Stony Brook University
College of Engineering and Applied Science

ESE 224.L02

Lab 8

Ryan Lin

Professor: Xin Wang

Task 1

task1.cpp

```cpp
void reverse_list(list<int> &ml, int left,int right){
    list<int>::iterator start = ml.begin();
    list<int>::iterator end = ml.begin();
    int temp = 0;

    //get left point
    for(int i = 0; i < left; i++){
        start++;
    }
    //get right point
    for(int i =0; i < right; i++){
        end++;
    }
    //reverse
    while(start != end)
    {
        temp = *start;
        *start = *end;
        *end = temp;

        start++;

        if(start==end)
        {
            break;
        }
        end--;
    }
}

int main(){
    list<int> myList = {0,1,2,3,4,5,6,7,8,9};
    reverse_list(myList, 1, 5);
    cout << "Output: ";
    for (int value : myList) {
        cout << value << " ";
    }
    cout << endl;
```

```
}
```

Output:

```
Output: 0 5 4 3 2 1 6 7 8 9
```

Task2:

task2.cpp

```cpp
struct ListNode{
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* addTwoNumbers(ListNode* l1, ListNode* l2){
    stack<int> stack1, stack2;

    while(l1){
        stack1.push(l1->val);
        l1 = l1-> next;
    }

    while(l2){
        stack2.push(l2->val);
        l2 = l2->next;
    }
    int carry = 0;
    ListNode* result = nullptr;

    while(!stack1.empty() || !stack2.empty() || carry){
        int sum = carry;
        if (!stack1.empty()) {
            sum += stack1.top();
            stack1.pop();
        }
        if (!stack2.empty()) {
            sum += stack2.top();
            stack2.pop();
        }

        carry = sum / 10;
```

```
        sum %= 10;

        ListNode* newNode = new ListNode(sum);
        newNode->next = result;
        result = newNode;
    }

    return result;
}
// Function to print a linked list.
void printList(ListNode* head) {
    cout << "Output: ";
    while (head) {
        cout << head->val;
        if (head->next) {
            cout << ", ";
        }
        head = head->next;
    }
    cout << endl;
}

ListNode* createList() {
    ListNode* head = nullptr;
    ListNode* tail = nullptr;

    cout << "Enter a number (-1 to end): ";
    int value;
    while (cin >> value && value != -1) {
        ListNode* newNode = new ListNode(value);
        if (!head) {
            head = newNode;
            tail = head;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return head;
```

```
}

int main() {
    cout << "Enter the first list of numbers:" << endl;
    ListNode* l1 = createList();

    cout << "Enter the second list of numbers:" << endl;
    ListNode* l2 = createList();

    // Add the two numbers.
    ListNode* result = addTwoNumbers(l1, l2);

    printList(result);

    return 0;
}
```

Output:

```
Enter the first list of numbers:
Enter a number (-1 to end): 7 2 4 3 -1
Enter the second list of numbers:
Enter a number (-1 to end): 5 6 4 -1
Output: 7, 8, 0, 7
```

Task 3

task3.cpp

```
class MyQueue{
public:
    stack<int> inputStack;
    stack<int> outputStack;
    void push(int x){
        inputStack.push(x);
        cout << "add element: " << x << endl;
    }

    int pop(){
        if(outputStack.empty()){
            while(!inputStack.empty()){
                outputStack.push(inputStack.top());
                inputStack.pop();
            }
        }
        int frontValue = outputStack.top();
        outputStack.pop();
        cout <<"pop out element: "<< frontValue << endl;
        return frontValue;
    }

    int front(){
        if (outputStack.empty()) {
            while (!inputStack.empty()) {
                outputStack.push(inputStack.top());
                inputStack.pop();
            }
        }
        cout << "get element: " << outputStack.top() << endl;
        return outputStack.top();
    }

    bool empty() {
        return inputStack.empty() && outputStack.empty();
    }
};
int main(){
```

```
    MyQueue* obj = new MyQueue();

    obj->push(1);
    obj->push(3);

    int res1 = obj ->front();
    cout << "=====> the top value is: " << res1 << endl;
    obj->pop();

    obj->push(5);

    int res2 = obj->front();
    cout << "=====> the top value is: " << res2 << endl;
    obj->pop();
    obj->pop();

    bool res3 = obj->empty();
    cout << "=====> is myQueue empty?: " << res3 << endl;

}
```

Output:

```
add element: 1
add element: 3
get element: 1
=====> the top value is: 1
pop out element: 1
add element: 5
get element: 3
=====> the top value is: 3
pop out element: 3
pop out element: 5
=====> is myQueue empty?: 1
```

Task4:

task4.cpp

```
class MyStack{
public:
    queue<int> q1;
    queue<int> q2;
```

```
void push(int x){
    cout << "add element: " << x << endl;
    if (q1.empty()){
        q1.push(x);
        while(!q2.empty()){
            q1.push(q2.front());
            q2.pop();
        }
    }
    else{
        q2.push(x);
        while(!q1.empty()){
            q2.push(q1.front());
            q1.pop();
        }
    }
}

int pop(){
    int topValue = top();
    if(!q1.empty()){
        q1.pop();
    }
    else{
        q2.pop();
    }
    cout << "pop out element: " << topValue << endl;
    return topValue;
}

int top(){
    if(!q1.empty()){
        return q1.front();
    }
    else{
        return q2.front();
    }
}
```

```
    bool empty(){
        return q1.empty() && q2.empty();
    }

};

int main(){
    MyStack* obj = new MyStack();

    obj->push(1);
    obj->push(3);

    int res1 = obj->top();
    cout << "=====> the top value is: " << res1 << endl;
    obj->pop();

    obj->push(5);

    int res2 = obj->top();
    cout << "=====> the top value is: " << res2 << endl;
    obj->pop();
    obj->pop();

    bool res3 = obj->empty();
    cout << "=====> is myStack empty?: " << res3 << endl;

    return 0;
}
```

Output:

```
add element: 1
add element: 3
=====> the top value is: 3
pop out element: 3
add element: 5
=====> the top value is: 5
pop out element: 5
pop out element: 1
=====> is myStack empty?: 1
```

Task 5

task5.cpp

```
class CustomContainer
{
private:
    stack<int> containerStack;

public:
    // Adds an integer to the container
    void add(int value)
    {
        containerStack.push(value);
    }

    // Returns the element at the specified index
    int get(int index)
    {
        if (index < 0 || index >= containerStack.size())
        {
            cout << "Invalid index input" << endl;
            return -1;
        }

        stack<int> tempStack;

        // pop elements until the element of the specified index is
reached
        for (int i = 0; i < index; ++i)
        {
```

```
            // push top value and pop
            tempStack.push(containerStack.top());
            containerStack.pop();
        }

        int resultElement = containerStack.top(); // value of element at
the index

        while (!tempStack.empty())
        {
            containerStack.push(tempStack.top());
            tempStack.pop();
        }

        return resultElement;
    }

    // Removes all occurrences of the specified value from the container
    void remove(int value)
    {
        stack<int> tempStack;

        for (int i = 0; i < containerStack.size(); ++i)
        {
            if (containerStack.top() == value)
            {
                containerStack.pop();
            }

            else
            {
                tempStack.push(containerStack.top());
                containerStack.pop();
            }
        }

        while (!tempStack.empty())
        {
            containerStack.push(tempStack.top());
            tempStack.pop();
```

```
        }
    }

    class CustomIterator
    {
    private:
        stack<int>* stackPtr;
        int currentIndex;

    public:
        // initialize constructor here
        CustomIterator(stack<int>* ptr, int index) : stackPtr(ptr),
currentIndex(index) {}

        int operator*() const
        {
            if (currentIndex < 0 || currentIndex >=
static_cast<int>(stackPtr->size()))
            {
                cout << ("Iterator is out of range") << endl;
            }

            stack<int> tempStack = *stackPtr;

            for (int i = 0; i < currentIndex; ++i)
            {
                tempStack.pop();
            }

            return tempStack.top();
        }

        // Moves the iterator to the next position
        void operator++()
        {
            if (currentIndex < 0 || currentIndex >=
static_cast<int>(stackPtr->size()))
            {
                cout << ("Iterator is out of range") << endl;
            }
```

```
            ++currentIndex;
        }

        // Check if two iterators are not equal
        bool operator!=(const CustomIterator& other) const
        {
            return currentIndex != other.currentIndex;
        }
    };

    CustomIterator begin() const
    {
        return CustomIterator(const_cast<stack<int> *>(&containerStack),
0);
    }

    CustomIterator end() const
    {
        return CustomIterator(const_cast<stack<int> *>(&containerStack),
containerStack.size());
    }
};

int main()
{
    CustomContainer container;

    container.add(1);
    container.add(2);
    container.add(3);
    container.add(2);
    container.add(4);

    container.remove(2);

    cout << "Container Elements: ";

    for (CustomContainer::CustomIterator it = container.begin(); it !=
container.end(); ++it)
```

```
    {
        cout << *it << " ";
    }

    cout << endl;
    return 0;
}
```

Output:

```
Container Elements: 4 3 2 1
```