

Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Ba, 20 tháng 2 2024, 6:01 AM
Kết thúc lúc	Chủ Nhật, 10 tháng 3 2024, 4:43 PM
Thời gian thực hiện	19 Các ngày 10 giờ

Câu hỏi 1

Đúng

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

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

Reset answer

```
1  template <class T>
2  void DLinkedList<T>::add(const T& e) {
3      /* Insert an element into the end of the list. */
4      Node* newNode = new Node(e);
5      if (head == NULL){
6          head = newNode;
7          tail = newNode;
8      } else {
9          tail->next = newNode;
10         newNode->previous = tail;
11         tail = newNode;
12     }
13     count++;
14 }
15
16 template<class T>
17 void DLinkedList<T>::add(int index, const T& e) {
18     /* Insert an element into the list at given index. */
19     if (index < 0 || index > count) throw out_of_range("Index out of range");
20     if (index == count){
21         add(e);
22         return;
23     }
24
25     Node* newNode = new Node(e);
26     if (index == 0){
27         newNode->next = head;
28         head->previous = newNode;
29         head = newNode;
30     } else {
31         Node* current = head;
32         for (int i = 0; i < index; i++) {
33             current = current->next;
34         }
35         newNode->next = current;
36         current->previous = newNode;
37     }
38     count++;
39 }
```

```
32     for (int i = 0, i < index - 1, i++) {
33         current = current->next;
34     }
35     newNode->next = current->next;
36     newNode->previous = current;
37     current->next->previous = newNode;
38     current->next = newNode;
39 }
40 count++;
41 }
42
43 template<class T>
44 int DLinkedList<T>::size() {
45     /* Return the length (size) of list */
46     return count;
47 }
```

	Test	Expected	Got	
✓	DLinkedList<int> list; int size = 10; for(int idx=0; idx < size; idx++){ list.add(idx); } 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; int size = 10; for(int idx=0; idx < size; idx++){ list.add(0, idx); } 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! ✓



Câu hỏi 2

Đúng

Đạt điểm 1,00

Implement methods **get**, **set**, **empty**, **indexOf**, **contains** in template class **DLinkedList** (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
<pre>DLinkedList<int> list; int size = 10; for(int idx=0; idx < size; idx++){ list.add(idx); } for(int idx=0; idx < size; idx++){ cout << list.get(idx) << " "; } </pre>	0 1 2 3 4 5 6 7 8 9
<pre>DLinkedList<int> list; int size = 10; int value[] = {2,5,6,3,67,332,43,1,0,9}; for(int idx=0; idx < size; idx++){ list.add(idx); } for(int idx=0; idx < size; idx++){ list.set(idx, value[idx]); } cout << list.toString(); </pre>	[2,5,6,3,67,332,43,1,0,9]


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

Reset answer

```
1  template<class T>
2  T DLinkedList<T>::get(int index) {
3      /* Give the data of the element at given index in the list. */
4      if (index < 0 || index >= count) throw out_of_range("Index out of range");
5      Node* current = head;
6      for (int i = 0; i < index; i++){
7          current = current->next;
8      }
9      return current->data;
10 }
11
12 template <class T>
13 void DLinkedList<T>::set(int index, const T& e) {
14     /* Assign new value for element at given index in the list */
15     if (index < 0 || index >= count) throw out_of_range("Index out of range");
16     Node* current = head;
17     for (int i = 0; i < index; i++){
18         current = current->next;
19     }
20     current->data = e;
21 }
22
23 template<class T>
```

```
24 bool DLinkedList<T>::empty() {
25     /* Check if the list is empty or not. */
26     return count == 0;
27 }
28
29 template<class T>
30 int DLinkedList<T>::indexOf(const T& item) {
31     /* Return the first index wheter item appears in list, otherwise return -1 */
32     int index = 0;
33     Node* current = head;
34     while (current != NULL){
35         if (current->data == item){
36             return index;
37         }
38         current = current->next;
39         index++;
40     }
41     return -1;
42 }
43
44 template<class T>
45 bool DLinkedList<T>::contains(const T& item) {
46     /* Check if item appears in the list */
47     return indexOf(item) != -1;
48 }
```

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

Câu hỏi 3

Đúng

Đạt điểm 1,00

Implement Iterator class in class DLinkedList.

Note: Iterator is a concept of repetitive elements on sequence structures. Iterator is implemented in class vector, list in STL container in C++ (<https://www.geeksforgeeks.org/iterators-c-stl/>). Your task is to implement the simple same class with iterator in C++ STL container.

```
template <class T>
class DLinkedList
{
public:
    class Iterator; //forward declaration
    class Node;     //forward declaration
protected:
    Node *head;
    Node *tail;
    int count;
public:
    DLinkedList() : head(NULL), tail(NULL), count(0){};
    ~DLinkedList();
    void add(const T &e);
    void add(int index, const T &e);
    T removeAt(int index);
    bool removeItem(const T &item);
    bool empty();
    int size();
    void clear();
    T get(int index);
    void set(int index, const T &e);
    int indexOf(const T &item);
    bool contains(const T &item);
    string toString();
    Iterator begin()
    {
        return Iterator(this, true);
    }
    Iterator end()
    {
        return Iterator(this, false);
    }
public:
    class Node
    {
    private:
        T data;
        Node *next;
        Node *previous;
        friend class DLinkedList<T>;

        Iterator begin()
        {
            return Iterator(this, true);
        }
        Iterator end()
        {
```

```
        return Iterator(this, false);
    }

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

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

class Iterator
{
private:
    DLinkedList<T> *pList;
    Node *current;
    int index; // is the index of current in pList
public:
    Iterator(DLinkedList<T> *pList, bool begin);
    Iterator &operator=(const Iterator &iterator);
    void set(const T &e);
    T &operator*();
    bool operator!=(const Iterator &iterator);
    void remove();

    // Prefix ++ overload
    Iterator &operator++();

    // Postfix ++ overload
    Iterator operator++(int);
};
};
```

Please read example carefully to see how we use the iterator.

For example:

Test	Result
<pre>DLinkedList<int> list; int size = 10; for(int idx=0; idx < size; idx++){ list.add(idx); } DLinkedList<int>::Iterator it = list.begin(); for(; it != list.end(); it++) { cout << *it << " "; }</pre>	0 1 2 3 4 5 6 7 8 9
<pre>DLinkedList<int> list; int size = 10; for (int idx = 0; idx < size; idx++) { list.add(idx); } DLinkedList<int>::Iterator it = list.begin(); while (it != list.end()) { it.remove(); it++; } cout << list.toString();</pre>	[]
<pre>DLinkedList<int> list; int size = 10; for (int idx = 0; idx < size; idx++) { list.add(idx); } DLinkedList<int>::Iterator it = list.begin(); for(; it != list.end(); it++) { it.remove(); } cout << list.toString();</pre>	[]

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

Reset answer

1 ▼

2

3

4

/*
 * TODO: Implement class Iterator's method
 * Note: method remove is different from SLinkedList, which is the advantage of DLink

```

4 //
5 template <class T>
6 DLinkedList<T>::Iterator::Iterator(DLinkedList<T> *pList, bool begin)
7 {
8     this->pList = pList;
9     if(pList ==NULL){index =-1;}
10    else if(begin){
11        if(this->pList->size() == 0){this->current =NULL;index =-1;}
12        else{ this->current = pList->head;index =0;}
13    }
14    else{
15        this->current =NULL;
16        if(this->pList->size() == 0) index = this->pList->size();
17        else index =pList->size();
18    }
19 }
20
21 template <class T>
22 typename DLinkedList<T>::Iterator& DLinkedList<T>::Iterator::operator=(const DLinkedList
23 {
24     this->pList = iterator.pList;
25     this->current = iterator.current;
26     this->index = iterator.index;
27     return *this;
28 }
29
30 template <class T>
31 void DLinkedList<T>::Iterator::set(const T &e)
32 {
33     if (current==NULL) throw out_of_range("Segmentation fault!");
34     current->data = e;
35 }
36
37 template<class T>
38 T& DLinkedList<T>::Iterator::operator*()
39 {
40     if(current ==NULL) throw out_of_range("Segmentation fault!");
41     return current->data;
42 }
43
44 template<class T>
45 void DLinkedList<T>::Iterator::remove()
46 {
47     /*
48     * TODO: delete Node in pList which Node* current point to.
49     * After that, Node* current point to the node before the node just deleted.
50     * If we remove first node of pList, Node* current point to nullptr.
51     * Then we use operator ++, Node* current will point to the head of pList.
52     */
53     if(this->current== NULL) throw out_of_range("Segmentation fault!");
54     int index =this->pList->indexOf(this->current->data);
55     if(index ==0)
56     {
57         this->pList->removeAt(index);
58         this->index =-1;

```

```
59     current = NULL;
60 }
61 else
62 {
63     T e = this->pList->removeAt(index-1);
64     this->index =index- 1;
65     current->data =e;
66 }
67 }
68
69 template<class T>
70 bool DLinkedList<T>::Iterator::operator!=(const DLinkedList::Iterator &iterator)
71 {
72     return !(iterator.index == this->index ||iterator.current == this->current);
73 }
74
75
```

	Test	Expected	Got	
✓	<pre>DLinkedList<int> list; int size = 10; for(int idx=0; idx < size; idx++){ list.add(idx); } DLinkedList<int>::Iterator it = list.begin(); for(; it != list.end(); it++) { cout << *it << " "; }</pre>	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	✓
✓	<pre>DLinkedList<int> list; int size = 10; for (int idx = 0; idx < size; idx++) { list.add(idx); } DLinkedList<int>::Iterator it = list.begin(); while (it != list.end()) { it.remove(); it++; } cout << list.toString();</pre>	[]	[]	✓

	Test	Expected	Got	
✓	<pre>DLinkedList<int> list; int size = 10; for (int idx = 0; idx < size; idx++) { list.add(idx); } DLinkedList<int>::Iterator it = list.begin(); for(; it != list.end(); it++) { it.remove(); } cout << list.toString();</pre>	[]	[]	✓

Passed all tests! ✓

Câu hỏi 4

Đúng

Đạt điểm 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
<pre>DLinkedList<int> list; int size = 10; int value[] = {2,5,6,3,67,332,43,1,0,9}; for(int idx=0; idx < size; idx++){ list.add(value[idx]); } list.removeAt(0); cout << list.toString();</pre>	[5,6,3,67,332,43,1,0,9]

Answer: (penalty regime: 0 %)

Reset answer

```
1  template <class T>
2  T DLinkedList<T>::removeAt(int index)
3  {
4      /* Remove element at index and return removed value */
5      T result;
6      if (index < 0 && index >= count) throw out_of_range("");
7      if (count == 1) {
8          result = head->data;
9          delete head;
10         head = tail = NULL;
11     } else if (index == 0){
12         Node* tmp = head;
13         head = head->next;
14         result = tmp->data;
15         delete tmp;
16         head->previous = NULL;
17     } else {
18         Node* tmp = head;
19         index--;
20         while(index){
21             tmp = tmp->next;
22             index--;
23         }
24         result = tmp->next->data;
25         if (tmp->next == tail){
26             delete tail;
27             tail = tmp;
28             tail->next = NULL;
29         } else {
30             Node* h = tmp->next;
31             tmp->next = tmp->next->next;
32             tmp->next->previous = tmp;
33             delete h;
34         }
35     }
36     count--;
```

```
36     count--,
37     return result;
38 }
39
40 template <class T>
41 bool DLinkedList<T>::removeItem(const T& item)
42 {
43     /* Remove the first apperance of item in list and return true, otherwise return false */
44     int index = indexOf(item);
45     if (index == -1) return false;
46     removeAt(index);
47     return true;
48 }
49
50 template<class T>
51 void DLinkedList<T>::clear(){}
52
```

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

Passed all tests! ✓



Câu hỏi 5

Đúng

Đạt điểm 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;



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

Reset answer

```
1 DataLog::DataLog()
2 {
3     /*
4      * TODO: add the first state with 0
5      */
6     logList.push_front(0);
7     currentState = logList.begin();
8 }
9
10 DataLog::DataLog(const int &data)
11 {
12     /*
13      * TODO: add the first state with data
14      */
15     logList.push_front(data);
16     currentState = logList.begin();
17 }
18
19 void DataLog::addCurrentState(int number)
20 {
21     /*
22      * TODO: Increase the value of current state by number
23      */
24     *currentState += number;
25 }
26
27 void DataLog::subtractCurrentState(int number)
28 {
29     /*
30      * TODO: Decrease the value of current state by number
31      */
32     *currentState -= number;
33 }
34
35 void DataLog::save()
36 {
37     /*
38      * TODO: This function will create a new state, copy the data of the currentState
39      *       and move the currentState Iterator to this new state. If there are other
40      *       currentState Iterator, we delete them all before creating a new state.
41      */
42     currentState++;
43     if (currentState != logList.end()){
44         list<int>::iterator curr = currentState;
45         list<int>::iterator prev = currentState;
46         while(curr != logList.end()){
47             curr++;
48             logList.erase(prev);
49             prev = curr;
50         }
51         currentState = logList.end();
52 }
```

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

Passed all tests! ✓



Câu hỏi 6

Đúng

Đạt điểm 1,00

Given the head of a doubly linked list, two positive integer a and b where $a \leq b$. 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:

$1 \leq \text{list.length} \leq 10^5$

$0 \leq \text{node.val} \leq 5000$

$1 \leq \text{left} \leq \text{right} \leq \text{list.length}$

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

Answer: (penalty regime: 0 %)

Reset answer

1	▼	ListNode* reverse(ListNode* head, int a, int b) {
2		unordered_map<int, ListNode*> v;
3		ListNode* tmp = head;

```
4      int index = 1;
5      int i = b;
6      while (tmp) {
7          if (index >= a && index <= b) {
8              v[i] = tmp;
9              i--;
10         }
11         else
12             v[index] = tmp;
13         tmp = tmp->right;
14         index++;
15     }
16     head = v[1];
17     head->left = nullptr;
18     tmp = head;
19     for (int i = 2; i < index; i++) {
20         tmp->right = v[i];
21         v[i]->left = tmp;
22         tmp = v[i];
23     }
24     tmp->right = nullptr;
25     return head;
26 }
27
28
29
```



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

Passed all tests! ✓

