Thời gian còn lại 0:13:00

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]`

`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 %)

Reset answer

```
template <class T>
void DLinkedList<T>::add(const T& e) {
    /* Insert an element into the end of the list. */
    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=nullptr;
    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;
        }
```

Precheck Kiểm tra

	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]`	✓

	Test	Expected	Got	

✓	`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 2

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

`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 %)

Reset answer

```
template<class T>
T DLinkedList<T>::get(int index) {
    /* Give the data of the element at given index in the list. */
    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;
}

template <class T>
void DLinkedList<T>::set(int index, const T& e) {
    /* Assign new value for element at given index in the list */
    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;
        }
    }
}

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

```cpp

template<class T>
int DLinkedList<T>::indexOf(const T& item) {
    /* Return the first index wheter item appears in list, otherwise return -1 */
```

Precheck   Kiểm tra

	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 3

Đúng một phần

Điểm 0,20 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

`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)
{
    T luu;
    if (index == 0)
    {
        Node* temp = head;
        luu = head->data;
        head = head->next;
        head->previous = NULL;
        delete temp;
        --(this->count);
        return luu;
    }
    else if (index == count - 1)
    {
        Node* temp = tail;
        luu = tail->data;
        tail = tail->previous;
        if (tail)
            tail->next = nullptr;
        delete temp;
        --count;
        return luu;
    }
    else
    {
        int currentIndex = 0;
        Node* current = head;

        while (currentIndex < index && current)
        {
            current = current->next;
            currentIndex++;
        }

        if (current)
        {
            Node* temp = current;
            luu = current->data;
            current->previous->next = current->next;
            current->next->previous = current->previous;
            delete temp;
            --count;
            return luu;
        }
    }
    return 1;
}
```

Kiểm tra

	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]`	✓

✗	`DLinkedList<int> list;`
	`int values[] = {10, 15, 2, 6, 4, 7, 40, 8};`
	`int index[] = {0, 1, 5, 3, 2, 1, 1, 0};`
	
	`for (int idx = 0; idx < 8; idx++)`
	`    list.add(values[idx]);`
	
	`for(int idx=0; idx < 8; idx++){`
	`    int idxRemoved = index[idx];`
	`    int rs = list.removeAt(idxRemoved);`
	`    cout << rs << "\n" << list.toString() << endl;`
	`}`
	`cout << list.empty();`
		10
		[15,2,6,4,7,40,8]
		2
		[15,6,4,7,40,8]
		8
		[15,6,4,7,40]
		7
		[15,6,4,40]
		4
		[15,6,40]
		6
		[15,40]
		40
		[15]
		15
		[]
		1
			10
			[15,2,6,4,7,40,8]
			2
			[15,6,4,7,40,8]
			8
			[15,6,4,7,40]
			7
			[15,6,4,40]
			4
			[15,6,40]
			6
			[15,40]
			40
			[15]
			
			Run error
			Segmentation fault (core dumped)
				✗

Testing was aborted due to error.

Show differences

Đúng một phần

Điểm cho bài nộp này: 0,20/1,00.

Câu hỏi 4

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;`	`3`	`10 9 8`
`    cin >> size;`	`8 9 10`	
`    int* list = new int[size];`	`1 3`	
`    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;`		

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 || a == b) {
        return head;
    }

    // Create dummy nodes for easier handling of edge cases.
    ListNode* dummy = new ListNode(0);
    dummy->right = head;
    head->left = dummy;

    ListNode* preA = dummy;
    ListNode* curA = head;

    // Move to position a.
    for (int i = 1; i < a; i++) {
        preA = curA;
        curA = curA->right;
    }

    ListNode* preB = curA;
    ListNode* curB = curA->right;

    // Move to position b.
    for (int i = a; i < b; i++) {
        preB = curB;
        curB = curB->right;
    }

    // Reverse the sublist from a to b.
    preB->right = nullptr;
```

Precheck

Kiểm tra

	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;`
	`    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`	`10 9 8`	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.