Status	Finished
Started	Sunday, 20 October 2024, 1:40 PM
Completed	Monday, 21 October 2024, 8:08 PM
Duration	1 day 6 hours
Marks	5.00/5.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 1.00 out of 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;
        }
    }

}

```

	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! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 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, 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;
```

```
}

template<class T>
int DLinkedList<T>::indexOf(const T& item) {
    /* Return the first index wheter item appears in list, otherwise return -1 */
    int idx=0;
    if(count==0) return -1;
    for(Node*temp_head=head;temp_head!=nullptr;temp_head=temp_head->next,idx++){
```

	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! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 3

Correct

Mark 1.00 out of 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)
{
    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) {
```

	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! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 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.

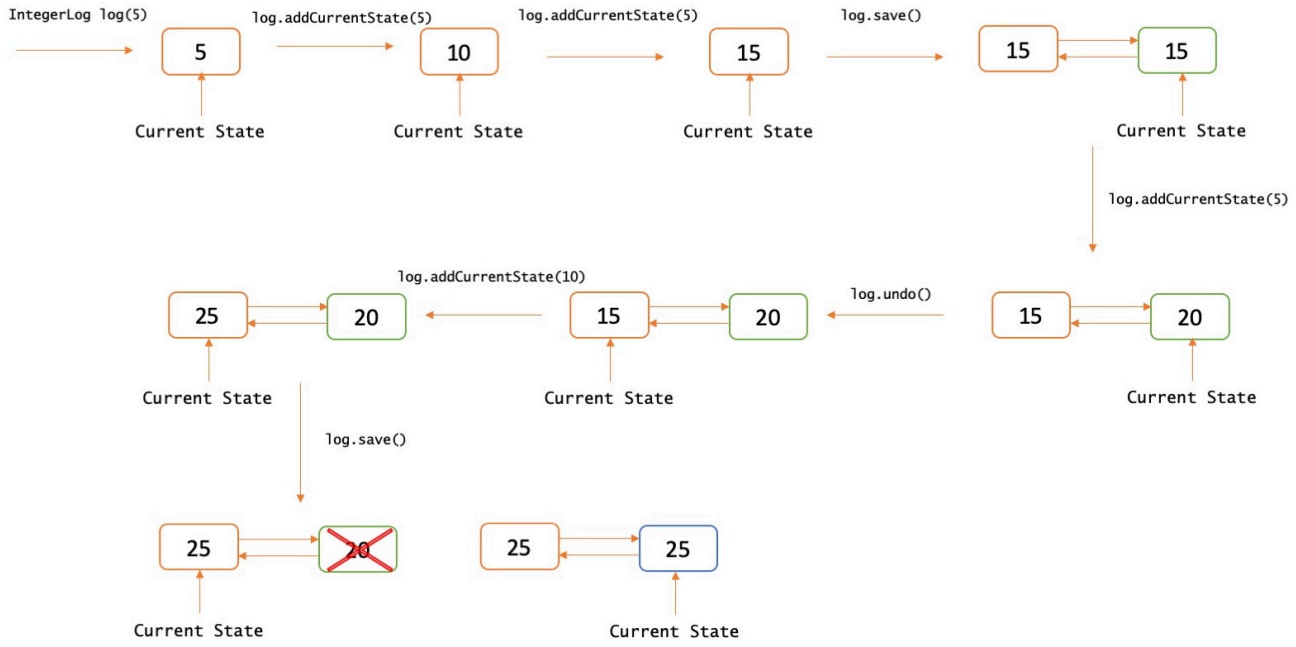
DataLog has a doubly linked list to store the states of data (an integer) and iterator to mark the current state. Each state is stored in a node, the transition of states is depicted in the figure below.

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();
```

```
}

void DataLog::addCurrentState(int number)
{
    /*
     * TODO: Increase the value of current state by number
     */
    (*currentState) += number;
}

void DataLog::subtractCurrentState(int number)
{
    /*
     * TODO: Decrease the value of current state by number
     */
    (*currentState) -= number;
}

void DataLog::save()
{
    /*
     * TODO: This function will create a new state, copy the data of the currentState
     *       and move the currentState Iterator to this new state. If there are other states behind the
     *       currentState Iterator, we delete them all before creating a new state.
     */
    list<int>::iterator it = currentState;
    it++;
    logList.erase(it, logList.end());
    logList.push_back(*currentState);
    currentState++;
}

void DataLog::undo()
{
    /*
     * TODO: Switch to the previous state of the data
```

	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! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 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;
	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! ✓

Correct

Marks for this submission: 1.00/1.00.