```
    1: all: ps5a
    2:
    3: ps5a: test.o RingBuffer.o
    4:         g++ test.o RingBuffer.o -o ps5a -lboost_unit_test_framework
    5:
    6: ps5a.o: test.cpp RingBuffer.hpp
    7:         g++ -c test.cpp -Wall -Werror -ansi -pedantic -lboost_unit_test_fram
ework -g
    8:
    9: RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
   10:         g++ -c RingBuffer.cpp -Wall -Werror -ansi -pedantic -lboost_unit_tes
t_framework -g
   11:
   12: clean:
   13:          rm *.o ps5a *~
```

```
 1: // Copyright 2015 <Angel Z'heondre Calcano>
 2: // PS5a
 3: #define BOOST_TEST_DYN_LINK
 4: #define BOOST_TEST_MODULE Main
 5: #include <boost/test/unit_test.hpp>
 6: #include <stdexcept>
 7: #include <exception>
 8: #include <iostream>
 9: #include <string>
10: #include "RingBuffer.hpp"
11:
12: BOOST_AUTO_TEST_CASE(RingBufferconsrtuct) {
13:   BOOST_REQUIRE_NO_THROW(RingBuffer(100));
14:   BOOST_CHECK_THROW(RingBuffer(0), std::exception);
15:   BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
16: }
17: BOOST_AUTO_TEST_CASE(Enqueue_Dequeue_peek) {
18:   RingBuffer a(3);
19:   a.enqueue(3);
20:   a.enqueue(2);
21:   a.enqueue(1);
22:   BOOST_REQUIRE(a.peek() == 3);
23:   BOOST_REQUIRE(a.dequeue() == 3);
24:   BOOST_REQUIRE(a.dequeue() == 2);
25:   BOOST_REQUIRE(a.dequeue() == 1);
26:   BOOST_REQUIRE_THROW(a.dequeue(), std::runtime_error);
27: }
28: BOOST_AUTO_TEST_CASE(EnqueueFullbuffer) {
29:   RingBuffer b(3);
30:   b.enqueue(5);
31:   b.enqueue(9);
32:   b.enqueue(7);
33:   BOOST_REQUIRE_THROW(b.enqueue(9), std::runtime_error);
34: }
35: BOOST_AUTO_TEST_CASE(peekonemptybuffer) {
36:   RingBuffer c(1);
37:   c.enqueue(5);
38:   BOOST_REQUIRE(c.dequeue() == 5);
39:   BOOST_CHECK_THROW(c.peek(), std::runtime_error);
40: }
```

```
 1: // Copyright 2015 <Angel Z'heondre Calcano>
 2: #ifndef _RingBuffer_
 3: #define _RingBuffer_
 4: #include <stdint.h>
 5: #include <iostream>
 6: #include <string>
 7: #include <vector>
 8: #include <stdexcept>
 9:
10: class RingBuffer{
11:   int _size, _first, _last, _currentcapacity;
12:   std::vector< double > _buffer;
13:  public:
14:   explicit RingBuffer(int x): _size(x) {
15:     if ( x < 1 )
16:       throw std::invalid_argument("Constructor capacity must be > than 0");
17:     _first = _last = _currentcapacity = 0;
18:     for (int i = 0 ; i < x; i++)
19:       _buffer.push_back(100);
20:   }
21:   void RB();
22:   int size();
23:   bool isEmpty();
24:   bool isFull();
25:   void enqueue(int16_t x);
26:   int16_t dequeue();
27:   int16_t peek();
28: };
29: #endif
```

```
 1: // Copyright 2015 <Angel Z'heondre Calcano>
 2: // PS5a
 3: #include <stdint.h>
 4: #include <stdexcept>
 5: #include <iostream>
 6: #include <string>
 7: #include <vector>
 8: #include "RingBuffer.hpp"
 9:
10: int RingBuffer::size() { return _currentcapacity; }
11:
12: bool RingBuffer::isEmpty() { return _buffer.empty(); }
13:
14: bool RingBuffer::isFull() {
15:   if (_buffer.size() == (unsigned)_size)
16:     return true;
17:   if (_buffer.size() >(unsigned)_size)
18:     return false;
19:   else
20:     return false;
21: }
22:
23: void RingBuffer::enqueue(int16_t x) {
24:   if (_currentcapacity == _size)
25:     throw
26:       std::runtime_error("Can't enqueue on a full ring");
27:   _buffer[_last] = x;
28:   _last++;
29:   _currentcapacity++;
30:   if (_last == _size)
31:     _last = 0;
32: }
33:
34: int16_t RingBuffer::dequeue() {
35:   if (_currentcapacity <= 0)
36:     throw
37:       std::runtime_error(" Can't dequeue from empty ring");
38:   double _hold;
39:   _hold = _buffer[_first];
40:   _first++;
41:   _currentcapacity--;
42:   if (_first == _last) _first = _last = 0;
43:   if (_first == _size) _first = 0;
44:   return _hold;
45: }
46:
47: int16_t RingBuffer::peek() {
48:   if (_currentcapacity == 0)
49:     throw
50:       std::runtime_error("Can't peek on an empty ring");
51:   if (_buffer.empty())
52:     throw
53:       std::runtime_error("Can't peek on an empty vector array");
54:   return _buffer[_first];
55: }
```