CSI 333 – Programming at the Hardware-Software Interface – Spring 2017  
Project III

The total grade for the assignment is 100 points.

You must follow the programming and documentation guidelines available in the Blackboard module Projects.

This is a team project (except for those students who have opted to work on their own). Group of two students may work on this project together.

# Description:

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.

Loop

5

Search

4

Maxval

4

begin

2

/

HEAD

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.)

1. **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.

1. 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.
2. 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.

1. **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.

1. 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.
2. 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.

1. **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.

1. **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.

1. The number of nodes in the list.
2. The maximum count in the list.
3. The minimum count in the list.
4. The average count in the list. (Bear in mind that, in general, the average count is a *real number.*)
5. **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.)

1. **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*.)

1. **Print Prefix Command:** The syntax for the print prefix command is as follows:

ppr str

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”.

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.)

1. **Print Suffix Command:** The syntax for the command is as follows:

psu *str*

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”.

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.)

1. **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.

1. 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.)
2. 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.)
3. 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.
4. 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.
5. 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:

1. Prompt the user for a command.
2. Read the command.
3. While command is not "end":
   1. Read the value(s) for the command, if necessary.
   2. Process the command.
   3. Prompt the user for the next command.
   4. 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.
5. Use I/O redirection facility of Unix while testing your program.
6. Use fflush(stdout) after each call to printf.

# Electronic Submission:

There are two mandatory submissions for each project. Both submissions are the same files but to be used for different purposes. You may do both submissions at one go:

* source code for the evaluation using turnin-csi333 on the ITS Unix machine,
* source code for the record – the standard Blackboard procedure for the assignments.

For the team project each team must make only one turnin-csi333 submission. That is, in each team, ONLY ONE member must do this. Team submissions must include additional documentation in the source file as explained below.

But the Blackboard submission must be done individually by EACH STUDENT even for the team project.

Important Notes: ignoring any of the following rules will result in penalty or even ZERO grade for the project.

* 1. For Project 3 you must turn in the file **named** “**p3.c”**.
  2. At the top of your C source file the following information must appear in the form of comments:
     1. For students working by themselves: You must have the following information at the beginning of your source file (p3.c) in the form of comments: your name, your Unix login id and the day/time of your weekly discussion class.
     2. For students working in a team: You must have the following information at the beginning of your source file (p3.c) in the form of comments:
        1. The names and UNIX login IDs of the two team members and the day/time of the weekly discussion class.
        2. A clear explanation of how the work for the project was divided among the two team members. Indicate clearly who developed each function and how the testing work was divided between the team members.
  3. Make sure that your programs compile and produce correct results on the UNIX machines (itsunix.albany.edu) supported by Information Technology Services (ITS) unit of UAlbany. Programs that cause compiler or linker errors on the ITS UNIX machines will NOT receive any credit.
  4. Using the turnin-csi333 program as discussed below is the ONLY acceptable way of submitting programming assignments in this course. You should NOT email the files to the instructor or to the TAs.
  5. Remember that you must submit only your C source files. DON'T turn in unnecessary files (e.g. object files with extension “.o” created by compiling C source files, executable files such as “a.out”, etc.).

To submit your files electronically, you must have the source files on one of the ITS UNIX machines. For this project the file p2.c must be in your working directory and you must be logged on to one of those machines to actually carry out the electronic submission.

To perform submission you should type the following command to the UNIX operating system:

/usr/local/bin/turnin-csi333 -c csi333 -p hw3 p3.c

After you issue the above command, the system responds with:

The sections of csi333 are:

FR\_1025

FR\_1235

FR\_0140

WE\_1235

Enter your section:

Depending upon the day and time of your weekly discussion section, you would type the appropriate section. For example, if your discussion class meets on Wednesdays at 12:35 PM, you would type WE\_1235 followed by the return key. The system will then respond with

Your files have been submitted to csi333, hw3 for grading.

In the above message, "hw3" refers to the name of the project that is currently active. If you submit your program during two days after due date the name of the project will be "hw3-late". Attempts to submit the program after the two day grace period will fail.

Additional information about the turnin program:

* 1. If you use the turnin command above again at a later time (before the deadline), then the files submitted previously would be completely replaced by the newly submitted files.
  2. At any time, you can obtain the names of the files that you have submitted to the current project using the following command:

/usr/local/bin/turnin-csi333 -c csi333 -v

# Some sample data to test your program:

Important Note: Some sample inputs that can be used to test your programs are given below. However, you should remember that when we compile and run your source files, we will use other data. Just because your programs work for the sample inputs given below, you shouldn't assume that they will work for all inputs. Therefore, you should test your programs thoroughly with other input values.

Attached there is sample test input for this program in the file t1.txt. The file contains a sequence of commands to be executed by your program.

The file t1\_with\_output.txt shows respectively the commands in t1.txt along with the correct output for each command that produces output. (Recall that some of the commands don't produce any output.) In the file that shows the outputs, a blank line has been inserted between successive commands to make it easier to see the outputs for each command. Your programs need not include such blank lines.

Important Note:

* 1. You should use Unix command to have the sample test inputs and outputs mentioned above available in your working directory.
  2. Remember that your program MUST read its input from stdin and produce its output to stdout. Use the I/O redirection facility of Unix while testing your program.
  3. It is very important to construct many sequences of commands and thoroughly test your programs. Be sure to test your program with sequences of commands so that the list becomes empty in the middle of a command sequence and the remaining commands in the sequence allow the list to grow.

# Program Grading:

Programs will be graded using a script written by the TAs. The script will compile your source program, generate the executable version and run the executable on new test data. The TAs will grade the version that you submit; once the submission is closed, you won’t be allowed to make any changes to your program.

If the ins, del and fde commands (which can modify the list) don’t work correctly, the answers for all subsequent commands are likely to be incorrect. Test your code thoroughly. If your program crashes during the middle of a sequence of commands, you won’t get credit for any of the subsequent commands in that sequence.

For students working individually:

1. Correctness: 85 points
2. Str. & doc.: 15 points

For students working in a team:

1. Correctness: 65 points
2. Str. & doc.: 15 points
3. Team work: 20 points

Each team member must participate in developing, documenting and testing the program. Each team should include additional documentation at the beginning of the source file indicating how the work for the project was divided between the two team members. (Indicate clearly who developed each function and how the testing work was divided between the team members.) After the submission deadline, each team must meet with their TA. During the meeting, the TA will ask questions about the team’s program and determine the points for team work. (The two team members may receive different scores for team work.)

# Example of program execution:

unix2> p3.out

Command? ins Loop

Command? ins Search

Command? ins begin

Command? prl

Loop 1

Search 1

begin 1

Command? ins begin

Command? pcr 2 3

begin 2

Command? ins Loop

Command? ins Long

Command? ppr Lo

Loop 2

Long 1

Command? ins Starch

Command? psu arch

Search 1

Starch 1

Command? pst

No. of nodes = 5

Max. count = 2

Min. count = 1

Avg. count = 1.4

Command? fde 1

Command? del begin

Command? prl

Loop 2

begin 1

Command? end

unix2>