```
    1: /**********************************************************************
***********/
    2: /* MarkovModel.cpp
          */
    3: /* Yoo Min Cha
          */
    4: /* Markov's Model
          */
    5: /* Professor Martin
          */
    6: /* 07 April 2014
          */
    7: /**********************************************************************
***********/
    8:
    9: #include <iostream>
   10: #include <ctime>
   11: #include <random>
   12: #include <vector>
   13: #include <cstring>
   14: #include "MarkovModel.hpp"
   15:
   16: using namespace std;
   17:
   18: MarkovModel::MarkovModel(string text, int k):_order(k), _alphabet(text)
   19: {
   20:   srand(time(0));
   21:   if (k < 0)
   22:     throw invalid_argument("Order k must be higher than zero.");
   23:   if (k >= text.size())
   24:     throw invalid_argument("Order k must be less than the length of text.
");
   25:
   26:   string temp = "";
   27:   if (k>0) temp = text.substr(0,k);
   28:
   29:   for (int i = k; i < text.size()+k; i++)
   30:   {
   31:     if (_kgrams[text[i%text.size()]].count(temp) == 0){
   32:       kgramList.insert(temp);
   33:       //kplusList.push_back(i);
   34:       _kgrams[text[i%text.size()]][temp] = 1;
   35:     } else {
   36:       _kgrams[text[i%text.size()]][temp]++;
   37:     }
   38:     temp += text[i%text.size()];
   39:     temp = temp.substr(1);
   40:
   41:   }
   42: }
   43:
   44: int MarkovModel::order()
   45: {
   46:   return _order;
   47: }
   48:
   49: int MarkovModel::freq(string kgram)
   50: {
   51:   // cout << "+" <<  kgram << "+" <<kgram.size()<<"+"<<_order<<endl;
   52:
   53:   if (kgram.size() != _order)
   54:     throw runtime_error("kgram must be the same size as k1");
   55:
   56:   int ans = 0;
   57:   for(int i=0;i<128;i++)
```

```
   58:   {
   59:     ans += _kgrams[i][kgram];
   60:   }
   61:     return ans;
   62: }
   63: int MarkovModel::freq(string kgram, char c)
   64: {
   65:   if (kgram.size() != _order)
   66:     throw runtime_error("kgram must be the same size as k2");
   67:   if (!_kgrams[c].count(kgram))
   68:     return 0;
   69:   else
   70:     return _kgrams[c][kgram];
   71: }
   72:
   73: char MarkovModel::randk(string kgram)
   74: {
   75:   if (kgram.size() != _order)
   76:     throw runtime_error("kgram must be the same size as k3");
   77:   int freqArr[128], total=0;
   78:   for(int i=0;i<128;i++){
   79:     freqArr[i] = MarkovModel::freq(kgram,i);
   80:     total += freqArr[i];
   81:   }
   82:   if (total!=0){
   83:     int ran = rand()%total+1;
   84:
   85:     for(int i=0;i<128;i++){
   86:       // cout<<"ran="<<ran<<endl;
   87:       ran -= freqArr[i];
   88:       if (ran<=0) {
   89:         //cout<<"return "<<i <<" "<<freqArr[i]<<endl;
   90:         return (char) i;
   91:       }
   92:     }
   93:   } else
   94:       throw runtime_error("No such kgram");
   95:
   96: }
   97:
   98: string MarkovModel::gen(string kgram, int T)
   99: {
  100:   if (kgram.size() != _order)
  101:     throw runtime_error("kgram must be the same size as k");
  102:
  103:   string result = "";
  104:   for(int i=0;i<T;i++){
  105:     //cout<<kgram<<"-\n";
  106:     char nextChar = randk(kgram);
  107:     //cout<<"next char="<<nextChar<<endl;
  108:     result += nextChar;
  109:     kgram += nextChar;
  110:     kgram = kgram.substr(1);
  111:   }
  112:   return result;
  113: }
  114:
  115: ostream& operator<< (ostream& out, MarkovModel& mm)
  116: {
  117:   out << "k-gram\tfreq" << endl;
  118:   out << "-------------" << endl;
  119:   for (auto it = mm.kgramList.begin(); it != mm.kgramList.end(); it++)
  120:   {
  121:     int a = mm.freq(*it);
  122:     out << *it << '\t' << a << endl;
```

```
  123:   }
  124:   out << endl;
  125:   out << "k+1 gram\tfreq" << endl;
  126:   out << "----------------------" << endl;
  127:   for (auto it = mm.kgramList.begin(); it != mm.kgramList.end(); it++)
  128:   {
  129:     int a;
  130:     for(int i = 0; i < 128; i++) if ((a=mm.freq(*it, (char) (i)))>0)
  131:     {
  132:       out << *it << " " << (char)(i) << "\t\t" << a << endl;
  133:     }
  134:   }
  135:   return out;
  136: }
```