Programming Assignment #2 – CSE 222 Winter 2019

In this assignment, you're going to implement a linked list using dynamic memory.

MAIN NODE STRUCTURE

Your linked list will be constructed from nodes, similar to what you used in PA1:

```
struct node{
  int data;
  struct node *next;
};
```

where

data is the integer actually stored in the list; and
next is the **memory location** of the next element, or NULL for the end of the list.

Declare a sentinel node pointer in your main program as follows:

```
struct node *list;
```

NECESSARY FUNCTIONS

You should implement the following functions:

struct node *init()

(this is the one function whose prototype changes from PA1)
This should return a pointer to an empty list. An empty list has a sentinel node pointing to NULL (indicating there are no other nodes in the list besides the sentinel).

int add(struct node*list,int number)

This should add a node the list, to store number in the list. The list should always be sorted, smallest to largest. The function returns 1 if successful, 0 if not successful (i.e. there is no memory left for creating a new node).

void print(struct node*list)

This routine will print the nodes in the linked list “list.” Since the nodes are stored in sorted order, the printed list should automatically appear in sorted order too.

int delete(struct node*list, int number)

This routine deletes the node containing “number” from the list “list.” This function returns 1 if the node as deleted, 0 if not (i.e. the node was not found in the list).

int search(struct node*list, int number)

This searches the linked list for a node containing the given number. It returns 1 if the node is found, 0 if it is not found.

The above routines represent the interface between a main program and the linked list as an abstract data type (ADT). They must behave exactly as described above, in the way described below.

Each element in the linked list (collection of struct nodes) contains 2 fields:

- a data field, which is the stored number; and
- a next field, which has the address of the next node, or the value NULL.

To add a number to the list, create a new node, search the list for the proper point to insert the new node, adjust the prior node's “next” field to point to the new node, and adjust the new node's “next” field accordingly.

Similar actions are taken for printing, searching and deleting, as discussed in class.

You should use malloc(sizeof(struct node)) to get the address of a new node; and free(ptr) to release a node from memory (after deleting it, or at the end of the program).

MAIN PROGRAM

You should use the above functions to implement a main program which lets the user manipulate a linked list. **It is important that the main program never manipulate the data structure directly**. All manipulation should be done via the above routines (this is the idea of encapsulation/information hiding/ADTs).

The first thing your program should do is initialize an empty linked list.

Next, the program should prompt the user for input using the prompt
>
and then wait for the user to enter a command followed by the ENTER key. The program then processes the command, as follows:

i number (where number is any legal integer).
This will insert the number into the list, in order (smallest to largest), provided the node is not already in the list. It should then print the message “SUCCESS”; “NODE ALREADY IN LIST”; or “OUT OF MEMORY.”

p
This will print the list in order, on one line, with the numbers separated by spaces

s number
This searches for the given number, and prints either “FOUND” or “NOT FOUND”

d number
This deletes the node containing the given number, and prints either “SUCCESS” or “NODE NOT FOUND”

x
This should cause the program to exit **after freeing all allocated memory** (use valgrind to confirm this)

(anything else)
should print a synopsis of legal commands.

The program should return to showing a prompt, and wait for another command.

You should catch errors such as

i haha

or

i

but don't worry about errors like

i 25 hahaha
(just insert 25 and ignore the hahaha)

of

i 384874238749823472387483274982374 98
(just use whatever sscanf returns)

DELIVERABLES

Your submission should include the following:

- all of the above routines, implemented as described;
- a main program which tests these routines, and operates as described above;
- plenty of commenting in your code. Specifically, for each function, you should describe its behavior, inputs, outputs, expectations, and so on. This should be explained in a set of comments appearing immediately before the function definition. Additionally, your code should be commented throughout.
- A comment block in the beginning of your main program, containing your name, class, date, assignment #, and a synopsis (in your own words) of what the program does.
- A makefile for creating an executable image

All of this should be put into a single .tar file, and uploaded to Canvas.