Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Ba, 25 tháng 2 2025, 1:57 PM
Kết thúc lúc	Thứ Hai, 10 tháng 3 2025, 11:57 PM
Thời gian thực hiện	13 Các ngày 10 giờ
Điểm	4,20/5,00
Điểm	**8,40** trên 10,00 (**84**%)

Câu hỏi 1

Đúng

Đạt điểm 1,00 trên 1,00

Implement all methods in class **Stack** with template type **T**. The description of each method is written as comment in frame code.

```
#ifndef STACK_H
#define STACK_H
#include "DLinkedList.h"
template<class T>
class Stack {
protected:
    DLinkedList<T> list;
public:
    Stack() {}
    void push(T item) ;
    T pop() ;
    T top() ;
    bool empty() ;
    int size() ;
    void clear() ;
};

#endif
```

You can use all methods in class **DLinkedList** without implementing them again. The description of class **DLinkedList** is written as comment in frame 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);
    T removeAt(int index);
    bool removeItem(const T& removeItem);
    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);
};
```

For example:

Test	Result
Stack<int> stack; cout << stack.empty() << " " << stack.size();	1 0

Test	Result

Stack<int> stack;
 	
int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };
for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);
 	
assert(stack.top() == 12);
 	
stack.pop();
stack.pop();
 	
cout << stack.top();
	8

Answer: (penalty regime: 0 %)

Reset answer

```
void push(T item) {
        // Thêm phần tử mới vào đầu danh sách (đỉnh của stack)
        list.add(0, item);
    }

    T pop() {
        // Nếu stack rỗng, ném ngoại lệ
        if (list.empty()) {
            throw std::out_of_range("Stack is empty.");
        }
        // Xóa và trả về phần tử đầu tiên (đỉnh của stack)
        return list.removeAt(0);
    }

    T top() {
        // Nếu stack rỗng, ném ngoại lệ
        if (list.empty()) {
            throw std::out_of_range("Stack is empty.");
        }
        // Trả về phần tử đầu tiên mà không xóa nó
        return list.get(0);
    }

    bool empty() {
        // Kiểm tra xem danh sách có rỗng không
        return list.empty();
    }

    int size() {
        // Trả về số lượng phần tử trong stack
        return list.size();
    }

    void clear() {
        // Xóa toàn bộ phần tử trong stack
        list.clear();
    }

```

	Test	Expected	Got	
✓	`Stack<int> stack;` `cout << stack.empty() << " " << stack.size();`	`1 0`	`1 0`	✓

✓	`Stack<int> stack;`
	
	`int item[] = { 3, 1, 4, 5, 2, 8, 10, 12 };`
	`for (int idx = 0; idx < 8; idx++) stack.push(item[idx]);`
	
	`assert(stack.top() == 12);`
	
	`stack.pop();`
	`stack.pop();`
	
	`cout << stack.top();`
		`8`	`8`	✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 1,00

Implement all methods in class **Queue** with template type **T**. The description of each method is written as comment in frame code.

```
#ifndef QUEUE_H
#define QUEUE_H
#include "DLinkedList.h"
template<class T>
class Queue {
protected:
    DLinkedList<T> list;
public:
    Queue() {}
    void push(T item) ;
    T pop() ;
    T top() ;
    bool empty() ;
    int size() ;
    void clear() ;
};

#endif /* QUEUE_H */
```

You can use all methods in class **DLinkedList** without implementing them again. The description of class **DLinkedList** is written as comment in frame 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);
    T removeAt(int index);
    bool removeItem(const T& removeItem);
    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);
};
```

For example:

Test	Result

`Queue<int> queue;`
`  assert(queue.empty());`
`  assert(queue.size() == 0);`	

Answer: (penalty regime: 0 %)

Reset answer

```
1 void push(T item) {
```

```
        // Thêm phần tử mới vào cuối queue
        list.add(item);  // Giả sử add(item) thêm phần tử vào cuối danh sách
    }

    T pop() {
        // Nếu queue rỗng, ném ngoại lệ
        if (list.empty()) {
            throw std::out_of_range("Queue is empty.");
        }
        // Xóa và trả về phần tử ở đầu queue (index 0)
        return list.removeAt(0);
    }

    T top() {
        // Nếu queue rỗng, ném ngoại lệ
        if (list.empty()) {
            throw std::out_of_range("Queue is empty.");
        }
        // Trả về phần tử ở đầu queue mà không xóa nó (index 0)
        return list.get(0);
    }

    bool empty() {
        // Kiểm tra xem queue có rỗng không
        return list.empty();
    }

    int size() {
        // Trả về số lượng phần tử hiện có trong queue
        return list.size();
    }

    void clear() {
        // Xóa toàn bộ các phần tử trong queue
        list.clear();
    }
```

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 1,00

Hiện thực hàm **void reverseQueue**, thực hiện việc đảo ngược các phần tử trong queue sử dụng stack.
Thư viện đã được thêm vào: stack

For example:

Test	Result

`int arr[] = {1, 2, 3};`
`int size = sizeof(arr) / sizeof(arr[0]);`
`queue<int> q;`
`for (int i = 0; i < size; ++i) {`
`  q.push(arr[i]);`
`}`
`cout << queueToStr(q) << endl;`
`reverseQueue(q);`
`cout << queueToStr(q) << endl;`
	`FRONT| 1 2 3 |REAR`
	`FRONT| 3 2 1 |REAR`

Answer: (penalty regime: 0 %)

Reset answer

```
1
2  void reverseQueue(std::queue<int>& q) {
3      std::stack<int> s;
4
5      // Chuyển tất cả các phần tử từ queue sang stack
6      while (!q.empty()) {
7          s.push(q.front());
8          q.pop();
9      }
10
11     // Chuyển các phần tử từ stack trở lại queue
12     while (!s.empty()) {
13         q.push(s.top());
14         s.pop();
15     }
16 }
17
```

	Test	Expected	Got	

✓	`int arr[] = {1, 2, 3};`
	`int size = sizeof(arr) / sizeof(arr[0]);`
	`queue<int> q;`
	`for (int i = 0; i < size; ++i) {`
	`  q.push(arr[i]);`
	`}`
	`cout << queueToStr(q) << endl;`
	`reverseQueue(q);`
	`cout << queueToStr(q) << endl;`
		`FRONT| 1 2 3 |REAR`
		`FRONT| 3 2 1 |REAR`
			`FRONT| 1 2 3 |REAR`
			`FRONT| 3 2 1 |REAR`
				✓

✓	`int arr[] = {};`
	`int size = sizeof(arr) / sizeof(arr[0]);`
	`queue<int> q;`
	`for (int i = 0; i < size; ++i) {`
	`  q.push(arr[i]);`
	`}`
	`cout << queueToStr(q) << endl;`
	`reverseQueue(q);`
	`cout << queueToStr(q) << endl;`
		`FRONT| |REAR`
		`FRONT| |REAR`
			`FRONT| |REAR`
			`FRONT| |REAR`
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 1,00

Hiện thực hàm **void reverseStack**, thực hiện việc đảo ngược các phần tử trong stack sử dụng queue.
Thư viện đã được thêm vào: queue

For example:

Test	Result

`int arr[] = {1, 2, 3};`
`int size = sizeof(arr) / sizeof(arr[0]);`
`stack<int> st;`
`for (int i = 0; i < size; ++i) {`
`  st.push(arr[i]);`
`}`
`cout << stackToStr(st) << endl;`
`reverseStack(st);`
`cout << stackToStr(st) << endl;`
	`TOP-> 3 2 1|BOTTOM`
	`TOP-> 1 2 3|BOTTOM`

Answer: (penalty regime: 0 %)

Reset answer

```
1
2  // Hàm chèn một phần tử vào đáy của stack
3  void insertAtBottom(stack<int>& st, int value) {
4      if (st.empty()) {
5          st.push(value);
6      } else {
7          int temp = st.top();
8          st.pop();
9          insertAtBottom(st, value);
10         st.push(temp);
11     }
12 }
13
14 void reverseStack(stack<int>& st) {
15     if (st.empty())
16         return;
17
18     int topVal = st.top();
19     st.pop();
20
21     // Đệ quy đảo ngược phần còn lại của stack
22     reverseStack(st);
23
24     // Chèn phần tử đã lấy ra vào đáy của stack
25     insertAtBottom(st, topVal);
26 }
27
```

	Test	Expected	Got	

✓	`int arr[] = {1, 2, 3};`
	`int size = sizeof(arr) / sizeof(arr[0]);`
	`stack<int> st;`
	`for (int i = 0; i < size; ++i) {`
	`  st.push(arr[i]);`
	`}`
	`cout << stackToStr(st) << endl;`
	`reverseStack(st);`
	`cout << stackToStr(st) << endl;`
		TOP-> 3 2 1|BOTTOM
		TOP-> 1 2 3|BOTTOM
			TOP-> 3 2 1|BOTTOM
			TOP-> 1 2 3|BOTTOM
				✓

✓	`int arr[] = {};`
	`int size = sizeof(arr) / sizeof(arr[0]);`
	`stack<int> st;`
	`for (int i = 0; i < size; ++i) {`
	`  st.push(arr[i]);`
	`}`
	`cout << stackToStr(st) << endl;`
	`reverseStack(st);`
	`cout << stackToStr(st) << endl;`
		TOP->|BOTTOM
		TOP->|BOTTOM
			TOP->|BOTTOM
			TOP->|BOTTOM
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **5**

Đúng một phần

Đạt điểm 0,20 trên 1,00

Hiện thực hàm **class QueueS**, gồm 4 phương thức:

1. enqueue(int value): thêm một phần tử vào cuối queue
2. dequeue(): loại bỏ phần tử ở đầu queue và trả về giá trị của phần tử này.
3. front(): trả về giá trị của phần tử ở đầu queue
4. rear(): trả về giá trị ở phần tử cuối.

Trong các phương thức 2, 3, 4, nếu queue rỗng, throw out_of_range("Empty queue").
Điểm đặc biệt trong QueueS là bạn cần dùng Stack để hiện hiện thực Queue.

For example:

Test	Result

`int arr[] = {1, 2, 3};`
`int size = sizeof(arr) / sizeof(arr[0]);`
`QueueS qs;`
`for (int i = 0; i < size; ++i) {`
`  qs.enqueue(arr[i]);`
`}`
`cout << qs.toString() << endl;`
	`FRONT| 1 2 3|REAR`

Answer: (penalty regime: 0 %)

Reset answer

```
#include <vector>
#include <string>
#include <sstream>
#include <stdexcept>
using namespace std;

class QueueS {
private:
    vector<int> data;  // Lưu trữ các phần tử của queue
public:
    // Thêm một phần tử vào cuối queue
    void enqueue(int value) {
        data.push_back(value);
    }

    // Lấy và xóa phần tử ở đầu queue
    int dequeue() {
        if (data.empty()) {
            throw out_of_range("Queue is empty.");
        }
        int frontVal = data.front();
        data.erase(data.begin());
        return frontVal;
    }

    // Lấy giá trị của phần tử ở đầu queue mà không xóa
    int front() {
        if (data.empty()) {
            throw out_of_range("Queue is empty.");
        }
        return data.front();
    }

    // Lấy giá trị của phần tử ở cuối queue mà không xóa
    int rear() {
        if (data.empty()) {
            throw out_of_range("Queue is empty.");
        }
        return data.back();
    }

    // Trả về chuỗi biểu diễn các phần tử trong queue
    string toString() {
        ostringstream oss;
```

```
45          oss << "FRONT|";
46          for (int value : data) {
47              oss << value << " ";
48          }
49          oss << "|REAR";
50          return oss.str();
51      }
52  };
53
```

	Test	Expected	Got	

✗	`int arr[] = {1, 2, 3};`
	`int size = sizeof(arr) / sizeof(arr[0]);`
	`QueueS qs;`
	`for (int i = 0; i < size; ++i) {`
	`  qs.enqueue(arr[i]);`
	`}`
	`cout << qs.toString() << endl;`
		`FRONT| 1 2 3|REAR`	`FRONT|1 2 3 |REAR`	✗

✓	`int arr[] = {};`
	
	`int size = sizeof(arr) / sizeof(arr[0]);`
	`QueueS qs;`
	`for (int i = 0; i < size; ++i) {`
	`  qs.enqueue(arr[i]);`
	`  cout << qs.toString() << endl;`
	`}`
	`cout << qs.toString() << endl;`
		`FRONT||REAR`	`FRONT||REAR`	✓

Some hidden test cases failed, too.

Show differences

Đúng một phần

Marks for this submission: 0,20/1,00.