WORKSHEET 4

Manish Mahato

The British College
Student ID: 23085136

WORKSHEET 4

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

Source code for this:

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>

using namespace std;

void addPerson(map<string, int>& people, vector<string>& names) {
    string name;
    int age;

    cout << "Enter name: ";
    cin.ignore(); // Clear leftover input
    getline(cin, name);

    cout << "Enter age: ";
    cin >> age;

    people[name] = age;
    names.push_back(name);
}

void showPeopleAboveAge(const map<string, int>& people, int limit) {
    bool found = false;
    cout << "People older than " << limit << ":\n";
    for (auto& person : people) {
        if (person.second > limit) {
            cout << person.first << " (" << person.second << ")\n";
            found = true;
        }
    }
    if (!found) cout << "No one found above that age.\n";
}

void showSortedNames(const vector<string>& names) {
```

```
    if (names.empty()) {
        cout << "No names to display.\n";
        return;
    }

    vector<string> sortedNames = names;
    sort(sortedNames.begin(), sortedNames.end());

    cout << "Names in alphabetical order:\n";
    for (const string& name : sortedNames) {
        cout << name << endl;
    }
}

int main() {
    map<string, int> people;
    vector<string> names;
    int choice;

    do {
        cout << "\nMenu:\n";
        cout << "1. Add person\n";
        cout << "2. Show people above a certain age\n";
        cout << "3. Show sorted names\n";
        cout << "4. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        if (cin.fail()) {
            cin.clear();
            cin.ignore(1000, '\n');
            cout << "Invalid input. Try again.\n";
            continue;
        }

        switch (choice) {
            case 1:
                addPerson(people, names);
                break;
            case 2: {
                int age;
                cout << "Enter age limit: ";
                cin >> age;
                showPeopleAboveAge(people, age);
                break;
            }
            case 3:
                showSortedNames(names);
                break;
            case 4:
                cout << "Goodbye!\n";
                break;
            default:
                cout << "Invalid option. Try again.\n";
        }

    } while (choice != 4);
```

```
    return 0;
}
```

Output:

```
Run    task1.cpp

1. Add person
2. Show people above a certain age
3. Show sorted names
4. Exit
Enter choice: 1
Enter name: manish
Enter age: 22

Menu:
1. Add person
2. Show people above a certain age
3. Show sorted names
4. Exit
Enter choice: 2
Enter age limit: 10
People older than 10:
manish (22)

Menu:
1. Add person
2. Show people above a certain age
3. Show sorted names
4. Exit
Enter choice: 3
Names in alphabetical order:
manish

Menu:
1. Add person
2. Show people above a certain age
3. Show sorted names
4. Exit
Enter choice: 4
Goodbye!
```

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

Source Code for this:

```
#include <iostream>
using namespace std;

class Stack {
private:
    int stack[10];
    int top;

public:
    Stack() { top = -1; }

    void push(int value) {
        if (top == 9) {
            cout << "Stack is full!" << endl;
            return;
        }
        stack[++top] = value;
    }

    void pop() {
        if (top == -1) {
            cout << "Stack is empty!" << endl;
            return;
        }
        cout << "Popped value: " << stack[top--] << endl;
    }

    void findMiddle() {
        if (top == -1) {
            cout << "Stack is empty!" << endl;
            return;
        }
        cout << "Middle element: " << stack[top / 2] << endl;
    }

    void reverseBottomHalf() {
        if (top < 1) {
            cout << "Not enough elements to reverse bottom half!" << endl;
            return;
        }

        int mid = top / 2;
        for (int i = 0; i <= mid / 2; i++) {
```

```
            swap(stack[i], stack[mid - i]);
        }

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

    void display() {
        if (top == -1) {
            cout << "Stack is empty!" << endl;
            return;
        }

        cout << "Stack elements: ";
        for (int i = 0; i <= top; i++) {
            cout << stack[i] << " ";
        }
        cout << endl;
    }

    int spaceLeft() {
        return 9 - top;
    }
};

int main() {
    Stack s;
    int choice, value, count;

    do {
        cout << "\nMenu:\n";
        cout << "1. Push up to 5 numbers\n";
        cout << "2. Pop\n";
        cout << "3. Find Middle\n";
        cout << "4. Reverse Bottom Half\n";
        cout << "5. Display Stack\n";
        cout << "6. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "How many numbers do you want to push (max 5)? ";
                cin >> count;

                if (count < 1 || count > 5) {
                    cout << "Please enter a number between 1 and 5." << endl;
                    break;
                }

                if (count > s.spaceLeft()) {
                    cout << "Not enough space in stack. You can push up to "
<< s.spaceLeft() << " more." << endl;
                    break;
                }

                cout << "Enter " << count << " number(s): ";
                for (int i = 0; i < count; ++i) {
```

```
                cin >> value;
                s.push(value);
            }
            break;

        case 2:
            s.pop();
            break;

        case 3:
            s.findMiddle();
            break;

        case 4:
            s.reverseBottomHalf();
            break;

        case 5:
            s.display();
            break;

        case 6:
            cout << "Goodbye!" << endl;
            break;

        default:
            cout << "Invalid choice." << endl;
    }

    } while (choice != 6);

    return 0;
}
```

Output:

Run Task2.cpp

```
/Users/manish/Desktop/worksheet4/Task2

Menu:
1. Push up to 5 numbers
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Enter choice: 1
How many numbers do you want to push (max 5)? 4
Enter 4 number(s): 1 2 3 4

Menu:
1. Push up to 5 numbers
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Enter choice: 2
Popped value: 4

Menu:
1. Push up to 5 numbers
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Enter choice: 3
Middle element: 2
```

```
Menu:
1. Push up to 5 numbers
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Enter choice: 4
Bottom half reversed.

Menu:
1. Push up to 5 numbers
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Enter choice: 5
Stack elements: 2 1 3

Menu:
1. Push up to 5 numbers
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Enter choice:
```

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

Source code for this:

```
#include <iostream>
using namespace std;

class Queue {
private:
    int queue[10];
    int front, rear, size;

public:
    Queue() {
        front = rear = size = 0;
    }

    void enqueue(int value) {
        if (size == 10) {
            cout << "Queue is full!\n";
            return;
        }
        queue[rear] = value;
        rear = (rear + 1) % 10;
        size++;
    }

    int dequeue() {
        if (size == 0) {
            cout << "Queue is empty!\n";
            return -1;
        }
        int val = queue[front];
        front = (front + 1) % 10;
        size--;
        return val;
    }

    void reverseFirstK(int k) {
        if (k > size || k < 1) {
            cout << "Invalid value of K.\n";
            return;
        }

        int temp[10];
        for (int i = 0; i < k; i++) temp[i] = dequeue();
        for (int i = k - 1; i >= 0; i--) enqueue(temp[i]);
    }

    void interleave() {
```

```
        if (size % 2 != 0) {
            cout << "Queue size must be even to interleave.\n";
            return;
        }

        int half = size / 2;
        int first[5], second[5];

        for (int i = 0; i < half; i++) first[i] = dequeue();
        for (int i = 0; i < half; i++) second[i] = dequeue();

        for (int i = 0; i < half; i++) {
            enqueue(first[i]);
            enqueue(second[i]);
        }
    }

    void display() {
        if (size == 0) {
            cout << "Queue is empty.\n";
            return;
        }

        int index = front;
        cout << "Queue: ";
        for (int i = 0; i < size; i++) {
            cout << queue[index] << " ";
            index = (index + 1) % 10;
        }
        cout << endl;
    }
};

int main() {
    Queue q;
    int choice, value, k;

    do {
        cout << "\nMenu:\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Reverse First K Elements\n";
        cout << "4. Interleave Halves\n";
        cout << "5. Display Queue\n";
        cout << "6. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value: ";
                cin >> value;
                q.enqueue(value);
                break;
            case 2:
                value = q.dequeue();
                if (value != -1) cout << "Dequeued: " << value << endl;
```

```
                break;
            case 3:
                cout << "Enter K: ";
                cin >> k;
                q.reverseFirstK(k);
                break;
            case 4:
                q.interleave();
                break;
            case 5:
                q.display();
                break;
            case 6:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice.\n";
        }

    } while (choice != 6);

    return 0;
}
```

Output:

Run Task3.cpp

```
Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value: 1

Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value: 2

Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value: 3
```

Run Task3.cpp

```
Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 2
Dequeued: 1

Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 3
Enter K: 2

Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 4
```

```
Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 5
Queue: 3 2

Menu:
1. Enqueue
2. Dequeue
3. Reverse First K Elements
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 6
Exiting...

Process finished with exit code 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

Source code for this:

```
#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; }

    void insertAtStart(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    void insertAtEnd(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }

    void insertAtPosition(int val, int pos) {
        if (pos <= 1) {
            insertAtStart(val);
            return;
        }
        Node* newNode = new Node(val);
        Node* temp = head;
```

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

        if (!temp) {
            cout << "Position out of range.\n";
            return;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }

    void findNthFromEnd(int n) {
        Node *main = head, *ref = head;
        for (int i = 0; i < n; ++i) {
            if (!ref) {
                cout << "Position too big.\n";
                return;
            }
            ref = ref->next;
        }
        while (ref) {
            main = main->next;
            ref = ref->next;
        }
        cout << n << "th node from end: " << main->data << endl;
    }

    void reverseInGroups(int k) {
        head = reverseK(head, k);
    }

    Node* reverseK(Node* node, int k) {
        Node *prev = nullptr, *curr = node, *next = nullptr;
        int count = 0;

        while (curr && count < k) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
            count++;
        }
        if (next) node->next = reverseK(next, k);
        return prev;
    }

    void detectAndRemoveLoop() {
        Node *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                removeLoop(slow);
                cout << "Loop removed.\n";
                return;
            }
        }
        cout << "No loop detected.\n";
```

```
    }

    void removeLoop(Node* loopNode) {
        Node* ptr1 = head;
        while (ptr1->next != loopNode->next) {
            ptr1 = ptr1->next;
            loopNode = loopNode->next;
        }
        loopNode->next = nullptr;
    }

    void createLoop(int pos) {
        if (pos <= 0) return;
        Node *loopNode = nullptr, *temp = head;
        int count = 1;
        while (temp->next) {
            if (count == pos) loopNode = temp;
            temp = temp->next;
            count++;
        }
        if (loopNode) temp->next = loopNode;
    }

    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL\n";
    }
};

int main() {
    LinkedList ll;

    ll.insertAtEnd(10);
    ll.insertAtEnd(20);
    ll.insertAtEnd(30);
    ll.insertAtEnd(40);
    ll.insertAtEnd(50);
    ll.display();

    ll.insertAtStart(5);
    ll.insertAtPosition(15, 3);
    ll.display();

    ll.findNthFromEnd(3);

    ll.reverseInGroups(2);
    cout << "After reversing in groups of 2:\n";
    ll.display();

    ll.createLoop(3);
    ll.detectAndRemoveLoop();
    ll.display();
```

```
    return 0;
}
```

Output:

```
Run    Task3.cpp    Task4.cpp

/Users/manish/Desktop/worksheet4/Task4
10 -> 20 -> 30 -> 40 -> 50 -> NULL
5 -> 10 -> 15 -> 20 -> 30 -> 40 -> 50 -> NULL
3th node from end: 30
After reversing in groups of 2:
10 -> 5 -> 20 -> 15 -> 40 -> 30 -> 50 -> NULL
Loop removed.
10 -> 5 -> 20 -> 15 -> 40 -> 30 -> 50 -> NULL

Process finished with exit code 0
```