ECE250-Project 1

Bingjian Du, b23du Jan 31th, 2020

1. Overview of Classes

Class: Node

Description: this is a basic unit of imformation.

Member variables: *prev, *next, data

Member functions:

Node(int i): create a node with data=i;

~Node(): default destructor;

Class: linked list

Description: this is a linked list.

Member variables:

*head: the first node of the list *tail: the last node of the list length: the size of the list

deque empty{}: used for exception handler

Member functions:

linked list(): initialize member variables

~linked list(): delete every node

void enqueue_front(int i): add an element from front void enqueue_back(int i): add an element from back void dequeue_front(): delete an element from front void dequeue_back(): delete an element from back

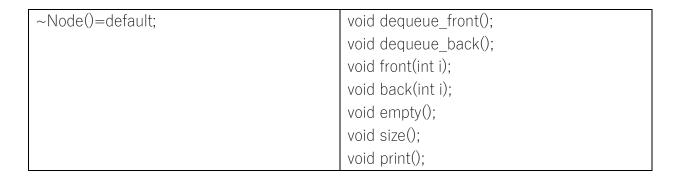
void front(int i): check if the data in the first element equals to i void back(int i): check if the data in the last element equals to i

void empty():check if the list is empty
void size(): return the length of the lisdt

void print(): print the data stored in the list from head to tail, tail to head

Class diagrams

Node	Linked_list
	Node *head,*tail;
	int length;
	class deque_empty{};
int data;	linked_list();
Node *next;	~linked_list();
Node *prev;	void enqueue_front(int i);
Node(int i);	void enqueue_back(int i);



2. Constructors/Destructor

Class Node: the constructor is intended to store an integer, so I pass an int to the constructor.

There is no specific requirement for destructor, so I keep it as default

Class linked_list: the constructor is to create an empty list, so I set the length=0, head and tail as nullptr. The destructor is intended to clean the list that every node should be deleted.

3. Test Cases

There are 2 cases I tested in addition to the example tests.

Test1: use dequeue_front to clean the list Test2: use dequeue back to clean the list

4. Performance

```
void enqueue_front(int i);
void enqueue_back(int i);
void dequeue_front();
void dequeue_back();
void front(int i);
void back(int i);
void empty();
```

These functions are implemented in O(1) times since there is no interation or loops in the function.

```
void print();
~linked list();
```

These functions are implemented in O(n) times because the times of operations are proportional to the number of elements (n) in the list. To be clearer, there is a while loop in the destructor which has 5 statements. The number of operations is 5n which has an upper bound O(n).