Đã bắt đầu vào lúc	Thứ bảy, 24 Tháng chín 2022, 3:01 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ tư, 12 Tháng mười 2022, 2:27 PM
Thời gian thực hiện	17 ngày 23 giờ
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();
    ~Deque();
    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

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;
	43 1234 223 568 90 193 2109
	7

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;
	14 13 12 11 10 9 8 7 6 5
	10

Answer: (penalty regime: 0 %)

Reset answer

```
Deque::Deque() {
    head = nullptr;
    tail = nullptr;
    curSize = 0;
}

Deque::~Deque() {
}

void Deque::clear() {
    while(head != nullptr){
        popBack();
    }
}

int Deque::getSize() {
    return curSize;
}

void Deque::pushFront(int i) {
    if (head == nullptr){
        Node* newNode = new Node(i,nullptr,nullptr);
        head = newNode;
        tail = newNode;
    } else {
        Node* newNode = new Node(i,nullptr,head);
        head->left = newNode;
        head = newNode;
    }
    curSize += 1;
}

void Deque::pushBack(int i) {
    if (head == nullptr){
        Node* newNode = new Node(i,nullptr,nullptr);
        head = newNode;
```

```
        tail = newNode;
    } else {
        Node* newNode = new Node(i,tail,nullptr);
        tail->right = newNode;
        tail = newNode;
    }
    curSize += 1;
}

int Deque::popFront() {
    if (head == nullptr)
        return -1;

    int ans = head->value;
    Node* newHead = head->right;
    if (newHead != nullptr){
        newHead->left = nullptr;
        delete head;
        head = newHead;
    } else {
        delete head;
        head = nullptr;
        tail = nullptr;
    }
    curSize -= 1;
    return ans;
}

int Deque::popBack() {
    if (head == nullptr)
        return -1;

    int ans = tail->value;
    Node* newTail = tail->left;
    if (newTail != nullptr){
        newTail->right = nullptr;
        delete tail;
        tail = newTail;
    } else {
        delete tail;
        head = nullptr;
        tail = nullptr;
    }
    curSize -= 1;
    return ans;
}

void Deque::printDequeReverse() {
   Node*temp=tail;
   while(temp!=nullptr){
        cout<<temp->value<<" ";
       temp=temp->left;
   }
   cout<<endl;
}

void Deque::printDeque() {
    Node*temp=head;
    while(temp!=nullptr){
        cout<<temp->value<<" ";
        temp=temp->right;
    }
    cout<<endl;
}
```

	Test	Expected	Got	

✔	`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;`
		43 1234 223 568 90 193 2109
		7
			43 1234 223 568 90 193 2109
			7
				✔

✔	`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;`
		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

Điểm 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

`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

```
template <class T>
void DLinkedList<T>::add(const T& e) {
    if (count == 0)
    {
    Node* newNode = new Node(e);
    head = newNode;
    tail = newNode;
    tail->next = NULL;
    ++(this->count);
    return;
    }
    Node* newNode = new Node(e);
    tail->next = newNode;
    newNode->previous = tail;
    newNode->next = NULL;
    tail = newNode;
    ++(this->count);
    return;
}

template<class T>
void DLinkedList<T>::add(int index, const T& e) {
    /* Insert an element into the list at given index. */
if (count == 0) {add(e);return;}
if (index == 0)
{
Node* newNode = new Node(e);
newNode->next = head;
head->previous = newNode;
head = newNode;
++(this->count);
return;
}
    if (index == this->count) {add(e); return;}
    int idx = 0;
    Node* front = head;
    Node* back = NULL;
    for (;front != NULL; back = front, front = front->next, ++idx)
    {
        if (idx == index)
        {
        Node* newNode = new Node (e);
        ++(this->count);
        back->next = newNode;
        newNode->next = front;
        front->previous = newNode;
        return;
        }
    }

}

template<class T>
int DLinkedList<T>::size() {
    /* Return the length (size) of list */
    int cnt=0;
    Node*temp=head;
    while(temp!=nullptr){
        temp=temp->next;
        cnt++;
    }
    return cnt;
}
```

	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! ✔

Chính 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 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

Test	Result

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 |

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]

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

Reset answer

```
template<class T>
T DLinkedList<T>::get(int index) {
if (count == 0) return -1;
if (index == this->count - 1) return tail->data;
if (index == 0) return head->data;
int idx = 0;
for (Node* h = head; h != NULL; h = h->next, ++idx)
{
if (idx == index) return h->data;
}
return -1;
/* Give the data of the element at given index in the list. */
}
template <class T>
void DLinkedList<T>::set(int index, const T& e) {
if (count == 0) return;
if (index == 0)
{
head->data = e;
return;
}
if (index == this->count - 1)
{
tail->data = e;
return;
}
int idx = 0;
for (Node* h = head; h != NULL; h = h->next, ++idx)
{
if (idx == index)
{
h->data = e;
return;
}
}
/* Assign new value for element at given index in the list */
}

template<class T>
bool DLinkedList<T>::empty() {
/* Check if the list is empty or not. */
if (count ==0) return true;
return false;

}
```

```
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

`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 << " |";`
`}`
	`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(idx);`
`}`
 	
`DLinkedList<int>::Iterator it = list.begin();`
`while (it != list.end())`
`{`
`  it.remove();`
`  it++;`
`}`
`cout << list.toString();`
	`[]`

`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();`
	`[]`

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  {
```

```
    this->pList = pList;
    if (begin == true) {
        if (pList != NULL) {
            this->current = pList->head;
            this->index = 0;
        }
        else {
            this->current = NULL;
            this->index = -1;
        }
    }
    else {
        this->current = NULL;
        if (pList != NULL) this->index = pList->count;
        else this->index = 0;
    }
}

template <class T>
typename DLinkedList<T>::Iterator& DLinkedList<T>::Iterator::operator=(const DLinkedList<T>::Iterator &iter
{
    this->current = iterator.current;
    this->index = iterator.index;
    this->pList = iterator.pList;
    return *this;
}

template <class T>
void DLinkedList<T>::Iterator::set(const T &e)
{
    if (current == NULL) throw std::out_of_range("Segmentation fault!");
    current->data = e;
}

template<class T>
T& DLinkedList<T>::Iterator::operator*()
{
    if (current == NULL) throw std::out_of_range("Segmentation fault!");
    return current->data;
}

template<class T>
void DLinkedList<T>::Iterator::remove()
{
    /*
    * TODO: delete Node in pList which Node* current point to.
    *       After that, Node* current point to the node before the node just deleted.
    *       If we remove first node of pList, Node* current point to nullptr.
    *       Then we use operator ++, Node* current will point to the head of pList.
    */
    if (current == NULL) throw std::out_of_range("Segmentation fault!");
    if (index == 0) {
        pList->head = current->next;
        delete current;
        pList->count--;
        if (pList->count == 0) pList->tail = pList->head;
        current = nullptr;
        index = -1;
    }
    else {
        Node* prev = pList->head;
        while (prev->next != current) prev = prev->next;
        prev->next = current->next;
        delete current;
        pList->count--;
        if (index == pList->count) pList->tail = prev;
        current = prev;
        index--;
    }
}

template<class T>
bool DLinkedList<T>::Iterator::operator!=(const DLinkedList::Iterator &iterator)
```

```
{
    if (this->current == iterator.current && this->index == iterator.index) return false;
    else return true;
}

template<class T>
typename DLinkedList<T>::Iterator& DLinkedList<T>::Iterator::operator++()
{
    //if (current == NULL) throw std::out_of_range("Segmentation fault!");
    if (index < 0) {
        current = pList->head;
        index = 0;
    }
    else {
        current = current->next;
        index++;
    }
    return *this;
}

template<class T>
typename DLinkedList<T>::Iterator DLinkedList<T>::Iterator::operator++(int)
{
    Iterator ret = *this;
    //if (current == NULL) throw std::out_of_range("Segmentation fault!");
    if (index < 0) {
        current = pList->head;
        index = 0;
    }
    else {
        current = current->next;
        index++;
    }
    return ret;
}
```

	Test	Expected	Got	

✔	`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 << " |";`
	`}`
		`0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |`	`0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |`	✔

	Test	Expected	Got	

✔	`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();`
		[]	[]	✔

✔	`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();`
		[]	[]	✔

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

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]

Answer: (penalty regime: 0 %)

Reset answer

```
template <class T>
T DLinkedList<T>::removeAt(int index)
{
    Node* temp = head;
    /* Remove element at index and return removed value */
    if(temp==nullptr){return 0;}
    T ret;
    if(index==0){
        Node*hold=head->next;
        if(head==tail){
            tail=nullptr;
        }
        ret=head->data;
        delete head;
        head=hold;
        if(head!=nullptr)
        head->previous=nullptr;
        //return ret;
    }
    else if(index==count-1){
        Node*hold=tail->previous;
        ret = tail->data;
        delete tail;
        tail=hold;
        tail->next=nullptr;
        //return ret;
    }
    else{
        Node*dummy_head=head;
        for(int i=0;i<index;i++){
            dummy_head=dummy_head->next;
        }
        Node*temp_prev = dummy_head->previous;
        Node*temp_next =  dummy_head->next;
        ret= dummy_head->data;
        delete dummy_head;
        temp_prev->next=temp_next;
        temp_next->previous=temp_prev;
    }
    //T a = temp->data;
    //delete temp;
    count--;
    return ret;
}

template <class T>
bool DLinkedList<T>::removeItem(const T& item)
{
    /* Remove the first apperance of item in list and return true, otherwise return false */
    Node* temp = head;
    for (int i = 0; temp != nullptr; i++) {
        if (temp->data == item) {
            removeAt(i);
            return true;
        }
```

```
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.

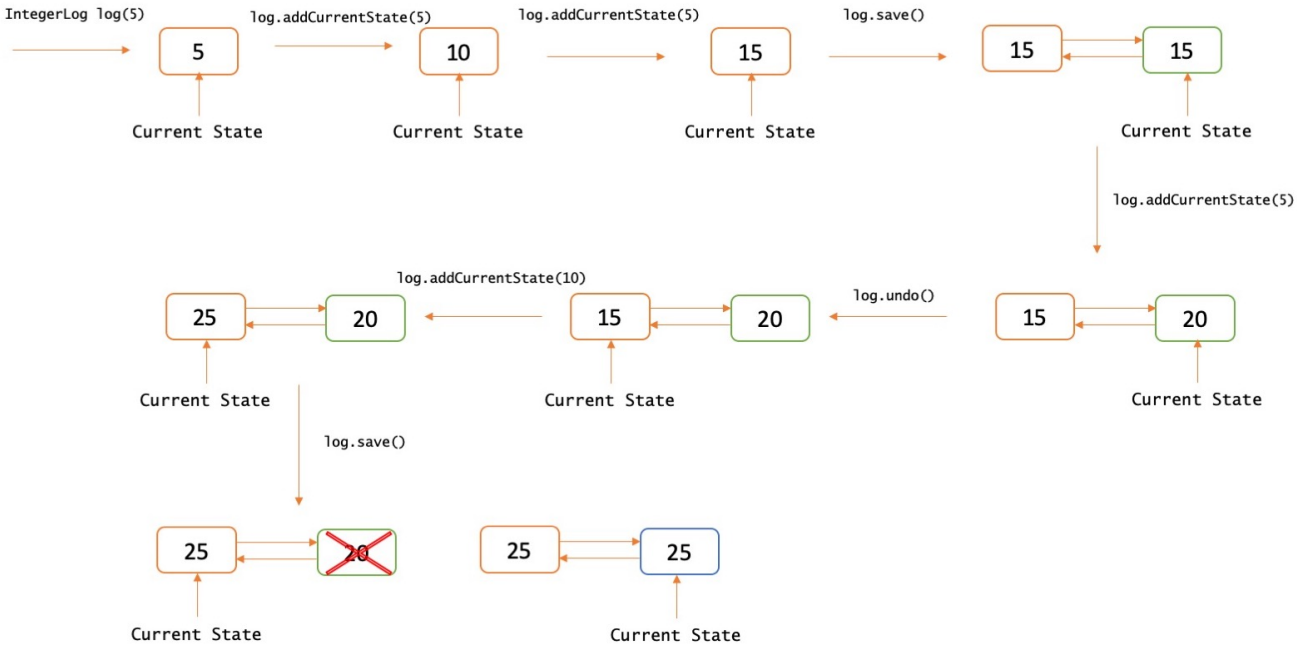
DataLog has a doubly linked list to store the states of data (an integer) and iterator to mark the current state. Each state is stored in a node, the transition of states is depicted in the figure below.

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

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

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

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

Reset answer

```
DataLog::DataLog()
{
    /*
     * TODO:  add the first state with 0
    */
    logList.push_back(0);
    currentState = logList.begin();
}

DataLog::DataLog(const int &data)
{
    /*
     * TODO:  add the first state with data
     */
    logList.push_back(data);
    currentState = logList.begin();
}
```

```
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	

	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	✔

✔	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! ✔

Chính 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 impletement

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 < 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 < 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 <= steps <= 100

1<= len(homepage), len(ele) <= 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

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;
	Smith
	Bob
	Smith
	Peter
	Smith

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

Reset answer

```
1 class LineStreet {
```

```
    public:
    class Node;
    private:
    Node* head;
    Node* curr;
    Node*tail;
    bool clear_up;
public:

    LineStreet(string homepage) {
        head=new Node(homepage,nullptr,nullptr);
        tail=head;
        curr=head;
    };

    void addNewElement(string url) {
        if(curr==tail){
            tail->next=new Node(url,tail,nullptr);
            tail=tail->next;
            tail->prev=tail;
        }
        else if(curr=head){

        }

    };

    string back(int steps) {
        Node*temp=curr;
        for(int i=0;i<steps;i++){
            if(temp->prev!=nullptr){
                temp=temp->prev;
            }
            else break;
        }
        curr=temp;
        return temp->s;
    };

    string forward(int steps) {
        Node*temp=curr;
        for(int i=0;i<steps;i++){
            if(temp->next!=nullptr){
                temp=temp->next;
            }
            else break;
        }
        curr=temp;
        return temp->s;
    };
    class Node{
        public:
        Node*next;
        Node*prev;
        string s;
        Node(string s="",Node*prev=nullptr,Node*next=nullptr){
            this->s=s;
            this->prev=prev;
            this->next=next;
        }
    };
};
```

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 <= 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

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;
	5
	3 4 5 6 7
	2 4
		3 6 5 4 7

Test	Input	Result

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;
	3
	8 9 10
	1 3
		10 9 8

Answer: (penalty regime: 0 %)

Reset answer

```
/*
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==nullptr){
        return head;
    }
    ListNode*temp=head;
    int i;
    for(i=1;i<a;i++){
        temp=temp->right;
    }
    ListNode*head_rev_start=temp;
    ListNode*start_part=temp->left;
    int delta =b-a;
    while(delta>=0){
        ListNode*dummy = temp->right;
        temp->right=temp->left;
        temp->left=dummy;
        if(delta!=0){
            temp=dummy;
        }
        delta--;
    }
    ListNode*end_part=temp->left;
    // link the start the rev and the end
    if(start_part!=nullptr){
        start_part->right=temp;
        temp->left=start_part;
    }
    else{
        head=temp;
        temp->left=nullptr;
    }
    if(end_part!=nullptr){
        end_part->left=head_rev_start;
        head_rev_start->right=end_part;
    }
```

```
    else {
        head_rev_start->right=nullptr;
    }
    return head;
}
```

	Test	Input	Expected	Got	

✔	`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;`
		5
		3 4 5 6 7
		2 4
			3 6 5 4 7	3 6 5 4 7	✔

✔	`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;`
		3
		8 9 10
		1 3
			10 9 8	10 9 8	✔

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

elearning@hcmut.edu.vn