```
    1: #include <string>
    2: #include <vector>
    3: #include <iostream>
    4: #include <exception>
    5: #include <stdexcept>
    6: #include <map>
    7: #include <algorithm>
    8: #include "MarkovModel.hpp"
    9:
   10: MarkovModel::MarkovModel(std::string text, int k)
   11: {
   12:         Order = k;
   13:         _alphabet = text;
   14:         _text = text;
   15:
   16:         int kgramEnd;
   17:         if(Order >= _text.length()){
   18:                 throw std::runtime_error("k is too large");
   19:         }
   20:         std::sort(_alphabet.begin(), _alphabet.end());
   21:         _alphabet.erase(std::unique(_alphabet.begin(), _alphabet.end()),
_alphabet.end());
   22:
   23:         for(unsigned int i = 1; i < text.length()+1; i++){
   24:                 std::string kgram_string, kplus_string, wraparound_string
;
   25:                 int wrapLength;
   26:
   27:                 kgramEnd = i + k - 1;
   28:
   29:                 if(kgramEnd >= text.length()){
   30:                         wrapLength = kgramEnd - text.length() + 1;
   31:                         wraparound_string = text.substr(0, wrapLength);
   32:                         kgram_string = text.substr(i, text.length()-1) +
wraparound_string;
   33:                         kplus_string = kgram_string + text.at(wrapLength)
;
   34:                 }
   35:                 else{
   36:                         kgram_string = text.substr(i, k);
   37:                         if(kgramEnd+1 >= text.length())
   38:                         {
   39:                                 kplus_string = kgram_string + text.at(0);
   40:                         }
   41:                         else {
   42:                                 kplus_string = text.substr(i, k+1);
   43:                         }
   44:                 }
   45:
   46:                 std::map<std::string, int>::iterator kgram_it = _kgrams.f
ind(kgram_string);
   47:                 if(kgram_it != _kgrams.end()){
   48:                         kgram_it -> second += 1;
   49:                 }
   50:                 else{
   51:                         _kgrams[kgram_string] = 1;
   52:                 }
   53:
   54:                 std::map<std::string, int>::iterator kplus_it = _kgrams.f
ind(kplus_string);
   55:                 if(kplus_it != _kgrams.end()){
   56:                         kplus_it->second += 1;
   57:                 }
   58:                 else{
   59:                         _kgrams[kplus_string] = 1;
```

```
   60:                 }
   61:         }
   62: }
   63:
   64: MarkovModel::~MarkovModel(){
   65:
   66: }
   67:
   68: int MarkovModel::order()
   69: {
   70:         return Order;
   71: }
   72:
   73: char MarkovModel::randk(std::string kgram)
   74: {
   75:         std::vector<char> character_count;
   76:         int kgram_freq;
   77:         int index;
   78:
   79:         if(freq(kgram) == 0){
   80:                 throw std::runtime_error("kgram does not exist in the tex
t");
   81:         }
   82:         else{
   83:                 kgram_freq = freq(kgram);
   84:         }
   85:
   86:         for(unsigned int i = 0; i < _alphabet.length(); i++){
   87:                 char kplus_char = _alphabet.at(i);
   88:                 int kplus_freq = freq(kgram, kplus_char);
   89:                 for(int j = 0; j < kplus_freq; j++){
   90:                         character_count.push_back(kplus_char);
   91:                 }
   92:         }
   93:
   94:         index = rand() % character_count.size();
   95:         return character_count.at(index);
   96: }
   97:
   98: std::string MarkovModel::gen(std::string kgram, int T)
   99: {
  100:         if(kgram.length() != Order){
  101:                 throw std::runtime_error("kgram length must == order");
  102:         }
  103:
  104:         std::string text = kgram;
  105:
  106:         if(Order == 0 && T == 0)
  107:                 return _text;
  108:
  109:         while(text.length() < T){
  110:                 char a = randk(kgram);
  111:                 std::string string(1,a);
  112:                 text.append(string);
  113:                 if(Order > 0){
  114:                         kgram = kgram.substr(1);
  115:                 }
  116:                 else{
  117:                         kgram = kgram;
  118:                 }
  119:                 std::string next_kgram = kgram + string;
  120:                 kgram = next_kgram;
  121:         }
  122:         return text;
  123: }
```

```
  124:
  125: int MarkovModel::freq(std::string kgram)
  126: {
  127:         if(Order != kgram.length()){
  128:                 throw std::runtime_error("kgram length must == order");
  129:         }
  130:         std::map<std::string, int>::iterator it = _kgrams.find(kgram);
  131:         if(it != _kgrams.end()){
  132:                 return it->second;
  133:         }
  134:         else
  135:                 return 0;
  136: }
  137:
  138:
  139: int MarkovModel::freq(std::string kgram, char c)
  140: {
  141:         int frequency = 0;
  142:
  143:         if(Order == 0){
  144:                 for(unsigned int i = 0; i < _text.length(); i++){
  145:                         if(_text.at(i) == c){
  146:                                 frequency++;
  147:                         }
  148:                 }
  149:                 return frequency;
  150:         }
  151:
  152:         if(Order != kgram.length()){
  153:                 throw std::runtime_error("kgram length must == order");
  154:         }
  155:
  156:         std::string string(1, c);
  157:         std::string kplus_string = kgram + string;
  158:         std::map<std::string, int>::iterator it = _kgrams.find(kplus_stri
ng);
  159:         if(it != _kgrams.end()){
  160:                 return it->second;
  161:         }
  162:         else
  163:                 return 0;
  164:
  165: }
  166:
  167: std::ostream& operator << (std::ostream& out, MarkovModel& MM)
  168: {
  169:         std::cout << std::endl << "Order = " << MM.Order << std::endl;
  170:         std::cout << "Available alphabet = " << MM._alphabet << std::endl
;
  171:
  172:         for(std::map<std::string, int>::iterator it = MM._kgrams.begin();
 it != MM._kgrams.end(); ++it)
  173:         {
  174:                 if(it->first.length() == MM.Order){
  175:                         std::cout << std::endl;
  176:                         std::cout << "kgram:    ";
  177:                         std::cout << it->first << "   | frequency: " << i
t->second;
  178:                         std::cout << std::endl;
  179:                 }
  180:                 else{
  181:                         std::cout << "          ";
  182:                         std::cout << it->first << " ----->" << it->second
;
  183:                         std::cout << std::endl;
```

```
184:                     }
185:             }
186:
187:             return out;
188: }
```