COMPRO2 (S17B, S18B, S19B)
MACHINE PROBLEM SPECIFICATIONS PHASE 1
(50% of MP Grade)

1. OBJECTIVES

   To design and implement the (a) data structures and (b) efficient algorithms for word search similar to `morewords.com`.

2. GENERAL DESCRIPTION

   One of the fundamental problems in computer science is searching. Given a universe of values, the problem involves the design and implementation an algorithm that will find if a certain key value is in the universe or not. Since the search function may involve a big universe (i.e., a lot of values), and that the search operation maybe invoked many times, it is important that the algorithm be computationally efficient.

   In COMPRO2, you will learn several data structures such as arrays, structures, and linked lists. You also learned about static and dynamic memory allocation. Your task in this machine problem is to apply what you learned to design and implement an efficient search algorithm.

3. SPECIFIC TASKS

   (a) Point your web browser to `morewords.com`. Read the FAQ. Download the file named `enable2k.txt`. This file contains the universe of words for searching.

   (b) On the same website, try the following word searches:

      i. Exact word search
      Search, for example, for the word "computer" (enter the word without the double quotes).

      ii. Word search with ? wildcard
      Search, for example, for "a?t". This search means that you want to search for three letter words that starts with 'a', and ends with 't'. The question mark is a wildcard which means any character. Try other possibilities such as: "b?e?t", "?co?r?"

      iii. Word search with * wild card
      Search, for example, for "a*t". This search means that you want to search for words that starts with 'a', and ends with 't'. The * (asterisk) is a wildcard which means any number of characters. The difference between "a?t" and "a*t" is that "a?t" is limited to words of length 3 characters (for example "art"), while "a*t" can be more than 3 characters in length (for example: "about"). Try other possiblities such as: "art*" (search for all words that start with "art"), and "*art" (search for all words that end with "art").

   (c) Once you have a good idea of the word searches above, you can then proceed to the design and implementation of your word search data structure and algorithms. Remember that it is important for you to produce an algorithm that is computationally efficient, i.e., to produce the output in the least amount of time. You may need to come up with several alternative design and implementations. You will need to test and compare their relative performance.

   Implement the following functions:

```
void Initialize(_______ universe)
{
        /* implement the body of this function */

       / * you may write other functions that can be
            called inside this function */

}
```

```
void Search (_________ universe, Str30 word)
{

      /* implement the body of this function */

     / * you may write other functions that can be
          called inside this function */


}
```

- The parameter named `universe` is the data structure that contains all the words (from `enable2k.txt`). The underline means that you have to specify your own data type.
- Function `Initialize()` should initialize the contents of your data structure, i.e., the storage representation of the words in the universe (of words). The `Initialize()` function should be called only once inside `main()`.
- The parameter named `word` is the word that you would like to search. The word may contain wild card characters, i.e., ? and *. Although it is possible to have both wildcards in the same word in `morewords.com` search (for example search for "ab?c*", we will not adapt this scheme to the machine problem. That is, we restrict our word search to a word with ? or with * but not both. The `Search()` maybe called several times. Examples of `Search()` function calls are shown below:

```
Search(universe, "computer");
Search(universe, "COmPuTeR");   // should produce same result as above
Search(universe, "a?t");
Search(universe, "a*t");
Search(universe, "art*");
Search(universe, "*art");
Search(universe, "wxyz");      // this will not produce a match
```

  There are two possible search results.
  - First, the search will produce hits, i.e., there is a word that matches (for exact search) or several words that match (for search with wildcards) the search key. If there is a match, the `Search()` function should output (via `printf()`) the corresponding word(s), one word per line of output.
  - Second, there is NO match. For example, `Search(universe, ''wxyz'')` will not produce a match. If there is NO match, the `Search()` function should output the string `NOMATCH` in capital letters.

4. ASSUMPTION

   Do not assume that the number of words in `enable2k.txt` is known in advance. Your program should work even if the number of words in the said file is modified. (For example, "google" is not yet in `enable2k.txt`, but it maybe added in future versions).

5. SKELETON FILE

   You will be provided a skeleton file named `lastname_mp1.c`. Rename the file using your own lastname. For example, if your last name is SANTOS, then the file should be renamed as `santos_mp1.c`. Use the skeleton file as base code.

Please make sure that your READ and FOLLOW the instructions (written as comments) in the skeleton file.

6. DELIVERABLES and DEADLINES

   You need to submit four items, namely:

   (a) C program source code (`lastname_mp1.c file`)

   Note: It is your responsibility to check/test the solution (several times!) for syntax and semantic errros before submission. Once submitted, no replacement will be accepted. The source code should be RECEIVED as an email attachment before 12 midnight of November 14, 2014 (Friday) local time.

   Send your email to two addresses: `florante.salvador@dlsu.edu.ph` and `pulingfe@yahoo.com`. Make sure that you CC also your own email account.

   Email subject should be: COMPRO2 <section> LASTNAME MP1. For example: COMPRO2 S12A SANTOS MP1.

   (b) Accomplished MP submission checklist (print and accomplish `mp_checklist.pdf`).

   (c) Hardcopy/printout of your C program source code; use short bond paper printed back-to-back.

   (d) A discussion of the design of your data structure, and a description of your search algorithm. You should provide an illustration/drawing to help the reader understand your document. The document should not exceed two pages of short bond paper printed back-to-back. Use 10pt or 12pt font size.

   Staple the MP checklist on top of your discussion, and source code printout. Submit these documents within the first 5 minutes of our class on Nov 17 (Mon class) / Nov 18 (Tue class). INCOMPLETE SUBMISSION and/or NON-COMPLIANCE with the instructions and specifications will result into point deductions.

   LATE SUBMISSION will be accepted but with a deduction of two points for every 1 hour of late submission (or a fraction thereof) in fairness to all students who submitted on time.

7. CHECKING, TESTING and GRADING SCHEME

   Your source code will be compiled, linked and tested using the GCC compiler/linker installed in our computer laboratory. We will use black box testing.

   Note that by default you will receive a perfect grade of 100%. It will remain 100% if there are no warnings, syntax errors, semantic errors, and that you complied properly with all the MP specifications and instructions.

   WARNING: A program with syntax error will automatically be given a grade of ZERO.

8. HONESTY POLICY

   - HONESTY policy applies. You should work on this machine problem independently. Consulting/discussing the idea/solution to the machine problem with another person (other than your teacher) is considered a breach of trust.
   - Please email/consult me if you have any question about this machine problem.

– end of this document –