ESE 2025 - Week 6 Report - LinkedList
Instructor: dr. Takis Zourntos
Student: Vy Nguyen

Introduction:
This report based on the starter code from dr. Takis Zourntos to build a linked- list library.
The source code can be found on:
https://github.com/takiszourntos/teaching/tree/master/lambton/2020/summer/ese2025/week_6/workspace/linked_list_lib

Discussion:
The tasks are to create two functions: inserting a node in the middle of the linked list and deleting a selection node.
The algorithm of inserting node pseudo-code:

set an index i
while (i < insertion point - 1) // a while loop is to find the position.
pointer points to next node
increment of i++
end while loop
assigns an after-Node from the previous pointer
points the previous node pointer to the new node.
points the new node pointer to after - Node.
return the new node.

C code:

```
_t* insertNode(ll_t *pHead, data_t payload, data_key_t insertionPoint)


ll_t *node;
ll_t *afterNode;
ll_t *point;
node = createNode();                // create a new node
setPayload(node, payload);          // set up data to new node
point = pHead;                              // set up a temperate point
size_t i = 0;
        while(i < insertionPoint-1)  // finding the position of the insertionPoint
        {
                point = point->pNext; //     skip the unwanted position
                i++;
        }
        afterNode = point->pNext;    //     node after insertionNode
        point->pNext = node;         //     insert the new node
        node->pNext = afterNode;     // links next pointer of new node to afterNode

return node;
```

Pseudocode for deleting a node:

check if the list is empty
set an index i = 0

while (i < deleting node - 1)
pointer point to next node
increment of i++
end while loop
check if the position is larger than the number of nodes.
assign a node after deleted node
free memory of the node in the deleted position
assigns the node after deleted node to next pointer
return

C code:

```
void deleteNode(ll_t *pHead, data_key_t nodeToDeleteKey)
{
	ll_t *nextPoint;
	ll_t *temp;                                    // Temperate node.
	size_t i = 0;
	temp = pHead;
	if (pHead == NULL)                          // check if linked list is empty
	{
		printf("Empty linked list");
		return;
	}
	//find the position of the node to delete.
	while (i != nodeToDeleteKey-1)      // position of the deleting node
	{
		temp=temp->pNext;                   // skipping unwanted node
		i++;
	}
	// position is larger than no. of nodes
	if (temp == NULL || temp->pNext == NULL)
		return;
	nextPoint = temp->pNext->pNext; // node after deleted node
	// Unlink the node from linked list.
	free(temp->pNext);                          // free memory
	temp->pNext = nextPoint;            // assign nextPoint to next pointer
	return;
}
```

Summary:

This assignment is used to practice working with a linked list. Recommend improving on inserting a node in a different position (first, last) and check if the list is empty.

Appendix

test_ll.c:

```
/*
 ============================================================================
 Name        : ese2025week6.c
 Author      : takis
 Modified by : Vy
 Version     :
 Copyright   : Your copyright notice
 Description : , Ansi-style
 ============================================================================
```

```
 */


#include <stdlib.h>
#include <stdio.h>
#include "ll.h"

int main(void)
{
	ll_t *pLLHead=NULL; // pointer to list, must be initialized to NULL
	data_t token;

	// create the linked list from standard input;
	// user indicates end of data by entering "9999"
	// for X, Y and key values.
	printf("\nLoading data...\n");
	scanf("%lf %lf %lu", &token.X, &token.Y, &token.key);
	pLLHead = addNode(pLLHead, token);
	int size;
	while (token.X != 9999 && token.Y != 9999 && token.key != 9999)
	{
		scanf("%lf %lf %lu", &token.X, &token.Y, &token.key);
		addNode(pLLHead, token);
		size++;
	}

	printf("     ... done.\n\n");

	// send linked list to standard output
	printf("\nPrinting the entire linked list to standard output:\n");
	ll_t *pW = pLLHead;
	while (pW != NULL)
	{
		token.X = pW->payload.X;
		token.Y = pW->payload.Y;
		token.key = pW->payload.key;
		printf("%lf %lf %lu\n", token.X, token.Y, token.key);
		pW = pW->pNext;
	}

	// terminate
	printf("\n\n ...done!\n\n");


	// Insert node function (middle)
	data_key_t insertPoint ;
	char ans;
	printf("Do you want to insert a node? ");
	for(;;)
	{
		scanf("%c",&ans);
		if(ans == 'y'|| ans =='Y')
		{
```

```
                printf("Enter the position you want to insert:");
                scanf("%u",&insertPoint);
                printf("Enter your data point:");
                scanf("%lf %lf %lu", &token.X, &token.Y, &token.key);
                insertNode(pLLHead, token, insertPoint);
                ll_t *pW = pLLHead;
                while (pW != NULL)
                {
                        token.X = pW->payload.X;
                        token.Y = pW->payload.Y;
                        token.key = pW->payload.key;
                        printf("%lf %lf %lu\n", token.X, token.Y, token.key);
                        pW = pW->pNext;
                }
                break;
        }
        else if(ans == 'n' || ans =='N')
        {
                printf("Bye bye\n");
                break;
        }
        else
                printf("Do you want to insert a node? Please answer y or n\n");
}


// Deleting a node
data_key_t deletingNode;
char del;
for (;;)
{
        scanf("%c",&del);
        if (del == 'y'|| del =='Y')
        {
                printf("Enter the node position you want to delete: \n");
                scanf("%u",&deletingNode);
                deleteNode(pLLHead,deletingNode);
                ll_t *pW = pLLHead;
                while (pW != NULL)
                {
                        token.X = pW->payload.X;
                        token.Y = pW->payload.Y;
                        token.key = pW->payload.key;
                        printf("%lf %lf %lu\n", token.X, token.Y, token.key);
                        pW = pW->pNext;
                }
                break;
        }
        else if (del == 'n'|| del == 'N')
        {
                break;
        }
        else
```

```
                        printf("Do you want to delete a node?, Enter y or n\n");
        }
        return 0;
}
```

ll.c:

```
/*
 * ll.c
 *
 *  Created on: Jul. 13, 2020
 *       Author      : takis
 *        Modified by : Vy
 */

#include <stdlib.h>
#include "ll.h"

/*
 * setPayload():
 */
void setPayload(ll_t* node, data_t payload)
{
        node->payload.X = payload.X;
        node->payload.Y = payload.Y;
        node->payload.key = payload.key;
}

/*
 * createNode():
 *
 */
ll_t* createNode(void)
{
        /* create a pointer for the new node */
        ll_t *node;

        /* allocate the node from heap */
        node = (ll_t*) malloc(sizeof(struct linkedList));

        /* make next point to NULL */
        node->pNext = NULL; //


        /* return the pointer to the new node */
        return node;
}

/*
 * addNode():
 *
 */
ll_t* addNode(ll_t *pHead, data_t payload)
{
```

```
        /* create two node pointers */
        ll_t *pNode;
        ll_t *pW;

        /* prepare the new node to be added */
        pNode = createNode();
        setPayload(pNode, payload); /* set the new element's data field to value */

        if (pHead == NULL)
        {
                pHead = pNode; /* if the linked list has no nodes to begin with */
        }
        else
        {
                /* search through list until tail node is found */
                pW = pHead;
                while ((pW->pNext) != NULL)
                {
                        pW = pW->pNext;
                }
                /* set the pointer from NULL to temp */
                pW->pNext = pNode;
        }
        return pHead;
}

/*
 * insertNode():
 *
 */
ll_t* insertNode(ll_t *pHead, data_t payload, data_key_t insertionPoint)
{

        ll_t *node;
        ll_t *afterNode;
        ll_t *point;
        node = createNode();                // create a new node
        setPayload(node, payload);          // set up data to new node
        point = pHead;                              // set up a temperate point
        size_t i = 0;
                while(i < insertionPoint-1)  // finding the position of the insertionPoint
                {
                        point = point->pNext; //     skip the unwanted position
                        i++;
                }
                afterNode = point->pNext;    //     node after insertionNode
                point->pNext = node;         //     insert the new node
                node->pNext = afterNode;     // links next pointer of new node to
afterNode

        return node;
}
```

```
/*
 * deleteNode():
 *
 */
void deleteNode(ll_t *pHead, data_key_t nodeToDeleteKey)
{
	ll_t *nextPoint;
	ll_t *temp;                                          // Temperate node.
	size_t i = 0;
	temp = pHead;
	if (pHead == NULL)                          // check if linked list is empty
	{
		printf("Empty linked list");
		return;
	}
	//find the position of the node to delete.
	while (i != nodeToDeleteKey-1)      // position of the deleting node
	{
		temp=temp->pNext;                   // skipping unwanted node
		i++;
	}
	// position is larger than no. of nodes
	if (temp == NULL || temp->pNext == NULL)
		return;
	nextPoint = temp->pNext->pNext; // node after deleted node
	// Unlink the node from linked list.
	free(temp->pNext);                          // free memory
	temp->pNext = nextPoint;            // assign nextPoint to next pointer
	return;
}
```

ll.h:

```
/*
 * ll_t.h
 *
 *  Created on: Jul. 13, 2020
 *      Author: takis
 *      Modified by Vy
 */

#ifndef DSTRUCTS_LL_H_
#define DSTRUCTS_LL_H_

#include <stdlib.h>

/*************************************************************
 * NODE STRUCT GOES HERE
 *************************************************************/
// nodes can be referenced by a KEY, of this type

typedef unsigned int data_key_t;

// nodes contain a PAYLOAD consisting of various elements and the key
```

```
struct data_struct
{
        double X;
        double Y;
        data_key_t key;
};
typedef struct data_struct data_t;

// actual node struct/typedef
struct linkedList
{
        data_t payload;
        struct linkedList *pNext; // recursively defined "next" pointer
};
typedef struct linkedList ll_t;

/************************************************************
 * USEFUL LINKED LIST FUNCTIONS
 ************************************************************/
/*
 * setPayload():
 *
 *             for the node pointed to by the first argument, this function
 *             sets the value of the node's payload to the function's
 *             second argument
 *
 */
void setPayload(ll_t*, data_t);


/*
 * createNode():
 *
 *             creates a node of type ll_t from the heap, and returns
 *             a pointer to this newly created node; sets the node's own
 *          pNext pointer to NULL;
 *
 */
ll_t* createNode(void);

/*
 * addNode():
 *
 *             adds a new node (with payload given by second argument) to the
 *             bottom/back of the list referenced by the pointer, head;
 *             if head==NULL, a new list is created, and the new head pointer
 *             is returned;
 *
 */
ll_t* addNode(ll_t*, data_t);

/*
 * insertNode():
```

```
 *
 *             for a list with head pointer given by the first argument, this
 *             function inserts a new node with payload given by the second argument
 *             at the point just BEFORE the node whose key matches the third
 *             argument; if head pointer returns NULL, a new list is created, and
 *             the new head pointer is returned;
 *
 */
ll_t* insertNode(ll_t*, data_t payload, data_key_t insertionPoint);

/*
 * deleteNode():
 *
 *             for a list with head pointer given by the first argument, this
 *             function deletes the node whose key data matches the second argument;
 *             if head pointer returns NULL, an error is generated, indicating that
 *             the node is not found;
 *
 */
void deleteNode(ll_t*, data_key_t);


#endif /* DSTRUCTS_LL_H_ */
```