PS6 Random Writer

For this assignment, we had to create a program to construct a Markov chain that can analyze k-grams (a fixed number of characters) and keep track of the probability of each character's occurrence. The program would then produce text using any input text and a Markov chain of given order k.

Key concepts

The RandWriter class utilizes a map with a kgram key and a map of characters and their frequency. The Mersenne Twister random number generator is also used by RandWriter in the kRand() function. RandWriter constructor creates a map that may be shown as a table-like output, containing each kgram and its frequency, as well as probability for each subsequent letter. When generating a new string from of previously produced characters, the function generate() employs the helper function kRand(), which picks a random next character from a kgram string.

What I Accomplished

Using probabilistic analysis on text to determine the next character/s in a sequence of length k words called kgrams is the Markov Model's name of the game. With the input of a string and order k the RandWriter class maps each of the kgrams in the string to it's following character and frequency. With the given kgram it can then generate a new string based on the probability of each of the following characters. Using this function, the TextWriter class is able to analyze words in text file and generate a pseudorandom string of L length.

What I Learned

Prior to beginning this project, I took it upon myself to absorb more knowledge about how Markov chains are commonly used in systems like online search engines, information retrieval, speech recognition, and gene prediction. I was able to observe firsthand how the Markov model can generate decent text using a trajectory through a table of probabilities of k-grams throughout the assignment.

```
 1: CC= g++
 2: CFLAGS= -g -Wall -std=c++0x -pedantic
 3: Boost= -lboost_unit_test_framework
 4:
 5: all:
 6:         make TextWriter TestWriter
 7:
 8: TextWriter:  RandWriter.o TextWriter.o
 9:         $(CC) -o TextWriter TextWriter.o RandWriter.o $(CFLAGS)
10:
11: TestWriter: RandWriter.o Test.o
12:         $(CC) -o TestWriter RandWriter.o Test.o $(Boost)
13:
14: Test.o: RandWriter.h Test.cpp
15:         $(CC) -c Test.cpp $(CFLAGS)
16:
17: TextWriter.o: RandWriter.h TextWriter.cpp
18:         $(CC) -c TextWriter.cpp $(CFLAGS)
19:
20: RandWriter.o: RandWriter.h RandWriter.cpp
21:         $(CC) -c RandWriter.cpp $(CFLAGS)
22:
23: clean:
24:         rm -f *.o *~ TextWriter TestWriter
```

```
 1: // Copyright 2022 Matthew Lorette Anaya
 2:
 3: #include "RandWriter.h"
 4: #include <fstream>
 5:
 6: int main(int argc, char *argv[]) {
 7:     if (argc != 3) {
 8:         std::cerr << "Usage: ./TextWriter k L < input.txt" << std::endl;
 9:         exit(-1);
10:     }
11:     int k = std::atoi(argv[1]);
12:     int L = std::atoi(argv[2]);
13:
14:     int count = 0;
15:     int length = 0;
16:     std::string input;
17:     std::string output;
18:
19:     // read input line by line and generate pseudo-random text
20:     while (std::getline(std::cin, input) && count < L) {
21:         if (input.length() > static_cast<unsigned int>(k)) {
22:             try {
23:                 RandWriter rw(input, k);
24:                 if (static_cast<int>(input.length()) > L) {
25:                     length = L;
26:                 } else if (static_cast<int>(input.length()) + count > L)
{
27:                     length = L - count;
28:                 } else {
29:                     length = input.length();
30:                 }
31:                 output = rw.generate(input.substr(0, k), length);
32:                 count += output.length();
33:                 std::cout << output << std::endl;
34:             }
35:             catch (std::invalid_argument err) {
36:                 std::cerr << err.what() << std::endl;
37:                 exit(-1);
38:             }
39:             catch (std::runtime_error err) {
40:                 std::cerr << err.what() << std::endl;
41:                 exit(-1);
42:             }
43:         }
44:     }
45:
46:     return 0;
47: }
```

```
 1: // Copyright 2022 Matthew Lorette Anaya
 2: #ifndef RANDWRITER_H
 3: #define RANDWRITER_H
 4:
 5: #include <iostream>
 6: #include <string>
 7: #include <map>
 8: #include <exception>
 9: #include <utility>
10: #include <random>
11:
12: class RandWriter {
13:  private:
14:
15:     int rw_k;
16:     std::string rw_txt;
17:     std::map<std::string, std::map<char, int>> rw_table;
18:
19:  public:
20:
21:     RandWriter(std::string text, int k);
22:
23:     int orderK() const;
24:     int freq(std::string kgram) const;
25:     int freq(std::string kgram, char c) const;
26:
27:     std::string getText() const;
28:     std::string generate(std::string kgram, int L) const;
29:
30:     std::map<std::string, std::map<char, int>> get_table() const;
31:
32:     char kRand(std::string kgram) const;
33:
34:     friend std::ostream& operator<<(std::ostream& out, const RandWriter&
35:     rw);
36:
37: };
38: #endif
```

```
    1: // Copyright 2022 Matthew Lorette Anaya
    2:
    3: #include "RandWriter.h"
    4:
    5: RandWriter::RandWriter(std::string text, int k) {
    6:    rw_txt = text;
    7:     rw_k = k;
    8:
    9:     if (rw_txt.length() < static_cast<unsigned int>(rw_k)) {
   10:         throw std::invalid_argument("RandWriter(string text, int k): orde
r k"
   11:         " must be less than or equal to text length.");
   12:     }
   13:
   14:     // Table setup
   15:     unsigned int pos = 0;
   16:     for (unsigned int i = 0; i < rw_txt.length(); i++) {
   17:         std::string kgram;
   18:         std::map<char, int> freq_table;
   19:
   20:         // kgrams parsing
   21:         for (unsigned int j = i; j < i + rw_k; j++) {
   22:             if (j >= rw_txt.length()) {
   23:                 pos = j - rw_txt.length();
   24:             } else {
   25:                 pos = j;
   26:             }
   27:             kgram += rw_txt.at(pos);
   28:         }
   29:
   30:         // Frequency table setup
   31:         pos++;
   32:         if (pos >= rw_txt.length()) { pos -= rw_txt.length(); }
   33:         freq_table.insert(std::make_pair(rw_txt.at(pos), 0));
   34:
   35:         // Mapping
   36:         if (rw_table.count(kgram) == 0) {
   37:             rw_table.insert(std::make_pair(kgram, freq_table));
   38:         }
   39:
   40:         rw_table[kgram][rw_txt.at(pos)]++;
   41:     }
   42: }
   43:
   44: int RandWriter::orderK() const {
   45:         return rw_k;
   46: }
   47:
   48: std::string RandWriter::getText() const {
   49:         return rw_txt;
   50: }
   51:
   52: std::map<std::string, std::map<char, int>> RandWriter::get_table() const
{
   53:     return rw_table;
   54: }
   55:
   56: int RandWriter::freq(std::string kgram) const {
   57:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
   58:         throw std::runtime_error("freq(string kgram): kgram must be of"
   59:         " length greater than or equal to order k.");
   60:     }
   61:     int count = 0;
   62:     for (unsigned int i = 0; i < rw_txt.length(); i++) {
   63:         unsigned int pos = 0;
```

```
   64:         std::string kg;
   65:         // parse input text for kgrams
   66:         for (unsigned int j = i; j < i + rw_k; j++) {
   67:             // get characters for kgrams
   68:             if (j >= rw_txt.length()) {
   69:                 pos = j - rw_txt.length();
   70:             } else {
   71:                 pos = j;
   72:             }
   73:             kg += rw_txt.at(pos);
   74:         }
   75:         if (kgram == kg) { count++; }
   76:     }
   77:     return count;
   78: }
   79:
   80: int RandWriter::freq(std::string kgram, char c) const {
   81:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
   82:         throw std::runtime_error("freq(string kgram, char c): kgram must
be"
   83:         " of length greater than or equal to order k.");
   84:     }
   85:     return rw_table.at(kgram).at(c);
   86: }
   87:
   88: char RandWriter::kRand(std::string kgram) const {
   89:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
   90:         throw std::runtime_error("kRand(string kgram): kgram must be of"
   91:         " length greater than or equal to order k.");
   92:     }
   93:     if (rw_table.count(kgram) == 0) {
   94:         throw std::runtime_error("kRand(string kgram): kgram does not"
   95:         " exist.");
   96:     }
   97:     std::string alphabet;
   98:     for (auto const &var1 : rw_table) {
   99:         if (var1.first == kgram) {
  100:             for (auto const &var2 : var1.second) {
  101:                 alphabet += var2.first;
  102:             }
  103:         }
  104:     }
  105:     std::random_device device;
  106:     std::mt19937 mt_rand(device());
  107:     std::uniform_int_distribution<int> distribution(0, alphabet.length()
  108:     - 1);
  109:
  110:     return alphabet[distribution(mt_rand)];
  111: }
  112:
  113: std::string RandWriter::generate(std::string kgram, int L) const {
  114:     if (kgram.length() < static_cast<unsigned int>(rw_k)) {
  115:         throw std::runtime_error("generate(string kgram, int L): kgram mu
st"
  116:         " be of length greater than or equal to order k.");
  117:     }
  118:     std::string generated = kgram;
  119:     // generate new characters based on kgrams
  120:     for (int i = rw_k; i < L; i++) {
  121:         generated += kRand(generated.substr(i - rw_k, rw_k));
  122:     }
  123:     return generated;
  124: }
  125:
  126: std::ostream& operator<<(std::ostream& out, const RandWriter& rw) {
```

```
  127:     out << "Markov Model\tOrder: " << rw.rw_k << std::endl;
  128:     out << "kgram:\tfrequency:\tfrqncy of next char:\tprob of next char:"
 <<
  129:     std::endl;
  130:
  131:     for (auto const &var1 : rw.rw_table) {
  132:         // var1.first = kgram
  133:         out << var1.first << "\t";
  134:         out << rw.freq(var1.first) << "\t\t";
  135:         for (auto const &var2 : var1.second) {
  136:             // var2.first = next char
  137:             // var2.second = data
  138:             out << var2.first << ":" << var2.second << " ";
  139:         }
  140:         out << "\t\t\t";
  141:         for (auto const &var2 : var1.second) {
  142:             out << var2.first << ":" << var2.second << "/" <<
  143:             rw.freq(var1.first) << " ";
  144:         }
  145:         out << std::endl;
  146:     }
  147:     return out;
  148: }
```

```
    1: // Copyright 2022 Matthew Lorette Anaya
    2: #include "RandWriter.h"
    3:
    4: #define BOOST_TEST_DYN_LINK
    5: #define BOOST_TEST_MODULE Main
    6: #include <boost/test/unit_test.hpp>
    7:
    8: BOOST_AUTO_TEST_CASE(base_test) {
    9:     std::cout << "******************** Test Case 1 ********************"
<<
   10:     std::endl;
   11:
   12:     int k = 2;
   13:     std::string str = "gagggagaggcgagaaa";
   14:     RandWriter rw(str, k);
   15:
   16:     std::cout << "Printing out Markov Table for string:\n" <<
   17:     str << std::endl << std::endl;
   18:     std::cout << rw << std::endl;
   19:
   20:     std::cout << "Testing orderK and freq functions" << std::endl;
   21:     BOOST_REQUIRE(rw.orderK() == k);
   22:     BOOST_REQUIRE(rw.freq("gg") == 3);
   23:     BOOST_REQUIRE(rw.freq("ga", 'g') == 4);
   24:
   25:     std::cout << "Testing kRand function" << std::endl;
   26:     char rand = rw.kRand("aa");
   27:     BOOST_REQUIRE(rand == 'a' || rand == 'g');
   28:
   29:     std::cout << "Testing generate function" << std::endl << std::endl;
   30:     BOOST_REQUIRE(rw.generate("ga", 10).length() == 10);
   31: }
   32:
   33: BOOST_AUTO_TEST_CASE(exception_test) {
   34:     std::cout << "******************** Test Case 2 ********************"
<<
   35:     std::endl;
   36:     std::cout << "Testing construction exception: RandWriter('ADF', 4)" <
<
   37:     std::endl;
   38:
   39:     BOOST_REQUIRE_THROW(RandWriter("ADF", 4), std::invalid_argument);
   40:
   41:     std::cout << "Testing function exceptions" << std::endl;
   42:     RandWriter testMM("abc", 3);
   43:     BOOST_REQUIRE_THROW(testMM.freq("a"), std::runtime_error);
   44:     BOOST_REQUIRE_THROW(testMM.freq("ab", 'b'), std::runtime_error);
   45:     BOOST_REQUIRE_THROW(testMM.kRand("g"), std::runtime_error);
   46: }
```