UCS 1312 Data Structures Lab

A2: List ADT using pointers and its application

--Dr. R. Kanchana

Best Practices to be adapted

- Design before coding
- Modular design and coding using versions
- Verification of algorithm using Hand-trace
- Function signatures in the beginning and implementation in the end of the header file
- Selecting suitable programming constructs for iteration, avoiding break/global variables

^^^

Design the algorithm and implement in C. **(CO1, K3)**

1. Create an ADT for a List data structure using pointers, with the following functions (List.h):
   a. Create a linked list of names (Max. size of the string is 4)
   b. Create the following operations

insertLast (linklist, item)	to insert the item in the end
insertFirst (linklist, item)	to insert the item in the front
deleteMiddle(linklist, item)	to delete the first occurrence of the item
deleteFirst (linklist)	to delete the first node and return the item
deleteLast (linklast)	to delete the last node and return the item
insertMiddle(linklist, item1,item2)	to insert item2 after the last occurrence of item1
search (linklist, item)	to return TRUE/FALSE if found/not found
length (linklist)	to return the length of the list
getData (linklist)	to return the name in the node pointed by linklist

2. Use List.h and write an application (a2List.c) for the following:
   a. Write an application program in C that includes options (12 options) to read items of lists, to test each of the list operations and the application operation.
   b. Implement *reverse(linklist)* to return the reverse of the items in the given *linklist* without creating a new linked list
   c. Implement *createSorted (linklist, item)* that inserts an item in the list such that all the items in the list are in lexically ascending order
   d. Write a function *display(linklist)* to display the items in the list
   e. Write a function rotateLeft(linklist) that rotates the items to the left
   f. Test your program with the test data given below and check your results with the expected result

Input Linked list	Operation	Expected output
l1 = Null	*display(l1)*	No names
	length(l1)	0
	insertLast(l1,"cat")	cat
	insertLast(l1,"mat")	cat mat
	insertFirst(l1,"rat")	rat cat mat
	insertLast(l1,"cat")	rat cat mat cat
	insertMiddle(l1,"cat","bat")	rat cat mat cat bat
	search(l1,"mat")	Found

	search(l1,"sat")	Not found
	getData(l1)	rat
	length(l1)	5
	l2=reverse(l1)	l2 = bat cat mat cat rat l1 = rat cat mat cat bat
	deleteMiddle(l1,"cat")	bat mat cat rat
	deleteFirst(l1)	mat cat rat
	deleteLast(l1)	mat cat
	createSorted(l1,"pat")	mat cat pat
	createSorted(l1,"eat")	eat mat cat pat
	createSorted(l1,"ant")	ant eat mat cat pat
	rotateLeft(l1)	eat mat cat pat ant

3. Define an ADT for Polynomial using linked list with the following operations (Poly.h):
   a. *InsertOrder (poly, coefft, expt)* to insert the term with coefft and expt in the right position so that all the terms in the polynomial poly are in the order of the expt
   b. *Coefft(poly,expt)* to return the coefficient of the term with *expt*

4. Use Poly.h and write an application (a2Poly.c) for the following:
   a. Create a user interface to read the polynomial terms given by the user
   b. Implement display(poly) to display all the terms in the polynomial - display it in the form of a X n + b X n-1 + …+ c
   c. Implement *sumPoly(poly1, poly2)* to return a polynomial that is the sum of poly1 and poly2
   d. Test your program with the following test cases

Input	Operation	Expected output
poly1 = 2x5-7x6+4x2 poly2=3x2+6x6-8	p3= sumPoly (poly1, poly2)	p3=-x6+2x5+7x2-8 poly1= -7x6+2x5+4x2 poly2=6x6+3x2-8
poly1 = 3x poly2 =null	p3= sumPoly (poly1, poly2)	p3=3x poly1=3x; poly2=null

****