CSI 333 – Programming at the Hardware-Software Interface – Fall 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.

# 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 255, 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 **text** and the **index** for that node respectively. Initially, the list is *empty*. At all times, the existing list must satisfy the following *requirements*:

1. The index is a number of the node in the list, i.e. the first node has index 1, and when the list is scanned from the beginning to the end, the value of indexes is increasing by 1.
2. The texts appearing in the list are all *distinct*; that is, no two nodes have the same text.

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. **Command *Insert After*:** The syntax for this command is as follows:

ina *num str*

Here, ina represents the name of the command, *num* represents a positive integer number, and *str* represents a text. The interpretation of this command is as follows.

1. A new node with the text specified in the command must be inserted in the list after a node whose index is equal to the number specified in the command, indexes of the list should be changed to keep increasing order, and the following message must be printed “Ok”.
2. If the list contains a node whose text is identical to the text specified in the command, then no new node must be created and the following message must be printed “Such text exists already”. **pass**
3. If the list does *not* contain a node whose index is equal to the number specified in the command, then a new node must be inserted at the end of the list and the following message must be printed “Text inserted at the end”. **pass**
4. **Command *Insert Before*:** The syntax for this command is as follows: **pass**

inb *num str*

Here, inb represents the name of the command, *num* represents a positive integer number, and *str* represents a text. The interpretation of this command is as follows.

1. A new node with the text specified in the command must be inserted the list *before* a node whose index is equal to the number specified in the command, indexes of the list should be changed to keep increasing order, and the following message must be printed “Ok”. **pass**
2. If the list contains a node whose text is identical to the text specified in the command, then no new node must be created and the following message must be printed “Such text exists already”. **Pass**
3. If the list does *not* contain a node whose index is equal to the number specified in the command, then a new node must be inserted at the beginning of the list and the following message must be printed “Text inserted at the beginning”. **Pass**
4. **Command *Delete*:** The syntax for this command is as follows: **pass**

del *num*

Here, del represents the name of the command and *num* represents a positive integer number. The interpretation of this command is as follows.

1. If the list contains a node whose index is equal to the number specified in the command, then the node must be *removed* from the list, indexes of the list should be changed to keep increasing order, and the following message must be printed “Deleted”. **pass**
2. If the list does *not* contain a node whose index is equal to the number specified in the command, then the program must leave the list unchanged and the following message must be printed “No such index”. **pass**
3. **Command *Replace*:** The syntax for this command is as follows: **pass**

rep *num* *str*

Here, *num* represents a positive integer number, and *str* represents a text. The interpretation of this command is as follows.

1. If the list contains a node whose index is equal to the number specified in the command, then the node text must be *replaced* with the text specified in the command, and the following message must be printed “Replaced”. **pass**
2. If the list does *not* contain a node whose index is equal to the number specified in the command, then the program must leave the list unchanged, and the following message must be printed “No such index”. **pass**
3. **Print List Command:** The syntax for this command is as follows: **pass**

prn

Here, prn 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 index and the corresponding text 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. **End Command:** The syntax for this command is as follows: **pass**

end

In response to this command, your program must stop.

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

1. The command given by the user will be one of ina, inb, del, rep, prn, or end. (The command names are case sensitive.)
2. 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.
3. Each string specified in a command won’t include any whitespace characters.

Thus, there is no need to deal with any erroneous commands; if such command is entered it should be ignored. 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. (If ina or inb entered when the list is empty, the first node will be created.)
   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 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 "%d" format to read the integer number specified as argument to the commands.
3. Use the "%s" format to read the string specified as an argument to the commands.
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 file 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 to keep your records – the standard Blackboard procedure for the assignments.

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 each of your C source file the following information must appear in the form of comments:
  3. your name,
  4. your Unix login ID,
  5. the name of your lab instructor and
  6. the day and time of your lab class.
  7. 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.
  8. 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.
  9. 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 p3.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:

turnin-csi333 -c csi333 -p hw3 p3.c

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

The sections of csi333 are:

WE\_0920

WE\_0415

FR\_1130

FR\_0140

MO\_1025

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 09:20 AM, you would type WE\_0920 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 should be "hw3-late":

turnin-csi333 -c csi333 -p hw3-late p3.c

Lateness penalty is 10 points per day. 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:

turnin-csi333 -c csi333 -v

* 1. You cannot submit your work if the project is closed. The project is starting from the date the homework is on Bb till the due date. To see the list of open/closed projects, use the following command:  
     turnin-csi333 -c csi333 -l

# 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.

Points: 85 points for correctness and 15 points for structure/documentation.

# Example of program execution:

unix%> p3.out

Command? inb 2 Loop

Text inserted at the beginning

Command? inb 2 Search

Ok

Command? ina 2 begin

Ok

Command? prn

1 Search

2 Loop

3 begin

Command? ina 10 begin

Such text exists already

Command? del 3

Deleted

Command? del 10

No such index

Command? inb 2 begin

Ok

Command? prn

1 Search

2 begin

3 Loop

Command? xyz

Command? end

unix%>