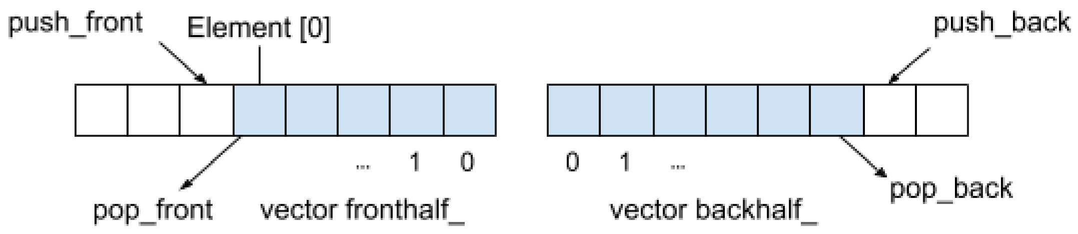
Project 3: Double-ended Queue—deque<T>(“deck”)

CPSC 131 Fall 2018

Introduction

The Standard Library provides stack and queue classes (std::stack<T> and std::queue<T>), but they are actually adapters (a kind of wrapper) for the Standard Library’s deque class (std::deque<T>). Deques allow you to add items or remove them from either end of the deque (e.g., push_back, push_front, pop_back, and pop_front).
Most Standard Library implementations of std::deque<T> use two back-to-back vectors, a fronthalf vector that you call push_back on when you need to push to the front of the deque, and a backhalf vector that you call push_back on when you need to push to the back of the queue.



minideque<T>	std::vector<T> fronthalf	std::vector<T> backhalf
push_front(value)	fronthalf.push_back(value)	
push_back(value)		backhalf.push_back(value)
pop_front(value)	fronthalf.pop_back(value) *	
pop_back()		backhalf.pop_back() *
front()	usually calls fronthalf.back() -- if fronthalf is empty, will have to call backhalf.front()	
back()		usually calls backhalf.back() -- if backhalf is empty, will have to call fronthalf.front()
operator[](i)	returns the i-th element of the deque, could be in fronthalf or backhalf. Note that element 0 is the “top” element of frontvector.	

The logic needs some adjusting if pop_back is called when the backhalf vector is empty, or if pop_front is called when the fronthalf vector is empty. In those cases, you will want to **re-balance the values between the two vectors,** making sure you keep the queue items in order - move half the values from fronthalf to backhalf, or vice-versa. It will take a little time to get this part to work correctly. Do not expect it to take 5-10 minutes.

Objective

You are given a partial implementation of a “mini-deque” class. `minideque<T>` is a templated class that holds the deque. Unlike the actual std::deque<T> class, minideque will **NOT** have the iterator classes (iterator, const_iterator, reverse_iterator, and const_reverse_iterator). This means you will also **NOT** have to provide the iterator-based functions: erase, begin and end.

For this project, you should use the std::vector class from the C++ standard library.

Complete the implementation of class `minideque<T>`, adding public/private member variables and functions as needed. Your code is tested in the provided `main.cpp.`

Source Code Files

You are given “skeleton” code files with declarations that may be incomplete and without any implementation. Implement the code and ensure that all the tests in `main.cpp` pass successfully.

`minideque.h`: This is to be completed
main.cpp: This calls a test function (in minideque.h) that tests the output of your minideque<T> class. You may wish to add additional tests, based on the ones already provided.
`README.md`: You must edit this file to include the name and CSUF email of each student in your group. Do this even if you are working by yourself. This information will be used so that we can enter your grades into Titanium.

Hints

As you start implementing the `minideque<T>` class, comment out the tests in main.cpp that you haven’t implemented yet in your class. As you fill in the code for your class, uncomment the matching tests to make sure your class is working. Keep testing your code as you implement it. It is much easier to debug one method in your class than all of the methods at the same time.

Do not wait till the very end to test your code.
When you complete your project, all of the tests should run correctly. Expected output of running the program is provided below.

Speak to your teachers if you need help designing your approach, or are having trouble compiling or debugging your code. const errors can frequently prevent your code from compiling. Note also that it is possible to be 95% of the way finished, but have the impression you are miles away. Your instructor can help you get over that final 5% if you need help. Don’t hesitate to ask.

EXPECTED OUTPUT:

```
PASSED dq[0] == dq.front(): expected and received 1
PASSED dq[0] == 9: expected and received 9
PASSED dq.front() == 1: expected and received 1
PASSED dq.back() == 9: expected and received 9
PASSED dq.front() == dq.push_front(el): expected and received 2
PASSED dq.front() == dq.push_front(el): expected and received 3
PASSED dq.front() == dq.push_front(el): expected and received 4
PASSED dq.front() == dq.push_front(el): expected and received 5
PASSED dq.front() == dq.push_front(el): expected and received 6
PASSED dq.front() == dq.push_front(el): expected and received 7
PASSED dq.front() == dq.push_front(el): expected and received 8
PASSED dq.back() == dq.push_back(el): expected and received 10
PASSED dq.back() == dq.push_back(el): expected and received 11
PASSED dq.back() == dq.push_back(el): expected and received 12
PASSED dq.back() == dq.push_back(el): expected and received 13
PASSED dq.back() == dq.push_back(el): expected and received 14
PASSED dq.back() == dq.push_back(el): expected and received 15
PASSED dq.back() == dq.push_back(el): expected and received 16
PASSED dq.back() == dq.push_back(el): expected and received 17
PASSED dq.back() == dq.push_back(el): expected and received 18
PASSED dq.back() == dq.push_back(el): expected and received 19
PASSED assign to array index: expected and received 9999
PASSED read from array index: expected and received 9999

clearing the minideque...
NOTE: the minideque keeps REBALANCING the front/back to have similar # entries
dq.pop_front() ==> 7 6 5 4 3 2 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:7,
back:19, size:18
dq.pop_front() ==> 6 5 4 3 2 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:6,
back:19, size:17
dq.pop_front() ==> 5 4 3 2 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:5, back:19,
size:16
dq.pop_front() ==> 4 3 2 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:4, back:19,
size:15
dq.pop_front() ==> 3 2 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:3, back:19,
size:14
dq.pop_front() ==> 2 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:2, back:19,
size:13
dq.pop_front() ==> 1 | 9 10 11 12 13 14 15 16 17 18 19 , front:1, back:19, size:12
dq.pop_front() ==> | 9 10 11 12 13 14 15 16 17 18 19 , front:9, back:19, size:11
dq.pop_front() ==> 10 11 12 13 14 | 15 16 17 18 19 , front:10, back:19, size:10
PASSED rebalancing fronthalf and backhalf vectors: expected and received 5
dq.pop_front() ==> 11 12 13 14 | 15 16 17 18 19 , front:11, back:19, size:9
dq.pop_front() ==> 12 13 14 | 15 16 17 18 19 , front:12, back:19, size:8
dq.pop_front() ==> 13 14 | 15 16 17 18 19 , front:13, back:19, size:7
dq.pop_front() ==> 14 | 15 16 17 18 19 , front:14, back:19, size:6
dq.pop_front() ==> | 15 16 17 18 19 , front:15, back:19, size:5
dq.pop_front() ==> 16 17 | 18 19 , front:16, back:19, size:4
PASSED rebalancing fronthalf and backhalf vectors: expected and received 2
dq.pop_front() ==> 17 | 18 19 , front:17, back:19, size:3
dq.pop_front() ==> | 18 19 , front:18, back:19, size:2
dq.pop_front() ==> | 19 , front:19, back:19, size:1
dq.pop_front() ==> minideque is empty

Passed: 25 out of: 25 total tests.
            ...done.
Program ended with exit code: 0
```

Obtaining and submitting code

Click the assignment link to fork your own copy of the skeleton code to your PC. One student from a group clicks on the link below and forms a new team. The team name must begin with your section number (e.g., 2-Brians-team). This student then invites his/her project partner as an "Outside collaborator" in the settings menu.

Do not fork your repository to your personal github account. Your code should have a URL like https://github.com/CSUF-CPSC-131-Fall2018/project3-2-brians-team, NOT https://github.com/brian/project3-2-brians-team.

https://classroom.github.com/g/Ee6Y8QM5

Here is a video that shows the steps: https://www.youtube.com/watch?v=-52quDR2QSc
Then edit your code locally as you develop it. As you make progress, commit and push your changes to your repository regularly. This ensures that your work is backed up, and that you will receive credit for making a submission.

Development environment

The test platform is Linux with the g++ -std=c++14 compiler. For this reason, the recommended development platform is Linux.

Linux environment

To attempt to compile the test program, use the following command:

```
$ clang++ -g -std=c++14 *.cpp -o test
```

To attempt to run the compiled test program, use the following command:

```
$ ./test
```

Grading rubric

Your grade will be comprised of two parts, *Form* and *Function*.

Function refers to whether your code works properly as tested by the main function (80%).
Form refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is October 28th at 11:55pm.
You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline.

Your code must compile/build for it to be tested and graded. If you only complete part of the project, make sure that it compiles before submitting.