

```
1: // Copyright 2022 Anson Cheang
2: #include "RandWriter.h"
3: #include <utility>
4:
5: using namespace std; // NOLINT
6:
7: // -----
8: // create a Markov model of order k from given text
9: // Assume that text has length at least k.
10: RandWriter::RandWriter(string text, int k) {
11:     order = k;
12:     string storage = text;
13:     int length = text.length();
14:     int count = 0, l;
15:     char temp;
16:     string stringTemp, size;
17:     bool isThere;
18:
19:     for (int i = 0; i < order; i++) {
20:         storage = storage + text[i];
21:     }
22:     for (int i = 0; i < length; i++) {
23:         temp = text.at(i);
24:         isThere = false;
25:         l = alphabet.length();
26:         for (int j = 0; j < l; j++) {
27:             if (alphabet.at(j) == temp) {
28:                 isThere = true;
29:             }
30:         }
31:         if (!isThere) {
32:             alphabet = alphabet + temp;
33:         }
34:     }
35:     for (int i = 0; i < length; i++) {
36:         stringTemp.clear();
37:         stringTemp = storage.substr(i, order);
38:         RM.insert(make_pair(stringTemp, map<char, int>()));
39:         RM.at(stringTemp).insert(make_pair('\0', 0));
40:     }
41:
42:     char placement;
43:     for (int i = 0; i < length; i++) {
44:         stringTemp.clear();
45:         stringTemp = storage.substr(i, order+1);
46:         placement = stringTemp[order];
47:         stringTemp.clear();
48:         stringTemp = storage.substr(i, order);
49:         RM.at(stringTemp).insert(make_pair(placement, 0));
50:     }
51:     map<char, int>::iterator p;
52:
53:     for (int j = 0; j < length; j++) {
54:         stringTemp.clear();
55:         stringTemp = storage.substr(j, order);
56:         p = RM.at(stringTemp).find('\0');
57:         count = p->second;
58:         count++;
59:         RM.at(stringTemp).at('\0') = count;
60:     }
61:     for (int j = 0; j < length; j++) {
62:         stringTemp.clear();
63:         stringTemp = storage.substr(j, order+1);
64:         placement = stringTemp[order];
65:         stringTemp.clear();
```

```
66:         stringTemp = storage.substr(j, order);
67:         p = RM.at(stringTemp).find(placement);
68:         count = p->second;
69:         count++;
70:         RM.at(stringTemp).at(placement) = count;
71:     }
72: }
73: // -----
74: // order k of Markov model
75: int RandWriter::orderK() const {
76:     return order;
77: }
78: // -----
79: // number of occurrences of kgram in text
80: // throw an exception if kgram is not of
81: // length k
82: int RandWriter::freq(string kgram) const {
83:     if (kgram.length() != static_cast<unsigned int> (order)) {
84:         throw runtime_error("freq(string):kgram is not of length k(order)
");
85:     }
86:
87:     map<char, int>::const_iterator p;
88:
89:     p = RM.at(kgram).find('\\0');
90:
91:     return p->second;
92: }
93: // -----
94: // number of times that character c follows kgram
95: // if order=0, return num of times char c appears
96: // (throw an exception if kgram is not of length k)
97: int RandWriter::freq(string kgram, char c) const {
98:     if (kgram.length() != static_cast<unsigned int> (order)) {
99:         throw runtime_error(
100:             "freq(string, char):kgram is not of length k(order)");
101:     }
102:
103:     string sub = kgram + c;
104:
105:     map<char, int>::const_iterator p;
106:     p = RM.at(kgram).find(c);
107:     return p->second;
108: }
109: // -----
110: // random character following given kgram
111: // (Throw an exception if kgram is not of length k.
112: // Throw an exception if no such kgram.)
113: char RandWriter::kRand(string kgram) {
114:     if (kgram.length() != static_cast<unsigned int> (order)) {
115:         throw runtime_error("kRand:kgram is not of length k(order)");
116:     }
117:     map<string, map<char, int>>::iterator t;
118:     map<char, int>::iterator p;
119:     t = RM.find(kgram);
120:     if (t == RM.end()) {
121:         throw runtime_error("kRand: could not find given kgram");
122:     }
123:     int amount = freq(kgram);
124:
125:     srand(static_cast<int>(time(NULL)));
126:     int rv = rand() % amount; //NOLINT
127:     p = RM.at(kgram).begin();
128:     p++;
129:     while (p != RM.at(kgram).end() && rv > p->second) {
```

```
130:         rv = rv - p->second;
131:     }
132:     return p->first;
133: }
134: // -----
135: // generate a string of length L characters
136: // by simulating a trajectory through the corresponding
137: // Markov chain. The first k characters of the newly
138: // generated string should be the argument kgram.
139: // Throw an exception if kgram is not of length k.
140: // Assume that L is at least k.
141: string RandWriter::generate(string kgram, int L) {
142:     if (kgram.length() != static_cast<unsigned int> (order)) {
143:         throw runtime_error("generate:kgram is not of length k(order)");
144:     }
145:     string returnString = kgram;
146:
147:     for (int i = 0; i < (L - order); i++) {
148:         returnString = returnString + kRand(returnString.substr(i, order)
);
149:     }
150:     return returnString;
151: }
152: // -----
153: // overload the stream insertion operator and display
154: // the internal state of the Markov Model. Print out
155: // the order, the alphabet, and the frequencies of
156: // the k-grams and k+1-grams.
157: ostream& operator<<(ostream& out, RandWriter& markov) {
158:     return out;
159: }
```