0; // Handle unexpected operators

}

}

}

return s.top(); // The final result will be at the top of the stack

}

int main() {

string expression;

cout << "Enter a postfix expression: ";

getline(cin, expression); // Read the entire line for the expression

int result = evaluatePostfix(expression);

cout << "The result of the postfix expression is: " << result << endl;

return 0;

}

**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.

#include <iostream>

#include <stack>

#include <string>

using namespace std;

void displayMenu() {

cout << "\nText Editor Menu:" << endl;

cout << "1. Type text" << endl;

cout << "2. Undo last action" << endl;

cout << "3. Display current text" << endl;

cout << "4. Exit" << endl;

}

int main() {

stack<string> actions;

string currentText;

int choice;

do {

displayMenu();

cout << "Choose an option (1-4): ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1: {

string inputText;

cout << "Enter text to type: ";

getline(cin, inputText);

actions.push(currentText);

currentText += inputText;

cout << "Text added." << endl;

break;

}

case 2:

if (!actions.empty()) {

currentText = actions.top();

actions.pop();

cout << "Last action undone." << endl;

} else {

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

}

break;

case 3:

cout << "Current text: \"" << currentText << "\"" << endl;

break;

case 4:

cout << "Exiting the program." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 4);

return 0;

}

**Link List**

**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.

#include <iostream>

#include <string>

using namespace std;

struct Node {

    string item;

    Node\* next;

};

class ShoppingList {

private:

    Node\* head;

public:

    ShoppingList() : head(nullptr) {}

    void addItem(const string& item) {

        Node\* newNode = new Node();

        newNode->item = item;

        newNode->next = head;

        head = newNode;

        cout << item << " has been added to the shopping list." << endl;

    }

    void removeItem(const string& item) {

        Node\* current = head;

        Node\* previous = nullptr;

        while (current != nullptr) {

            if (current->item == item) {

                if (previous == nullptr) {

                    head = current->next;

                } else {

                    previous->next = current->next;

                }

                delete current;

                cout << item << " has been removed from the shopping list." << endl;

                return;

            }

            previous = current;

            current = current->next;

        }

        cout << item << " not found in the shopping list." << endl;

    }

    void displayList() const {

        Node\* current = head;

        if (current == nullptr) {

            cout << "Shopping list is empty." << endl;

            return;

        }

        cout << "Shopping List:" << endl;

        while (current != nullptr) {

            cout << "- " << current->item << endl;

            current = current->next;

        }

    }

    ~ShoppingList() {

        Node\* current = head;

        while (current != nullptr) {

            Node\* next = current->next;

            delete current;

            current = next;

        }

    }

};

int main() {

    ShoppingList list;

    int choice;

    string item;

    do {

        cout << "\nShopping List Menu:" << endl;

        cout << "1. Add Item" << endl;

        cout << "2. Remove Item" << endl;

        cout << "3. Display List" << endl;

        cout << "4. Exit" << endl;

        cout << "Choose an option (1-4): ";

        cin >> choice;

        cin.ignore();

        switch (choice) {

            case 1:

                cout << "Enter item to add: ";

                getline(cin, item);

                list.addItem(item);

                break;

            case 2:

                cout << "Enter item to remove: ";

                getline(cin, item);

                list.removeItem(item);

                break;

            case 3:

                list.displayList();

                break;

            case 4:

                cout << "Exiting the program." << endl;

                break;

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

    } while (choice != 4);

    return 0;

}

**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.

#include <iostream>

#include <string>

using namespace std;

enum class Priority {

    High,

    Medium,

    Low

};

struct TaskNode {

    string task;

    Priority priority;

    TaskNode\* next;

};

class TaskManager {

private:

    TaskNode\* head;

public:

    TaskManager() : head(nullptr) {}

    void addTask(const string& task, Priority priority) {

        TaskNode\* newNode = new TaskNode();

        newNode->task = task;

        newNode->priority = priority;

        newNode->next = nullptr;

        if (head == nullptr || head->priority > priority) {

            newNode->next = head;

            head = newNode;

        } else {

            TaskNode\* current = head;

            while (current->next != nullptr && current->next->priority <= priority) {

                current = current->next;

            }

            newNode->next = current->next;

            current->next = newNode;

        }

        cout << "Task \"" << task << "\" with priority " << (int)priority << " has been added." << endl;

    }

    void removeTask(const string& task) {

        TaskNode\* current = head;

        TaskNode\* previous = nullptr;

        while (current != nullptr) {

            if (current->task == task) {

                if (previous == nullptr) {

                    head = current->next;

                } else {

                    previous->next = current->next;

                }

                delete current;

                cout << "Task \"" << task << "\" has been removed." << endl;

                return;

            }

            previous = current;

            current = current->next;

        }

        cout << "Task \"" << task << "\" not found." << endl;

    }

    void displayTasks() const {

        if (head == nullptr) {

            cout << "No tasks in the list." << endl;

            return;

        }

        cout << "Tasks:" << endl;

        TaskNode\* current = head;

        while (current != nullptr) {

            cout << "- " << current->task << " (Priority: " << (int)current->priority << ")" << endl;

            current = current->next;

        }

    }

    ~TaskManager() {

        TaskNode\* current = head;

        while (current != nullptr) {

            TaskNode\* next = current->next;

            delete current;

            current = next;

        }

    }

};

Priority stringToPriority(const string& priority) {

    if (priority == "high") {

        return Priority::High;

    } else if (priority == "medium") {

        return Priority::Medium;

    } else {

        return Priority::Low;

    }

}

int main() {

    TaskManager manager;

    int choice;

    string task, priority;

    do {

        cout << "\nTask Management Menu:" << endl;

        cout << "1. Add Task" << endl;

        cout << "2. Remove Task" << endl;

        cout << "3. Display Tasks" << endl;

        cout << "4. Exit" << endl;

        cout << "Choose an option (1-4): ";

        cin >> choice;

        cin.ignore();

        switch (choice) {

            case 1:

                cout << "Enter task: ";

                getline(cin, task);

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

                getline(cin, priority);

                manager.addTask(task, stringToPriority(priority));

                break;

            case 2:

                cout << "Enter task to remove: ";

                getline(cin, task);

                manager.removeTask(task);

                break;

            case 3:

                manager.displayTasks();

                break;

            case 4:

                cout << "Exiting the program." << endl;

                break;

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

    } while (choice != 4);

    return 0;

}

**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.

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

using namespace std;

void displayMenu() {

    cout << "\nStudent Roll Call Menu:" << endl;

    cout << "1. Add Student" << endl;

    cout << "2. Remove Student" << endl;

    cout << "3. Print Student List" << endl;

    cout << "4. Exit" << endl;

}

int main() {

    vector<string> students;

    int choice = 0;

    string name;

    do {

        displayMenu();

        cout << "Choose an option (1-4): ";

        cin >> choice;

        cin.ignore(numeric\_limits<streamsize>::max(), '\n');

        switch (choice) {

            case 1: {

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

                getline(cin, name);

                students.push\_back(name);

                cout << "Student \"" << name << "\" has been added." << endl;

                break;

            }

            case 2: {

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

                getline(cin, name);

                auto it = find(students.begin(), students.end(), name);

                if (it != students.end()) {

                    students.erase(it);

                    cout << "Student \"" << name << "\" has been removed." << endl;

                } else {

                    cout << "Student \"" << name << "\" not found." << endl;

                }

                break;

            }

            case 3: {

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

                cout << "Student List:" << endl;

                if (students.empty()) {

                    cout << "No students in the list." << endl;

                } else {

                    for (const auto& student : students) {

                        cout << "- " << student << endl;

                    }

                }

                break;

            }

            case 4: {

                cout << "Exiting the program." << endl;

                break;

            }

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

    } while (choice != 4);

    return 0;

}

**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).

#include <iostream>

#include <list>

#include <string>

using namespace std;

void displayMenu() {

    cout << "\nTicketing System Menu:" << endl;

    cout << "1. Take a Ticket" << endl;

    cout << "2. Serve a Ticket" << endl;

    cout << "3. Display Tickets" << endl;

    cout << "4. Exit" << endl;

}

int main() {

    list<string> tickets;

    int choice;

    string ticket;

    do {

        displayMenu();

        cout << "Choose an option (1-4): ";

        cin >> choice;

        cin.ignore();

        switch (choice) {

            case 1:

                cout << "Enter your ticket name: ";

                getline(cin, ticket);

                tickets.push\_back(ticket);

                cout << "Ticket \"" << ticket << "\" has been taken." << endl;

                break;

            case 2:

                if (!tickets.empty()) {

                    string servedTicket = tickets.front();

                    tickets.pop\_front();

                    cout << "Ticket \"" << servedTicket << "\" has been served." << endl;

                } else {

                    cout << "No tickets in the queue to serve." << endl;

                }

                break;

            case 3:

                cout << "Current Tickets in Queue:" << endl;

                if (tickets.empty()) {

                    cout << "No tickets in the queue." << endl;

                } else {

                    for (const auto& t : tickets) {

                        cout << "- " << t << endl;

                    }

                }

                break;

            case 4:

                cout << "Exiting the program." << endl;

                break;

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

    } while (choice != 4);

    return 0;

}

**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.

#include <iostream>

#include <list>

#include <string>

#include <algorithm>

using namespace std;

void displayMenu() {

    cout << "\nMusic Playlist Manager Menu:" << endl;

    cout << "1. Add Song at Beginning" << endl;

    cout << "2. Add Song at End" << endl;

    cout << "3. Add Song at Specific Position" << endl;

    cout << "4. Remove Song" << endl;

    cout << "5. Display Playlist" << endl;

    cout << "6. Exit" << endl;

}

int main() {

    list<string> playlist;

    int choice = 0;

    string song;

    int position;

    do {

        displayMenu();

        cout << "Choose an option (1-6): ";

        cin >> choice;

        cin.ignore();

        switch (choice) {

            case 1: {

                cout << "Enter song name to add at beginning: ";

                getline(cin, song);

                playlist.push\_front(song);

                cout << "Song \"" << song << "\" has been added at the beginning." << endl;

                break;

            }

            case 2: {

                cout << "Enter song name to add at end: ";

                getline(cin, song);

                playlist.push\_back(song);

                cout << "Song \"" << song << "\" has been added at the end." << endl;

                break;

            }

            case 3: {

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

                getline(cin, song);

                cout << "Enter position to add the song (starting from 1): ";

                cin >> position;

                cin.ignore();

                if (position < 1 || position > playlist.size() + 1) {

                    cout << "Invalid position!" << endl;

                } else {

                    auto it = playlist.begin();

                    advance(it, position - 1);

                    playlist.insert(it, song);

                    cout << "Song \"" << song << "\" has been added at position " << position << "." << endl;

                }

                break;

            }

            case 4: {

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

                getline(cin, song);

                auto it = find(playlist.begin(), playlist.end(), song);

                if (it != playlist.end()) {

                    playlist.erase(it);

                    cout << "Song \"" << song << "\" has been removed." << endl;

                } else {

                    cout << "Song \"" << song << "\" not found in the playlist." << endl;

                }

                break;

            }

            case 5: {

                cout << "Current Playlist:" << endl;

                if (playlist.empty()) {

                    cout << "The playlist is empty." << endl;

                } else {

                    for (const auto& s : playlist) {

                        cout << "- " << s << endl;

                    }

                }

                break;

            }

            case 6: {

                cout << "Exiting the program." << endl;

                break;

            }

            default:

                cout << "Invalid choice. Please try again." << endl;

        }

    } while (choice != 6);

    return 0;

}