```
CC = g++
CFLAGS = -Wall -Werror -pedantic --std=c++14
LIBS = -lboost_unit_test_framework
DEPS = RandWriter.h
SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio

%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<

all: test

test: test.o RandWriter.o
	$(CC) -g $(CFLAGS) -o test $^  $(LIBS) $(SFMLFLAGS)
	cpplint --filter=-runtime/references *.cpp *.h

clean:
	rm *.o test
```

```
// Copyright 2022 Anson Cheang
// Copyright 2022 Anson Cheang, and Andy Nguyen
#ifndef _HOME_IIFORCE_BADNAME_COMP4_PS6_RANDWRITER_H_
#define _HOME_IIFORCE_BADNAME_COMP4_PS6_RANDWRITER_H_

#include <iostream>
#include <string>
#include <map>

using namespace std; //NOLINT

class RandWriter {
 public:
// create a Markov model of order k from given text
RandWriter(string text, int k);  // Assume that text has length at least
k.
// ------------------------------------------------------------------
int orderK() const;  // order k of Markov model
// ------------------------------------------------------------------
// number of occurrences of kgram in text
int freq(string kgram) const;  // throw an exception if kgram is not of
// length k
// ------------------------------------------------------------------
// number of times that character c follows kgram
// if order=0, return num of times char c appears
// (throw an exception if kgram is not of length k)
int freq(string kgram, char c) const;
// ------------------------------------------------------------------
// random character following given kgram
// (Throw an exception if kgram is not of length k.
// Throw an exception if no such kgram.)
char kRand(string kgram);
// ------------------------------------------------------------------
// generate a string of length L characters
// by simulating a trajectory through the corresponding
// Markov chain. The first k characters of the newly
// generated string should be the argument kgram.
// Throw an exception if kgram is not of length k.
// Assume that L is at least k.
string generate(string kgram, int L);
// ------------------------------------------------------------------
friend ostream& operator<<(ostream& out, RandWriter& markov);
// overload the stream insertion operator and display
// the internal state of the Markov Model. Print out
// the order, the alphabet, and the frequencies of
// the k-grams and k+1-grams.

 private:
string alphabet; //NOLINT
int order;
map<string, map<char, int>> RM;
};

ostream& operator<<(ostream& out, RandWriter& markov);

#endif  // _HOME_IIFORCE_BADNAME_COMP4_PS6_RANDWRITER_H_
```

```cpp
// Copyright 2022 Anson Cheang
#include "RandWriter.h"
#include <utility>

using namespace std;  // NOLINT

// ----------------------------------------------------------------------
// create a Markov model of order k from given text
// Assume that text has length at least k.
RandWriter::RandWriter(string text, int k) {
    order = k;
    string storage = text;
    int length = text.length();
    int count = 0, l;
    char temp;
    string stringTemp, size;
    bool isThere;

    for (int i = 0; i < order; i++) {
        storage = storage + text[i];
    }
    for (int i = 0; i < length; i++) {
        temp = text.at(i);
        isThere = false;
        l = alphabet.length();
        for (int j = 0; j < l; j++) {
            if (alphabet.at(j) == temp) {
                isThere = true;
            }
        }
        if (!isThere) {
            alphabet = alphabet + temp;
        }
    }
    for (int i = 0; i < length; i++) {
        stringTemp.clear();
        stringTemp = storage.substr(i, order);
        RM.insert(make_pair(stringTemp, map<char, int>()));
        RM.at(stringTemp).insert(make_pair('\0', 0));
    }

    char placement;
    for (int i = 0; i < length; i++) {
        stringTemp.clear();
        stringTemp = storage.substr(i, order+1);
        placement = stringTemp[order];
        stringTemp.clear();
        stringTemp = storage.substr(i, order);
        RM.at(stringTemp).insert(make_pair(placement, 0));
    }
    map<char, int>::iterator p;

    for (int j = 0; j < length; j++) {
        stringTemp.clear();
        stringTemp = storage.substr(j, order);
        p = RM.at(stringTemp).find('\0');
        count = p->second;
        count++;
        RM.at(stringTemp).at('\0') = count;
    }
    for (int j = 0; j < length; j++) {
        stringTemp.clear();
        stringTemp = storage.substr(j, order+1);
        placement = stringTemp[order];
        stringTemp.clear();
        stringTemp = storage.substr(j, order);
        p = RM.at(stringTemp).find(placement);
        count = p->second;
        count++;
        RM.at(stringTemp).at(placement) = count;
    }
}
// --------------------------------------------------------------------
// order k of Markov model
int RandWriter::orderK() const {
    return order;
}
// --------------------------------------------------------------------
// number of occurrences of kgram in text
// throw an exception if kgram is not of
// length k
int RandWriter::freq(string kgram) const {
    if (kgram.length() != static_cast<unsigned int> (order)) {
        throw runtime_error("freq(string):kgram is not of length k(order)
");
    }

    map<char, int>::const_iterator p;

    p = RM.at(kgram).find('\0');

    return p->second;
}
// --------------------------------------------------------------------
// number of times that character c follows kgram
// if order=0, return num of times char c appears
// (throw an exception if kgram is not of length k)
int RandWriter::freq(string kgram, char c) const {
    if (kgram.length() != static_cast<unsigned int> (order)) {
        throw runtime_error(
            "freq(string, char):kgram is not of length k(order)");
    }

    string sub = kgram + c;

    map<char, int>::const_iterator p;
    p = RM.at(kgram).find(c);
    return p->second;
}
// --------------------------------------------------------------------
// random character following given kgram
// (Throw an exception if kgram is not of length k.
// Throw an exception if no such kgram.)
char RandWriter::kRand(string kgram) {
    if (kgram.length() != static_cast<unsigned int> (order)) {
        throw runtime_error("kRand:kgram is not of length k(order)");
    }
    map<string, map<char, int>>::iterator t;
    map<char, int>::iterator p;
    t = RM.find(kgram);
    if (t == RM.end()) {
        throw runtime_error("kRand: could not find given kgram");
    }
    int amount = freq(kgram);

    srand(static_cast<int>(time(NULL)));
    int rv = rand() % amount; //NOLINT
    p = RM.at(kgram).begin();
    p++;
    while (p != RM.at(kgram).end() && rv > p->second) {
        rv = rv - p->second;
    }
    return p->first;
}
// ------------------------------------------------------------------
// generate a string of length L characters
// by simulating a trajectory through the corresponding
// Markov chain. The first k characters of the newly
// generated string should be the argument kgram.
// Throw an exception if kgram is not of length k.
// Assume that L is at least k.
string RandWriter::generate(string kgram, int L) {
    if (kgram.length() != static_cast<unsigned int> (order)) {
        throw runtime_error("generate:kgram is not of length k(order)");
    }
    string returnString = kgram;

    for (int i = 0; i < (L - order); i++) {
        returnString = returnString + kRand(returnString.substr(i, order)
);
    }
    return returnString;
}
// ------------------------------------------------------------------
// overload the stream insertion operator and display
// the internal state of the Markov Model. Print out
// the order, the alphabet, and the frequencies of
// the k-grams and k+1-grams.
ostream& operator<<(ostream& out, RandWriter& markov) {
    return out;
}
```

```cpp
// Copyright 2022 Anson Cheang
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main

#include "RandWriter.h"

#include <iostream>
#include <sstream>
#include <string>
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(order0) {
  BOOST_REQUIRE_NO_THROW(RandWriter("gagggagaggcgagaaa", 0));

  RandWriter mm("gagggagaggcgagaaa", 0);

  BOOST_REQUIRE(mm.orderK() == 0);
  BOOST_REQUIRE(mm.freq("") == 17);
  BOOST_REQUIRE_THROW(mm.freq("t"), std::runtime_error);

  BOOST_REQUIRE(mm.freq("", 'g') == 9);
  BOOST_REQUIRE(mm.freq("", 'a') == 7);
  BOOST_REQUIRE(mm.freq("", 'c') == 1);
  BOOST_REQUIRE(mm.freq("", 'x') == 0);
}

BOOST_AUTO_TEST_CASE(order1) {
  RandWriter mm("gagggagaggcgagaaa", 1);

  BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);

  BOOST_REQUIRE(mm.freq("a") == 7);
  BOOST_REQUIRE(mm.freq("g") == 9);
  BOOST_REQUIRE(mm.freq("c") == 1);

  BOOST_REQUIRE(mm.freq("a", 'a') == 2);
  BOOST_REQUIRE(mm.freq("a", 'c') == 0);
  BOOST_REQUIRE(mm.freq("a", 'g') == 5);

  BOOST_REQUIRE_NO_THROW(mm.kRand("a"));

  BOOST_REQUIRE_THROW(mm.kRand("t"), std::runtime_error);

  BOOST_REQUIRE_THROW(mm.kRand("tt"), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(order2) {
  RandWriter mm("gagggagaggcgagaaa", 2);

  BOOST_REQUIRE_NO_THROW(mm.freq("aa"));
  BOOST_REQUIRE_THROW(mm.freq("", 'g'), std::runtime_error);

  BOOST_REQUIRE(mm.freq("aa") == 2);
  BOOST_REQUIRE(mm.freq("aa", 'a') == 1);
}
```