WORKSHEET 4

Submitted By: Sandhya Aryal

Student ID: 24000860

Cyber Security and Digital Forensics

GITHUB LINK:

https://github.com/Sandhyaaaa1/Cpp_Worksheet/tree/f24a47926af8ebf09e597690a9fe7422df2b55cc/worksheet_sandhya/worksheet_4

1. STL Container Practice: Write a program using STL containers that: (40 marks)
    1. Uses vector<string> to store names (5 Marks)
    2. Uses map<string, int> to store age against each name (5 Marks)
    3. Implements functions to:
        1. Add new name-age pair (10 marks)
        2. Find all people above certain age (10 marks)
        3. Sort and display names alphabetically (10 marks)

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

using namespace std;

class PersonManager {
public:
    void addPerson() {
        string name;
        int age;

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

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

    void findSeniors() {
        int limit;
```

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

    cout << "People above " << limit << ":\n";
    for (auto& pair : ages) {
        if (pair.second > limit) {
            cout << pair.first << " (" << pair.second << ")\n";
        }
    }
}

void showSorted() {
    vector<string> temp = names;
    sort(temp.begin(), temp.end());

    cout << "Sorted names:\n";
    for (auto& name : temp) {
        cout << "- " << name << "\n";
    }
}

void runMenu() {
    while (true) {
        cout << "\n--- Menu ---\n";
        cout << "1. Add person\n";
        cout << "2. Find seniors\n";
        cout << "3. Show sorted names\n";
        cout << "4. Exit\n";
        cout << "Choose: ";

        int choice;
        cin >> choice;
```

```
        switch (choice) {
            case 1: addPerson(); break;
            case 2: findSeniors(); break;
            case 3: showSorted(); break;
            case 4: return;
            default: cout << "Invalid choice!\n";
        }
    }
  }

private:
  vector<string> names;
  map<string, int> ages;
};

int main() {
  PersonManager manager;
  manager.runMenu();
  return 0;
}
```

OUTPUT:

```
C:\Users\acer\Desktop\works

--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 1
Enter name: sandhya
Enter age: 21

--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 1
Enter name: pradeep
Enter age: 28

--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 1
Enter name: crishtina
Enter age: 20

--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 1
Enter name: samraj
Enter age: 24

--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice:
```

```
--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 2
Enter age limit: 18
People above 18:
crishtina (20)
pradeep (28)
samraj (24)
sandhya (21)
```

```
--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 3
Sorted names:
- crishtina
- pradeep
- samraj
- sandhya
```

```
--- Menu ---
1. Add new person
2. Find people above age
3. Show sorted names
4. Exit
Your choice: 4

Process returned 0 (0x0)   execution time : 86.504 s
Press any key to continue.
```

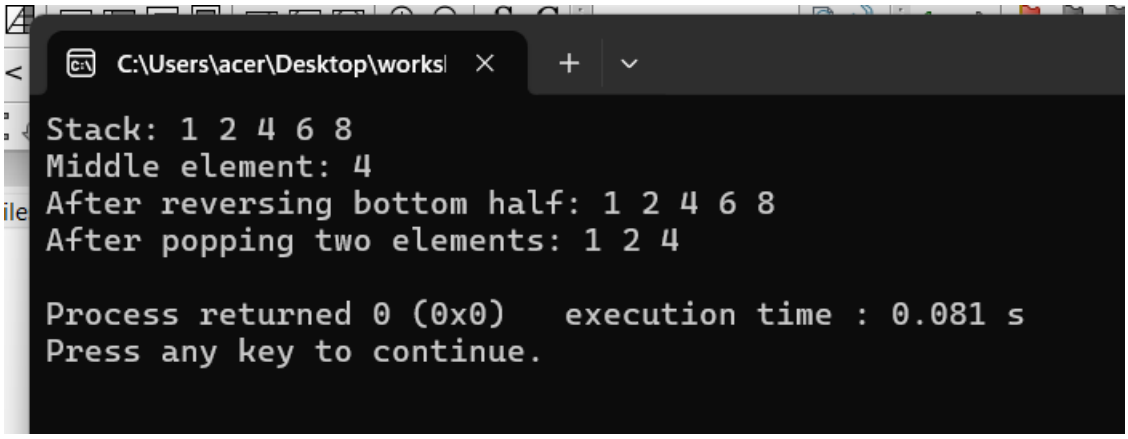
2. Stack Problem: Implement a stack using arrays (not STL) that: (20 marks)

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

```
#include <iostream>

const int MAX_SIZE = 10;

class Stack {
public:
    int data[MAX_SIZE];
    int top = -1;

    void push(int value) {
        if (top < MAX_SIZE - 1) {
            top++;
            data[top] = value;
        }
    }

    void pop() {
        if (top >= 0) {
            top--;
        }
    }

    int findMiddle() {
        if (top >= 0) {
            return data[top / 2];
        }
        return -1;
    }

    void reverseBottomHalf() {
        if (top >= 0) {
            int middle = top / 2;
            for (int i = 0; i < middle; i++) {
                int temp = data[i];
                data[i] = data[middle - i - 1];
                data[middle - i - 1] = temp;
            }
        }
    }

    void displayStack() {
        for (int i = 0; i <= top; i++) {
            std::cout << data[i] << " ";
        }
        std::cout << std::endl;
    }
};

int main() {
    Stack stack;

    stack.push(1);
```

```
    stack.push(2);
    stack.push(4);
    stack.push(6);
    stack.push(8);


    std::cout << "Stack: ";
    stack.displayStack();

    std::cout << "Middle element: " << stack.findMiddle() << std::endl;

    stack.reverseBottomHalf();

    std::cout << "After reversing bottom half: ";
    stack.displayStack();

    stack.pop();
    stack.pop();

    std::cout << "After popping two elements: ";
    stack.displayStack();

    return 0;
}
```

OUTPUT:



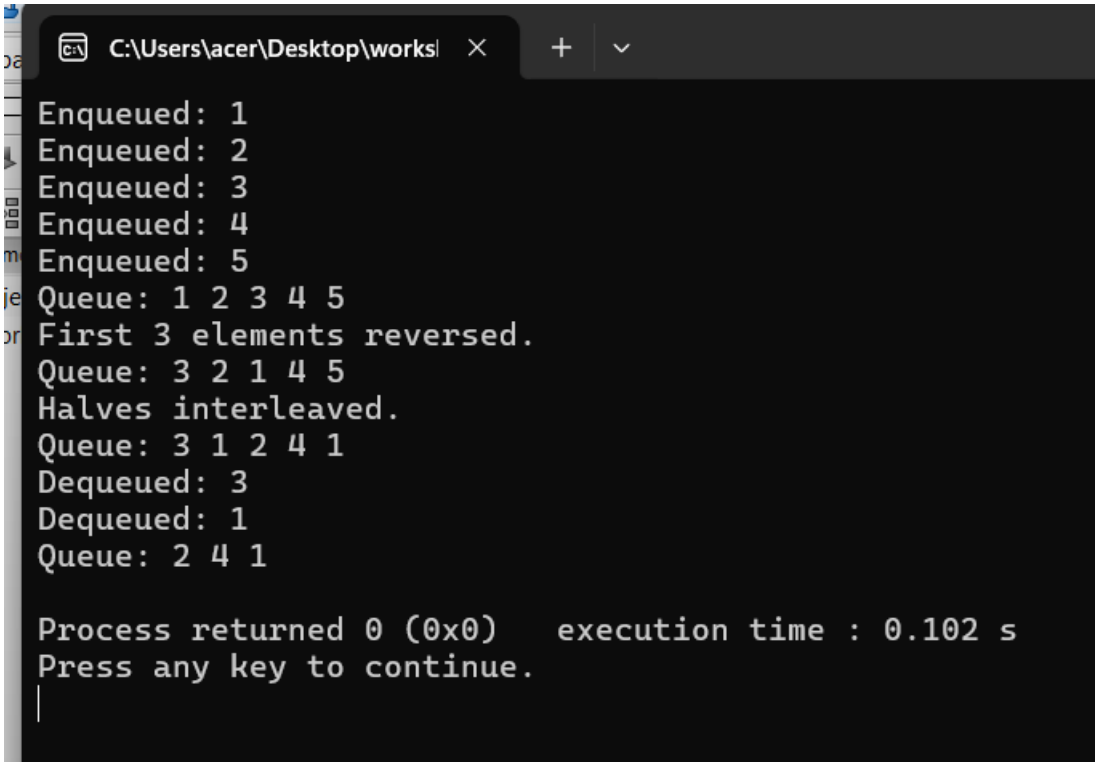
3. Queue Problem: Implement a queue using arrays (not STL) that: (20 marks)
    1. Has basic enqueue and dequeue operations
    2. Has a function to reverse first K elements
    3. Has a function to interleave first half with second half
    4. Handle queue overflow/underflow

```
#include <iostream>

const int MAX_SIZE = 10;

class Queue {
public:
    int data[MAX_SIZE];
    int front = 0;
    int rear = -1;
    int size = 0;


    void enqueue(int value) {
        if (size < MAX_SIZE) {
            rear = (rear + 1) % MAX_SIZE;
            data[rear] = value;
            size++;
            std::cout << "Enqueued: " << value << std::endl;
        } else {
            std::cout << "Queue is full!" << std::endl;
        }
    }


    void dequeue() {
        if (size > 0) {
            std::cout << "Dequeued: " << data[front] << std::endl;
            front = (front + 1) % MAX_SIZE;
            size--;
        } else {
            std::cout << "Queue is empty!" << std::endl;
        }
    }


    void reverseFirstK(int k) {
        if (k > size) {
            std::cout << "Not enough elements to reverse!" << std::endl;
            return;
        }

        int start = front;
        int end = (front + k - 1) % MAX_SIZE;

        while (start < end) {
            int temp = data[start];
            data[start] = data[end];
            data[end] = temp;

            start = (start + 1) % MAX_SIZE;
            if (start == end) break;
            end = (end - 1 + MAX_SIZE) % MAX_SIZE;
        }

        std::cout << "First " << k << " elements reversed." << std::endl;
        displayQueue();
```

```
    }


    void interleaveHalves() {
        if (size <= 1) {
            std::cout << "Not enough elements to interleave!" << std::endl;
            return;
        }

        int mid = size / 2;
        int temp[MAX_SIZE];

        int i = front;
        int j = (front + mid) % MAX_SIZE;
        int k = 0;

        while (k < size) {
            temp[k++] = data[i];
            i = (i + 1) % MAX_SIZE;

            if (k < size) {
                temp[k++] = data[j];
                j = (j + 1) % MAX_SIZE;
            }
        }

        for (int i = 0; i < size; i++) {
            data[(front + i) % MAX_SIZE] = temp[i];
        }

        std::cout << "Halves interleaved." << std::endl;
        displayQueue();
    }


    void displayQueue() {
        std::cout << "Queue: ";
        for (int i = 0; i < size; i++) {
            std::cout << data[(front + i) % MAX_SIZE] << " ";
        }
        std::cout << std::endl;
    }
};

int main() {
    Queue queue;


    queue.enqueue(1);
    queue.enqueue(2);
    queue.enqueue(3);
    queue.enqueue(4);
    queue.enqueue(5);

    queue.displayQueue();
```

```
    queue.reverseFirstK(3);

    queue.interleaveHalves();

    queue.dequeue();
    queue.dequeue();

    queue.displayQueue();

    return 0;
}
```

OUTPUT:



4. Linked List Problem: Create a singly linked list (not STL) that: (20 marks)
    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

```
#include <iostream>

class Node {

public:
```

```
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class LinkedList {
public:
    Node* head = nullptr;



    void insertAtStart(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
        } else {
            newNode->next = head;
            head = newNode;
        }
    }



    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
```

```
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}



void insertAtPosition(int value, int pos) {
    Node* newNode = new Node(value);
    if (pos == 0) {
        insertAtStart(value);
        return;
    }

    Node* temp = head;
    for (int i = 0; i < pos - 1 && temp != nullptr; i++) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        std::cout << "Position exceeds list length!" << std::endl;
        return;
```

```
    }

    newNode->next = temp->next;
    temp->next = newNode;
}


void detectAndRemoveLoop() {
    Node* slow = head;
    Node* fast = head;

    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast) {
            std::cout << "Loop detected!" << std::endl;



            slow = head;
            while (slow->next != fast->next) {
                slow = slow->next;
                fast = fast->next;
            }
            fast->next = nullptr;
            std::cout << "Loop removed." << std::endl;
            return;
```

```
        }
    }

    std::cout << "No loop found." << std::endl;
}



void findNthNodeFromEnd(int n) {
    Node* mainPtr = head;
    Node* refPtr = head;

    for (int i = 0; i < n; i++) {
        if (refPtr == nullptr) {
            std::cout << "n is greater than the no. of nodes in list" << std::endl;
            return;
        }
        refPtr = refPtr->next;
    }

    while (refPtr != nullptr) {
        mainPtr = mainPtr->next;
        refPtr = refPtr->next;
    }

    if (mainPtr != nullptr) {
        std::cout << "Node no. " << n << " from the end is " << mainPtr->data << std::endl;
    }
```

```
}



void reverseInGroupsOfK(int k) {
    Node* current = head;
    Node* prev = nullptr;
    Node* next = nullptr;

    while (current != nullptr) {
        Node* first = current;
        Node* last = current;



        for (int i = 0; i < k - 1 && current != nullptr; i++) {
            current = current->next;
        }

        if (current == nullptr) break;

        last = current;
        current = current->next;



        Node* prevGroup = nullptr;
        Node* nextGroup = nullptr;
        for (int i = 0; i < k; i++) {
            nextGroup = first->next;
```

```
            first->next = prevGroup;
            prevGroup = first;
            first = nextGroup;
        }



        if (head == last) {
            head = prevGroup;
        } else {
            Node* temp = head;
            while (temp->next != last) {
                temp = temp->next;
            }
            temp->next = prevGroup;
        }

        last->next = current;
        prevGroup = last;
    }

    std::cout << "List reversed in groups of " << k << "." << std::endl;
    displayList();
}



void displayList() {
    Node* temp = head;
```

```
        while (temp != nullptr) {
            std::cout << temp->data << " ";
            temp = temp->next;
        }
        std::cout << std::endl;
    }
};

int main() {
    LinkedList list;



    list.insertAtEnd(1);
    list.insertAtEnd(2);
    list.insertAtEnd(3);
    list.insertAtEnd(4);
    list.insertAtEnd(5);

    std::cout << "Initial List: ";
    list.displayList();

    list.insertAtStart(0);
    std::cout << "After inserting at start: ";
    list.displayList();

    list.insertAtPosition(6, 3);
    std::cout << "After inserting at position: ";
```

```
    list.displayList();

    list.detectAndRemoveLoop();

    list.findNthNodeFromEnd(2);

    list.reverseInGroupsOfK(2);
    std::cout << "Final List: ";
    list.displayList();

    return 0;
}
```