```
    1: all: TextGenerator mmtest
    2:
    3: mmtest: MarkovModel.o mmtest.o
    4:         g++ -o mmtest MarkovModel.o mmtest.o -lboost_unit_test_framework
    5:
    6: mmtest.o: mmtest.cpp MarkovModel.hpp
    7:         g++ -c -g mmtest.cpp -Wall -ansi -pedantic
    8:
    9: TextGenerator: TextGenerator.o MarkovModel.o
   10:         g++ -o TextGenerator TextGenerator.o MarkovModel.o -lboost_unit_test_fram
ework
   11:
   12: TextGenerator.o: TextGenerator.cpp MarkovModel.hpp
   13:         g++ -c -g TextGenerator.cpp -Wall -ansi -pedantic
   14:
   15: MarkovModel.o: MarkovModel.cpp MarkovModel.hpp
   16:         g++ -c -g MarkovModel.cpp -Wall -ansi -pedantic
   17:
   18: clean:
   19:         rm *.o TextGenerator mmtest
```

```
 1: //
 2: //  main.cpp
 3: //  ps6
 4: //
 5: //  Created by Jingxian Shi on 3/30/18.
 6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
 7: //
 8:
 9: #include <iostream>
10: #include <cstdlib>
11: #include <string>
12: #include "MarkovModel.hpp"
13:
14: int main(int argc, const char * argv[]) {
15:     if (argc != 3) {
16:         std::cout << "Wrong number of arguments" << std::endl;
17:         return -1;
18:     }
19:     int k = atoi(argv[1]);
20:     int T = atoi(argv[2]);
21:     std::string text = "";
22:     std::string current;
23:     while ((std::cin >> current)) {
24:         text += " " + current;
25:         current = "";
26:     }
27:     MarkovModel mm(text, k);
28:     std::cout << mm.gen(text.substr(0, k), T) << std::endl;
29:     std::cout << mm << std::endl;
30:     return 0;
31: }
```

```
    1: #include <iostream>
    2: #include <string>
    3: #include <exception>
    4: #include <stdexcept>
    5:
    6: #include "MarkovModel.hpp"
    7:
    8: #define BOOST_TEST_DYN_LINK
    9: #define BOOST_TEST_MODULE Main
   10: #include <boost/test/unit_test.hpp>
   11:
   12: using namespace std;
   13:
   14: BOOST_AUTO_TEST_CASE(order0) {
   15:     // normal constructor
   16:     BOOST_REQUIRE_NO_THROW(MarkovModel("gagggagaggcgagaaa", 0));
   17:
   18:     MarkovModel mm("gagggagaggcgagaaa", 0);
   19:
   20:     BOOST_REQUIRE(mm.order() == 0);
   21:     BOOST_REQUIRE(mm.freq("") == 17); // length of input in constructor
   22:     BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);
   23:
   24:     BOOST_REQUIRE(mm.freq("", 'g') == 9);
   25:     BOOST_REQUIRE(mm.freq("", 'a') == 7);
   26:     BOOST_REQUIRE(mm.freq("", 'c') == 1);
   27:     BOOST_REQUIRE(mm.freq("", 'x') == 0);
   28:
   29: }
   30:
   31: BOOST_AUTO_TEST_CASE(order1) {
   32:     // normal constructor
   33:     BOOST_REQUIRE_NO_THROW(MarkovModel("gagggagaggcgagaaa", 1));
   34:
   35:     MarkovModel mm("gagggagaggcgagaaa", 1);
   36:
   37:     BOOST_REQUIRE(mm.order() == 1);
   38:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
   39:     BOOST_REQUIRE_THROW(mm.freq("xx"), std::runtime_error);
   40:
   41:     BOOST_REQUIRE(mm.freq("g") == 9);
   42:     BOOST_REQUIRE(mm.freq("a") == 7);
   43:     BOOST_REQUIRE(mm.freq("c") == 1);
   44:
   45:     BOOST_REQUIRE(mm.freq("a", 'a') == 2);
   46:     BOOST_REQUIRE(mm.freq("a", 'c') == 0);
   47:     BOOST_REQUIRE(mm.freq("a", 'g') == 5);
   48:
   49:     BOOST_REQUIRE(mm.freq("c", 'a') == 0);
   50:     BOOST_REQUIRE(mm.freq("c", 'c') == 0);
   51:     BOOST_REQUIRE(mm.freq("c", 'g') == 1);
   52:
   53:     BOOST_REQUIRE(mm.freq("g", 'a') == 5);
   54:     BOOST_REQUIRE(mm.freq("g", 'c') == 1);
   55:     BOOST_REQUIRE(mm.freq("g", 'g') == 3);
   56:
   57:     BOOST_REQUIRE_NO_THROW(mm.randk("a"));
   58:     BOOST_REQUIRE_NO_THROW(mm.randk("c"));
   59:     BOOST_REQUIRE_NO_THROW(mm.randk("g"));
   60:
   61:     BOOST_REQUIRE_THROW(mm.randk("x"), std::runtime_error);
   62:
   63:     BOOST_REQUIRE_THROW(mm.randk("xx"), std::runtime_error);
   64:
   65: }
```

```
   66:
   67: BOOST_AUTO_TEST_CASE(order2) {
   68:     // normal constructor
   69:     BOOST_REQUIRE_NO_THROW(MarkovModel("gagggagaggcgagaaa", 2));
   70:
   71:     MarkovModel mm("gagggagaggcgagaaa", 2);
   72:
   73:     BOOST_REQUIRE(mm.order() == 2);
   74:
   75:     BOOST_REQUIRE_THROW(mm.freq(""), std::runtime_error);
   76:     BOOST_REQUIRE_THROW(mm.freq("x"), std::runtime_error);
   77:     BOOST_REQUIRE_NO_THROW(mm.freq("xx"));
   78:     BOOST_REQUIRE_THROW(mm.freq("", 'g'), std::runtime_error); // kgram is wrong
length
   79:     BOOST_REQUIRE_THROW(mm.freq("x", 'g'), std::runtime_error); // kgram is wrong
 length
   80:     BOOST_REQUIRE_THROW(mm.freq("xxx", 'g'), std::runtime_error); // kgram is wro
ng length
   81:
   82:
   83:     BOOST_REQUIRE(mm.freq("aa") == 2);
   84:     BOOST_REQUIRE(mm.freq("aa", 'a') == 1);
   85:     BOOST_REQUIRE(mm.freq("aa", 'c') == 0);
   86:     BOOST_REQUIRE(mm.freq("aa", 'g') == 1);
   87:
   88:     BOOST_REQUIRE(mm.freq("ag") == 5);
   89:     BOOST_REQUIRE(mm.freq("ag", 'a') == 3);
   90:     BOOST_REQUIRE(mm.freq("ag", 'c') == 0);
   91:     BOOST_REQUIRE(mm.freq("ag", 'g') == 2);
   92:
   93:     BOOST_REQUIRE(mm.freq("cg") == 1);
   94:     BOOST_REQUIRE(mm.freq("cg", 'a') == 1);
   95:     BOOST_REQUIRE(mm.freq("cg", 'c') == 0);
   96:     BOOST_REQUIRE(mm.freq("cg", 'g') == 0);
   97:
   98:     BOOST_REQUIRE(mm.freq("ga") == 5);
   99:     BOOST_REQUIRE(mm.freq("ga", 'a') == 1);
  100:     BOOST_REQUIRE(mm.freq("ga", 'c') == 0);
  101:     BOOST_REQUIRE(mm.freq("ga", 'g') == 4);
  102:
  103:     BOOST_REQUIRE(mm.freq("gc") == 1);
  104:     BOOST_REQUIRE(mm.freq("gc", 'a') == 0);
  105:     BOOST_REQUIRE(mm.freq("gc", 'c') == 0);
  106:     BOOST_REQUIRE(mm.freq("gc", 'g') == 1);
  107:
  108:     BOOST_REQUIRE(mm.freq("gg") == 3);
  109:     BOOST_REQUIRE(mm.freq("gg", 'a') == 1);
  110:     BOOST_REQUIRE(mm.freq("gg", 'c') == 1);
  111:     BOOST_REQUIRE(mm.freq("gg", 'g') == 1);
  112:
  113: }
```

```
    1: //
    2: //  MarkovModel.cpp
    3: //  ps6
    4: //
    5: //  Created by Jingxian Shi on 4/2/18.
    6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
    7: //
    8: #include "MarkovModel.hpp"
    9: #include <stdlib.h>
   10: #include <iostream>
   11: #include <string>
   12: #include <vector>
   13: #include <map>
   14: #include <exception>
   15: #include <stdexcept>
   16:
   17:
   18: MarkovModel::MarkovModel(std::string text, int k) {
   19:     _order = k;
   20:     _alphabet = "";
   21:     _length = text.size();
   22:     std::string temp = "";
   23:     temp = text + text.substr(0, k);
   24:     for (int i = 0; i < temp.size() - k; i++) {
   25:         std::string kgram = temp.substr(i, k);
   26:         _kgrams[kgram] += 1;
   27:     }
   28:     temp = text + text.substr(0, k+1);
   29:     for (int i = 0; i < temp.size() - k - 1; i++) {
   30:         std::string kp1gram = temp.substr(i, k+1);
   31:         _kgrams[kp1gram] += 1;
   32:         if (static_cast<int>(_alphabet.find(temp[i])) == -1)
   33:             _alphabet += temp[i];
   34:     }
   35: }
   36:
   37: MarkovModel::~MarkovModel() {
   38: }
   39:
   40: int MarkovModel::order() {
   41:     return _order;
   42: }
   43:
   44: int MarkovModel::freq(std::string kgram) {
   45:     if (kgram.size() != _order) {
   46:         throw std::runtime_error("kgram is not of length k");
   47:     }
   48:     if (_order == 0) {
   49:         return _length;
   50:     }
   51:     return _kgrams[kgram];
   52: }
   53:
   54: int MarkovModel::freq(std::string kgram, char c) {
   55:     if (kgram.size() != _order) {
   56:         throw std::runtime_error("kgram is not of length k");
   57:     }
   58:     std::string temp = kgram;
   59:     temp += c;
   60:     if (_kgrams.count(temp) == 0) {
   61:         return 0;
   62:     }
   63:     return _kgrams[temp];
   64: }
   65:
```

```
 66: char MarkovModel::randk(std::string kgram) {
 67:     if (kgram.size() != _order) {
 68:         throw std::runtime_error("kgram is not of length k");
 69:     }
 70:     if (_kgrams.count(kgram) == 0) {
 71:         throw std::runtime_error("no such kgram");
 72:     }
 73:     std::map<std::string, int> kgrams1;
 74:     for (int i = 0; i < _alphabet.size(); i++) {
 75:         std::string kgram1 = kgram + _alphabet[i];
 76:         kgrams1[kgram1] = freq(kgram, _alphabet[i]);
 77:     }
 78:     int freq_sum = 0;
 79:     std::map <std::string, int> sum;
 80:     std::map <std::string, int>::iterator p;
 81:     for (p = kgrams1.begin(); p != kgrams1.end(); p++) {
 82:         freq_sum += p->second;
 83:         sum[p->first] = freq_sum;
 84:     }
 85:     std::map <int, std::string> sort;
 86:     for (p = sum.begin(); p != sum.end(); p++)
 87:         sort[p->second] = p->first;
 88:     char c;
 89:     unsigned int seed = 1;
 90:     int rand_num = rand_r(&seed) % freq(kgram);
 91:     std::string temp;
 92:     std::map <int, std::string>::iterator p1;
 93:     for (p1 = sort.begin(); p1 != sort.end(); p1++) {
 94:         if (p1->first > rand_num) {
 95:             temp = p1->second;
 96:             break;
 97:         }
 98:     }
 99:     c = temp[_order];
100:     return c;
101: }
102:
103: std::string MarkovModel::gen(std::string kgram, int T) {
104:     if (kgram.size() != _order) {
105:         throw std::runtime_error("kgram is not of length k");
106:     }
107:     std::string temp = kgram;
108:     std::string current = kgram;
109:     for (int i = 0; i < T; i++) {
110:         char c = randk(current);
111:         temp += c;
112:         current.erase(0, 1);
113:         current += c;
114:     }
115:     return temp;
116: }
117:
118: std::ostream& operator<< (std::ostream &out, MarkovModel &mm) {
119:     out << "kgram order: " << mm._order << std::endl;
120:     out << "Alphabet: " << mm._alphabet << std::endl;
121:     out << "K-gram | Frequency" << std::endl;
122:     std::map<std::string, int>::iterator p;
123:     for (p = mm._kgrams.begin(); p != mm._kgrams.end(); p++)
124:     std::cout << p->first << " | " << p->second << std::endl;
125:     return out;
126: }
```

```
 1: //
 2: //  MarkovModel.hpp
 3: //  ps6
 4: //
 5: //  Created by Jingxian Shi on 4/2/18.
 6: //  Copyright Â© 2018 Jingxian Shi. All rights reserved.
 7: //
 8:
 9: #ifndef MarkovModel_hpp
10: #define MarkovModel_hpp
11:
12: #include <stdio.h>
13: #include <string>
14: #include <map>
15:
16: class MarkovModel {
17:  public:
18:     MarkovModel(std::string text, int k);
19:     ~MarkovModel();
20:     int order();
21:     int freq(std::string kgram);
22:     int freq(std::string kgram, char c);
23:     char randk(std::string kgram);
24:     std::string gen(std::string kgram, int T);
25:     friend std::ostream& operator<< (std::ostream &out, MarkovModel &mm);
26:  private:
27:     int _order;
28:     int _length;
29:     std::map <std::string, int> _kgrams;
30:     std::string _alphabet;
31: };
32: #endif /* MarkovModel_hpp */
```