**COP3530 Data Structures and Algorithms**

**Project 1a: Implement a Double-Ended Queue**

**version 3**

**due 2/11/15**

**Brief Description:**

Develop a double-ended queue (Deque) using only arrays. The queue shall automatically grow and shrink depending on the usage with an initial and minimum size of eight (8). Its size shall double when it is full and shrink to half its size when it is less than 1/4 full (and not already of minimum size).

The project will be developed and evaluated in two parts. Part a is to implement a Deque that holds strings.

Your deque shall implement the following functions.

* push\_front - Inserts an element at the front of the queue
* push\_back - Inserts an element at the back of the queue
* pop\_front - Deletes the element at the front of the queue
* pop\_back - Deletes the element at the back of the queue
* empty - checks to see if the queue is empty
* size - returns the size of the queue
* toStr – concatenates the contents of the queue from front to back into a return string with each string item followed by a newline character in the return string – for example, if the deque has had push\_front(“alpha”), then push\_front(“beta”), then push\_back(“gamma”), the toStr method should produce “beta\nalpha\ngamma\n” as the returned string.
* appropriate constructors and destructors

**Deliverables**

* Well commented and well-structured source code. Your header file shall be named sDeque.h Your part a implementations need not catch and throw exceptions, but part b shall do this. Your code must be reasonably efficient as well (e.g., it must NOT be quadratic in the worst case).
* Makefile that compiles your code with our sDeque\_main.cpp on the CISE linux machines and produces executable sDeque. We will provide the main file. The program must run on CISE linux machines, so be sure to test it there (use remote access as required).
* PDF document containing a description of code organization, any special diagnostics, test cases and test results, as well as known bugs. This shall be named according to usual convention.

**Interface**

class Deque{

private:

 // your private stuff to implement the deque

public:

 //Constructor

 Deque() {

 }

 //Destructor

 ~Deque() {

 }

 // Inserts the element at the front of the queue.

 void push\_front(std::string item) {

 }

 // Inserts the element at the back of the queue

 void push\_back(std::string item) {

 }

 // Removes and returns the element at the front of the queue.

 std::string pop\_front() {

 }

 // Removes and returns the element at the back of the queue.

 std::string pop\_back() {

 }

 // Returns the number of elements in the queue.

 int size() {

 }

 // Tells whether the queue is empty or not.

 bool empty() {

 }

 /\* Puts the contents of the queue from front to back into a

retrun string with each string item followed by a newline

 \*/

 std::string toStr() {

 }

};

You may want to also provide helper methods (of course), as well as other public methods that are not included in this interface in order to help you develop/diagnose/debug your class. For example, you may want a public (for now) method that allows you to print out all the array contents with their indices from a start index through an end index. Obviously, this will not be part of the Deque ADT, but you may find it useful to see what is going on.

**Grading Rubric for Part a**

|  |  |
| --- | --- |
| **Criterion** | **Points** |
| Code compiles without error from makefile | 5 |
| Code efficiently gives correct answers to test cases | 20 |
| Testing (Writing appropriate test cases and testing the code) | 5 |
| Proper code comments | 5 |
| Good code documentation (structure, how to use) | 5 |
| Following good programming practices | 5 |
| Total | 45 |

Remember, ALL of your submissions MUST compile on the CISE department machines using your makefile. You may develop your code on your own system/environment, but upload it to your CISE account and test it on the CISE machines well before you submit it. Code that does not compile on the CISE machines from the makefile WILL BE RETURNED ungraded and points (see rubric) will be deducted, as we will NOT debug or port either you code or your makefile.