A red sign with white text

AI-generated content may be incorrect.A logo with blue and red text

AI-generated content may be incorrect.

SUBMITTED BY: SADAF AKHTAR ANSARI

STUDENT ID: 24030177

WORKSHEET 4

# **Question 1.1**

1. STL Container Practice: Write a program using STL containers that:
   1. Uses vector<string> to store names
   2. Uses map<string, int> to store age against each name
   3. Implements functions to:
      1. Add new name-age pair
      2. Find all people above certain age
      3. Sort and display names alphabetically

**CODE:**

#include <iostream>

#include <vector>

#include <string>

#include <map>

#include <algorithm>

using namespace std;

// Function to add a new name and age

void addNameAgePair(vector<string>& names, map<string, int>& ageMap, const string& name, int age) {

names.push\_back(name); // Add name to list

ageMap[name] = age; // Add name and age to map

}

// Function to find people older than a given age

void findPeopleAboveAge(const map<string, int>& ageMap, int threshold) {

cout << "People older than " << threshold << ":\n";

for (const auto& person : ageMap) {

if (person.second > threshold) {

cout << person.first << " (" << person.second << " years old)" << endl;

}

}

}

// Function to sort and show names in order

void sortAndDisplayNames(vector<string>& names) {

vector<string> sortedNames = names; // Copy names

sort(sortedNames.begin(), sortedNames.end()); // Sort names

cout << "Names in alphabetical order:\n";

for (const auto& name : sortedNames) {

cout << name << endl;

}

}

int main() {

vector<string> names; // List to store names

map<string, int> ageMap; // Map to store name and age

int choice;

do {

// Show menu

cout << "\nMenu:\n";

cout << "1. Add new name and age\n";

cout << "2. Find people older than a certain age\n";

cout << "3. Sort and display names\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

string name;

int age;

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

addNameAgePair(names, ageMap, name, age);

cout << "Added successfully.\n";

}

else if (choice == 2) {

int threshold;

cout << "Enter age limit: ";

cin >> threshold;

findPeopleAboveAge(ageMap, threshold);

}

else if (choice == 3) {

sortAndDisplayNames(names);

}

else if (choice == 4) {

cout << "Goodbye!\n";

}

else {

cout << "Invalid option. Try again.\n";

}

} while (choice != 4); // Keep running until user chooses to exit

return 0;

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

# **Question 1.2**

1. Stack Problem: Implement a stack using arrays (not STL) that:
   1. Has basic push and pop operations
   2. Has a function to find middle element
   3. Has a function to reverse only bottom half of stack
   4. Maintain stack size of 10

**CODE:**

#include <iostream>

using namespace std;

#define MAX\_SIZE 10 // Max size of stack

class Stack {

private:

int arr[MAX\_SIZE]; // Array to store stack items

int top; // Top index of stack

public:

Stack() {

top = -1; // Stack starts empty

}

// Add value to stack

void push(int value) {

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

cout << "Stack full! Cannot push " << value << endl;

return;

}

arr[++top] = value;

}

// Remove value from top of stack

int pop() {

if (top < 0) {

cout << "Stack empty! Cannot pop.\n";

return -1;

}

return arr[top--];

}

// Show middle value of stack

void findMiddle() {

if (top < 0) {

cout << "Stack is empty!\n";

return;

}

int middleIndex = top / 2;

cout << "Middle element: " << arr[middleIndex] << endl;

}

// Reverse only the bottom half of the stack

void reverseBottomHalf() {

if (top < 1) {

cout << "Not enough items to reverse bottom half.\n";

return;

}

int halfSize = (top + 1) / 2;

for (int i = 0; i < halfSize / 2; i++) {

swap(arr[i], arr[halfSize - 1 - i]);

}

cout << "Bottom half reversed.\n";

}

// Show all elements of stack

void display() {

if (top < 0) {

cout << "Stack is empty.\n";

return;

}

cout << "Stack (top to bottom): ";

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

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

}

cout << endl;

}

};

int main() {

Stack stack;

int choice, value;

do {

cout << "\nMenu:\n";

cout << "1. Push\n2. Pop\n3. Find Middle\n4. Reverse Bottom Half\n5. Display Stack\n6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to push: ";

cin >> value;

stack.push(value);

break;

case 2:

value = stack.pop();

if (value != -1)

cout << "Popped: " << value << endl;

break;

case 3:

stack.findMiddle();

break;

case 4:

stack.reverseBottomHalf();

break;

case 5:

stack.display();

break;

case 6:

cout << "Goodbye!\n";

break;

default:

cout << "Wrong option. Try again.\n";

}

} while (choice != 6);

return 0;

}

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

# **Question 1.3**

1. Queue Problem: Implement a queue using arrays (not STL) that:
   1. Has basic enqueue and dequeue operations
   2. Has a function to reverse first K elements
   3. Has a function to interleave first half with second half
   4. Handle queue overflow/underflow

**CODE:**

#include <iostream>

using namespace std;

#define MAX\_SIZE 10 // Maximum size of queue

class Queue {

private:

int arr[MAX\_SIZE]; // Array to hold queue elements

int front, rear, size; // Front, rear indices and size of queue

public:

// Constructor to initialize queue

Queue() {

front = 0;

rear = -1;

size = 0;

}

// Check if queue is empty

bool isEmpty() {

return size == 0;

}

// Check if queue is full

bool isFull() {

return size == MAX\_SIZE;

}

// Add value to queue

void enqueue(int value) {

if (isFull()) {

cout << "Queue Overflow! Cannot add " << value << endl;

return;

}

rear = (rear + 1) % MAX\_SIZE;

arr[rear] = value;

size++;

}

// Remove value from queue

int dequeue() {

if (isEmpty()) {

cout << "Queue Underflow! No element to remove.\n";

return -1;

}

int removedValue = arr[front];

front = (front + 1) % MAX\_SIZE;

size--;

return removedValue;

}

// Reverse the first K elements in the queue

void reverseFirstK(int k) {

if (k <= 0 || k > size) {

cout << "Invalid K value.\n";

return;

}

for (int i = 0; i < k / 2; i++) {

swap(arr[(front + i) % MAX\_SIZE], arr[(front + k - 1 - i) % MAX\_SIZE]);

}

cout << "First " << k << " elements reversed.\n";

}

// Interleave the first half with the second half of the queue

void interleaveQueue() {

if (size % 2 != 0) {

cout << "Queue size must be even to interleave.\n";

return;

}

int halfSize = size / 2;

int temp[MAX\_SIZE];

// Merge first half and second half into temp array

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

temp[i \* 2] = arr[(front + i) % MAX\_SIZE];

temp[i \* 2 + 1] = arr[(front + halfSize + i) % MAX\_SIZE];

}

// Copy the interleaved result back to the queue

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

arr[(front + i) % MAX\_SIZE] = temp[i];

}

cout << "Queue interleaved successfully.\n";

}

// Display all elements in the queue

void display() {

if (isEmpty()) {

cout << "Queue is empty.\n";

return;

}

cout << "Queue (front to rear): ";

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

cout << arr[(front + i) % MAX\_SIZE] << " ";

}

cout << endl;

}

};

int main() {

Queue queue;

int choice, value, k;

// Menu-driven interface for queue operations

do {

cout << "\nMenu:\n";

cout << "1. Enqueue\n2. Dequeue\n3. Reverse First K Elements\n4. Interleave Queue\n5. Display Queue\n6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to enqueue: ";

cin >> value;

queue.enqueue(value); // Add value to queue

break;

case 2:

value = queue.dequeue(); // Remove value from queue

if (value != -1) cout << "Dequeued: " << value << endl;

break;

case 3:

cout << "Enter K value: ";

cin >> k;

queue.reverseFirstK(k); // Reverse first K elements

break;

case 4:

queue.interleaveQueue(); // Interleave first half with second half

break;

case 5:

queue.display(); // Display queue elements

break;

case 6:

cout << "Exiting program.\n"; // Exit the program

break;

default:

cout << "Invalid choice. Try again.\n"; // Handle invalid choice

}

} while (choice != 6); // Continue until user selects exit

return 0;

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

# **Question 1.4**

1. Linked List Problem: Create a singly linked list (not STL) that:
   1. Has functions to insert at start/end/position
   2. Has a function to detect and remove loops
   3. Has a function to find nth node from end
   4. Has a function to reverse list in groups of K nodes

**CODE:**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = nullptr;

}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr; // Initially, list is empty

}

// Insert a node at the beginning

void insertAtStart(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

}

// Insert a node at the end

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode; // If the list is empty, make new node the head

return;

}

Node\* temp = head;

while (temp->next) // Traverse to the last node

temp = temp->next;

temp->next = newNode; // Add new node at the end

}

// Insert a node at a specific position

void insertAtPosition(int value, int position) {

if (position <= 0) {

cout << "Invalid position!\n";

return;

}

if (position == 1) {

insertAtStart(value); // Insert at the beginning if position is 1

return;

}

Node\* newNode = new Node(value);

Node\* temp = head;

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

temp = temp->next; // Traverse to the node just before the position

}

if (!temp) {

cout << "Position out of bounds!\n";

return;

}

newNode->next = temp->next; // Insert new node at the specified position

temp->next = newNode;

}

// Detect and remove loop in the list

void detectAndRemoveLoop() {

Node\* slow = head, \* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next; // Move fast pointer 2 steps, slow pointer 1 step

if (slow == fast) {

cout << "Loop detected! Removing...\n";

removeLoop(slow); // Remove loop if detected

return;

}

}

cout << "No loop detected.\n";

}

// Helper function to remove the loop

void removeLoop(Node\* loopNode) {

Node\* ptr1 = head;

Node\* ptr2 = loopNode;

while (ptr1->next != ptr2->next) {

ptr1 = ptr1->next;

ptr2 = ptr2->next; // Move both pointers until they meet at the loop entry point

}

ptr2->next = nullptr; // Break the loop by setting the loop node's next to null

}

// Find the Nth node from the end of the list

void findNthFromEnd(int n) {

Node\* first = head;

Node\* second = head;

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

if (!first) {

cout << "N is larger than the list size!\n";

return;

}

first = first->next; // Move first pointer N steps ahead

}

while (first) {

first = first->next;

second = second->next; // Move both pointers one step at a time until first reaches the end

}

cout << "The " << n << "th node from the end is: " << second->data << endl;

}

// Reverse the list in groups of K nodes

Node\* reverseInGroups(Node\* head, int k) {

if (!head || k <= 1) return head; // If the list is empty or K is 1, return as is

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

int count = 0;

Node\* temp = head;

for (int i = 0; i < k && temp; i++, temp = temp->next) count++; // Count if there are at least K nodes

if (count < k) return head; // If less than K nodes, no reversal

// Reverse the first K nodes

count = 0;

while (current && count < k) {

next = current->next;

current->next = prev;

prev = current;

current = next;

count++;

}

// Recursively reverse the rest of the list

if (next) head->next = reverseInGroups(next, k);

return prev;

}

// Reverse the list in groups of K nodes (public function)

void reverseInGroupsK(int k) {

head = reverseInGroups(head, k);

cout << "List reversed in groups of " << k << endl;

}

// Display the list

void display() {

Node\* temp = head;

while (temp) {

cout << temp->data << " -> "; // Print each node's data

temp = temp->next;

}

cout << "NULL\n"; // End of list

}

};

int main() {

LinkedList list;

int choice, value, pos, k;

do {

cout << "\nMenu:\n";

cout << "1. Insert at Start\n2. Insert at End\n3. Insert at Position\n4. Detect & Remove Loop\n5. Find Nth Node from End\n6. Reverse in Groups of K\n7. Display\n8. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value: ";

cin >> value;

list.insertAtStart(value);

break;

case 2:

cout << "Enter value: ";

cin >> value;

list.insertAtEnd(value);

break;

case 3:

cout << "Enter value: ";

cin >> value;

cout << "Enter position: ";

cin >> pos;

list.insertAtPosition(value, pos);

break;

case 4:

list.detectAndRemoveLoop();

break;

case 5:

cout << "Enter N: ";

cin >> pos;

list.findNthFromEnd(pos);

break;

case 6:

cout << "Enter K: ";

cin >> k;

list.reverseInGroupsK(k);

break;

case 7:

list.display();

break;

case 8:

cout << "Exiting program.\n";

break;

default:

cout << "Invalid choice. Try again.\n";

}

} while (choice != 8);

return 0;

}

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.