Programming Assignment #1 – CSE 222 Winter 2020

In this assignment, you're going to implement a linked list using an array of structures as described in class. There are several pieces to this assignment, and it's important that you do all the pieces as described (some of these may also appear as One Day Program assignments).

MAIN NODE STRUCTURE

Your linked list will be an array of struct nodes, where a node is defined as follows:

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

where

data is the integer actually stored in the list;
next is the index in the array of the next element, or -1 for the end of the list; and
valid is a flag: 1 means the node is valid, 0 means it is invalid (unused or deleted)

You should define a symbol “MYNULL” and set it equal to -1:

```
#define MYNULL -1
```

and use that in checking the .next field to see if a node is at the end of the list.

Declare a list in your main program as follows:

```
struct node list[100];
```

This defines a linked list that can hold up to 100 nodes.

NECESSARY FUNCTIONS

You should implement the following functions:

void init(struct node*ll)
This should initialize ll to be an empty linked list. An empty list has the node at index [0] pointing to MYNULL (indicating there are no other nodes in the list besides the sentinel). Additionally, the valid flag in all other nodes should be cleared.

int add(struct node*ll,int number)
This should add a node the ll, 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 space left in the array, **or the number is already in the list**).

void print(struct node*ll)
This routine will print the nodes in the linked list ll. Since the nodes are stored in sorted order, the printed list should automatically appear in sorted order too.

int delete(struct node*ll, int number)
This routine deletes the node containing “number” from the list ll. This function returns 1 if the node is deleted, 0 if not (i.e. the node was not found in the list).

int search(struct node*ll, 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 (array of struct nodes) contains 3 fields:

- a data field, which is the stored number;
- a next field, which has the index to the next node, or the value MYNULL; and
- a valid field, which is used to mark deleted nodes (valid=0).

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.

The following routines should also be implemented, and will support the above routines:

int get_node(struct node*ll)
This looks for the next free spot in the ll array (by checking the valid field of each entry); **sets that node's valid field to 1**; and returns the node's **index**, or MYNULL if there is no free space left. **You must use get_node when determining where to create a new entry in the linked list array. This will be important for PA2.**

void release_node(struct node*ll,int indexnumber)
This will delete the node **whose index is “indexnumber.”** While deleting a node, this routine should be called to clear the valid flag in the deleted node. **You must use release_node to reset the valid flag after deleting a node**. **This will be important for PA2.**

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 either the message “SUCCESS” or “NODE ALREADY IN LIST” or “OUT OF SPACE” if there is nowhere available to insert the number.

Since the add() function doesn’t distinguish between these last 2 cases, the main program should first call search(), and only then call add() if the number is not already in the list.

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

(anything else)

should print a synopsis of legal commands.

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

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 available to me in a GITLab repo named “LinkedList1” Make sure I have reporter access.