

Đã bắt đầu vào lúc	Thứ năm, 12 Tháng mười 2023, 1:32 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ sáu, 20 Tháng mười 2023, 1:29 PM
Thời gian thực hiện	7 ngày 23 giờ
Điểm	6,00/6,00
Điểm	10,00 của 10,00 (100%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 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]

Test	Result
<pre>DLinkedList<int> list; int size = 10; for(int idx=0; idx < size; idx++){ list.add(0, idx); } cout << list.toString();</pre>	[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      if(this->count == 0) {
5          this->head = this->tail = new Node(e);
6          this->count++;
7          return;
8      }
9      Node* pNew = new Node(e);
10     tail->next = pNew;
11     pNew->previous = tail;
12     tail = tail->next;
13     this->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 > this->count) throw std::out_of_range("Naruto");
20     if(index == this->count) {
21         add(e);
22         return;
23     }
24     else if(index == 0) {
25         Node* pNew = new Node(e);
26         pNew->next = head;
27         head->previous = pNew;
28         head = head->previous;
29         this->count++;
30         return;
31     }
32     else {
33         Node* pNew = new Node(e);
34         Node* temp = head;
35         for(int i = 0; i < index-1; i++) temp = temp->next;
36         pNew->previous = temp;
37         pNew->next = temp->next;
38         temp->next = pNew;
39         pNew->next->previous = pNew;
40         this->count++;
41     }
42 }
43
44 template<class T>
45 int DLinkedList<T>::size() {
46     /* Return the length (size) of list */
47     return this->count;
48 }
```

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

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 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>	<pre>0 1 2 3 4 5 6 7 8 9 </pre>
<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>	<pre>[2,5,6,3,67,332,43,1,0,9]</pre>

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 std::out_of_range("Naruto");
5      Node* temp = this->head;
6      for(int i = 0; i < index; i++) temp = temp->next;
7      return temp->data;
8  }
9
10 template <class T>
11 ▼ void DLinkedList<T>::set(int index, const T& e) {
12     /* Assign new value for element at given index in the list */
13     if(index < 0 || index >= count) throw std::out_of_range("Naruto");
14     Node* temp = this->head;
15     for(int i = 0; i < index; i++) temp = temp->next;
16     temp->data = e;
17 }
18
19 template<class T>
20 ▼ bool DLinkedList<T>::empty() {
21     /* Check if the list is empty or not. */
22     if(this->count == 0) return true;
23     else return false;
24 }
25
26 template<class T>
27 ▼ int DLinkedList<T>::indexOf(const T& item) {
28     /* Return the first index wheter item appears in list, otherwise return -1 */
29     Node* temp = head;
30     int i = 0;
31 ▼ while(temp != nullptr) {
32     if(temp->data == item) return i;
33     temp = temp->next;
34     i++;
35 }
36 return -1;
37 }
38
39 template<class T>
40 ▼ bool DLinkedList<T>::contains(const T& item) {
41     /* Check if item appears in the list */
42     if(this->indexOf(item) != -1) return true;
43     return false;
44 }
```

	Test	Expected	Got	
✓	<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	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]	[2,5,6,3,67,332,43,1,0,9]	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 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   * TODO: Implement class Iterator's method
3   * Note: method remove is different from SLinkedList, which is the advantage of DLinkedList
4   */
5  template <class T>
6  DLinkedList<T>::Iterator::Iterator(DLinkedList<T> *pList, bool begin)
7  {
8      this->pList = pList;
9      if(begin) {
10         if(this->pList != nullptr) {
11             this->current = this->pList->head;
12             this->index = 0;
13         }
14         else {
15             this->current = nullptr;
16             this->index = -1;
17         }
18     }
19     else {
20         this->current = nullptr;
21         if(this->pList != nullptr) this->index = this->pList->size();
22         else this->index = 0;
23     }
24 }
25
26 template <class T>
27 typename DLinkedList<T>::Iterator& DLinkedList<T>::Iterator::operator=(const DLinkedList<T>::Iterator &itera
28 {
29     this->pList = iterator.pList;
30     this->current = iterator.current;
31     this->index = iterator.index;
32     return *this;
33 }
34
35 template <class T>
36 void DLinkedList<T>::Iterator::set(const T &e)
37 {
38     if(this->current == nullptr) throw std::out_of_range("Segmentation fault!");
39     else this->current->data = e;
40 }
41
42 template<class T>
43 T& DLinkedList<T>::Iterator::operator*()
44 {
45     if(this->current == nullptr) throw std::out_of_range("Segmentation fault!");
46     return this->current->data;
47 }
48 template<class T>
49 void DLinkedList<T>::Iterator::remove()
50 {
51     /*
52     * TODO: delete Node in pList which Node* current point to.
53     *       After that, Node* current point to the node before the node just deleted.
54     *       If we remove first node of pList, Node* current point to nullptr.
55     *       Then we use operator ++, Node* current will point to the head of pList.
56     */
57     if(this->current == nullptr) throw std::out_of_range("Segmentation fault!");
58     else if(this->current == this->pList->head) {
59         this->current = nullptr;
60         this->pList->removeAt(0);
61         this->index = -1;
62     }
63     else {
64         this->current = this->current->previous;
65         this->pList->removeAt(this->index);
66         --this->index;
67     }
68 }
69

```

```

70 | template<class T>
71 | bool DLinkedList<T>::Iterator::operator!=(const DLinkedList::Iterator &iterator)
72 | {
73 |     if(this->pList != iterator.pList || this->current != iterator.current) return true;
74 |     else return false;
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>	[]	[]	✓
✓	<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! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 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      if(index < 0 || index >= count) throw std::out_of_range("Naruto");
6      if(this->count == 1) {
7          int save = head->data;
8          delete head;
9          head = nullptr;
10         tail = nullptr;
11         this->count = 0;
12         return save;
13     }
14     if(index == 0) {
15         Node* temp = head;
16         head = head->next;
17         head->previous = nullptr;
18         temp->next = nullptr;
19         int save = temp->data;
20         delete temp;
21         temp = nullptr;
22         this->count--;
23         return save;
24     }
25     else if(index == this->count - 1) {
26         Node* temp = tail;
27         tail = tail->previous;
28         tail->next = nullptr;
29         temp->previous = nullptr;
30         int save = temp->data;
31         delete temp;
32         temp = nullptr;
33         this->count--;
34         return save;
35     }
36     else {
37         Node* temp = head;
38         for(int i = 0; i < index; i++) {
39             temp = temp->next;
40         }
41         temp->previous->next = temp->next;
42         temp->next->previous = temp->previous;
43         temp->next = nullptr;
44         temp->previous = nullptr;
45         int save = temp->data;
46         delete temp;
47         temp = nullptr;
48         this->count--;
49         return save;
50     }

```

	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! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 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 %)

```

1 //Use_Push_Back()
2 DataLog::DataLog()
3 {
4     /*
5      * TODO: add the first state with 0
6      */
7     logList.push_back(0);
8     currentState = logList.begin();
9 }
10
11 DataLog::DataLog(const int &data)
12 {
13     /*
14      * TODO: add the first state with data
15      */
16     logList.push_back(data);

```

```

17     currentState = logList.begin();
18 }
19
20 void DataLog::addCurrentState(int number)
21 {
22     /*
23      * TODO: Increase the value of current state by number
24      */
25     *(this->currentState) = *(this->currentState) + number;
26 }
27
28 void DataLog::subtractCurrentState(int number)
29 {
30     /*
31      * TODO: Decrease the value of current state by number
32      */
33     *(this->currentState) = *(this->currentState) - number;
34 }
35
36 void DataLog::save()
37 {
38     /*
39      * TODO: This function will create a new state, copy the data of the currentState
40      *       and move the currentState Iterator to this new state. If there are other states behind the
41      *       currentState Iterator, we delete them all before creating a new state.
42      */
43     list<int>::iterator now = currentState;
44     now++;
45     while(now != logList.end()) now = logList.erase(now);
46     logList.push_back(*currentState);
47     currentState++;
48 }
49
50 void DataLog::undo()

```

	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! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 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>

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> 3 8 9 10 1 3 </pre>	<pre> 10 9 8 </pre>

Answer: (penalty regime: 0 %)

Reset answer

```

8  */
9 void swap(ListNode* a, ListNode* b) {
10     ListNode* temp = b->left;
11     if(a->left != nullptr) {
12         a->left->right = b;
13         b->left = a->left;
14         a->left = temp;
15         temp->right = a;
16     }
17     else {
18         b->left = nullptr;
19         a->left = temp;
20         temp->right = a;
21     }
22     temp = a->right;
23     if(b->right != nullptr) {
24         b->right->left = a;
25         a->right = b->right;
26         b->right = temp;
27         temp->left = b;
28     }
29     else {
30         a->right = nullptr;
31         b->right = temp;
32         temp->left = b;
33     }
34 }
35 ListNode* reverse(ListNode* head, int a, int b) {
36     //To Do
37     if(head == nullptr || (head->left == nullptr && head->right == nullptr))
38         return head;
39     if(a == b) return head;
40     ListNode* p1 = head;
41     ListNode* p2 = head;
42     bool flag = false;
43     for(int i = 0; i < a-1; i++) p1 = p1->right;
44     for(int i = 0; i < b-1; i++) p2 = p2->right;
45     while(p1 != p2 && p1->left != p2) {
46         swap(p1, p2);

```

	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>	3 6 5 4 7	3 6 5 4 7	✓
✓	<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>	10 9 8	10 9 8	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

BÁCH KHOA E-LEARNING



WEBSITE

HCMUT

MyBK

BKSI

LIÊN HỆ

📍 268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

☎ (028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

✉ elearning@hcmut.edu.vn

Copyright 2007-2022 BKEL - Phát triển dựa trên Moodle