| | |
|---|---|
| **Trạng thái** | Đã xong |
| **Bắt đầu vào lúc** | Thứ Ba, 25 tháng 2 2025, 1:56 PM |
| **Kết thúc lúc** | Chủ Nhật, 9 tháng 3 2025, 10:16 PM |
| **Thời gian thực hiện** | 12 Các ngày 8 giờ |
| **Điểm** | 4,95/5,00 |
| **Điểm** | **9,90** trên 10,00 (**99**%) |

Câu hỏi **1**

Đúng

Đạt điểm 1,00 trên 1,00

Implement methods **add, size** in template class **DLinkedList (which implements List ADT)** representing the doubly linked list with type T with the initialized frame. The description of each method is given in the code.

```
template <class T>
class DLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    DLinkedList();
    ~DLinkedList();
    void    add(const T &e);
    void    add(int index, const T &e);
    int     size();
public:
    class Node
    {
    private:
        T data;
        Node *next;
        Node *previous;
        friend class DLinkedList<T>;

    public:
        Node()
        {
            this->previous = NULL;
            this->next = NULL;
        }

        Node(const T &data)
        {
            this->data = data;
            this->previous = NULL;
            this->next = NULL;
        }
    };

};
```

In this exercise, we have include <iostream>, <string>, <sstream> and using namespace std.

**For example:**

| Test | Result |
|---|---|
| `DLinkedList<int> list;`<br>`int size = 10;`<br>`for(int idx=0; idx < size; idx++){`<br>`   list.add(idx);`<br>`}`<br>`cout << list.toString();` | `[0,1,2,3,4,5,6,7,8,9]` |
| `DLinkedList<int> list;`<br>`int size = 10;`<br>`for(int idx=0; idx < size; idx++){`<br>`   list.add(0, idx);`<br>`}`<br>`cout << list.toString();` | `[9,8,7,6,5,4,3,2,1,0]` |

**Answer:** (penalty regime: 0, 0, 0, 5, 10 %)

Reset answer

```
template <class T>
void DLinkedList<T>::add(const T &e) {
    Node* newNode = new Node(e);
    if (!head) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->previous = tail;
        tail = newNode;
    }
    count++;
}

template <class T>
void DLinkedList<T>::add(int index, const T &e) {
    if (index < 0 || index > count) {
        throw out_of_range("Index out of bounds");
    }

    Node* newNode = new Node(e);
    if (index == 0) {
        newNode->next = head;
        if (head) head->previous = newNode;
        head = newNode;
        if (!tail) tail = newNode;
    } else if (index == count) {
        add(e);
        return;
    } else {
        Node* current = head;
        for (int i = 0; i < index - 1; i++) {
            current = current->next;
        }
        newNode->next = current->next;
        newNode->previous = current;
        if (current->next) current->next->previous = newNode;
        current->next = newNode;
    }
    count++;
}

template <class T>
int DLinkedList<T>::size() {
    return count;
}

```

| | Test | Expected | Got | |
|---|---|---|---|---|
| ✓ | `DLinkedList<int> list;`<br>`int size = 10;`<br>`for(int idx=0; idx < size; idx++){`<br>`   list.add(idx);`<br>`}`<br>`cout << list.toString();` | `[0,1,2,3,4,5,6,7,8,9]` | `[0,1,2,3,4,5,6,7,8,9]` | ✓ |
| ✓ | `DLinkedList<int> list;`<br>`int size = 10;`<br>`for(int idx=0; idx < size; idx++){`<br>`   list.add(0, idx);`<br>`}`<br>`cout << list.toString();` | `[9,8,7,6,5,4,3,2,1,0]` | `[9,8,7,6,5,4,3,2,1,0]` | ✓ |

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 0,95 trên 1,00

Implement methods **get, set, empty, indexOf, contains** in template class D**LinkedList (which implements List ADT)** representing the singly linked list with type T with the initialized frame. The description of each method is given in the code.

```
template <class T>
class DLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    DLinkedList();
    ~DLinkedList();
    void    add(const T &e);
    void    add(int index, const T &e);
    int     size();
    bool    empty();
    T       get(int index);
    void    set(int index, const T &e);
    int     indexOf(const T &item);
    bool    contains(const T &item);
public:
    class Node
    {
    private:
        T data;
        Node *next;
        Node *previous;
        friend class DLinkedList<T>;

    public:
        Node()
        {
            this->previous = NULL;
            this->next = NULL;
        }

        Node(const T &data)
        {
            this->data = data;
            this->previous = NULL;
            this->next = NULL;
        }
    };

};
```

In this exercise, we have include <iostream>, <string>, <sstream> and using namespace std.

**For example:**

| Test | Result |
|---|---|
| `DLinkedList<int> list;`<br>`int size = 10;`<br>`for(int idx=0; idx < size; idx++){`<br>`  list.add(idx);`<br>`}`<br>`for(int idx=0; idx < size; idx++){`<br>`  cout << list.get(idx) << " \|";`<br>`}` | `0 \|1 \|2 \|3 \|4 \|5 \|6 \|7 \|8 \|9 \|` |

| Test | Result |
|---|---|
| `DLinkedList<int> list;`<br>`int size = 10;`<br>`int value[] = {2,5,6,3,67,332,43,1,0,9};`<br>`for(int idx=0; idx < size; idx++){`<br>`  list.add(idx);`<br>`}`<br>`for(int idx=0; idx < size; idx++){`<br>`  list.set(idx, value[idx]);`<br>`}`<br>`cout << list.toString();` | `[2,5,6,3,67,332,43,1,0,9]` |

**Answer:** (penalty regime: 0, 0, 0, 5, 10 %)

Reset answer

```cpp

template <class T>
bool DLinkedList<T>::empty() {
    return count == 0;
}

template <class T>
T DLinkedList<T>::get(int index) {
    if (index < 0 || index >= count) throw out_of_range("Index out of range");
    Node* current = head;
    for (int i = 0; i < index; i++) {
        current = current->next;
    }
    return current->data;
}

template <class T>
void DLinkedList<T>::set(int index, const T &e) {
    if (index < 0 || index >= count) throw out_of_range("Index out of range");
    Node* current = head;
    for (int i = 0; i < index; i++) {
        current = current->next;
    }
    current->data = e;
}

template <class T>
int DLinkedList<T>::indexOf(const T &item) {
    Node* current = head;
    int index = 0;
    while (current) {
        if (current->data == item) return index;
        current = current->next;
        index++;
    }
    return -1;
}

template <class T>
bool DLinkedList<T>::contains(const T &item) {
    return indexOf(item) != -1;
}
```

| | Test | Expected | Got | |
|---|---|---|---|---|
| ✓ | `DLinkedList<int> list;`<br>`int size = 10;`<br>`for(int idx=0; idx < size; idx++){`<br>`  list.add(idx);`<br>`}`<br>`for(int idx=0; idx < size; idx++){`<br>`  cout << list.get(idx) << " \|";`<br>`}` | `0 \|1 \|2 \|3 \|4 \|5 \|6 \|7 \|8 \|9 \|` | `0 \|1 \|2 \|3 \|4 \|5 \|6 \|7 \|8 \|9 \|` | ✓ |
| ✓ | `DLinkedList<int> list;`<br>`int size = 10;`<br>`int value[] = {2,5,6,3,67,332,43,1,0,9};`<br>`for(int idx=0; idx < size; idx++){`<br>`  list.add(idx);`<br>`}`<br>`for(int idx=0; idx < size; idx++){`<br>`  list.set(idx, value[idx]);`<br>`}`<br>`cout << list.toString();` | `[2,5,6,3,67,332,43,1,0,9]` | `[2,5,6,3,67,332,43,1,0,9]` | ✓ |

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00. Accounting for previous tries, this gives **0,95/1,00**.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 1,00

Implement methods **removeAt, removeItem, clear** in template class **SLinkedList (which implements List ADT)** representing the singly linked list with type T with the initialized frame. The description of each method is given in the code.

```
template <class T>
class DLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    DLinkedList();
    ~DLinkedList();
    void    add(const T &e);
    void    add(int index, const T &e);
    int     size();
    bool    empty();
    T       get(int index);
    void    set(int index, const T &e);
    int     indexOf(const T &item);
    bool    contains(const T &item);
    T       removeAt(int index);
    bool    removeItem(const T &item);
    void    clear();
public:
    class Node
    {
    private:
        T data;
        Node *next;
        Node *previous;
        friend class DLinkedList<T>;

    public:
        Node()
        {
            this->previous = NULL;
            this->next = NULL;
        }

        Node(const T &data)
        {
            this->data = data;
            this->previous = NULL;
            this->next = NULL;
        }
    };

};
```

In this exercise, we have include <iostream>, <string>, <sstream> and using namespace std.

**For example:**

| Test | Result |
|---|---|
| `DLinkedList<int> list;`<br>`int size = 10;`<br>`int value[] = {2,5,6,3,67,332,43,1,0,9};`<br><br>`for(int idx=0; idx < size; idx++){`<br>`  list.add(value[idx]);`<br>`}`<br>`list.removeAt(0);`<br>`cout << list.toString();` | `[5,6,3,67,332,43,1,0,9]` |

**Answer:** (penalty regime: 0 %)

Reset answer

```
template <class T>
T DLinkedList<T>::removeAt(int index) {
    if (index < 0 || index >= count) throw out_of_range("Index out of range");

    Node* current = head;
    T removedData;

    if (index == 0) { // Remove head
        removedData = head->data;
        head = head->next;
        if (head) head->previous = nullptr;
        else tail = nullptr; // List becomes empty
    } else {
        for (int i = 0; i < index; i++) {
            current = current->next;
        }
        removedData = current->data;
        current->previous->next = current->next;
        if (current->next) current->next->previous = current->previous;
        else tail = current->previous; // Update tail if removing last element
    }

    delete current;
    count--;
    return removedData;
}
template <class T>
bool DLinkedList<T>::removeItem(const T& item) {
    Node* current = head;
    while (current) {
        if (current->data == item) {
            if (current == head) {
                head = head->next;
                if (head) head->previous = nullptr;
                else tail = nullptr; // List becomes empty
            } else {
                current->previous->next = current->next;
                if (current->next) current->next->previous = current->previous;
                else tail = current->previous; // Update tail if removing last element
            }

            delete current;
            count--;
            return true;
        }
        current = current->next;
    }
    return false;
}
template<class T>
void DLinkedList<T>::clear() {
    Node* current = head;
```

| | Test | Expected | Got | |
|---|---|---|---|---|
| ✓ | `DLinkedList<int> list;`<br>`int size = 10;`<br>`int value[] = {2,5,6,3,67,332,43,1,0,9};`<br><br>`for(int idx=0; idx < size; idx++){`<br>`  list.add(value[idx]);`<br>`}`<br>`list.removeAt(0);`<br>`cout << list.toString();` | [5,6,3,67,332,43,1,0,9] | [5,6,3,67,332,43,1,0,9] | ✓ |

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 1,00

In this exercise, we will use Standard Template Library List (click open in other tab to show more) to implement a Data Log.

This is a simple implementation in applications using undo and redo. For example in Microsoft Word, you must have nodes to store states when Ctrl Z or Ctrl Shift Z to go back or forward.

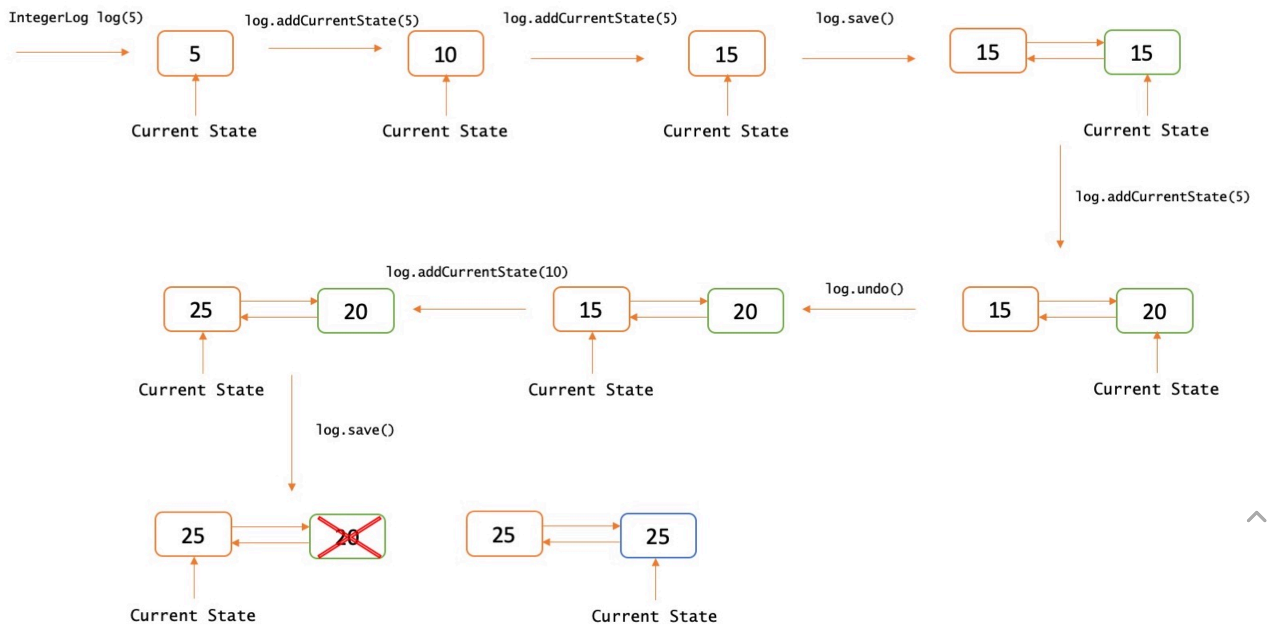
DataLog has a doubly linked list to store the states of data (an integer) and iterator to mark the current state. Each state is stored in a node, the transition of states is depicted in the figure below.

Your task in this exercise is implement functions marked with /*  * TODO   */.

```
class DataLog
{
private:
    list<int> logList;
    list<int>::iterator currentState;

public:
    DataLog();
    DataLog(const int &data);
    void addCurrentState(int number);
    void subtractCurrentState(int number);
    void save();
    void undo();
    void redo();

    int getCurrentStateData()
    {
        return *currentState;
    }

    void printLog()
    {
        for (auto i = logList.begin(); i != logList.end(); i++) {
            if(i == currentState) cout << "Current state: ";
            cout << "[ " << *i << " ] => ";
        }
        cout << "END_LOG";
    }
};
```

Note: Normally, when we say a List, we talk about doubly linked list. For implementing a singly linked list, we use forward list.

We have include <iostream> <list> and using namespace std;



**For example:**

| Test | Result |
|---|---|
| `DataLog log(10);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.undo();`<br>`log.printLog();` | `[ 10 ] => Current state: [ 25 ] => [ 40 ] => END_LOG` |
| `DataLog log(10);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.save();`<br>`log.subtractCurrentState(5);`<br>`log.printLog();` | `[ 10 ] => [ 25 ] => [ 40 ] => Current state: [ 35 ] => END_LOG` |

**Answer:** (penalty regime: 0, 0, 0, 5, 10 %)

Reset answer

```
DataLog::DataLog()
{
    logList.push_back(0);
    currentState = logList.begin();
}
DataLog::DataLog(const int &data)
{
    logList.push_back(data);
    currentState = logList.begin();
}
void DataLog::addCurrentState(int number)
{
    *currentState += number;
}
void DataLog::subtractCurrentState(int number)
{
    *currentState -= number;
}
void DataLog::save()
{
    auto it = currentState;
        it++;
        logList.erase(it, logList.end()); // Remove all states after current state
        logList.push_back(*currentState);
        currentState = prev(logList.end());
}
void DataLog::undo()
{
    if (currentState != logList.begin())
        {
            currentState--;
        }
}
void DataLog::redo()
{
     auto it = currentState;
        it++;
        if (it != logList.end())
        {
            currentState++;
        }
}
```

| | Test | Expected | Got | |
|---|---|---|---|---|
| ✓ | `DataLog log(10);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.undo();`<br>`log.printLog();` | `[ 10 ] => Current state: [ 25 ] => [ 40 ] => END_LOG` | `[ 10 ] => Current state: [ 25 ] => [ 40 ] => END_LOG` | ✓ |
| ✓ | `DataLog log(10);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.save();`<br>`log.addCurrentState(15);`<br>`log.save();`<br>`log.subtractCurrentState(5);`<br>`log.printLog();` | `[ 10 ] => [ 25 ] => [ 40 ] => Current state: [ 35 ] => END_LOG` | `[ 10 ] => [ 25 ] => [ 40 ] => Current state: [ 35 ] => END_LOG` | ✓ |

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **5**

Đúng

Đạt điểm 1,00 trên 1,00

Given the head of a doubly linked list, two positive integer a and b where a <= b. Reverse the nodes of the list from position a to position b and return the reversed list

Note: the position of the first node is 1. It is guaranteed that a and b are valid positions. You MUST NOT change the val attribute in each node.

```
struct ListNode {
    int val;
    ListNode *left;
    ListNode *right;
    ListNode(int x = 0, ListNode *l = nullptr, ListNode* r = nullptr) : val(x), left(l), right(r) {}
};
```

Constraint:
1 <= list.length <= 10^5
0 <= node.val <= 5000
1 <= left <= right <= list.length

Example 1:
Input: list = {3, 4, 5, 6, 7} , a = 2, b = 4
Output: 3 6 5 4 7

Example 2:
Input: list = {8, 9, 10}, a = 1, b = 3
Output: 10 9 8

**For example:**

| Test | Input | Result |
|---|---|---|
| <pre>int size;<br>    cin >> size;<br>    int* list = new int[size];<br>    for(int i = 0; i < size; i++) {<br>        cin >> list[i];<br>    }<br>    int a, b;<br>    cin >> a >> b;<br>    unordered_map<ListNode*, int> nodeValue;<br>    ListNode* head = init(list, size, nodeValue);<br>    ListNode* reversed = reverse(head, a, b);<br>    try {<br>        printList(reversed, nodeValue);<br>    }<br>    catch(char const* err) {<br>        cout << err << '\n';<br>    }<br>    freeMem(head);<br>    delete[] list;</pre> | 5<br>3 4 5 6 7<br>2 4 | 3 6 5 4 7 |

| Test | Input | Result |
|---|---|---|
| <pre>int size;<br>    cin >> size;<br>    int* list = new int[size];<br>    for(int i = 0; i < size; i++) {<br>        cin >> list[i];<br>    }<br>    int a, b;<br>    cin >> a >> b;<br>    unordered_map<ListNode*, int> nodeValue;<br>    ListNode* head = init(list, size, nodeValue);<br>    ListNode* reversed = reverse(head, a, b);<br>    try {<br>        printList(reversed, nodeValue);<br>    }<br>    catch(char const* err) {<br>        cout << err << '\n';<br>    }<br>    freeMem(head);<br>    delete[] list;</pre> | 3<br>8 9 10<br>1 3 | 10 9 8 |

**Answer:** (penalty regime: 0 %)

Reset answer

```cpp
/*
struct ListNode {
    int val;
    ListNode *left;
    ListNode *right;
    ListNode(int x = 0, ListNode *l = nullptr, ListNode* r = nullptr) : val(x), left(l), right(r) {}
};
*/
ListNode* reverse(ListNode* head, int a, int b) {
    if (!head || a == b) return head;

    ListNode* prev = nullptr;
    ListNode* current = head;

    for (int i = 1; current && i < a; i++) {
        prev = current;
        current = current->right;
    }

    ListNode* conn = prev;
    ListNode* tail = current;
    ListNode* next = nullptr;

    for (int i = a; current && i <= b; i++) {
        next = current->right;
        current->right = prev;
        current->left = next;
        prev = current;
        current = next;
    }

    if (conn) {
        conn->right = prev;
    } else {
        head = prev;
    }

    if (tail) {
        tail->right = current;
        if (current) current->left = tail;
    }

    return head;
}

```

| | Test | Input | Expected | Got | |
|---|---|---|---|---|---|
| ✓ | `int size;`<br>`    cin >> size;`<br>`    int* list = new int[size];`<br>`    for(int i = 0; i < size; i++) {`<br>`        cin >> list[i];`<br>`    }`<br>`    int a, b;`<br>`    cin >> a >> b;`<br>`    unordered_map<ListNode*, int> nodeValue;`<br>`    ListNode* head = init(list, size, nodeValue);`<br>`    ListNode* reversed = reverse(head, a, b);`<br>`    try {`<br>`        printList(reversed, nodeValue);`<br>`    }`<br>`    catch(char const* err) {`<br>`        cout << err << '\n';`<br>`    }`<br>`    freeMem(head);`<br>`    delete[] list;` | 5<br>3 4 5 6 7<br>2 4 | 3 6 5 4 7 | 3 6 5 4 7 | ✓ |
| ✓ | `int size;`<br>`    cin >> size;`<br>`    int* list = new int[size];`<br>`    for(int i = 0; i < size; i++) {`<br>`        cin >> list[i];`<br>`    }`<br>`    int a, b;`<br>`    cin >> a >> b;`<br>`    unordered_map<ListNode*, int> nodeValue;`<br>`    ListNode* head = init(list, size, nodeValue);`<br>`    ListNode* reversed = reverse(head, a, b);`<br>`    try {`<br>`        printList(reversed, nodeValue);`<br>`    }`<br>`    catch(char const* err) {`<br>`        cout << err << '\n';`<br>`    }`<br>`    freeMem(head);`<br>`    delete[] list;` | 3<br>8 9 10<br>1 3 | 10 9 8 | 10 9 8 | ✓ |

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.