```cpp
/*Copyright 2017 Sam Pickell*/
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include <string>
#include <exception>
#include "RingBuffer.hpp"

BOOST_AUTO_TEST_CASE(RBcontructor) {
        // normal constructor
        BOOST_REQUIRE_NO_THROW(RingBuffer(100));

        // this should fail
        BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
        BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
        RingBuffer rb(3);

        rb.enqueue(2);
        rb.enqueue(1);
        rb.enqueue(0);

        BOOST_REQUIRE(rb.dequeue() == 2);
        BOOST_REQUIRE(rb.dequeue() == 1);
        BOOST_REQUIRE(rb.dequeue() == 0);

        BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(RBpeek) {
        RingBuffer rb(3);

        rb.enqueue(42);
        rb.enqueue(100);
        rb.enqueue(9001);

        // Can’t enqueue full buffer
        BOOST_REQUIRE_THROW(rb.enqueue(1), std::runtime_error);

        // Check peek
        BOOST_REQUIRE(rb.peek() == 42);

        BOOST_REQUIRE(rb.dequeue() == 42);
        BOOST_REQUIRE(rb.dequeue() == 100);
        BOOST_REQUIRE(rb.dequeue() == 9001);

        BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
        // Can’t peek into an empty buffer
        BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error);
}
```

```cpp
/* Copyright 2017 Sam Pickell */
#ifndef RINGBUFFER_HPP
#define RINGBUFFER_HPP

#include <stdint.h>
#include <stdexcept>
#include <iostream>
#include <vector>

class RingBuffer {
 public:
  RingBuffer();
  explicit RingBuffer(int u_capacity);
  ~RingBuffer();

  int size() { return my_size; }
  int get_capacity() { return capacity; }
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();

 private:
  int my_size, capacity;
  std::vector<int16_t> data;
};
#endif
```

```
/*Copyright 2017 Sam Pickell*/
#include "RingBuffer.hpp"

RingBuffer::RingBuffer() {
  my_size = 0;
  capacity = 1;
}

RingBuffer::RingBuffer(int u_capacity) {
  my_size = 0;

  if (u_capacity < 1) {
       throw std::invalid_argument(
            "RB constructor: capacity must be greater than zero.");
    } else {
       capacity = u_capacity;
    }
}

RingBuffer::~RingBuffer() {}

bool RingBuffer::isEmpty() {
return (my_size == 0);
}

bool RingBuffer::isFull() {
return (my_size == capacity);
}

void RingBuffer::enqueue(int16_t x) {
  if (this->isFull()) {
      throw std::runtime_error("enqueue: can't enqueue to a full ring.");
    } else {
      data.push_back(x);
      my_size++;
    }
}

int16_t RingBuffer::dequeue() {
  int16_t temp;
  if (this->isEmpty()) {
      throw std::runtime_error("dequeue: nothing to dequeue, empty.");
    } else {
      temp = data.front();
      data.erase(data.begin());
      my_size--;
    }
  return temp;
}

int16_t RingBuffer::peek() {
  if (this->isEmpty()) {
      throw std::runtime_error("peek: nothing to see, empty.");
    }
  return data.front();
}
```

```
C=g++ -g -Wall --std=c++98 -Werror
E=.cpp
O=test.o RingBuffer.o
P=ps5a
BOOST= -lboost_unit_test_framework
SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
all: $(P)
$(P):$(O)
	$(C) -o $(P) $(O) $(BOOST) $(SFML)

$(E).o:
	$(C) -c $<  -o $@

clean:
	rm $(O) $(P)
```