```
 1: // Copyright 2023 Thomas O'Connor
 2: #include "RandWriter.hpp"
 3:
 4: // Constructor:
 5: RandWriter::RandWriter(string text, int k) : _K(k) {
 6:     // if order 0, the text IS the dictionary of probabilities
 7:     if (!k) {
 8:         MarkovModel[ORDERZERO].second.assign(text);
 9:         MarkovModel[ORDERZERO].first = static_cast<int>(text.size());
10:     }
11:     // else
12:     // Append the first k+1 characters to the end of the text
13:     string kgram = text.substr(0, k+1);
14:     text.append(kgram);
15:     kgram.pop_back();
16:     // for each char in the text, generate a kgram
17:     // starting with the first k-characters:
18:     for (size_t i = k; i < text.size()-1; i++) {
19:         // at this kgram push back the following char
20:         // into the personal dictionary for that kgram
21:         MarkovModel[kgram].first++;
22:         MarkovModel[kgram].second.push_back(text[i]);
23:         // generate new kgram
24:         cycleString(kgram, text[i]);
25:     }
26:     // the MarkovModel is now full of kgrams
27:     // and personal probability dictionaries
28: }
29:
30: // Number of occurences of kgram in text
31: // Throw an exception if kgram is not length k
32: int RandWriter::freq(string kgram) const {
33:     if (kgram.size() != static_cast<size_t>(_K)) {
34:         throw std::invalid_argument("String length != k in FREQ");
35:     } else {
36:         if (!_K) kgram = ORDERZERO;
37:         return MarkovModel.at(kgram).first;
38:     }
39: }
40:
41: // Number of times that character c follows kgram
42: // if order=0, return num of times that char c appears
43: // (throw an exception if kgram is not of length k)
44: int RandWriter::freq(string kgram, char c) const {
45:     if (kgram.size() != static_cast<size_t>(_K)) {
46:         throw std::invalid_argument("String length != k in FREQ");
47:     } else {
48:         if (!_K) kgram = ORDERZERO;
49:         return std::count(MarkovModel.at(kgram).second.begin(),
50:                           MarkovModel.at(kgram).second.end(), c);
51:     }
52: }
53:
54: // Random character following given kgram
55: // (throw an exception if kgram is not of length k)
56: // (throw an exception if no such kgram)
57: char RandWriter::kRand(string kgram) {
58:     if (kgram.size() != static_cast<size_t>(_K)) {
59:         throw std::invalid_argument("String length != k in KRAND");
60:     } else {
61:         // check for order zero
62:         if (!_K) kgram = ORDERZERO;
63:         // get bounds of dictionary
64:         int lengthOfDictionary = MarkovModel.at(kgram).second.size() - 1;
65:         // set randomizing variables
```

```
   66:         std::random_device rd;
   67:         std::mt19937 gen(rd());
   68:         std::uniform_int_distribution<> distrib(0, lengthOfDictionary);
   69:         return MarkovModel.at(kgram).second[distrib(gen)];
   70:     }
   71: }
   72:
   73: // Generate a string of length L characters by simulating a trajectory
   74: // through the corresponding Markov chain. The first k characters of
   75: // the newly generated string should be the argument kgram.
   76: // Throw an excpetion if kgram is not of length k.
   77: // Assume that L is at least k
   78: string RandWriter::generate(string kgram, int L) {
   79:     if (kgram.size() != static_cast<size_t>(_K)) {
   80:         throw std::invalid_argument("String length != k in GENERATE " + k
gram);
   81:     }
   82:     string outputString;
   83:     char newItem;
   84:     for (int i = 0; i < L; i++) {
   85:         newItem = kRand(kgram);
   86:         outputString.push_back(newItem);
   87:         cycleString(kgram, newItem);
   88:     }
   89:     outputString.push_back('\n');
   90:     return outputString;
   91: }
   92:
   93: // Overload the stream insertion operator << and display the internal sta
te
   94: // of the Markov model. Print out the order, alphabet, and the frequencie
s
   95: // of the k-grams and k+1-grams
   96: std::ostream& operator<<(std::ostream& out, RandWriter& obj) {
   97:     out << "Order: " << obj.orderK() << endl;
   98:     for (auto const &outerMap : obj.MarkovModel) {
   99:         cout << "Kgram:  " << outerMap.first << "  Frequency:  " << outer
Map.second.first;
  100:         cout << "  Dictionary:  " << outerMap.second.second << endl;
  101:         list<char> usedChars;
  102:         for (char kp1gram : outerMap.second.second) {
  103:             // if it could not find the element in the used chars list, c
reate a k+1 gram
  104:             if (std::find(usedChars.begin(), usedChars.end(), kp1gram) ==
 usedChars.end()) {
  105:                 cout << "\tK+1gram:  " << outerMap.first << kp1gram << "
  Frequency:  " <<
  106:                     std::count(outerMap.second.second.begin(),
  107:                     outerMap.second.second.end(), kp1gram)
  108:                     << endl;
  109:                 usedChars.push_back(kp1gram);
  110:             }
  111:         }
  112:     }
  113:     return out;
  114: }
  115:
  116: // helper funcion
  117: void cycleString(string& str, char item) {
  118:     if (!str.size()) return;
  119:     str.erase(0, 1);
  120:     str.push_back(item);
  121: }
```