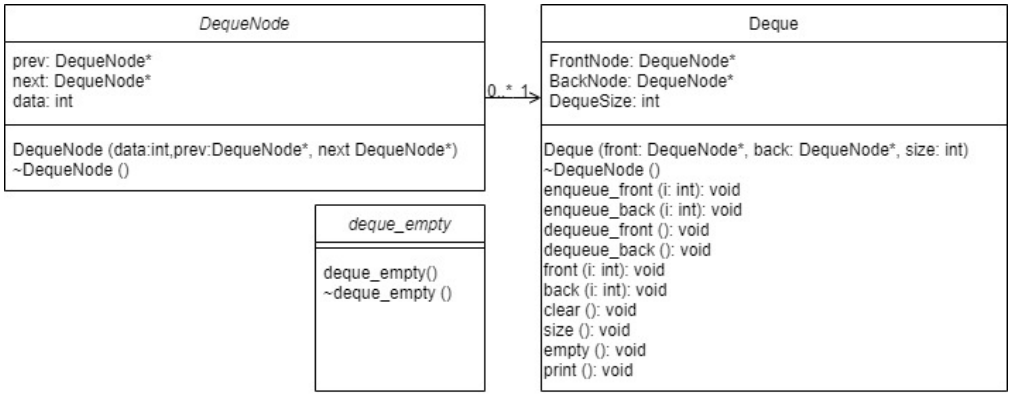
ECE250 – Project 1
Deque Driver
Design Document

Qinying Wu, q227wu
Jan 31th, 2020

1. Overview of Classes

What class(es) did you design? What are the member variables and member functions for each of these classes?

Class Name: deque_empty,
Description: the exception class for handling empty deque



Class Name: DequeNode
Description: an object representing a single node in the deque.
Member variables:

- *Prev* – DequeNode pointer type stores the base address of the previous node
- *next* – DequeNode pointer type stores the base address of the next node
- *data* – integer type stores the integer data associated with the current node

Class Name: Deque, **Description:** the deque containing the DequeNode objects
Member variables:

- *FrontNode* – DequeNode* pointer type stores the base address of the first node in the deque
- *BackNode* – DequeNode* pointer type stores the base address of the last node in the deque
- *DequeSize* – integer type stores the total number of DequeNode objects in the deque

Member functions (operations):

- void enqueue_front: add an element to the front of the deque
  - Takes in an integer parameter *i* that represents the data stored in the node
  - Outputs success if the element is inserted to the front successfully
- void enqueue_back: add an element to the back of the deque
  - Takes in an integer parameter *i* that represents the data stored in the node
  - Outputs success if the element is inserted to the back successfully
- void dequeue_front: remove the first element from the deque
  - outputs string "success" if deque is not empty
  - throws an exception, outputs string "failure" and return the current node if deque is empty
- void dequeue_back: remove the last element from the deque
  - outputs string "success" if deque is not empty
  - throws an exception, outputs string "failure" and return the current node if deque is empty
- void clear: erase all the elements in the deque if not already empty
- void front: access the first element in the deque and compare its stored value with the *i* parameter
  - takes in an integer parameter *i*
  - outputs string "success" if the two values are equal and that the deque is not empty
  - throws an exception and outputs string "failure" other wise
- void back: access the last element in the deque and compare its stored value with the *i* parameter
  - takes in an integer parameter *i*
  - output string "success" if the two values are equal and that the deque is not empty
  - throws an exception and outputs string "failure" otherwise
- void empty: test if the current deque is empty
  - outputs string "success" if empty, otherwise "failure"
- void size: output the value of the DequeSize member variable, which represents the total number of nodes
- void print: prints the integer values stored in the nodes of the deque from front to back and back to front
  - Outputs the integer values stored in each node first front to back, then in reverse order on a new line

2. Constructors

For each class, what are your design decisions regarding constructors?

The constructor for the *deque_empty* class is empty since it is a placeholder for passing an exception object. The constructor for *DequeNode* takes in three parameters. The integer parameter *i* is assigned to data. The pointer parameter *prev* is assigned to the same-name member variable as the link to the previous element (will be nullptr if the current element is the front of the

deque). The pointer parameter *next* is assigned to the same-name member variable as the link to the next element (will be nullptr if the current element is the back of the deque).
The constructor for *Deque* takes in three parameters. The DequeNode pointer *head* is assigned to FrontHead and *back* is assigned to BackNode. The integer *size* is 0 upon initialization since the deque is empty when first created.
For each class, what are your design decisions regarding destructors?
The destructor for *deque_empty* is empty since there is nothing to deallocate or to free up the memory. The destructor for DequeNode dereferences the current *prev* and *next* to null pointer, as well as the other pointers that are pointing to the current node. The destructor for the Deque removes all the nodes from the current deque.

3. Asymptotic Upper Bounds

All member functions in the Deque class except clear() and print() has constant run time. In enqueue_front and enqueue_back, the front and back of the deque can be changed by directly changing the *FrontNode* and *BackNode* of the Deque object, respectively. The same applies for *dequeue_front* and *dequeue_back* for removing a node and *front()* and *back* for comparing the data stored in the first and last node, respectively. When checking for the size of the deque, the size() function simply returns the value of *DequeSize* and the empty() function simply checks whether *DequeSize* is zero. There are no iterative loops such as for loops or while loops present in these functions, thus each of their run time is constant. The clear() and print() functions each contain a while loop. Therefore they have linear run time, which depends on the size of the deque.

4. <u>Test Cases</u>

a) Enqueue/Dequeue the front and back of the deque and print the deque (both positive and negative integers)
   a. When the deque is empty
   b. When the deque has only one element
      i. dequeue_back when enqueue_front
      ii. dequeue_front when enqueue_back
   c. When the deque is non-empty
      i. All the elements are retained (no change)
      ii. Some existing elements are removed

b) Check the front/back value of the deque and print the deque
   a. When the deque is empty
   b. When the deque has only one element
   c. When the deque is non-empty
      i. All the elements are retained (no change)
      ii. Some existing elements are removed

c) Check the size of the deque and test if it is empty or not and print the deque
   a. When the deque is empty
      i. Before and after clearing the deque
   b. When the deque has only one element
   c. When the deque is non-empty
      i. All the elements are retained (no change)
      ii. Some existing elements are removed
      iii. Before and after clearing the deque

Example:

enqueue_front 3 enqueue_front 4 enqueue_front 8 dequeue_back enqueue_back 9 enqueue_front -4 dequeue_front front 4 front 8 back 9 back 8 empty print clear print	size empty print enqueue_back 2 front 2 back 2 size print enqueue_front 5 enqueue_back 2 dequeue_front front 2 back 2 empty print	dequeue_back dequeue_front size enqueue_front 8 size dequeue_back enqueue_front 9 dequeue_back back 9 front 8	clear empty enqueue_back -2 enqueue_front 6 enqueue_front 5 size print empty clear enqueue_front 3 enqueue_back 2 clear size print empty