
Đã bắt đầu vào Thứ bảy, 24 Tháng chín 2022, 3:01 PM
lúc

Tình trạng Đã hoàn thành

Hoàn thành vào Thứ tư, 12 Tháng mười 2022, 2:27 PM
lúc

Thời gian thực 17 ngày 23 giờ
hiện

Điểm 7,00 của 8,00 (87,5%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

A double-ended queue or deque (pronounced "deck") is like a queue or a stack but supports adding and removing items at both ends. A deque stores a collection of items and supports the following methods:

```

+ getSize(): int           => number of items in the deque
+ pushFront(int item): void => add an item to the left end
+ pushBack(int item): void  => add an item to the right end
+ popFront(): int           => remove and return an item from the left end
+ popBack(): int            => remove and return an item from the right end
+ clear(): void             => erase all items in the deque

// For checking purposes
+ printQueue(): void         => print all items in the deque from left to right, separated by a space, with a new line (i.e.
'\n') at the end.
+ printQueueReverse(): void   => print all items in the deque from right to left, separated by a space, with a new line at the
end.

Note: if the deque is empty, every pop method return -1;

class Deque {
private:
    class Node {
private:
        int value;
        Node* left;
        Node* right;
        friend class Deque;
public:
        Node(int val = 0, Node* l = nullptr, Node* r = nullptr) : value(val), left(l), right(r) { }
    };
private:
    Node* head;
    Node* tail;
    int curSize;
public:
    Deque();
    ~Dequeue();
    int getSize();
    void pushFront(int item);
    void pushBack(int item);
    int popFront();
    int popBack();
    void clear();
    void printQueueReverse();
    void printQueue();
};

```

For example:

Test	Result
------	--------

Test	Result
<pre>Deque* deque = new Deque(); vector<int> arr = {223, 1234, 43, 568, 90, 193, 2109}; for(int i = 0; i < (int)arr.size(); i++) { if (i < (int)arr.size() / 2) deque->pushFront(arr[i]); else { deque->pushBack(arr[i]); } } deque->printDeque(); cout << deque->getSize(); delete deque;</pre>	43 1234 223 568 90 193 2109 7
<pre>Deque* deque = new Deque(); int size = 20; for(int i = 0; i < size; i++) { deque->pushBack(i); } for(int i = 0; i < size / 2; i++) { if (i % 2 == 0) deque->popBack(); else deque->popFront(); } deque->printDequeReverse(); cout << deque->getSize(); delete deque;</pre>	14 13 12 11 10 9 8 7 6 5 10

Answer: (penalty regime: 0 %)

[Reset answer](#)

```

1 ✓ Deque::Deque() {
2     head = nullptr;
3     tail = nullptr;
4     curSize = 0;
5 }
6
7 ✓ Deque::~Deque() {
8 }
9
10 ✓ void Deque::clear() {
11     while(head != nullptr){
12         popBack();
13     }
14 }
15
16 ✓ int Deque::getSize() {
17     return curSize;
18 }
19
20 ✓ void Deque::pushFront(int i) {
21     if (head == nullptr){
22         Node* newNode = new Node(i,nullptr,nullptr);
23         head = newNode;
24         tail = newNode;
25     } else {
26         Node* newNode = new Node(i,nullptr,head);
27         head->left = newNode;
28         head = newNode;
29     }
30     curSize += 1;
31 }
32
33 ✓ void Deque::pushBack(int i) {
34     if (head == nullptr){
35         Node* newNode = new Node(i,nullptr,nullptr);
36         head = newNode;

```

```
37     tail = newNode;
38 } else {
39     Node* newNode = new Node(i,tail,nullptr);
40     tail->right = newNode;
41     tail = newNode;
42 }
43 curSize += 1;
44 }
45
46 int Deque::popFront() {
47     if (head == nullptr)
48         return -1;
49
50     int ans = head->value;
51     Node* newHead = head->right;
52     if (newHead != nullptr){
53         newHead->left = nullptr;
54         delete head;
55         head = newHead;
56     } else {
57         delete head;
58         head = nullptr;
59         tail = nullptr;
60     }
61     curSize -= 1;
62     return ans;
63 }
64
65 int Deque::popBack() {
66     if (head == nullptr)
67         return -1;
68
69     int ans = tail->value;
70     Node* newTail = tail->left;
71     if (newTail != nullptr){
72         newTail->right = nullptr;
73         delete tail;
74         tail = newTail;
75     } else {
76         delete tail;
77         head = nullptr;
78         tail = nullptr;
79     }
80     curSize -= 1;
81     return ans;
82 }
83
84 void Deque::printDequeReverse() {
85     Node*temp=tail;
86     while(temp!=nullptr){
87         cout<<temp->value<<" ";
88         temp=temp->left;
89     }
90     cout<<endl;
91 }
92
93 void Deque::printDeque() {
94     Node*temp=head;
95     while(temp!=nullptr){
96         cout<<temp->value<<" ";
97         temp=temp->right;
98     }
99     cout<<endl;
100 }
```

	Test	Expected	Got	
✓	<pre>Deque* deque = new Deque(); vector<int> arr = {223, 1234, 43, 568, 90, 193, 2109}; for(int i = 0; i < (int)arr.size(); i++) { if (i < (int)arr.size() / 2) deque->pushFront(arr[i]); else { deque->pushBack(arr[i]); } } deque->printDeque(); cout << deque->getSize(); delete deque;</pre>	43 1234 223 568 90 193 2109 7	43 1234 223 568 90 193 2109 7	✓
✓	<pre>Deque* deque = new Deque(); int size = 20; for(int i = 0; i < size; i++) { deque->pushBack(i); } for(int i = 0; i < size / 2; i++) { if (i % 2 == 0) deque->popBack(); else deque->popFront(); } deque->printDequeReverse(); cout << deque->getSize(); delete deque;</pre>	14 13 12 11 10 9 8 7 6 5 10	14 13 12 11 10 9 8 7 6 5 10	✓

Passed all tests! ✓

Chính xác

Diểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

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
<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]
<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     if (count == 0)
4     {
5         Node* newNode = new Node(e);
6         head = newNode;
7         tail = newNode;
8         tail->next = NULL;
9         ++(this->count);
10        return;
11    }
12    Node* newNode = new Node(e);
13    tail->next = newNode;
14    newNode->previous = tail;
15    newNode->next = NULL;
16    tail = newNode;
17    ++(this->count);
18    return;
19 }
20
21 template<class T>
22 void DLinkedList<T>::add(int index, const T& e) {
23     /* Insert an element into the list at given index. */
24     if (count == 0) {add(e); return;}
25     if (index == 0)
26     {
27         Node* newNode = new Node(e);
28         newNode->next = head;
29         head->previous = newNode;
30         head = newNode;
31         ++(this->count);
32         return;
33     }
34     if (index == this->count) {add(e); return;}
35     int idx = 0;
36     Node* front = head;
37     Node* back = NULL;
38     for (;front != NULL; back = front, front = front->next, ++idx)
39     {
40         if (idx == index)
41         {
42             Node* newNode = new Node (e);
43             ++(this->count);
44             back->next = newNode;
45             newNode->next = front;
46             front->previous = newNode;
47             return;
48         }
49     }
50 }
51 }
52
53 template<class T>
54 int DLinkedList<T>::size() {
55     /* Return the length (size) of list */
56     int cnt=0;
57     Node*temp=head;
58     while(temp!=nullptr){
59         temp=temp->next;
60         cnt++;
61     }
62     return cnt;
63 }
```



	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! ✓

Chỉnh xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

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
------	--------

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     if (count == 0) return -1;
4     if (index == this->count - 1) return tail->data;
5     if (index == 0) return head->data;
6     int idx = 0;
7     for (Node* h = head; h != NULL; h = h->next, ++idx)
8     {
9         if (idx == index) return h->data;
10    }
11    return -1;
12 /* Give the data of the element at given index in the list. */
13 }
14 template <class T>
15 void DLinkedList<T>::set(int index, const T& e) {
16     if (count == 0) return;
17     if (index == 0)
18     {
19         head->data = e;
20         return;
21     }
22     if (index == this->count - 1)
23     {
24         tail->data = e;
25         return;
26     }
27     int idx = 0;
28     for (Node* h = head; h != NULL; h = h->next, ++idx)
29     {
30         if (idx == index)
31         {
32             h->data = e;
33             return;
34         }
35     }
36 /* Assign new value for element at given index in the list */
37 }
38
39 template<class T>
40 bool DLinkedList<T>::empty() {
41     /* Check if the list is empty or not. */
42     if (count == 0) return true;
43     return false;
44 }
```

```

46
47 template<class T>
48 int DLinkedList<T>::indexOf(const T& item) {
49     /* Return the first index wheter item appears in list, otherwise return -1 */
50     int idx=0;
51     if(count==0) return -1;
52     for(Node*temp_head=head;temp_head!=nullptr;temp_head=temp_head->next, idx++){
53         if(item==temp_head->data){
54             return idx;
55         }
56     }
57     return -1;
58 }
59
60
61 template<class T>
62 bool DLinkedList<T>::contains(const T& item) {
63     /* Check if item appears in the list */
64     Node*temp=head;
65     while(temp!=nullptr){
66         if(temp->data==item){
67             return 1;
68         }
69         temp=temp->next;
70     }
71     return 0;
72 }
73 }
```

	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! ✓

 Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

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;
    friend class DLinkedList<T>;
public:
    Node()
    {
        next = 0;
    }
    Node(Node *next)
    {
        this->next = next;
    }
    Node(T data, Node *next = NULL)
    {
        this->data = data;
        this->next = next;
    }
};
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(DLinkedList<T> *pList, bool begin)
7 {

```

```

8     this->pList = pList;
9     if (begin == true) {
10        if (pList != NULL) {
11            this->current = pList->head;
12            this->index = 0;
13        }
14    } else {
15        this->current = NULL;
16        this->index = -1;
17    }
18 }
19 else {
20     this->current = NULL;
21     if (pList != NULL) this->index = pList->count;
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 &iterator)
28 {
29     this->current = iterator.current;
30     this->index = iterator.index;
31     this->pList = iterator.pList;
32     return *this;
33 }
34
35 template <class T>
36 void DLINKEDLIST<T>::Iterator::set(const T &e)
37 {
38     if (current == NULL) throw std::out_of_range("Segmentation fault!");
39     current->data = e;
40 }
41
42 template<class T>
43 T& DLINKEDLIST<T>::Iterator::operator*()
44 {
45     if (current == NULL) throw std::out_of_range("Segmentation fault!");
46     return current->data;
47 }
48
49 template<class T>
50 void DLINKEDLIST<T>::Iterator::remove()
51 {
52     /*
53     * TODO: delete Node in pList which Node* current point to.
54     * After that, Node* current point to the node before the node just deleted.
55     * If we remove first node of pList, Node* current point to nullptr.
56     * Then we use operator ++, Node* current will point to the head of pList.
57     */
58     if (current == NULL) throw std::out_of_range("Segmentation fault!");
59     if (index == 0) {
60         pList->head = current->next;
61         delete current;
62         pList->count--;
63         if (pList->count == 0) pList->tail = pList->head;
64         current = nullptr;
65         index = -1;
66     }
67     else {
68         Node* prev = pList->head;
69         while (prev->next != current) prev = prev->next;
70         prev->next = current->next;
71         delete current;
72         pList->count--;
73         if (index == pList->count) pList->tail = prev;
74         current = prev;
75         index--;
76     }
77 }
78
79 template<class T>
80 bool DLINKEDLIST<T>::Iterator::operator!=(const DLINKEDLIST::Iterator &iterator)

```

```

81 {
82     if (this->current == iterator.current && this->index == iterator.index) return false;
83     else return true;
84 }
85
86 template<class T>
87 typename DLinkedList<T>::Iterator& DLinkedList<T>::Iterator::operator++()
88 {
89     //if (current == NULL) throw std::out_of_range("Segmentation fault!");
90     if (index < 0) {
91         current = pList->head;
92         index = 0;
93     }
94     else {
95         current = current->next;
96         index++;
97     }
98     return *this;
99 }
100
101 template<class T>
102 typename DLinkedList<T>::Iterator DLinkedList<T>::Iterator::operator++(int)
103 {
104     Iterator ret = *this;
105     //if (current == NULL) throw std::out_of_range("Segmentation fault!");
106     if (index < 0) {
107         current = pList->head;
108         index = 0;
109     }
110     else {
111         current = current->next;
112         index++;
113     }
114     return ret;
115 }

```

	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	✓

	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(); 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 5

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
------	--------

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     Node* temp = head;
5     /* Remove element at index and return removed value */
6     if(temp==nullptr){return 0;}
7     T ret;
8     if(index==0){
9         Node* hold=head->next;
10    if(head==tail){
11        tail=nullptr;
12    }
13    ret=head->data;
14    delete head;
15    head=hold;
16    if(head!=nullptr)
17        head->previous=nullptr;
18    //return ret;
19    }
20    else if(index==count-1){
21        Node* hold=tail->previous;
22        ret = tail->data;
23        delete tail;
24        tail=hold;
25        tail->next=nullptr;
26        //return ret;
27    }
28    else{
29        Node* dummy_head=head;
30        for(int i=0;i<index;i++){
31            dummy_head=dummy_head->next;
32        }
33        Node*temp_prev = dummy_head->previous;
34        Node*temp_next = dummy_head->next;
35        ret= dummy_head->data;
36        delete dummy_head;
37        temp_prev->next=temp_next;
38        temp_next->previous=temp_prev;
39    }
40    //T a = temp->data;
41    //delete temp;
42    count--;
43    return ret;
44 }
45
46 template <class T>
47 bool DLinkedList<T>::removeItem(const T& item)
48 {
49     /* Remove the first apperance of item in list and return true, otherwise return false */
50     Node* temp = head;
51     for (int i = 0; temp != nullptr; i++) {
52         if (temp->data == item) {
53             removeAt(i);
54             return true;
55         }
56     }
57 }
```

```

56     temp = temp->next;
57 }
58 return false;
59 }
60 }
61
62 template<class T>
63 void DLinkedList<T>::clear() {
64     /* Remove all elements in list */
65     Node* temp = head;
66     while (temp != nullptr) {
67         //cout << temp->data << endl;
68         Node* next = temp->next;
69         delete temp;
70         temp = next;
71     }
72     head = nullptr;
73     tail = nullptr;
74     count = 0;
75 }
```

	Test	Expected	Got	
✓	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();	[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 6

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;

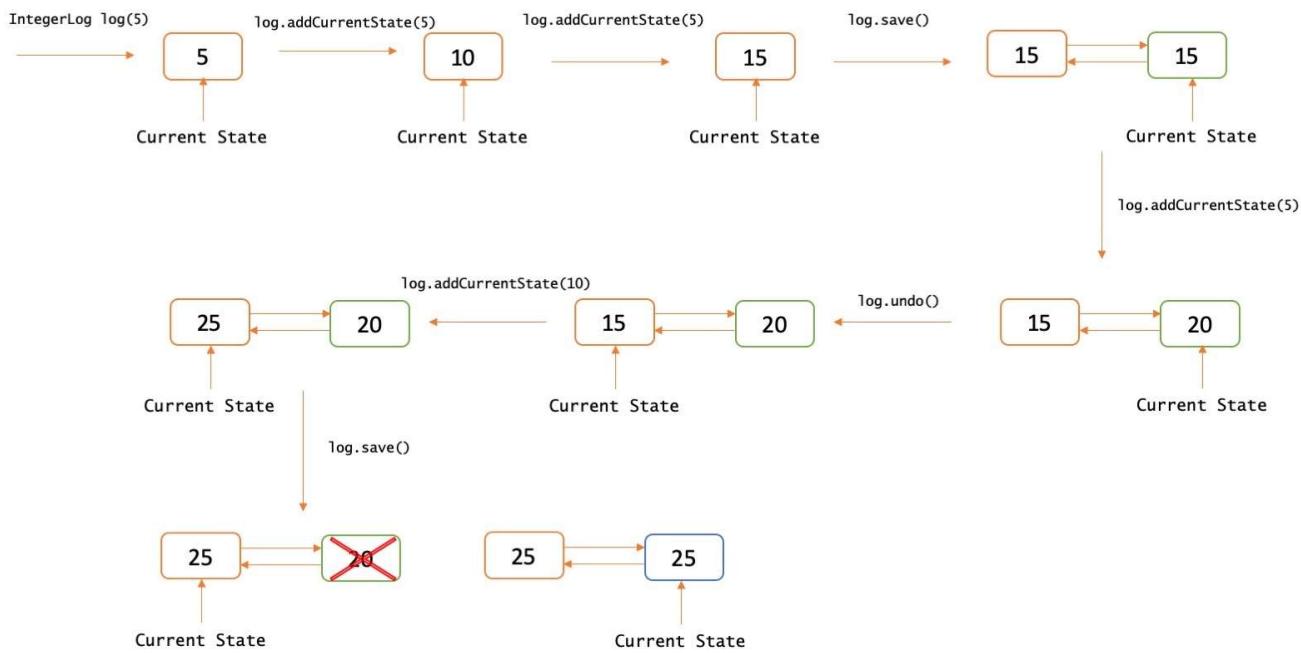
public:
    DataLog();
    DataLog(const int &data);
    void addCurrentState(int number);
    void subtractCurrentState(int number);
    void save();
    void undo();
    void redo();

    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;



For example:

Test	Result
<pre>DataLog log(10); log.save(); log.addcurrentState(15); log.save(); log.addcurrentState(15); log.undo(); log.printLog();</pre>	[10] => Current state: [25] => [40] => END_LOG
<pre>DataLog log(10); log.save(); log.addcurrentState(15); log.save(); log.addcurrentState(15); log.save(); log.subtractcurrentState(5); log.printLog();</pre>	[10] => [25] => [40] => Current state: [35] => END_LOG

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_back(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_back(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 states behind the
40     *         currentState Iterator, we delete them all before creating a new state.
41     */
42     list<int>::iterator it = currentState;
43     it++;
44     logList.erase(it, logList.end());
45     logList.push_back(*currentState);
46     currentState++;
47 }
48
49 void DataLog::undo()
50 {
51     /*
52     * TODO: Switch to the previous state of the data
53     *         If this is the oldest state in the log, nothing changes
54     */
55     if (currentState == logList.begin()) return;
56     currentState--;
57 }
58
59 void DataLog::redo()
60 {
61     /*
62     * TODO: Switch to the latter state of the data
63     *         If this is the latest state in the log, nothing changes
64     */
65     list<int>::iterator it = currentState;
66     it++;
67     if (it == logList.end()) return;
68     currentState++;
69 }
```

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

	Test	Expected	Got	
✓	DataLog log(10); log.save(); log.addcurrentState(15); log.save(); log.addcurrentState(15); log.save(); log.subtractcurrentState(5); log.printLog();	[10] => [25] => [40] => Current state: [35] => END_LOG	[10] => [25] => [40] => Current state: [35] => END_LOG	✓

Passed all tests! ✓

Chỉnh xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Không chính xác

Điểm 0,00 của 1,00

Assume that you build a toy called line-street. You have to implement

some functions followed by some rules:

```
void LineStreet(string homepage): make a root for a line
void addNewElement(string ele): add new element in line, and clear up
```

all forward elements

```
void back(int steps): you have to back to element behind with steps.
```

If you can only back to n while $n < \text{steps}$. You should back to n steps.

```
void forward(int steps): you have to forward to element behind with steps.
```

If you can only forward to n while $n < \text{steps}$. You should

forward n steps.

Simple Example:

```
LineStreet* line = new LineStreet("home");
line->addNewElement("Bob");
line->addNewElement("Smith");
line->addNewElement("Ann");
line->back(1);                                --> return "Smith"
line->back(1);                                --> return "Bob"
line->forward(1);                             --> return "Smith"
line->addNewElement("Peter");
line->forward(2);                            --> return "Peter"
line->back(1);                                --> return "Smith"
```

Constraints: $1 \leq \text{steps} \leq 100$

$1 \leq \text{len(homepage)}, \text{len(ele)} \leq 20$

Note: In this exercise, Libraries `iostream`, `string` and `using namespace std;`

have been used. You can add other

functions for your answer, but you are not allowed to add other libraries.

For example:

Test	Result
<pre>LineStreet* obj = new LineStreet("home"); obj->addNewElement("Bob"); obj->addNewElement("Smith"); obj->addNewElement("Ann"); cout << obj->back(1) << endl; cout << obj->back(1) << endl; cout << obj->forward(1) << endl; obj->addNewElement("Peter"); cout << obj->forward(2) << endl; cout << obj->back(1) << endl;</pre>	Smith Bob Smith Peter Smith

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
1. class LineStreet {
```

```
2  public:
3  class Node;
4  private:
5  Node* head;
6  Node* curr;
7  Node*tail;
8  bool clear_up;
9 public:
10
11 LineStreet(string homepage) {
12     head=new Node(homepage,nullptr,nullptr);
13     tail=head;
14     curr=head;
15 };
16
17 void addNewElement(string url) {
18 if(curr==tail){
19     tail->next=new Node(url,tail,nullptr);
20     tail=tail->next;
21     tail->prev=tail;
22 }
23 else if(curr==head){
24
25 }
26
27 };
28
29 string back(int steps) {
30     Node*temp=curr;
31 for(int i=0;i<steps;i++){
32 if(temp->prev!=nullptr){
33     temp=temp->prev;
34     }
35     else break;
36 }
37     curr=temp;
38     return temp->s;
39 };
40
41 string forward(int steps) {
42     Node*temp=curr;
43 for(int i=0;i<steps;i++){
44 if(temp->next!=nullptr){
45     temp=temp->next;
46     }
47     else break;
48 }
49     curr=temp;
50     return temp->s;
51 };
52 class Node{
53 public:
54     Node*next;
55     Node*prev;
56     string s;
57 Node(string s="",Node*prev=nullptr,Node*next=nullptr){
58     this->s=s;
59     this->prev=prev;
60     this->next=next;
61     }
62     };
63 };
64 }
```



Syntax Error(s)

```
_tester_.cpp: In member function 'void LineStreet::addNewElement(std::__cxx11::string)':  
_tester_.cpp:29:21: error: suggest parentheses around assignment used as truth value [-Werror=parentheses]  
    else if(curr=head){  
        ~~~~~^~~~~~  
cc1plus: all warnings being treated as errors
```

Không chính xác

Điểm cho bài nộp này: 0,00/1,00.

Câu hỏi 8

Chính xác

Điểm 1,00 của 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:

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

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

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

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>	5 3 4 5 6 7 2 4	3 6 5 4 7

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

BÁCH KHOA E-LEARNING



WEBSITE

HCMUT

MvRK

```

1
2
3
4
5
6 Thuròng Kiệt, P.14, Q.10, TP.HCM
7
8 88 651 670 - (028) 38 647 256 (Ext: 5258, 5234)
9
10 vking@hcmut.edu.vn
11
12
13
14
15
16 for(i=1;i<a;i++){
17     temp=temp->right;
18 }
19 ListNode*head_rev_start=temp;
20 ListNode*start_part=temp->left;
21 int delta =b-a;
22 while(delta>=0){
23     ListNode*dummy = temp->right;
24     temp->right=temp->left;
25     temp->left=dummy;
26 if(delta!=0){
27     temp=dummy;
28 }
29 delta--;
30 }
31 ListNode*end_part=temp->left;
32 // link the start the rev and the end
33 if(start_part!=nullptr){
34     start_part->right=temp;
35     temp->left=start_part;
36 }
37 else{
38     head=temp;
39     temp->left=nullptr;
40 }
41 if(end_part!=nullptr){
42     end_part->left=head_rev_start;
43     head_rev_start->right=end_part;
44 }
```

```

45     }
46     head_rev_start->right=nullptr;
47   }
48   return head;
49 }
```

	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>	5 3 4 5 6 7 2 4	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>	3 8 9 10 1 3	10 9 8	10 9 8	✓

Passed all tests! ✓

Chỉnh xác

Điểm cho bài nộp này: 1,00/1,00.