Problem Set 4

Due before midnight on Friday, November 5, 2010.

1 Assignment Goals

1. Explore a significant code base.
2. Learn about iterators.
3. Learn more about operator extensions, including casts.
4. Learn how various C++ constructs interact.
5. Analyze the class structure of a program.
6. Extend an existing program in non-trivial ways.

---

2 Word Triangles

A *word triangle* is a sequence of words, each of length one less than the preceding word in the sequence, and ending with a minimal-length word. Moreover, each word is an anagram (letter rearrangement) of a subword of the previous word. An example should make this clear:

```
trilinear
airliner
airline
aliner
alien
ilea
ail
```

Of course, one can debate what is a "word". For the above example, I used the dictionary file `/usr/share/dict/words` found on the Zoo.

You will find a complete word triangle program on the Zoo under the path `/c/cs427/assignments/ps4/PS4-triangle` that implements the following algorithm:

1. Obtain four parameters: dictionary file name, filter string, minimum length word in the triangle, and maximum length word in the triangle.
2. Read the entire dictionary file into memory, discarding words that are too long or that contain letters not in the filter string.
3. Arrange the dictionary into lists of words of the same length.

4. Copy each word of the dictionary that can be the head of a triangle into a new triangle dictionary. A word is a *triangle head* if either its length is the minimum length allowed or it has a *parent* triangle word. A *parent* is a word that is one letter shorter than the original word and is an anagram of a subword of the original word. That is, it can be obtained from the original word by deleting one letter and rearranging the remaining letters. For each word copied, set a link to its first parent. (There can be several.)

5. Print each triangle head of maximum length followed by successively shorter triangle heads comprising the triangle.

3 Assignment

This assignment has two parts. The first part is to explore the given code and answer questions about it. The second part is to modify the code in a couple of directions.

3.1 Code Exploration

1. Draw a UML diagram of the class structure of this program.

2. Find each `friend` declaration. For each, find the lines that cause privacy violations if the declaration is removed.

3. Class `Word` contains a declaration `operator<=(const Word& w2)`. It is used in the line `(*p <= word)` in `WordList::findParent()`. What is the type of `*p`? What is the type of `word`? Explain why the aforementioned operator extension is used even though the argument types are not the same as the corresponding declared parameter types.

4. Class `WordEntry` contains a set-function. Where is it needed? What are the pros and cons of using a set-function rather than making the `parent` data member public? What are the pros and cons of instead adding another `friend` declaration?

5. In `Dictionary::readDict()`, there is a call `add(cword)`. Why does this work even though `Dictionary::add()` has only been defined for parameter type `const Word&`?

6. `Triangle::build()` contains the statement: `add(*p)->setParent(parent)`. Explain in detail how this works. For each operator or function in the expression, say what the types of the parameters and results are, which methods are actually called, and why.

7. `main()` initializes static variables declared in `word.hpp`. Which principle of OO-programming does this violate? What would be a better way to handle this initialization? Write the code to do it.

Iterators Class `WordList` contains an embedded class `iterator`. Iterators are generalizations of pointers. The intention is that an iterator can be used to iterate over a linear data structure in the same way that a pointer can be used to scan an array. For example, if `int a[100];` is an array of integers, then the following loop can be used to print the array:

```
int* p;
int* begin=&a[0];
int* end=&a[100];
for (p=begin; p!=end; p++) cout << *p;
```

We use iterators to scan over a `WordList`. See `WordList::print()` for an example of their use, and compare it with the above pointer code.

8. Explain what each of the operator definitions in class `iterator` does.

9. What happens if you comment out the definition for `WordEntry& operator*() const`? Explain.

10. What happens if you comment out the definition for `WordEntry* operator->() const`? Explain.

3.2 Extensions: Count Triangles

A word may, in general, have several parents. If it does, it may head several word triangles. Since each parent may itself have several parents, the number of triangles for a given word is potentially quite large.

The first extension is to modify the given code to count the number of word triangles for each word. It should appear to the user just like the present code, except that instead of printing out each head of a word triangle followed by the triangle, it will print out just the head word followed by a count of the number of triangles that it heads.

3.3 Extension: Find All Triangles

The second extension is to modify the given code not only to count but also to print out all triangles headed by a given word. This will require that you modify the data structures used to keep track of the words in a word triangle. You might find it useful to define one or more new classes for this purpose. You will be graded not only on the correctness of your code but also on its efficiency and how closely it follows object-oriented design principles. You should document your code with a UML diagram as well as with a report describing your data structures and why you believe your design follows good OO practices.

4 Deliverables

You should submit this assignment using the `submit` script that you will find in `/c/cs427/bin/` on the Zoo. The three parts should be submitted separately using the `-V` option.

1. To submit the answers to the code exploration questions, use

   ```
   /c/cs427/bin/submit -V1 4 files...
   ```

2. To submit the count-triangles extension, use

   ```
   /c/cs427/bin/submit -V2 4 files...
   ```

3. To submit the find-all-triangles extension, use

```
/c/cs427/bin/submit -V3 4 files...
```

Remember to include the following items with each code submission:

1. Source code and header files.
2. A `makefile` and any other files necessary to build your project. (If you're using Eclipse with the default settings, the makefiles will be found in the directory `Debug`.)
3. Test programs, output, and methodology that you used to test the correctness of your program.
4. Brief reports named `report1.txt` and `report2.txt` (or any other common format such as `.pdf`) describing the design choices you made in implementing the required code extensions. You can also put any other information here that the TA should know about your programs.