CEN 333/CSI 333 – Programming at the Hardware-Software Interface – Fall 2016

Programming Assignment III

Date given: Oct. 24, 2016 **Due date:** Nov. 6, 2016
Weightage: 11%

This is a *team* project (except for those students who have opted to work on their own). The regular deadline for this assignment is **11 PM, Sunday, November 6, 2016**. With lateness penalty, the program will be accepted until **11 PM, Tuesday, November 8, 2016**. The assignment will *not* be accepted after that deadline.

Important notes: The C source file for the program must be named `p3.c`. This file must be submitted using the `turnin-csi333` command by the deadline specified above. *Each team must make only one submission.* Additional information about the `turnin-csi333` command will be included in the `README` file for this assignment.

You must follow the programming and documentation guidelines indicated in the syllabus. Specific requirements regarding the structure of the program are given later in this handout.

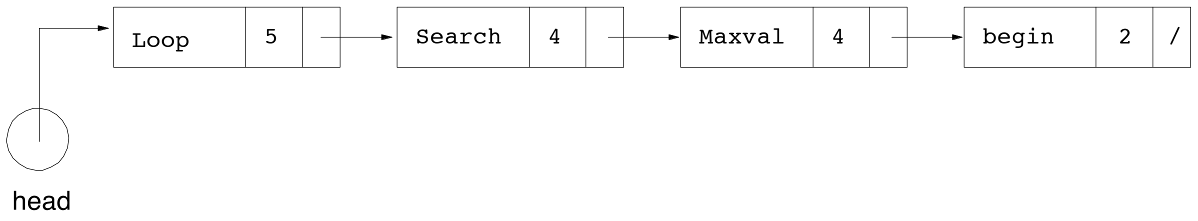
The total grade for the assignment is 100 points. The grading method for individuals and teams will be discussed in class.

You are required to write an interactive C program that prompts the user for commands, accepts commands from the keyboard (`stdin`) and executes those commands. When a command requires output, it must be written to `stdout`. The program must continue to accept and process commands until the user types the `end` command.

The program deals with linked lists. Each node of such a list contains a string of length at most 10, a positive integer (i.e., an integer value ≥ 1) and a pointer to the next node of the list. For any node, the string and the integer stored in that node will be referred to as the **symbol** and **count** for that node respectively. At all times, the list must satisfy the following two important properties.

1. The symbols appearing in the list are all *distinct*; that is, no two nodes have the same symbol.
2. When the list is scanned from left to right, *the counts must be in non-increasing order*.

An example of such a linked list is shown below.



Initially, we have an *empty* list. Some commands require your program to modify the list while others involve traversing the list to gather and print information about the list. The commands and their interpretations are as follows. (You should bear in mind that different parts of a command are separated by one or more spaces.)

(a) Insert Command: The syntax for this command is as follows:

`ins` *str*

Here, `ins` represents the name of the command and *str* represents a string. The interpretation of this command is as follows.

(i) If the list contains a node whose symbol is identical to the string specified in the command, then the count stored in the node must be incremented by 1. After this increment, if necessary, the node must be moved to an appropriate position in the list to ensure that the counts are in non-increasing order.

(ii) If the list does *not* contain a node whose symbol is identical to the string specified in the command, then a new node must be created. The symbol stored in the new node is the string specified in the command and the count stored in the new node must be 1. The new node must be inserted at the end of the list.

Note that the `ins` command does not produce any output.

(b) Delete Command: The syntax for this command is as follows:

`del` *str*

Here, `del` represents the name of the command and *str* represents a string. The interpretation of this command is as follows.

(i) If the list contains a node whose symbol is identical to the string specified in the command, then the count stored in the node must be decremented by 1. If the new count becomes 0, then the node must be *removed* from the list. If the new count is at least 1, the node must be moved, if necessary, to an appropriate position in the list to ensure that the counts are in non-increasing order.

(ii) If the list does *not* contain a node whose symbol is identical to the string specified in the command, then your program must leave the list unchanged.

Note that the `del` command does not produce any output.

(c) Forced Delete Command: The syntax for this command is as follows:

`fde` *val*

Here, `fde` represents the name of the command and *val* represents a positive integer value. The command must *remove* from the list, each node whose count is less than or equal to the integer value specified by *val*. If the list is empty or the count stored in each node is greater than the value specified by *val*, then your program must leave the list unchanged. Note that the `fde` command does not produce any output.

(d) Print Statistics Command: The syntax for this command is as follows:

```
pst
```

Here, `pst` represents the name of the command. If the list is empty, your program should simply print the message "`The list is empty.`". Otherwise (i.e., the list is non-empty), your program must compute and print the following quantities.

(a) The number of nodes in the list.

(b) The maximum count in the list.

(c) The minimum count in the list.

(d) The average count in the list. (Bear in mind that, in general, the average count is a *real number*.)

(e) Print List Command: The syntax for this command is as follows:

```
prl
```

Here, `prl` represents the name of the command. If the list is empty, your program should print the message "`The list is empty.`". Otherwise, your program should traverse the list (from left to right) and print each symbol and the corresponding count on a line by itself. (Thus, when the list is non-empty, the number of lines printed is the number of nodes in the list.)

(f) Print using Count Range Command: The syntax for this command is as follows:

`pcr` *v1* *v2*

Here, `pcr` represents the name of the command and *v1* and *v2* are non-negative integer values such that the value specified by *v1* is less than or equal to that specified by *v2*. If the list is empty, your program should simply print the message "`The list is empty.`". Otherwise, your program should traverse the list (from left to right); for each node whose count is in the integer range specified by *v1* and *v2* (i.e., the count is greater than or equal to the value specified by *v1* and less than or equal to the value specified by *v2*), the program must print the symbol and the count for that node on a line by itself. (Thus, when the list is non-empty, the number of lines printed is the number of nodes in the list whose counts are in the integer range specified by *v1* and *v2*.)

(g) Print Prefix Command: To discuss this command, we first note that a **prefix** is a substring that occurs at the *beginning* of a string. For example, the string "`val`" is a prefix of the symbol "`value`" and the string "`On`" is a prefix of the symbol "`Only`". (Note also that each string is a prefix of itself.) However, the string "`alu`" is *not* a prefix of the symbol "`value`" and the string "`ly`" is *not* a prefix of the symbol "`Only`".

The syntax for the print prefix command is as follows:

`ppr` *str*

Here, `ppr` represents the name of the command and *str* represents a string. If the list is empty, your program should simply print the message "`The list is empty.`". Otherwise, your program should traverse the list (from left to right); if the given string *str* is a prefix of the symbol stored in a node, then the command must print the symbol and the corresponding count on a line by itself. (Thus, when the list is non-empty, the number of lines printed is the number of symbols which have the string specified by *str* as a prefix.)

(h) Print Suffix Command: To discuss this command, we first note that a **suffix** is a substring that occurs at the *end* of a string. For example, the string "`ue`" is a suffix of the symbol "`value`" and the string "`y`" is a suffix of the symbol "`Only`". (Note also that each string is a suffix of itself.) However, the string "`va`" is *not* a suffix of the symbol "`value`" and the string "`nl`" is *not* a suffix of the symbol "`Only`".

The syntax for the command is as follows:

`psu` *str*

Here, `psu` represents the name of the command and *str* represents a string. If the list is empty, your program should print the message "`The list is empty.`". Otherwise, your program should traverse the list (from left to right); if the given string *str* is a suffix of the symbol stored in a node, then the command must print the symbol and the corresponding count on a line by itself. (Thus, when the list is non-empty, the number of lines printed is the number of symbols which have the string specified by *str* as a suffix.)

(i) End Command: The syntax for this command is as follows:

```
end
```

In response to this command, your program must stop.

Assumptions: In writing this program, you may assume the following.

(a) Symbols, prefixes, suffixes are all *case sensitive*. (Thus, the symbols "`value`" and "`Value`" are distinct. The string "`val`" is a prefix of the symbol "`value`" but not a prefix of the symbol "`Value`". Similar considerations apply to suffixes.)

(b) The command given by the user will be one of `ins`, `del`, `fde`, `pst`, `prl`, `pcr`, `ppr`, `psu` or `end`. (The command names are also case sensitive.)

(c) Each command will contain all and only the necessary arguments. (Thus, commands won't have missing or extraneous arguments.) Further, when a command has one or more arguments, the command name and the successive arguments will be separated by one or more spaces.

(d) Each string specified in a command will have a length of at least 1 and at most 10; further, the string won't include any whitespace characters.

(e) Integer values specified in commands will be non-negative; further, in the `pcr` command, the value specified by *v1* will be less than or equal to that specified by *v2*.

Thus, there is no need to deal with any erroneous commands. Your program should continue to prompt the user and process commands until the user types the `end` command.

Program Outline: An outline for your program is as follows.

```
1. Prompt the user for a command.
2. Read the command.
3. while (command is not "end") {
      (a) Read the value(s) for the command, if necessary.
      (b) Process the command.
      (c) Prompt the user for the next command.
      (d) Read the next command.
   }
```

Structural Requirements: In addition to `main`, you must have a *separate* function to implement each of the commands `ins`, `del`, `fde`, `pst`, `prl`, `pcr`, `ppr` and `psu` described above. (You may have other functions in addition to these.)

Suggestions:

1. Use the `"%s"` format to read the command as a string into a `char` array of size 4. (Since each command is exactly three characters long and each string must be properly terminated using the `'\0'` character, the size of the character array must be 4.)

2. Use the `"%s"` format to read the string specified as an argument to the commands `ins`, `del`, `ppr` and `psu`. (You may use a `char` array of size 11 to store such a string.)

3. Use the `"%d"` format to read the integer value(s) specified as argument(s) to the commands `fde` and `prc`.

4. Use the `strcmp` function in the string library (`<string.h>`) to identify which command is specified.