UFCFGL-30-1-Programming in C++

******

Computer Science and creative technologies

Tasks

Task 1: Solve the following programming problems:

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

2. 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

3. 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

4. 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

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

Answer:

#include <iostream>

#include <vector>

#include <map>

#include <string>

#include <algorithm>

using namespace std;

class PersonDatabase {

private:

vector<string> names;

map<string, int> ageMap;

public:

void addPerson(string name, int age) {

if (ageMap.find(name) == ageMap.end()) {

names.push\_back(name);

ageMap[name] = age;

cout << "Added: " << name << ", Age: " << age << endl;

} else {

cout << "Name already exists!" << endl;

}

}

vector<string> findPeopleAboveAge(int minAge) {

vector<string> result;

for (const auto& pair : ageMap) {

if (pair.second > minAge) {

result.push\_back(pair.first);

}

}

return result;

}

void displaySortedNames() {

vector<string> sortedNames = names;

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

cout << "\nNames in alphabetical order:" << endl;

for (const string& name : sortedNames) {

cout << name << " - Age: " << ageMap[name] << endl;

}

}

void displayAll() {

cout << "\nAll entries:" << endl;

for (const auto& pair : ageMap) {

cout << pair.first << " - Age: " << pair.second << endl;

}

}

};

int main() {

PersonDatabase db;

db.addPerson("Amit", 25);

db.addPerson("deepak", 30);

db.addPerson("shubham", 22);

db.addPerson("surya", 35);

db.displayAll();

cout << "\nPeople above age 25:" << endl;

vector<string> olderThan25 = db.findPeopleAboveAge(25);

for (const string& name : olderThan25) {

cout << name << endl;

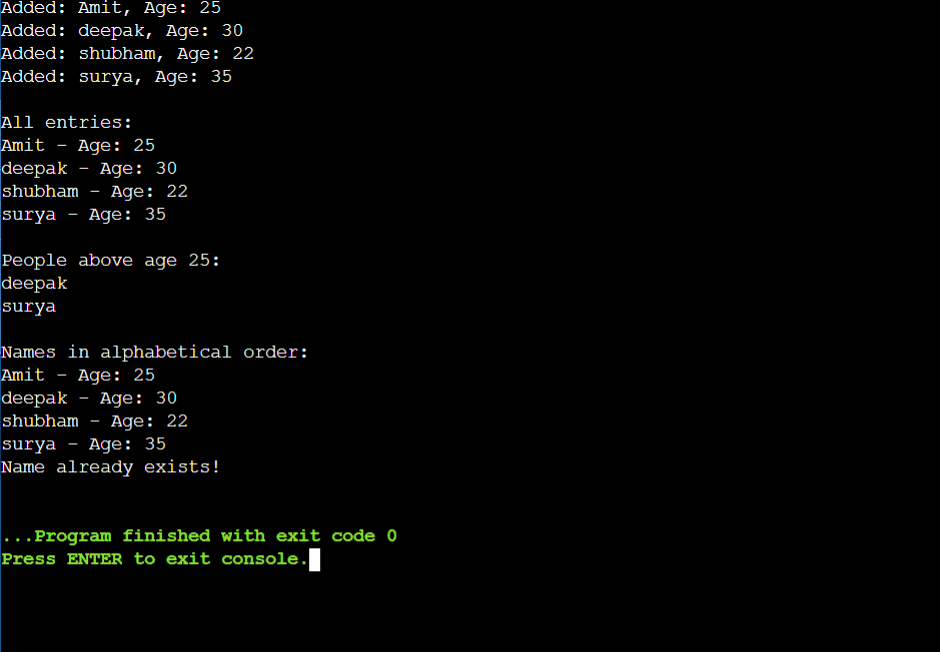
}

db.displaySortedNames();

db.addPerson("Amit", 26);

return 0;

}



2. 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

Answer:

#include <iostream>

using namespace std;

class CustomStack {

private:

static const int MAX\_SIZE = 10;

int arr[MAX\_SIZE];

int top;

public:

CustomStack() {

top = -1;

}

bool isFull() {

return top >= MAX\_SIZE - 1;

}

bool isEmpty() {

return top < 0;

}

void push(int value) {

if (isFull()) {

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

return;

}

arr[++top] = value;

cout << "Pushed: " << value << endl;

}

int pop() {

if (isEmpty()) {

cout << "Stack is empty! Cannot pop" << endl;

return -1;

}

int value = arr[top--];

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

return value;

}

int findMiddle() {

if (isEmpty()) {

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

return -1;

}

int midIndex = top / 2;

return arr[midIndex];

}

void reverseBottomHalf() {

if (isEmpty()) {

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

return;

}

int size = top + 1;

int half = size / 2;

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

// Swap elements

int temp = arr[i];

arr[i] = arr[half - 1 - i];

arr[half - 1 - i] = temp;

}

cout << "Bottom half reversed" << endl;

}

void display() {

if (isEmpty()) {

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

return;

}

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

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

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

}

cout << endl;

}

};

int main() {

CustomStack stack;

cout << "Pushing elements..." << endl;

for (int i = 1; i <= 8; i++) {

stack.push(i);

}

stack.display();

cout << "\nMiddle element: " << stack.findMiddle() << endl;

cout << "\nReversing bottom half..." << endl;

stack.reverseBottomHalf();

stack.display();

cout << "\nPopping elements..." << endl;

stack.pop();

stack.pop();

stack.display();

cout << "\nTrying to push beyond capacity..." << endl;

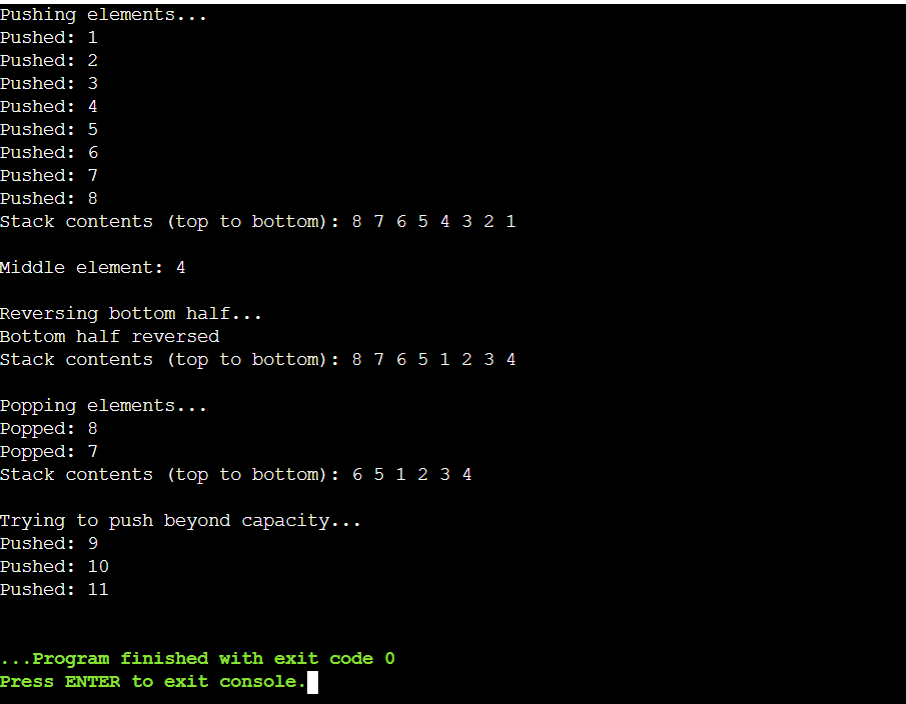
for (int i = 9; i <= 11; i++) {

stack.push(i);

}

return 0;

}



3. 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

Answer:

#include <iostream>

using namespace std;

class CustomQueue {

private:

static const int MAX\_SIZE = 10;

int arr[MAX\_SIZE];

int front, rear, size;

public:

CustomQueue() {

front = 0;

rear = -1;

size = 0;

}

bool isFull() {

return size >= MAX\_SIZE;

}

bool isEmpty() {

return size <= 0;

}

void enqueue(int value) {

if (isFull()) {

cout << "Queue overflow! Cannot enqueue " << value << endl;

return;

}

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

arr[rear] = value;

size++;

cout << "Enqueued: " << value << endl;

}

int dequeue() {

if (isEmpty()) {

cout << "Queue underflow! Cannot dequeue" << endl;

return -1;

}

int value = arr[front];

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

size--;

cout << "Dequeued: " << value << endl;

return value;

}

void reverseFirstK(int k) {

if (isEmpty()) {

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

return;

}

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

cout << "Invalid K value!" << endl;

return;

}

int start = front;

int end = (front + k - 1) % MAX\_SIZE;

while (start < end) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

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

if (end == 0) end = MAX\_SIZE - 1;

else end--;

}

cout << "First " << k << " elements reversed" << endl;

}

void interleaveHalves() {

if (isEmpty()) {

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

return;

}

if (size % 2 != 0) {

cout << "Queue size must be even for interleaving!" << endl;

return;

}

int half = size / 2;

int temp[MAX\_SIZE];

int i = 0;

int pos = front;

while (i < half) {

temp[i++] = arr[pos];

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

}

while (i < size) {

temp[i++] = arr[pos];

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

}

front = 0;

rear = -1;

size = 0;

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

enqueue(temp[j]); // First half element

enqueue(temp[j + half]); // Second half element

}

cout << "Halves interleaved" << endl;

}

void display() {

if (isEmpty()) {

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

return;

}

cout << "Queue contents: ";

int count = 0;

int i = front;

while (count < size) {

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

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

count++;

}

cout << endl;

}

};

int main() {

CustomQueue queue;

cout << "Enqueuing elements..." << endl;

for (int i = 1; i <= 8; i++) {

queue.enqueue(i);

}

queue.display();

cout << "\nReversing first 4 elements..." << endl;

queue.reverseFirstK(4);

queue.display();

cout << "\nDequeuing..." << endl;

queue.dequeue();

queue.display();

cout << "\nEnqueuing one more for even size..." << endl;

queue.enqueue(9);

queue.display();

cout << "\nInterleaving halves..." << endl;

queue.interleaveHalves();

queue.display();

cout << "\nTesting overflow..." << endl;

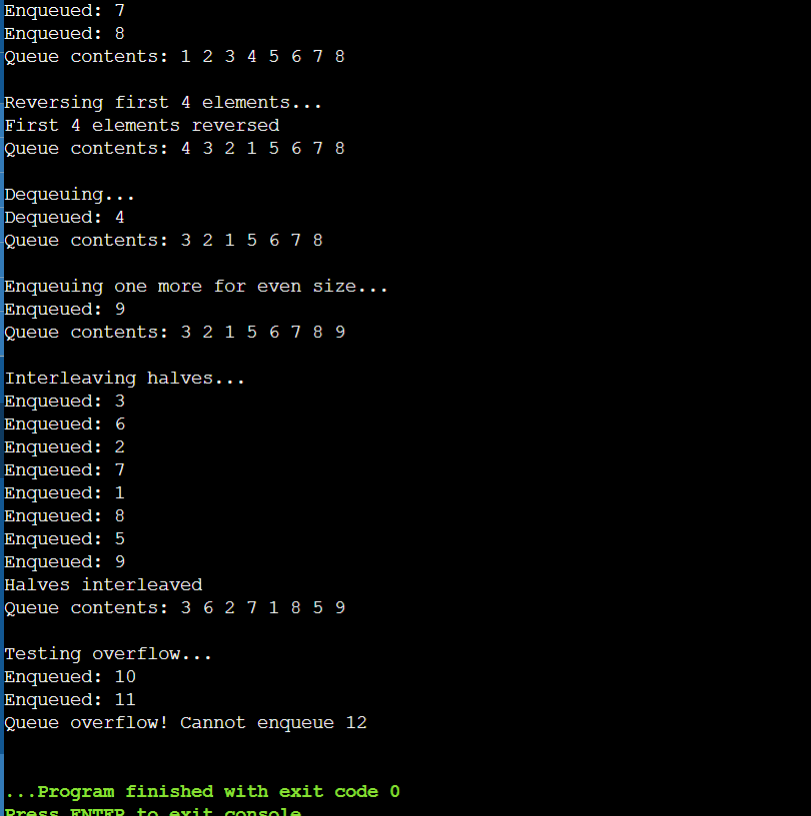
for (int i = 10; i <= 12; i++) {

queue.enqueue(i);

}

return 0;

}



4. 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

Answer:

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class LinkedList {

private:

Node\* head;

public:

LinkedList() : head(nullptr) {}

void insertAtStart(int value) {

Node\* newNode = new Node(value);

newNode->next = head;

head = newNode;

cout << "Inserted " << value << " at start" << endl;

}

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (!head) {

head = newNode;

} else {

Node\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

cout << "Inserted " << value << " at end" << endl;

}

void insertAtPosition(int value, int pos) {

if (pos < 0) {

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

return;

}

if (pos == 0) {

insertAtStart(value);

return;

}

Node\* newNode = new Node(value);

Node\* current = head;

for (int i = 0; i < pos - 1 && current; i++) {

current = current->next;

}

if (!current) {

cout << "Position out of range!" << endl;

delete newNode;

return;

}

newNode->next = current->next;

current->next = newNode;

cout << "Inserted " << value << " at position " << pos << endl;

}

void detectAndRemoveLoop() {

if (!head || !head->next) {

cout << "No loop possible" << endl;

return;

}

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

bool hasLoop = false;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

hasLoop = true;

break;

}

}

if (!hasLoop) {

cout << "No loop detected" << endl;

return;

}

slow = head;

while (slow != fast) {

slow = slow->next;

fast = fast->next;

}

while (fast->next != slow) {

fast = fast->next;

}

fast->next = nullptr;

cout << "Loop removed" << endl;

}

int findNthFromEnd(int n) {

if (!head || n <= 0) {

cout << "Invalid input!" << endl;

return -1;

}

Node \*main = head, \*ref = head;

int count = 0;

while (count < n && ref) {

ref = ref->next;

count++;

}

if (count < n) {

cout << "List shorter than " << n << " nodes!" << endl;

return -1;

}

while (ref) {

main = main->next;

ref = ref->next;

}

return main->data;

}

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

Node \*current = start, \*prev = nullptr, \*next = nullptr;

int count = 0;

while (current && count < k) {

next = current->next;

current->next = prev;

prev = current;

current = next;

count++;

}

if (next) {

start->next = reverseInGroups(next, k);

}

return prev;

}

void reverseGroups(int k) {

if (!head || k <= 1) return;

head = reverseInGroups(head, k);

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

}

void display() {

if (!head) {

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

return;

}

cout << "List: ";

Node\* current = head;

while (current) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void createLoop(int pos) {

if (!head || pos < 0) return;

Node\* current = head;

Node\* loopNode = nullptr;

int count = 0;

while (current->next) {

if (count == pos) loopNode = current;

current = current->next;

count++;

}

current->next = loopNode;

}

~LinkedList() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

LinkedList list;

list.insertAtStart(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

list.insertAtPosition(4, 1);

list.insertAtEnd(5);

list.display();

cout << "\n2nd node from end: " << list.findNthFromEnd(2) << endl;

cout << "\nReversing in groups of 2..." << endl;

list.reverseGroups(2);

list.display();

cout << "\nCreating loop and testing..." << endl;

LinkedList list2;

list2.insertAtEnd(1);

list2.insertAtEnd(2);

list2.insertAtEnd(3);

list2.insertAtEnd(4);

list2.createLoop(1);

list2.detectAndRemoveLoop();

list2.display();

return 0;

}

