CSC112 Fundamentals of Computer Science Spring 2016

Lab 8 – Linked Lists

In this lab we will implement an application to find antonyms. Your program will:

1. Read a file named "antonyms.txt". Each line of the file contains two words that are antonyms of each other. The file can be downloaded from **http://menehune.opt.wfu.edu**
2. Build a linked list of word pairs from the file.
3. Enter the user-loop:
   (a) Prompt the user for a word and read it from the keyboard.
   (b) Search your linked list for the word.
      i. Your search should check both words in the pair.
      ii. If the word is found, print the word and its antonym
      iii. otherwise, print "not found".
   (c) Enter a period (and return) to exit the program.

Implementation Guidelines:

1. Write a class named `antonyms` which implements a singly linked list of word pairs.
   (a) Class `antonyms` makes use of class `word_pair`. More on class `word_pair` below.
   (b) Your implementation of a singly linked list must include both a **head** and a **tail** pointer.
   (c) Each new word pair read from the input file must be appended to the end of the list. Maintain the `tail` pointer to make this efficient.
   (d) Class `antonyms` must have a constructor to initialize an empty list.
   (e) Class `antonyms` must have a destructor to release all memory used by the list.
   (f) Class `antonyms` must have methods **read_file**, **append** and **search**.
2. Use a class named `word_pair` to implement each word pair.
   (a) Suggested implementation for class `word_pair` :

```
class word_pair {
   private:
      char * word1 ;
      char * word2 ;
      word_pair * next ;
   public:
      word_pair()  ;
      ~word_pair() ;
      void print1() ;
      void print2() ;
      friend class antonym ;
} ;
```

(b) The constructor `word_pair()` should initialize both pointers `word1` and `word2` to `NULL`.

(c) When reading a word pair, you should read the input into a fixed-length array of characters. You may assume no word is longer than 63 characters (array size 64, to leave space for the null terminator). The array will be re-used on subsequent reading from the file. Remember to use `strdup` when assigning a pointer to `word1` and `word2`.

(d) In use, both `word1` and `word2` point to dynamically allocated memory. The destructor should free the memory used by each word.

(e) When searching, the word input by the user may match `word1`. For example, suppose the user has input the word "left" and your search program finds it matches `word1`. Your output should be:

```
word: 'left'  antonym: 'right'
```

The words in the file are alphabetized by the first word. If the user enters the word "right", then the search method (checking both words) will find the word "right" matching `word2`. In this case, the search has found the same word pair on the list, but your output should be:

```
word: 'right'  antonym: 'left'
```

The important observation is that sometimes the antonym for the user-supplied word is `word2` in class `antonym`, and sometimes it is `word1`. The methods `print1` and `print2` are intended to handle these two cases.

Sample Session:

```
gottlieb% g++ antonym.cc
gottlieb% a.out
919 word pairs read.
Begin entering words. Use period '.' to stop.
--> full
word: 'full'  antonym: 'devoid'
--> empty
word: 'empty'  antonym: 'full'
--> glass
word 'glass' not found.
--> go
word: 'go'  antonym: 'come'
--> stop
word: 'stop'  antonym: 'go'
--> .
```

Turn In:

Save all your work in a directory named "Lab8". Change to your home directory (the parent directory of "Lab8"), and create a file named "lab8.tar" using the command:

```
tar cf lab8.tar Lab8
```

Use `sftp` to upload the file "lab8.tar" to your account on telesto.