333 Section 6 - C++ Templates, STL Sample Solutions

Welcome back to Section! We're glad that you're here :)

C++ Templates

Exercise 1) Templates & Things

Fill in the blanks below for the definition of a simple templated struct `Node` for a singly-linked list. The struct has two public fields: a `value`, which is a pointer of template type `T` pointing to a heap-allocated payload, and a `next`, which is a pointer to another struct `Node`. The struct also has a two-argument constructor that takes a `T` pointer for `value` and another `Node<T>` pointer for `next`.

```
template <typename T>
struct Node {
  Node(T* val, Node<T>* node): value(val), next(node) {}
  ~Node() { delete value; }
  T* value;
  Node<T>* next;
};
```

Remember that struct in C++ by default has its members being public, so no need to specify the access modifiers explicitly here.

C++'s Standard Library

Exercise 2) Standard Template Library

Complete the function `ChangeWords` below. This function's inputs are a vector of strings, and a map of `<string, string>` key-value pairs. The function should return a new `vector<string>` value (not a pointer) that is a copy of the original vector, except that every string in the original vector that is found as a key in the map should be replaced by the corresponding value from that key-value pair.

Write your code below. Assume that all necessary headers have already been written for you.

```
using namespace std;
vector<string> ChangeWords(const vector<string> &words,
                           map<string,string> &subs) {
  vector<string> result;
  for (auto& word : words) {
    if (subs.find(word) != subs.end()) {
      result.push_back(subs[word]);
    } else {
      result.push_back(word);
    }
  }
  return result;

}
```

Exercise 3) T9 Example



Before smart phones, mobile phones used a predictive text system called T9, based on the mapping of a single numpad key to any of the corresponding letters shown in the image. Note that the `'1'`, `'*'`, and `'#'` keys won't be used and that `'0'` corresponds to [Space].

Our T9 class also has a field `map<char, char> letters_to_keys`, which maps letters to their corresponding number on the T9 keyboard. For this exercise, assume this map has already been initialized for you.

a) Complete the function to add a mapping from *each* prefix to the string itself to `predictions`. Assume the passed-in word is always lowercase. You may find the string member function `string substr(size_t pos, size_t len)` useful, which returns the substring of length `len` starting from position `pos`.

```
map<string, vector<string>> predictions; // global prediction map
void AddPrefixesToPredictions(const string& word) {
  string prefix;

  for (auto& c : word) {
    prefix += letters_to_keys[c];
    predictions[prefix].push_back(word);
  }
}
```

b) Complete the function below to print out the contents of `predictions`. For example, if we've added `"a"` and `"ax"`, it should print out the following (note the formatting):

```
2 : a, ax,
29 : ax,
```

```
void PrintPredictions() {

  // loop over every prediction pair
  for (auto& pred_pair : predictions) {
    cout << pred_pair.first << " : ";

    // loop over every vector entry
    for (auto& w : pred_pair.second) {
      cout << w << ", ";
    }

    cout << endl;
  }
}
```

Exercise 3) STL Debugging [extra]

Here is a little program that has a small class `Thing` and `main` function (assume that necessary `#includes` and `using namespace std;` are included).

```
class Thing {
 public:
  Thing(int n): n_(n) { }
  int getThing() const { return n_; }
  void setThing(int n) { n_ = n; }
 private:
  int n_;
};

int main() {
  Thing t(17);
  vector<Thing> v;
  v.push_back(t);
}
```

This code compiled and worked as expected, but then we added the following two lines of code (plus the appropriate `#include <set>`):

```
set<Thing> s;
s.insert(t);
```

The second line (`s.insert(t)`) failed to compile and produced dozens of spectacular compiler error messages, all of which looked more-or-less like this (edited to save space):

```
In file included from string:48:0, from bits/locale_classes.h:40, from
bits/ios_base.h:41,from ios:42,from ostream:38, from /iostream:39,from
thing.cc:3: bits/stl_function.h: In instantiation of 'bool
std::less<_Tp>::operator()(const _Tp&, const _Tp&) const [with _Tp =
Thing]': <<many similar lines omitted>> thing.cc:37:13: required from here
bits/stl_function.h:
387:20: error: no match for 'operator<' (operand types are 'const Thing'
and 'const Thing') { return ___x < __y; }
```

What on earth is wrong? Somehow class `Thing` doesn't work with `set<Thing>` even though insert is the correct function to use here. (a) What is the most likely reason, and (b) what would be needed to fix the problem? (Be brief but precise – you don't need to write code in your answer, but you can if that helps make your explanation clear.)

STL has to compare them using operator<. Add an appropriate operator< as either a member function in Thing, or as a free-standing function that compares two Thing& parameters.