You may not change the way the tests are written. E.g. You may not modify the tests so that a new function you create is called in the test case. Do not create a main function. The testing framework will generate it automatically.

You may use anything in the C++ STL (other than the deque class) to complete the project. However, you must back your deck with a doubly linked list you've created. I.e., you may not use a vector or list as the backing structure for your deque.

Interface Details

T refers to the type specialization of the templated deque.

Method	Description
`deque()`	Constructs a deque object, initializing necessary member variables, etc. The deque has no maximum size: the size if only limited by the amount of memory in your computer.
`void push_front(T data)`	Adds *data* to the front of the deque.
`void push_back(T data)`	Adds *data* to the back of the deque.
`void pop_front()`	Removes the item at the front of the deque. Throws an std::runtime_error if the deque is empty.
`void pop_back()`	Removes the item at the back of the deque. Throws an std::runtime_error if the deque is empty.
`T front()`	Returns (by value) the item at the front of the deque. Throws an std::runtime_error if the deque is empty.
`T back()`	Returns (by value) the item at the back of the deque. Throws an std::runtime_error if the deque is empty.
`size_t size()`	Returns the size of the deque().
`bool empty()`	Return true if the deque is empty and false otherwise.

Here is a short description of the files provided:

• deque.h - Contains the declaration and definition of the deque class and its member functions.

• catch.hpp – Catch framework header file. Do not modify this file.

• catch_main.cpp – Separate compilation unit for the catch framework to speed up the compilation process. Do not modify this file.

• tests.cpp – Contains sample Catch unit tests. You may add your own test cases to this file.

• makefile – Contains rules for compiling the project. Do not modify this file.

Testing To test your implementation, we will use the Catch testing framework. Sample test cases are provided to show how your class will be used and to show how Catch tests are written so you can write your own. A basic catch test has the following format:

```
TEST_CASE("Test case name", "[Test case tags]") {
    // test case setup
    // test case assertions
    // test case tear down
}
```

For example, a basic deque unit test may look like the following:

```
TEST_CASE("Test push_front", "[deque]") {
    deque<int> dq;
    dq.push_front(1);
    dq.push_front(2);
    CHECK(dq.back() == 1);
    CHECK(dq.front() == 2);
}
```

Compile & Run the Project Locally To compile the project, we will use make, a tool for controlling the generation of executables and nonsource files from the project's source files. For our purposes, you don't need to know the details of make as we will provide a makefile for you, but if you're curious here's a quick primer.

Make is made up of rules and targets, which are placed in a file aptly named "makefile". Rules take the following basic form:

```
target: dependencies
        commands
```

target is the name of the executable or file that will be created when the rule finishes running.

dependencies are a list of files on which the target depends. This allows make to skip executing a rule if the timestamp of the target is newer than the timestamps of all the dependencies.

commands is a list of commands (such as g++) that will be used to generate the target from the dependencies.

To compile the project, open your terminal and use the cd command to browse to the folder containing the project source files and makefile. Note that on the Windows Subsystem for Linux you can browse to your Windows user directory by cding to /mnt/c/<windows_user_name>. Once inside the project folder, run make to compile the project. The first time you run this it may take a few seconds, since Catch takes a long time to compile. Subsequent runs should be much faster since catch will not need to compile again.

Once the project is compiled, run the project with the command ./build/test_deque. The output will look like the following by default:

```
joe@JOE-SURFACE:/mnt/c/Users/josep/Documents/git/S20-PA0/skeleton$ ./build/test_deque

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

test_deque is a Catch v2.11.0 host application.
Run with -? for options

-------------------------------------------------------------------------------
Push front
-------------------------------------------------------------------------------
deque_tests.cpp:6
...............................................................................

deque_tests.cpp:9: FAILED:
  CHECK( dq.front() == 1 )
with expansion:
  -878869830 == 1

deque_tests.cpp:10: FAILED:
  CHECK( dq.back() == 1 )
with expansion:
  -878869830 == 1

deque_tests.cpp:12: FAILED:
  CHECK( dq.front() == 2 )
with expansion:
  -878869830 == 2

deque_tests.cpp:13: FAILED:
  CHECK( dq.back() == 1 )
with expansion:
```