Dynamic Data Structure & Linked Lists

----

1 Write a program that asks the user how many characters will be entered, and then asks for each character. Use dynamic char array to store the characters, as taught in lecture 1. Write function `removes()` to remove the first character. Write `insert()` to insert a character entered by the user to the beginning of the array, and remove the last character in the array if the number of characters is more than the size of the array. Sample output:

```
How many characters do you want to input: 5
Input the string:datas
The string is: datas
Do you want to 1-insert or 2-remove or 3-quit?: 1
What is the character you want to insert: a
Resulting string: adata
Do you want to 1-insert or 2-remove or 3-quit?: 2
Resulting string: data
Do you want to 1-insert or 2-remove or 3-quit?: 1
What is the character you want to insert: q
Resulting string: qdata
Do you want to 1-insert or 2-remove or 3-quit?: 3
```

2. Write a C program `main()` that requests a list of numbers from the user, terminated by a sentinel value of '-1'. After the linked list is built, your program should print all numbers stored in the linked list. Sample output:

```
Enter a list of numbers, terminated by the value -1:
1
2
3
4
5
-1
Current list: 1 2 3 4 5
```

3 Based on the program `main()` written in 2, write a function `searchList()` that examines a linked list of integers to determine if a given integer value is stored within the list. `searchList()` is called in `main()` once the linked list is built and printed.

```
int searchList(ListNode *head, int value);
```

The function should return the index of the node containing value if it is found and -1 otherwise. Sample output: (Test application uses a sentinel value of -1)

```
Current list: 1 2 3 5 6
Enter value to search for: 5
Value 5 found at index 3
```

4 Based on the program `main()` written in 3, suppose that you are not using the linked list anymore, then you add the following code to the end of `main()` for freeing up the memory:

```
ListNode *p = head;
while (p!= NULL) {
   free(p);
   p=p->next;
}
```

This code could crash your program! (Even worse – might work on your system and fail on others). Explain why. Therefore, the correct code should use a temporary pointer. Rewrite the program for freeing up the memory using a temporary pointer.