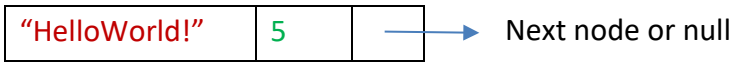
Part III Project description

A linked list of strings

Write an interactive C program that prompts users to select a choice by entering the corresponding command via keyboard(stdin). After the choice/command is entered, the program will execute with correct results and display the same menu until ENIT is selected. Outputs are written to screen(stdout). The following shows the six menu items. The corresponding commands will be discussed later.

A. INSERT AFTER: enter in this format: ***ina*** `index text` (e.g.: ***ina*** `1 "HelloWorld"`):
B. INSERT BEFORE: enter in this format: ***inb*** `index text` (e.g.: ***inb*** `1 "HelloWorld"`):
C. DELETE: enter in this format: ***del*** `index` (e.g.: ***del*** `2`):
D. REPLACE: enter in this format: ***rep*** `index text` (e.g.: ***inb*** `1 "HelloWorld"`):
E. PRINT: enter in this format: ***prn*** (e.g.: ***prn***):
F. EXIT: enter in this format: ***exit*** (e.g.: ***exit***):

A linked list must be designed and implemented to store a list of strings. A node contains three members:



- *text*: a string with a maximum length of 255(no spaces). The text must be unique. No two nodes will have the same text.
- *index*: an integer specifying the index of this node. The fist node has index of 1, the second node has an index of 2, …, etc.
- *next*: a pointer to the next node

The list is empty initially. The list can be manipulated by entering the following commands in the required formats:

A. **INSERT AFTER**: enter a command in this format: ***ina*** `index text(e.g.: ina 1 "HelloWorld"):`
The syntax for this command is ***ina*** `index text.`

   - `ina`: the name of the command **INSERT AFTER**
   - `index`: the position index after which the new node will be inserted
   - `text`: the text of the new node

   A new node containing `text` will be inserted after the node at position `index` after this command.
   a. If the new node isn't the last node, the indexes of all nodes after the new node must be adjusted accordingly. All index values of a linked list must be contiguous. "`Insertion After OK!`" should be printed.
   b. A new node with a duplicate text should not be inserted and "`Insertion of a duplicate failed!`" should be printed.
   c. If `index` is larger than the last index of the list, insert the new node at the end of the list and print "`Inserted at the end!`".

B. **INSERT BEFORE**: enter a command in this format: ***inb*** `index text(inb 1 "HelloWorld"):`
The syntax for this command is ***inb*** `index text.`

- `inb`: the name of the command **INSERT BEFORE**
- `index`: the position index before which the new node will be inserted
- `text`: the text of the new node

A new node containing `text` will be inserted before the node at position `index` after this command.

a. If the new node isn't the last node, the indexes of all nodes after the new node must be adjusted accordingly. All index values of a linked list must be contiguous. "`Insertion Before OK!`" should be printed.
b. A new node with a duplicate text should not be inserted and "`Insertion of a duplicate failed!`" should be printed.
c. If `index` is larger than the last index of the list, insert the new node at the front of the list and print "`Inserted at the front!`".

C. **DELETE**: enter in this format: ***del*** `index(e.g.:` ***del*** `2)`:
The syntax for this command is ***del*** `num.`

- `del`: the name of the command **DELETE**
- `index`: the position index from which the node will be removed

The node at position `index` will be removed after this command.

a. If the node isn't the last node, the indexes of all nodes after the new node must be adjusted accordingly. All index values of a linked list must be contiguous. "`Deletion OK!`" should be printed.
b. If `index` is larger than the last index of the list, the list remains unchanged. "`No changes!`" should be printed.

D. **REPLACE**: enter in this format: ***rep*** `index text` (e.g.: ***inb*** `1 "HelloWorld")`:
The syntax for this command is `rep index text`.

- `rep`: the name of the command **REPLACE**
- `index`: the position index from which the text of the node will be replaced
- `text`: the new text of the node at position `index`

The text of the node at position `index` will be replaced with a new text after this command.

a. If successful, display "`Replacement OK!`" should be printed.
b. If the list contains a node with the same text as the new text, the list remains unchanged. "`No changes!`" should be printed.
c. If `index` is larger than the last index of the list, the list remains unchanged. "`No changes!`" should be printed.

E. **PRINT**: enter in this format: ***prn*** (e.g.: ***prn***) :
The syntax for this command is `prn`.

- `prn`: the name of the command **PRINT**

The entire list of nodes are oriented in order after this command.

a. If the list is empty, display "`Empty List!`".
b. If the list is not empty, print all the nodes in order. Each node must be printed in a separate line with its index and text printed.

F. EXIT: enter in this format: ***`exit`*** (e.g.: ***`exit`***`)` **:**
The syntax for this command is `exit.` The program will be terminated after this command.

For this project, you can assume the following:

- Only the valid commands: `ina, inb, del, rep, prn, or exit` will be entered.
- The command names are case sensitive.
- Each command will be entered in correct syntax/format (name and arguments if any are correct.)
- If a command has one or more arguments, the name and the arguments are separated by one or more spaces.
- Each argument entered for `text` of a node doesn't include any whitespace characters. For example, a text containing multiple tokens such as "Hello World!" would be entered as a non-space string such as "HelloWorld!".

In addition to main, you **must** write a separate function along with possible helper functions for each command. Here are some suggestions:

- For a command,
  - use conversion specifier "%s" to read the name argument as a string into a char array of size 4(three characters of the command plus '\0' character).
  - Use conversion specifier "%d" to read the *index* argument as an integer.
  - Use conversion specifier "%s" to read the *text* argument as a string.
- Use function *strcmp* (<string.h>) to determine a command.
- Use I/O redirection when testing the program.
- Use *fflush*(stdout) after each call to *printf*.