```
/* <Copyright Abara Mehene*/

#include "MarkovModel.hpp"
#include <utility>
#include <map>
#include <string>

MarkovModel::MarkovModel(std::string text, int k) {
  _order = k;
  _alphabet = text;

  char character;
  bool value = 0;
  std::string circularString = text;
  std::string tempString;

  // INSERTING CHARS INTO THE ALPHABET

  // store the chars that appear in text into the alphabet
  for (int i = 0; i < _order; ++i) {
    // insert the new char into the alphabet
    circularString.push_back(text[i]);
  }

  // check if the char in the text is already in the alphabet
  for (unsigned int i = 0; i < text.length(); ++i) {
    character = text.at(i);
    value = 0;
    // check if we already have the char
    for (unsigned int j = 0; j < _alphabet.length(); ++j) {
      if (_alphabet.at(j) == character)
        value = 1;
    }
    // do we store the char into the alphabet?
    if (!value)
      _alphabet.push_back(character);
  }


  std::map<std::string, int>::iterator it;
  int temp_count = 0;

  // get a substring from the text and inserting it into kgram
  for (int x = _order; x <= _order + 1; ++x) {
    for (unsigned int y = 0; y < text.length(); ++y) {
      tempString = circularString.substr(y, x);
      _kgrams.insert(std::pair<std::string, int>(tempString, 0));
      it = _kgrams.find(tempString);
      temp_count = it->second;
      temp_count++;
      _kgrams[tempString] = temp_count;
    }
  }
}

// returns order
int MarkovModel::order() {
  return _order;
}

int MarkovModel::freq(std::string kgram) {
  // error check
  if ((unsigned)_order != kgram.length())
    throw std::runtime_error("Kgram is not of length k");
  // space
  std::map<std::string, int>::iterator numOfkGram;
  // go through the map and count how many
  // times we have the string kgram
  numOfkGram = _kgrams.find(kgram);

  // return the kgram we find
  if (numOfkGram == _kgrams.end())
    return 0;
  return numOfkGram->second;
}

int MarkovModel::freq(std::string kgram, char c) {
  // error check
  if (kgram.length() != (unsigned)_order)
    throw std::runtime_error("Kgram is not of length k");

  // put c into kgram then find the new kgram
  std::map<std::string, int>::iterator numOfkGram;
  kgram.push_back(c);
  numOfkGram = _kgrams.find(kgram);

  // if there is the kgram
  if (numOfkGram == _kgrams.end())
    return 0;
  return numOfkGram->second;
}

char MarkovModel::randk(std::string kgram) {
  unsigned int seed = time(NULL);
  int randomValue;
  std::string randomInput;

  // error check
  if (kgram.length() != (unsigned)_order)
    throw std::runtime_error("Kgram is not of length k");
  // space
  // try to find the kgram
  std::map<std::string, int>::iterator temp;
  temp = _kgrams.find(kgram);

  int kgram_freq = freq(kgram);

  // if there is no such kgram
  if (temp == _kgrams.end())
    throw std::runtime_error("No such kgram");

  for (;;) {
    // gets random value from the kgram_freq
    randomValue = rand_r(&seed) % kgram_freq;
    randomInput = kgram + _alphabet[randomValue];
    // if we are at the end return a random char
    if (temp != _kgrams.end()) {
      std::cout << randomInput << std::endl;
      return _alphabet[randomValue];
    }
  }
}
```

```

std::string MarkovModel::gen(std::string kgram, int T) {
  // error check
  if (kgram.length() != (unsigned)_order)
    throw std::runtime_error("Kgram is not of length k");

  // put the initial kgram into our string
  std::string tempkGram = kgram;

  // append the random character to the end of our output
  // until size the string is of size T
  for (int i = kgram.length(); i < T; ++i) {
    tempkGram.push_back(randk(kgram));
  }

  return tempkGram;
}

std::ostream& operator<<(std::ostream &out, MarkovModel &mm) {
  std::map<std::string, int>::iterator it;
  // basic output
  out << "Order: " << mm._order << std::endl;
  out << "Alphabet: " << mm._alphabet << std::endl;
  out << "Kgrams map: " << std::endl;

  for (it == mm._kgrams.begin(); it != mm._kgrams.end(); it++) {
    out << it->first << it->second << std::endl;
  }
  return out;
}

MarkovModel::~MarkovModel() {
  // destructor
}
```