```
 1: // Copyright 2023 Thomas O'Connor
 2: #include "RandWriter.hpp"
 3:
 4: #define BOOST_TEST_DYN_LINK
 5: #define BOOST_TEST_MODULE Main
 6: #include <boost/test/unit_test.hpp>
 7:
 8: #define TESTING_ITEM "gagggagaggcgagaaa"
 9: using std::invalid_argument;
10: using std::out_of_range;
11:
12: BOOST_AUTO_TEST_CASE(testOrderK) {
13:   RandWriter l(TESTING_ITEM, 3);
14:   BOOST_REQUIRE_EQUAL(l.orderK(), 3);
15:   RandWriter m(TESTING_ITEM, 0);
16:   BOOST_REQUIRE_EQUAL(m.orderK(), 0);
17:   RandWriter n(TESTING_ITEM, 100);
18:   BOOST_REQUIRE_EQUAL(n.orderK(), 100);
19: }
20:
21: BOOST_AUTO_TEST_CASE(testFreqKGram) {
22:   RandWriter l(TESTING_ITEM, 4);
23:   BOOST_REQUIRE_EQUAL(l.freq("gagg"), 2);
24:   BOOST_REQUIRE_NO_THROW(l.freq("gagg"));
25:   BOOST_REQUIRE_THROW(l.freq("gag"), invalid_argument);
26:   BOOST_REQUIRE_THROW(l.freq("gaggg"), invalid_argument);
27:   BOOST_REQUIRE_THROW(l.freq(" "), invalid_argument);
28:   BOOST_REQUIRE_THROW(l.freq(""), invalid_argument);
29:   RandWriter m(TESTING_ITEM, 1);
30:   BOOST_REQUIRE_EQUAL(m.freq("c"), 1);
31:   BOOST_REQUIRE_EQUAL(m.freq("g"), 9);
32:   BOOST_REQUIRE_EQUAL(m.freq("a"), 7);
33:   BOOST_REQUIRE_THROW(m.freq(" "), out_of_range);
34: }
35:
36: BOOST_AUTO_TEST_CASE(testFreqKGramC) {
37:   RandWriter l(TESTING_ITEM, 4);
38:   BOOST_REQUIRE_EQUAL(l.freq("gagg", 'c'), 1);
39:   BOOST_REQUIRE_NO_THROW(l.freq("gagg", 'c'));
40:   BOOST_REQUIRE_THROW(l.freq("gag", 'a'), invalid_argument);
41:   BOOST_REQUIRE_THROW(l.freq("gaggg", 'a'), invalid_argument);
42:   BOOST_REQUIRE_THROW(l.freq(" ", 'a'), invalid_argument);
43:   BOOST_REQUIRE_THROW(l.freq("", 'a'), invalid_argument);
44:   RandWriter m(TESTING_ITEM, 1);
45:   BOOST_REQUIRE_EQUAL(m.freq("c", 'g'), 1);
46:   BOOST_REQUIRE_EQUAL(m.freq("g", 'a'), 5);
47:   BOOST_REQUIRE_EQUAL(m.freq("a", 'g'), 5);
48:   BOOST_REQUIRE_NO_THROW(m.freq("c", 'g'));
49:   BOOST_REQUIRE_NO_THROW(m.freq("g", 'a'));
50:   BOOST_REQUIRE_NO_THROW(m.freq("a", 'g'));
51:   BOOST_REQUIRE_THROW(m.freq(" ", 'a'), out_of_range);
52:   RandWriter n(TESTING_ITEM, 0);
53:   BOOST_REQUIRE_EQUAL(n.freq("", 'c'), 1);
54:   BOOST_REQUIRE_EQUAL(n.freq("", 'g'), 9);
55:   BOOST_REQUIRE_EQUAL(n.freq("", 'a'), 7);
56: }
57:
58: BOOST_AUTO_TEST_CASE(testKRand) {
59:   RandWriter l(TESTING_ITEM, 6);
60:   BOOST_REQUIRE_EQUAL(l.kRand("gagagg"), 'c');
61:   BOOST_REQUIRE_NO_THROW(l.kRand("gagagg"));
62:   BOOST_REQUIRE_THROW(l.kRand("gag"), invalid_argument);
63:   BOOST_REQUIRE_THROW(l.kRand("gggggg"), out_of_range);
64:   RandWriter m(TESTING_ITEM, 2);
65:   BOOST_REQUIRE_EQUAL(m.kRand("cg"), 'a');
```

```
   66:   BOOST_REQUIRE_NO_THROW(m.kRand("cg"));
   67:   BOOST_REQUIRE_THROW(m.kRand("gag"), invalid_argument);
   68:   BOOST_REQUIRE_THROW(m.kRand("cc"), out_of_range);
   69: }
   70:
   71: BOOST_AUTO_TEST_CASE(testGenerate) {
   72:   RandWriter l(TESTING_ITEM, 3);
   73:   BOOST_REQUIRE_THROW(l.generate("gagagg", 4), invalid_argument);
   74:   BOOST_REQUIRE_NO_THROW(l.generate("gag", 4));
   75:   BOOST_REQUIRE_THROW(l.generate("ccc", 4), out_of_range);
   76:   RandWriter m(TESTING_ITEM, 0);
   77:   BOOST_REQUIRE_THROW(m.generate("cg", 4), invalid_argument);
   78:   BOOST_REQUIRE_NO_THROW(m.generate("", 4));
   79: }
```