CSI 333 – Programming at the Hardware-Software Interface  
SQUPT, Spring 2019

**Project 3**

The total grade for the assignment is 100 points.

You must follow the programming and documentation guidelines (see file *Programming Assignments Requirements and Recommendations.docx*).

**Due date: 11:59pm Sunday, April 21, 2019**

# 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”.
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”.
4. **Command *Insert Before*:** The syntax for this command is as follows:

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 in 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”.
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”.
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, indexes of the list should be changed to keep increasing order, and the following message must be printed “Text inserted at the beginning”.
4. **Command *Delete*:** The syntax for this command is as follows: eww23

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”.
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”.
3. **Command *Replace*:** The syntax for this command is as follows:

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”.
2. If the list contains a node whose text is identical to the text specified in the command, then the program must leave the list unchanged, and the following message must be printed “Such text exists already”.
3. 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”.
4. **Print List Command:** The syntax for this command is as follows:

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 in the increasing order of indexes 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:

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 OS while testing your program.
6. Use fflush(stdout) after each call to printf.

# Example of program execution

The following examples assume that the executable version of the program is in the file p3.exe (if Windows).

> p3.exe

Command? inb 2 Loop

Text inserted at the beginning

Command? inb 2 Search

Text inserted at the beginning

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

>

# Submission

You must perform submissions as directed by your co-instructor.

*Ignoring any of the following rules will result in penalty or even ZERO grade for the project!*

Submission should include:

* A file named as directed by your co-instructor with source code for the project. More details will be given in your lab classes.
* Screenshot with program output.

At the top of your source code file the following information must appear in the form of comments:

* 1. Course code and title (i.e. “CSI 333. Programming at the Hardware-Software Interface”),
  2. Semester (e.g., Spring 2019),
  3. The name of your lab classes supervisor,
  4. Your class (e.g., ZR170102),
  5. Your student ID,
  6. Your pinyin name.

Make sure that your program compiles and produces correct results on the lab machines. Programs that cause compiler or linker errors on these machines will NOT receive any credit.

# Project Grading

Program will be graded by co-instructors. The total grade for this assignment is 100 points, with 85 points for correctness and 15 points for structure/documentation.