C++ Worksheet Task_4

ASSESSMENT

WEIGHTAGE AND TYPE: 12.5%

YEAR: 2024-25

STUDENT NAME: SWOYAMRAJ SHRESTHA

STUDENT ID: 24030185

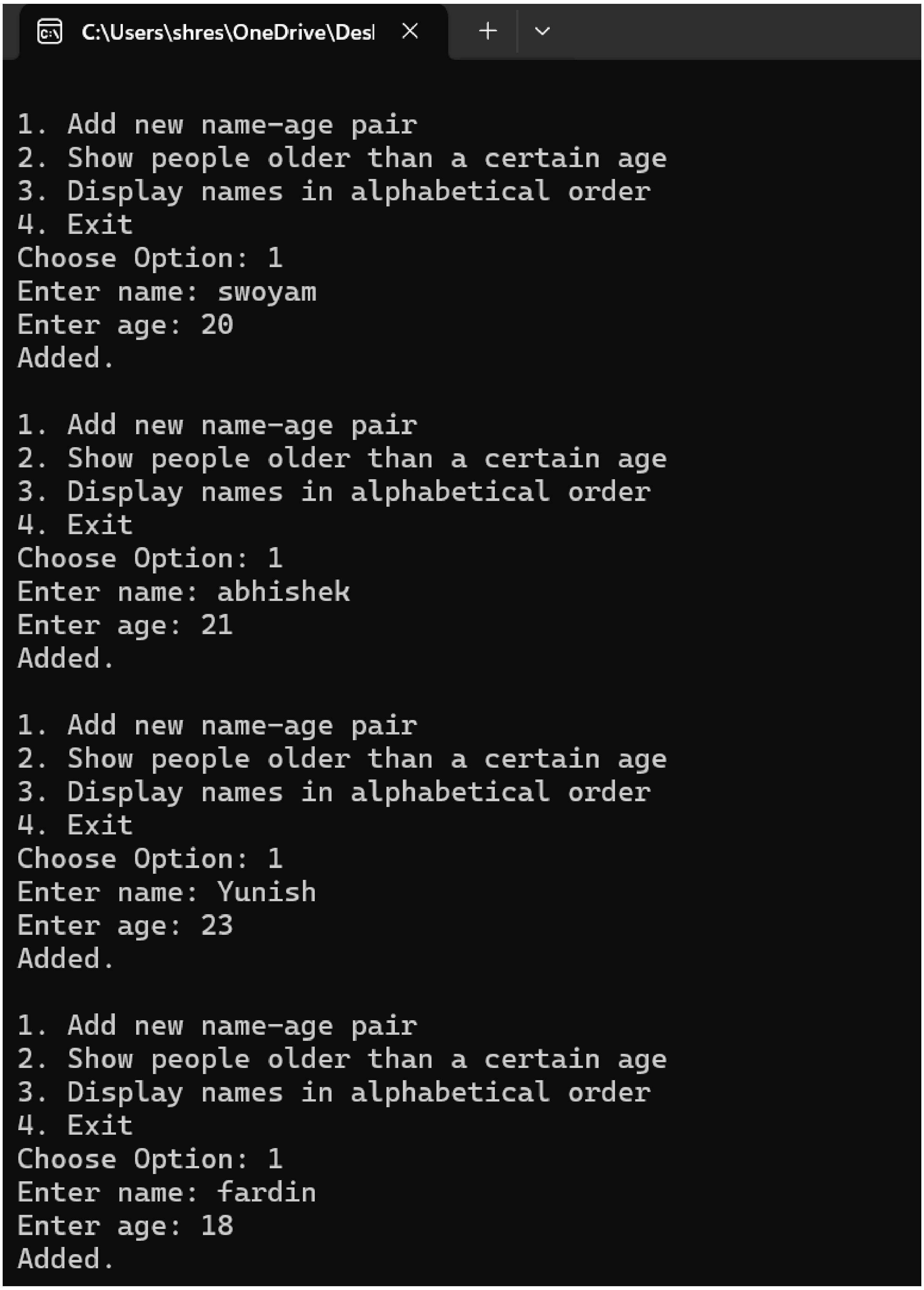
Question_1.1

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

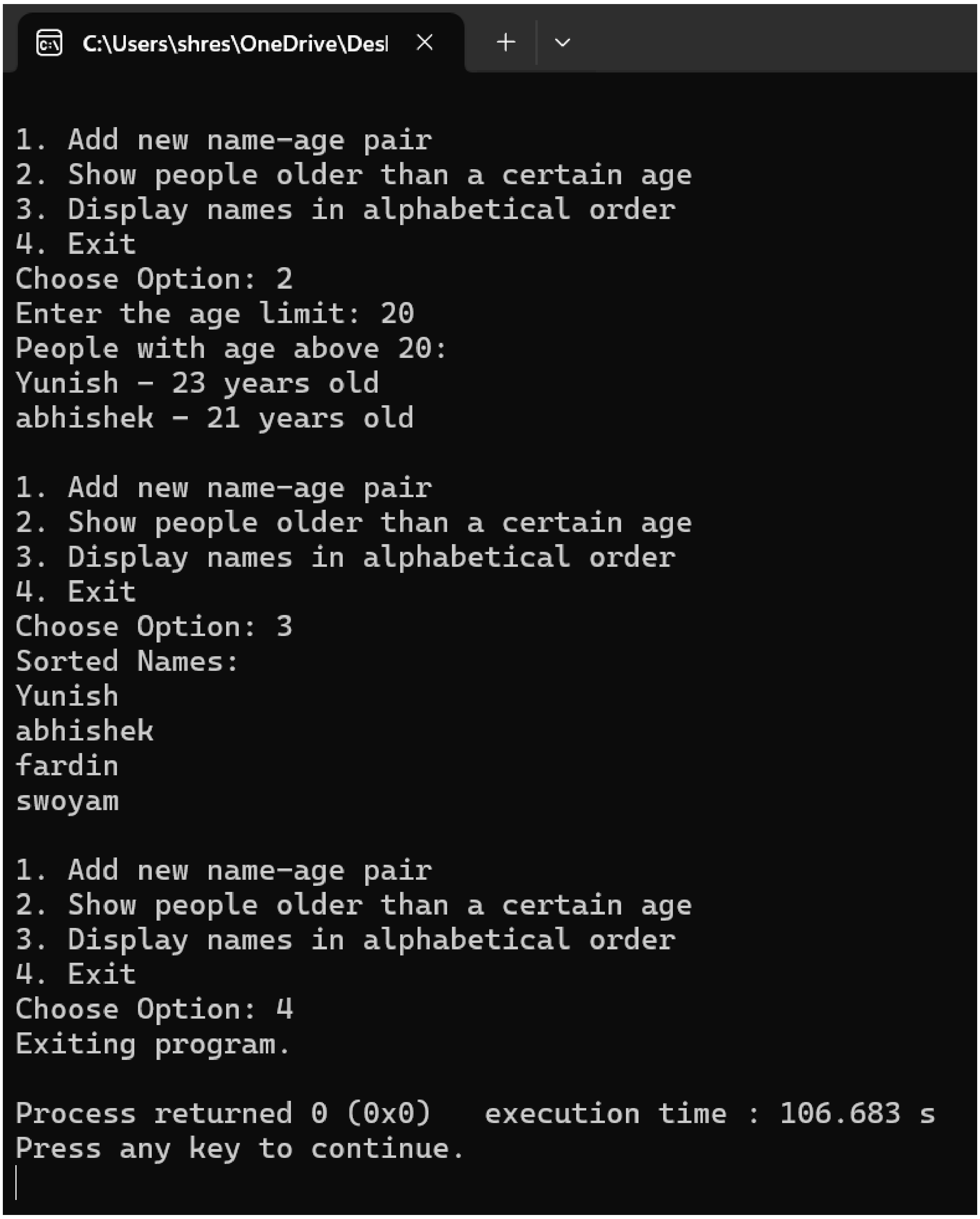
```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <algorithm>
using namespace std;

void addNNAP(vector<string>& names, map<string, int>& ages, const string&
name, int age)
{
   names.push_back(name);
   ages[name] = age;
}

void showPAA(const map<string, int>& ages, int limit)
{
   cout << "People with age above " << limit << ":\n";
   for (const auto& p : ages)
   {
      if (p.second > limit)
      {
         cout << p.first << " - " << p.second << " years old" << endl;
      }
   }
}
void DisplaySortedNames(vector<string>& names)
{
   vector<string> sortedname = names;
   sort(sortedname.begin(), sortedname.end());
   cout << "Sorted Names:\n";
   for (const auto& name : sortedname)
   {
      cout << name << endl;
```

```
    }
}
int main()
{
    vector<string> names;
    map<string, int> ages;
    int option;
    do
    {
        cout << "\n1. Add new name-age pair\n";
        cout << "2. Show people older than a certain age\n";
        cout << "3. Display names in alphabetical order\n";
        cout << "4. Exit\n";
        cout << "Choose Option: ";
        cin >> option;

        if (option == 1)
        {
            string name;
            int age;
            cout << "Enter name: ";
            cin >> name;
            cout << "Enter age: ";
            cin >> age;
            addNNAP(names, ages, name, age);
            cout << "Added.\n";
        }
        else if (option == 2)
        {
            int limit;
            cout << "Enter the age limit: ";
            cin >> limit;
            showPAA(ages, limit);
        }
        else if (option == 3)
        {
            DisplaySortedNames(names);
        }
        else if (option == 4)
        {
            cout << "Exiting program.\n";
        }
        else
        {
            cout << "Invalid option.\n";
        }

    }
    while (option != 4);

    return 0;
}
```

Output:



```
C:\Users\shres\OneDrive\Desl

1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 1
Enter name: swoyam
Enter age: 20
Added.

1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 1
Enter name: abhishek
Enter age: 21
Added.

1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 1
Enter name: Yunish
Enter age: 23
Added.

1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 1
Enter name: fardin
Enter age: 18
Added.
```



```
1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 2
Enter the age limit: 20
People with age above 20:
Yunish - 23 years old
abhishek - 21 years old

1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 3
Sorted Names:
Yunish
abhishek
fardin
swoyam

1. Add new name-age pair
2. Show people older than a certain age
3. Display names in alphabetical order
4. Exit
Choose Option: 4
Exiting program.

Process returned 0 (0x0)   execution time : 106.683 s
Press any key to continue.
```

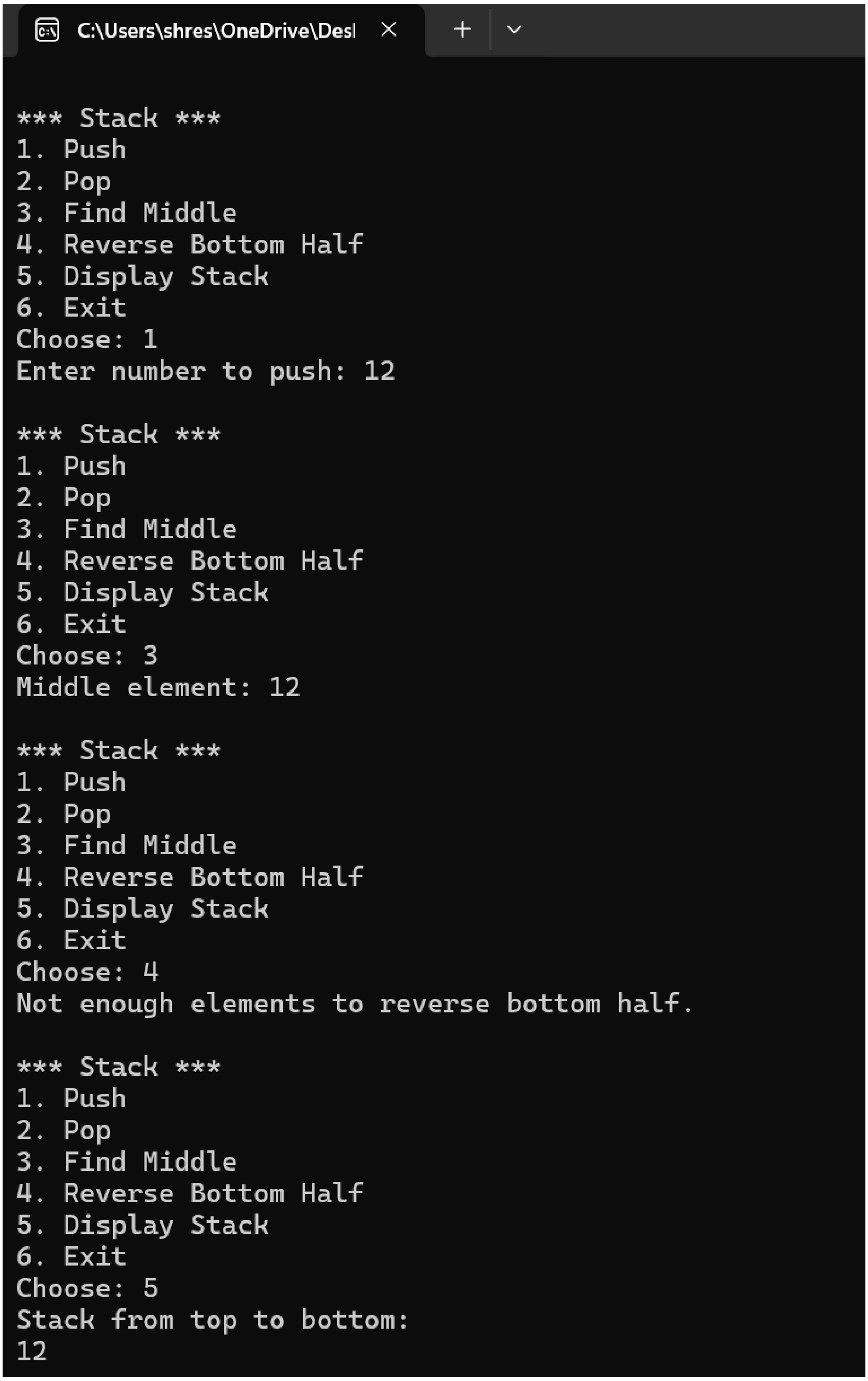
Question_1.2

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

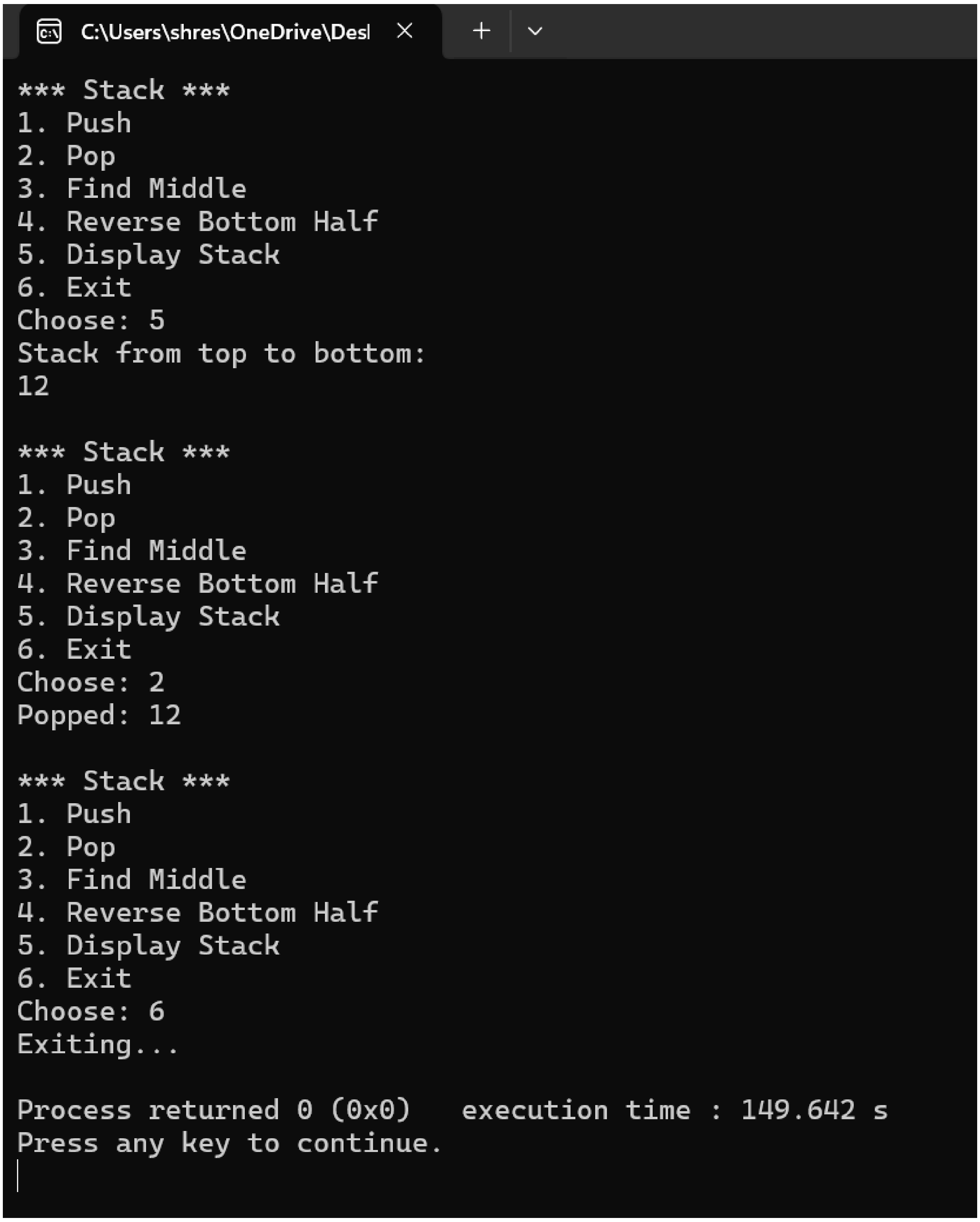
```
#include <iostream>
using namespace std;

const int SIZE = 10;

class MyStack
{
private:
    int stackArr[SIZE];
    int top;

public:
    MyStack()
    {
        top = -1;
    }
    void pushItem(int val)//pushes data into stack
    {
        if (top == SIZE - 1)
            {
            cout << "Stack is full. Can't push! " << val << endl;
            return;
            }
        top++;
        stackArr[top] = val;
    }

    int popItem() //removes top item from stack.
    {
        if (top == -1)
        {
            cout << "Stack is empty. Nothing to pop.\n";
            return -1;
        }
        int poppedVal = stackArr[top];
        top--;
        return poppedVal;
    }

    void showMiddle()  //shows the middle value.
    {
        if (top == -1)
        {
            cout << "Stack is empty.\n";
            return;
        }

        int mid = top / 2;
        cout << "Middle element: " << stackArr[mid] << endl;
    }

void reverseLowerHalf() //reverse bottom half of the stack
{
    if (top < 1)
    {
        cout << "Not enough elements to reverse bottom half.\n";
        return;
    }

int halfCount = (top + 1) / 2;

for (int i = 0; i < halfCount / 2; i++)
{
    int temp = stackArr[i];
    stackArr[i] = stackArr[halfCount - i - 1];
    stackArr[halfCount - i - 1] = temp;
}
    cout << "Bottom half reversed.\n";
    }

void printStack() //prints the stack
{
    if (top == -1)
{
    cout << "Stack is empty.\n";
    return;
}
```

```
    cout << "Stack from top to bottom:\n";
    for (int i = top; i >= 0; i--)
        {
            cout << stackArr[i] << " ";
        }
    cout << endl;
}
};

int main()
{
    MyStack s;
    int option, num;

    while (true)
        {
            cout << "\n*** Stack ***\n";
            cout << "1. Push\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 << "Choose: ";
            cin >> option;

        switch (option)
        {
            case 1:
                cout << "Enter number to push: ";
                cin >> num;
                s.pushItem(num);
                break;
            case 2:
                num = s.popItem();
                if (num != -1)
                    cout << "Popped: " << num << endl;
                break;
            case 3:
                s.showMiddle();
                break;
            case 4:
                s.reverseLowerHalf();
            break;
         case 5:
            s.printStack();
            break;
         case 6:
            cout << "Exiting...\n";
            return 0;
         default:
            cout << "Invalid input. Please try again.\n";
      }
   }

   return 0;
}
```

Output:



```

*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 1
Enter number to push: 12

*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 3
Middle element: 12

*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 4
Not enough elements to reverse bottom half.

*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 5
Stack from top to bottom:
12
```



```
*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 5
Stack from top to bottom:
12

*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 2
Popped: 12

*** Stack ***
1. Push
2. Pop
3. Find Middle
4. Reverse Bottom Half
5. Display Stack
6. Exit
Choose: 6
Exiting...

Process returned 0 (0x0)   execution time : 149.642 s
Press any key to continue.
```

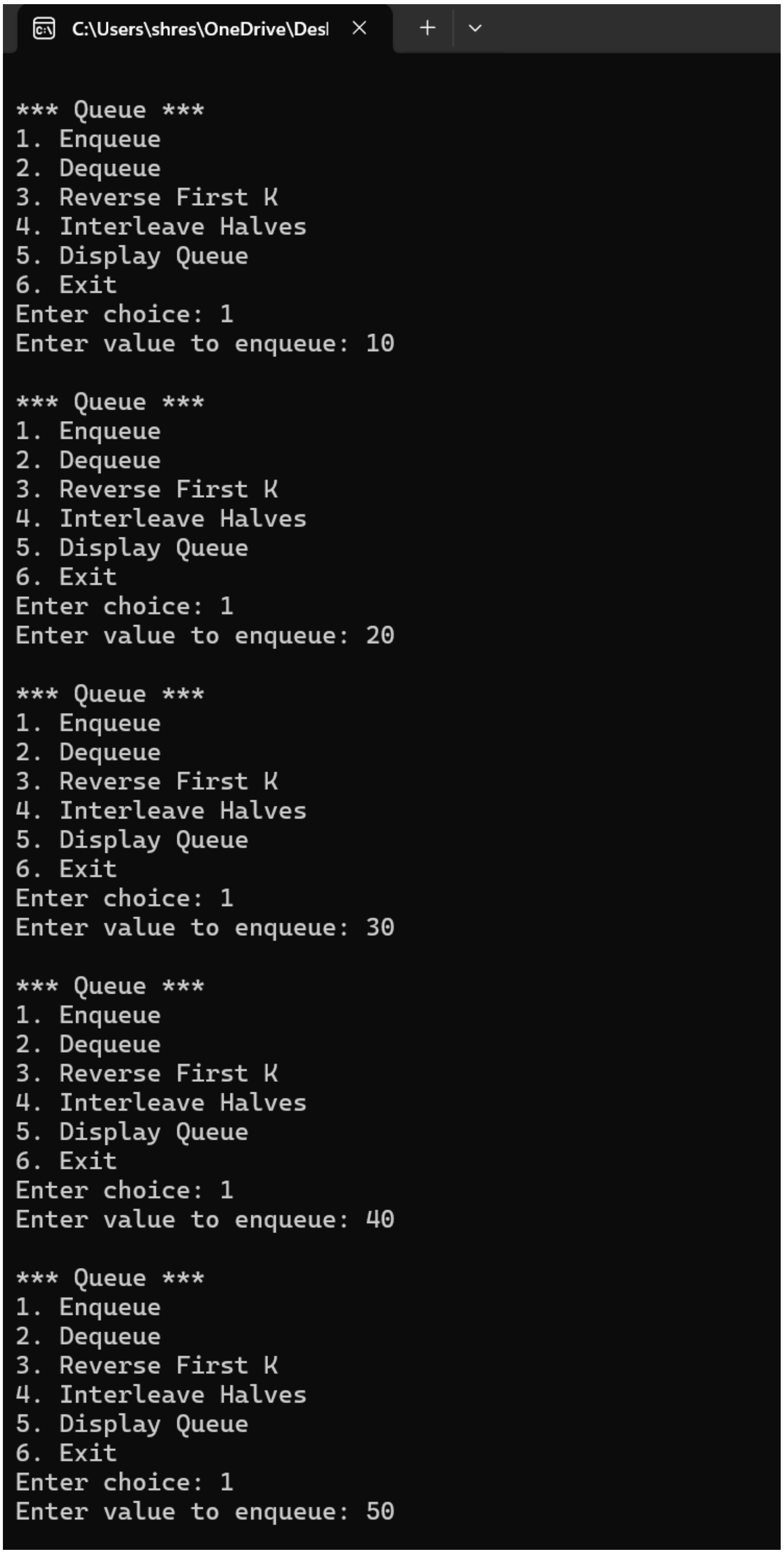
Question_1.3

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

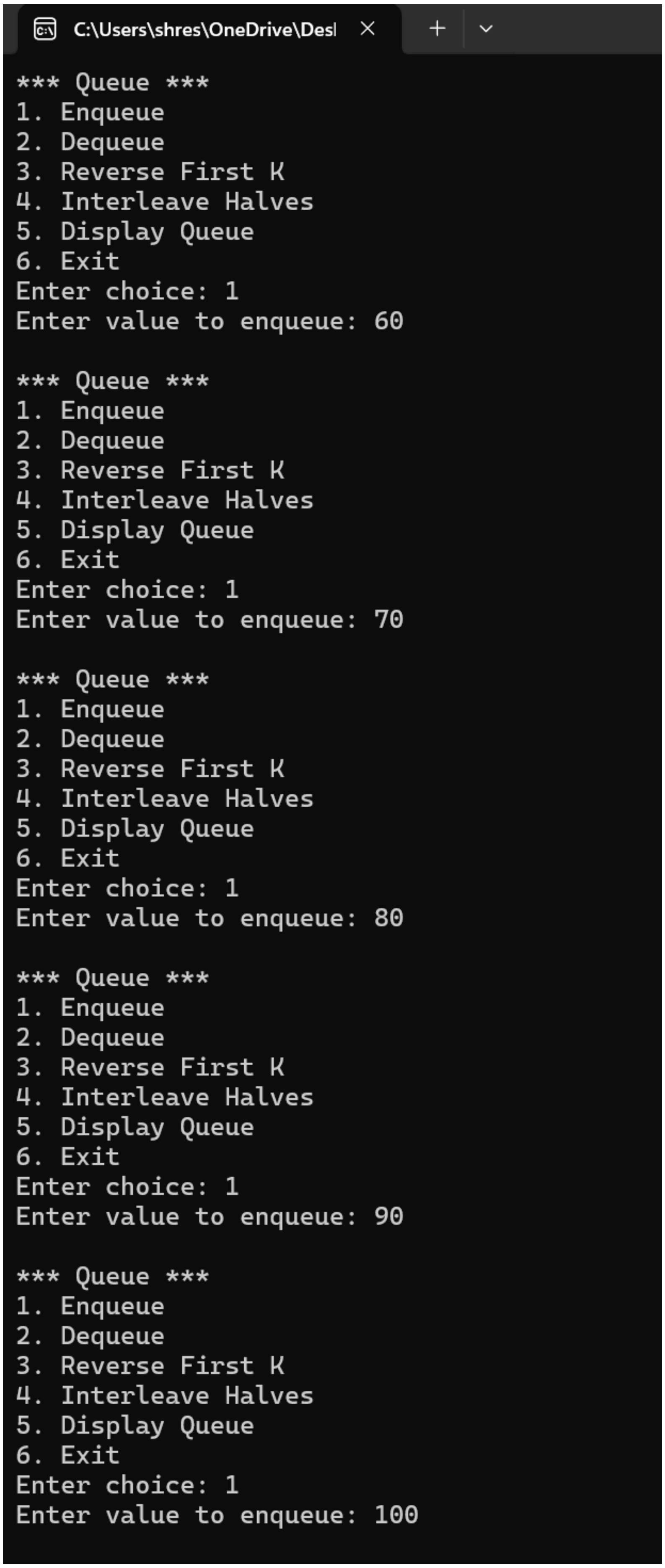
```
#include <iostream>
using namespace std;

const int MAX = 10;

class Queue
{
private:
    int arr[MAX];
    int front, rear, Size;

public:
    Queue()
    {
        front = 0;
        rear = -1;
        Size = 0;
    }

    bool isFull()
    {
        return Size == MAX;
    }

    bool isEmpty()
    {
        return Size == 0;
    }

    void add(int val)
    {
        if (isFull())
        {
            cout << "Queue is full. Can't enqueue " << val << endl;
            return;
        }
        rear = (rear + 1) % MAX;
        arr[rear] = val;
        Size++;
    }

int remove()
{
    if (isEmpty())
    {
        cout << "Queue is empty. Nothing to dequeue.\n";
        return -1;
    }
    int removed = arr[front];
    front = (front + 1) % MAX;
    Size--;
    return removed;
}

void reverseK(int k)
{
    if (k > Size || k <= 0)
    {
        cout << "K is invalid for current queue.\n";
        return;
    }
    for (int i = 0; i < k / 2; i++)
    {
        int a = (front + i) % MAX;
        int b = (front + k - 1 - i) % MAX;
        swap(arr[a], arr[b]);
    }
    cout << "First " << k << " elements reversed.\n";
}
void interleave()
{
        if (Size % 2 != 0)
        {
            cout << "Queue must have even number of elements to interleave.\n";
            return;
        }
```

```
int temp[MAX];
int half = Size / 2;

for (int i = 0; i < half; i++)
    {
        temp[2 * i] = arr[(front + i) % MAX];
        temp[2 * i + 1] = arr[(front + half + i) % MAX];
    }

for (int i = 0; i < Size; i++)
{
    arr[(front + i) % MAX] = temp[i];
}
cout << "Queue interleaved successfully.\n";
}

void show()
{
    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] << " ";
        }
    cout << endl;
}
};

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

    while (true)
    {
        cout << "\n*** Queue ***\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Reverse First K\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 to enqueue: ";
                cin >> num;
                q.add(num);
                break;
            case 2:
                num = q.remove();
                if (num != -1)
                    cout << "Dequeued: " << num << endl;
                break;
            case 3:
                cout << "Enter value of K: ";
                cin >> k;
                q.reverseK(k);
                break;
            case 4:
                q.interleave();
                break;
            case 5:
                q.show();
                break;
            case 6:
                cout << "Program ended.\n";
                return 0;
            default:
                cout << "Invalid option. Please try again.\n";
        }
    }
    return 0;
}
```

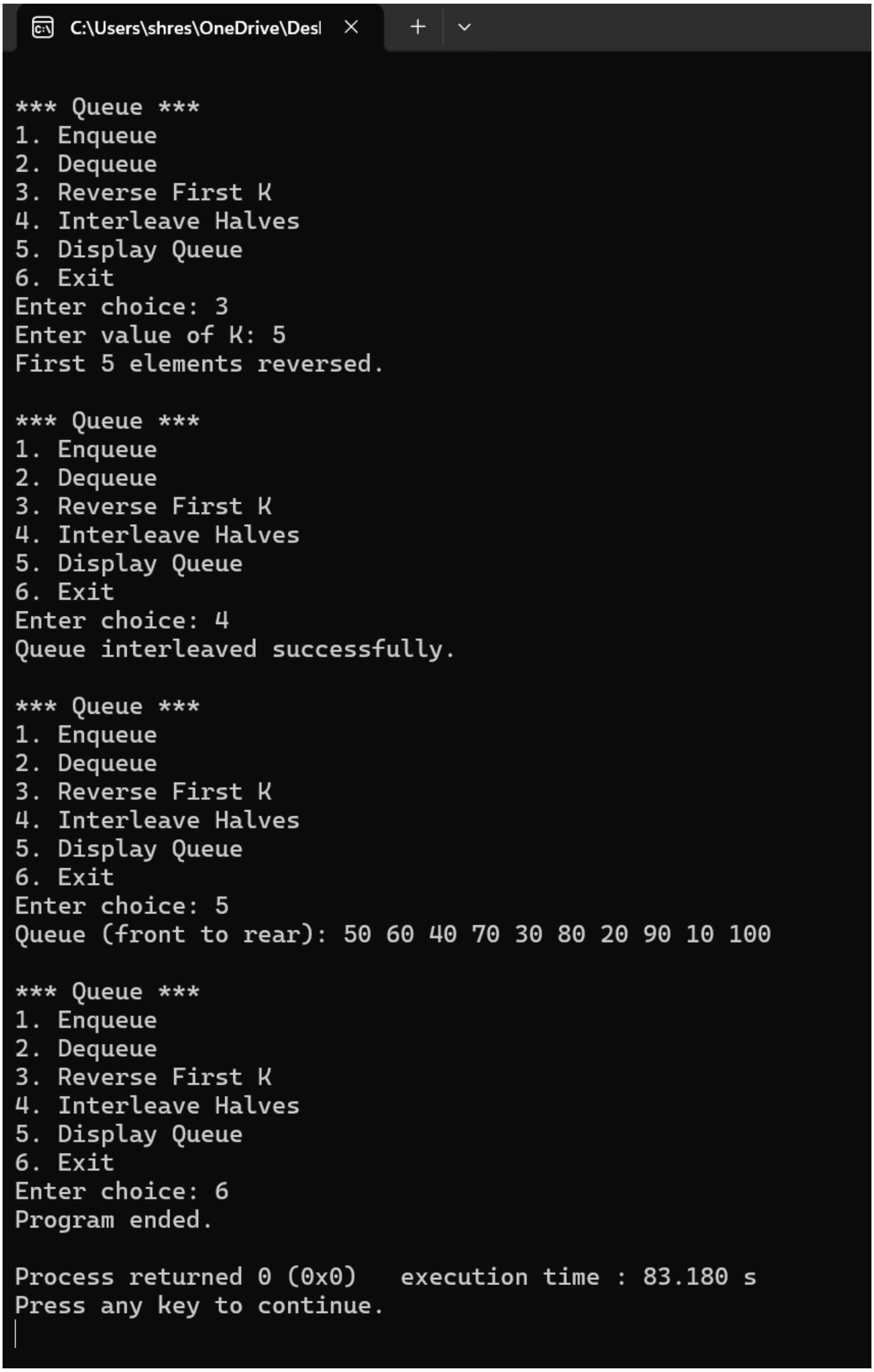
Output:



```
*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 10

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 20

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 30

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 40

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 50
```



```
*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 60

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 70

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 80

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 90

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 1
Enter value to enqueue: 100
```



```
*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 3
Enter value of K: 5
First 5 elements reversed.

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

*** Queue ***
1. Enqueue
2. Dequeue
3. Reverse First K
4. Interleave Halves
5. Display Queue
6. Exit
Enter choice: 5
Queue (front to rear): 50 60 40 70 30 80 20 90 10 100

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

Process returned 0 (0x0)   execution time : 83.180 s
Press any key to continue.
```

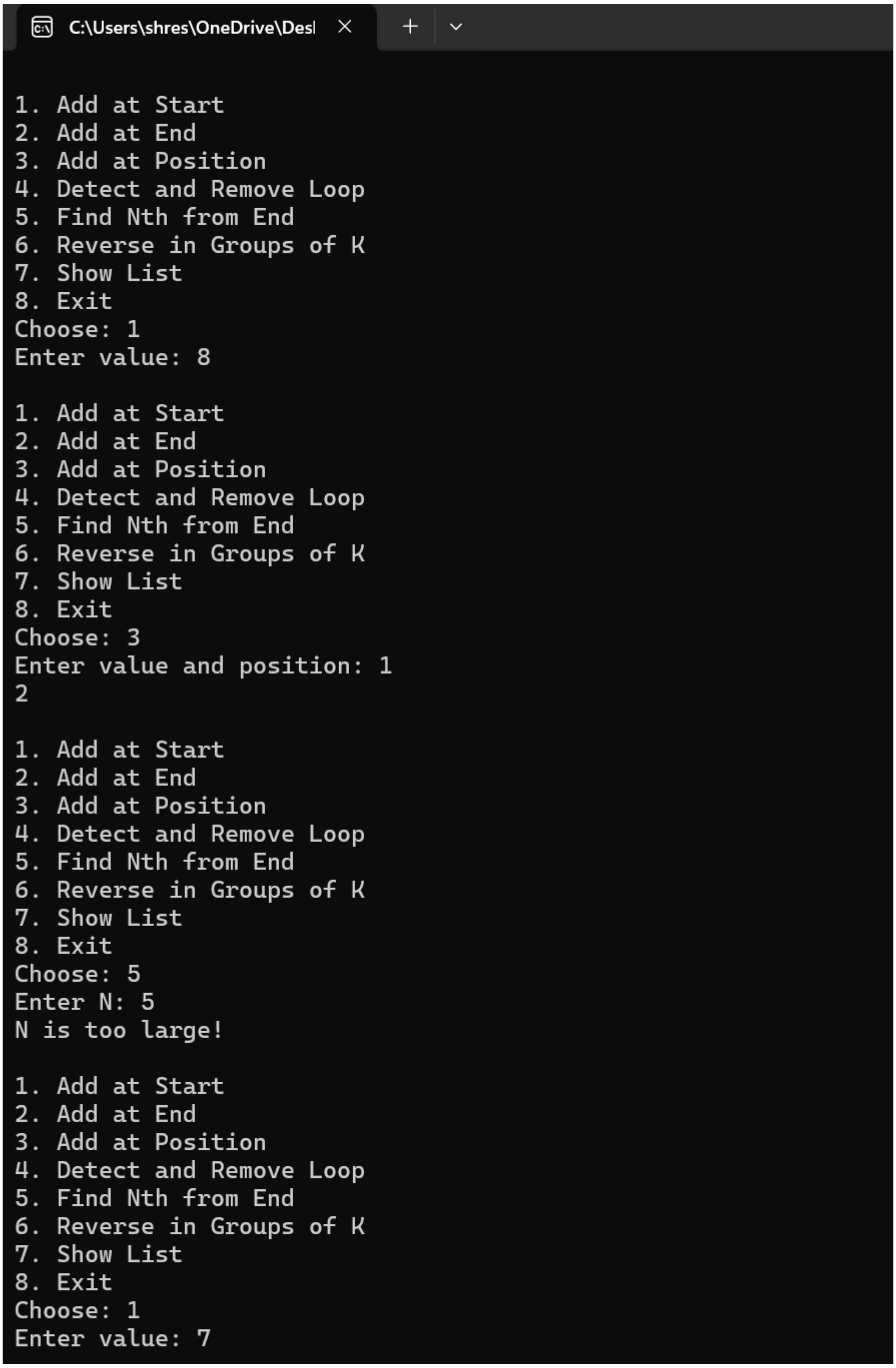
Question_1.4

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

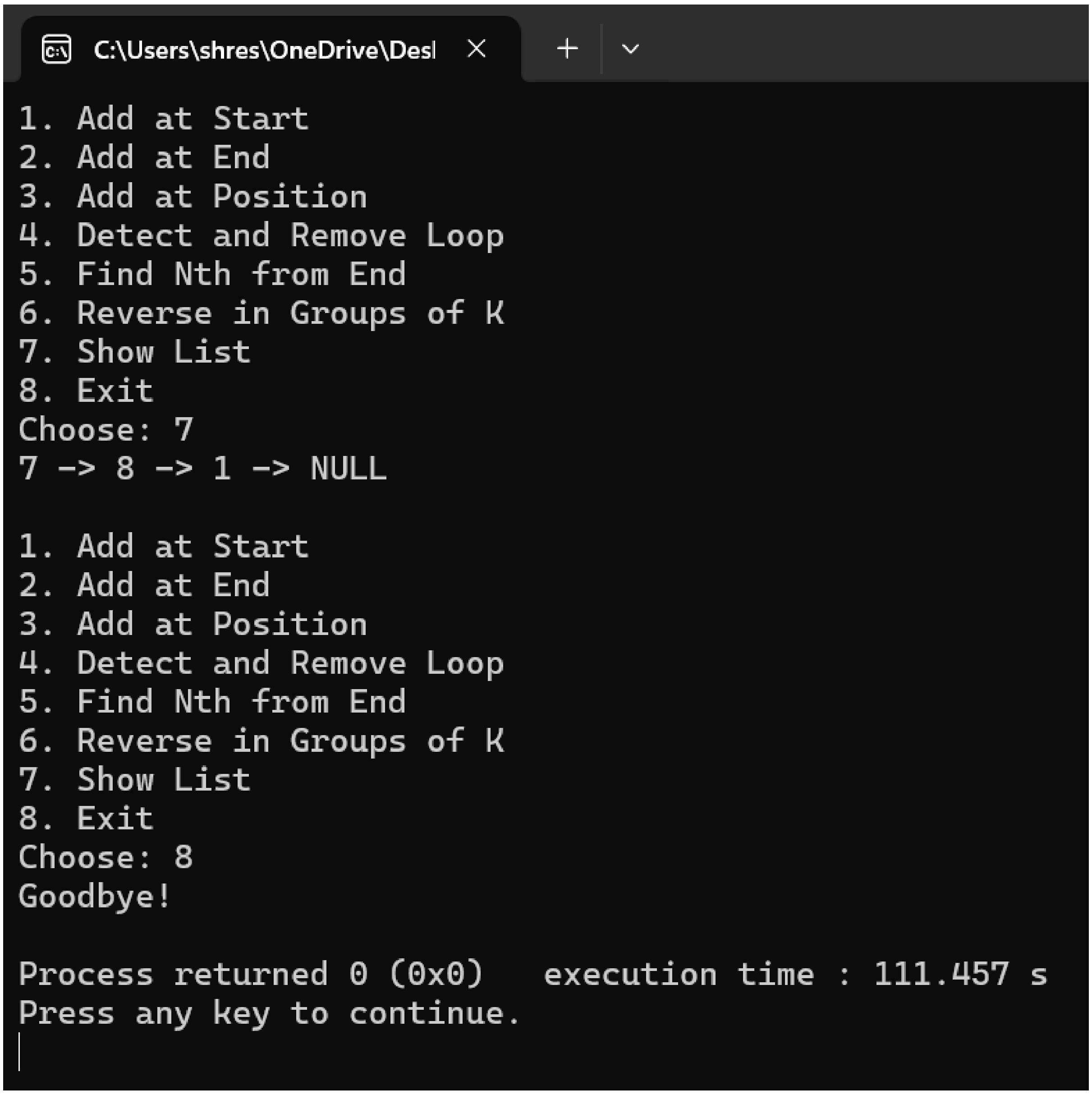
```
#include <iostream>
using namespace std;

class Node
{
public:
    int data;
    Node* next;
    Node(int val)
    {
        data = val;
        next = NULL;
    }
};

class MyList
{
    Node* head;

public:
    MyList()
    {
        head = NULL;
    }

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

    void addAtEnd(int val)
    {
```

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

void addAtPos(int val, int pos)
{
    if (pos <= 0)
        {
            cout << "Invalid position!" << endl;
            return;
        }
    if (pos == 1)
        {
            addAtStart(val);
            return;
        }
    Node* newNode = new Node(val);
    Node* temp = head;
    for (int i = 1; i < pos - 1 && temp != NULL; i++)
        {
            temp = temp->next;
        }
    if (temp == NULL)
        {
            cout << "Position out of range!" << endl;
            return;
        }
        newNode->next = temp->next;
        temp->next = newNode;
}

void findLoopAndRemove()
{
    Node* slow = head;
    Node* fast = head;
    while (fast && fast->next)
        {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast)
            {
                cout << "Loop found, removing..." << endl;
                removeLoop(slow);
                return;
            }
        }
    cout << "No loop found." << endl;
}

void removeLoop(Node* loopNode)
{
    Node* ptr1 = head;
    Node* ptr2 = loopNode;
    while (ptr1->next != ptr2->next)
    {
        ptr1 = ptr1->next;
        ptr2 = ptr2->next;
    }
    ptr2->next = NULL;
}

void printNthFromEnd(int n)
{
    Node* p1 = head;
    Node* p2 = head;
    for (int i = 0; i < n; i++)
        {
        if (p1 == NULL)
        {
            cout << "N is too large!" << endl;
            return;
        }
        p1 = p1->next;
        }
    while (p1 != NULL)
    {
        p1 = p1->next;
        p2 = p2->next;
    }
    cout << n << "th node from end is: " << p2->data << endl;
}

Node* reverseK(Node* node, int k)
{
    if (!node || k <= 1) return node;
    Node* prev = NULL;
    Node* curr = node;
    Node* next = NULL;
    int cnt = 0;
    Node* temp = node;

    for (int i = 0; i < k && temp; i++)
        {
        temp = temp->next;
        cnt++;
        }

    if (cnt < k) return node;

    cnt = 0;
    while (curr && cnt < k)
        {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
        cnt++;
        }

    if (next != NULL)
        {
            node->next = reverseK(next, k);
        }

    return prev;
}

void reverseInK(int k)
{
    head = reverseK(head, k);
    cout << "Reversed in group of " << k << endl;
    }

    void show()
    {
        Node* temp = head;
        while (temp != NULL)
            {
                cout << temp->data << " -> ";
                temp = temp->next;
            }
        cout << "NULL" << endl;
    }
};

int main() {
    MyList mylist;
    int ch, val, pos, k;

    do {
        cout << "\n1. Add at Start\n";
        cout << "2. Add at End\n";
        cout << "3. Add at Position\n";
        cout << "4. Detect and Remove Loop\n";
        cout << "5. Find Nth from End\n";
        cout << "6. Reverse in Groups of K\n";
        cout << "7. Show List\n";
        cout << "8. Exit\n";
        cout << "Choose: ";
        cin >> ch;

        switch (ch)
        {
            case 1:
                cout << "Enter value: ";
                cin >> val;
                mylist.addAtStart(val);
                break;
            case 2:
                cout << "Enter value: ";
                cin >> val;
                mylist.addAtEnd(val);
                break;
            case 3:
                cout << "Enter value and position: ";
                cin >> val >> pos;
                mylist.addAtPos(val, pos);
                break;
            case 4:
                mylist.findLoopAndRemove();
                break;
            case 5:
                cout << "Enter N: ";
                cin >> pos;
                mylist.printNthFromEnd(pos);
                break;
            case 6:
                cout << "Enter K: ";
                cin >> k;
                mylist.reverseInK(k);
                break;
            case 7:
                mylist.show();
                break;
            case 8:
                cout << "Goodbye!" << endl;
                break;
            default:
                cout << "Invalid option!" << endl;
        }

    } while (ch != 8);

    return 0;
}
```

Output:



```
1. Add at Start
2. Add at End
3. Add at Position
4. Detect and Remove Loop
5. Find Nth from End
6. Reverse in Groups of K
7. Show List
8. Exit
Choose: 1
Enter value: 8

1. Add at Start
2. Add at End
3. Add at Position
4. Detect and Remove Loop
5. Find Nth from End
6. Reverse in Groups of K
7. Show List
8. Exit
Choose: 3
Enter value and position: 1
2

1. Add at Start
2. Add at End
3. Add at Position
4. Detect and Remove Loop
5. Find Nth from End
6. Reverse in Groups of K
7. Show List
8. Exit
Choose: 5
Enter N: 5
N is too large!

1. Add at Start
2. Add at End
3. Add at Position
4. Detect and Remove Loop
5. Find Nth from End
6. Reverse in Groups of K
7. Show List
8. Exit
Choose: 1
Enter value: 7
```



```
1. Add at Start
2. Add at End
3. Add at Position
4. Detect and Remove Loop
5. Find Nth from End
6. Reverse in Groups of K
7. Show List
8. Exit
Choose: 7
7 -> 8 -> 1 -> NULL

1. Add at Start
2. Add at End
3. Add at Position
4. Detect and Remove Loop
5. Find Nth from End
6. Reverse in Groups of K
7. Show List
8. Exit
Choose: 8
Goodbye!

Process returned 0 (0x0)   execution time : 111.457 s
Press any key to continue.
```