<u>ESE_2025: EMBEDDED OPERATING SYSTEMS</u>
<u>LINKED LIST LIBRARY</u>

Student Name & ID: **ANNA JOY (C0769402)**

INTRODUCTION

In this exercise, we need to build our own linked list library by completing the insertnode() and deletenode() functions and should adhere to the conventions already established in the provided code base.

DISCUSSION

The linked lists and arrays can be compared together to get a better idea of their operations. Arrays are of fixed size and it cannot be changed when it is needed. But in the case of linked lists, the size can be changed whenever needed.
The task here is to write, and then demonstrate the functioning of, the functions insertNode() and deleteNode(). Please note that you have been provided with a data file for the code. Its use is described in the comments of test_ll.c.

The insertNode() function is to insert a node in between the existing linked list. Each node in the linked list will be pointing to the next node and if the pointer is showing "NULL", then it indicates the endpoint of the linked list. So let us assume A, B, C, and D are consecutive nodes of a linked list. If we need to insert a node 'X' between B and C, these steps are to be followed;
Initially, B points to C, and to insert X between them, B should point to X and X should point to C canceling the link between B and C.
<u>Function for inserting</u>

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

    ll_t *temp,*prev,*newn;
    temp=pHead;
    prev=pHead;
    newn=createNode();
    setPayload(newn,payload);

    if(pHead==NULL){
      pHead=newn;
      return pHead;
    }
    while(temp!=NULL){
    if(insertionPoint==temp->payload.key){

      // head special case
           if (prev==temp){
           newn->pNext=pHead;
           pHead=newn;
      }else{
           newn->pNext=temp;
           prev->pNext=newn;
      }
    }
    prev=temp;
    temp=temp->pNext;
    }

    return pHead;
}
```

The deleteNode() function is to cancel a node inside a linked list. If A, B, C, and D are nodes of a linked list, and C needed to be deleted, then node B should point to D and C should let free.

<u>Function for deleting</u>

```c
void deleteNode(ll_t *pHead, data_key_t nodeToDeleteKey)
{
    ll_t *temp,*prev;

    temp=pHead;
    prev=pHead;

    if(pHead==NULL){
      printf("the node is not found");
      return ;
    }

    while(temp!=NULL){
      if(nodeToDeleteKey==temp->payload.key){

           // head special case
           if (prev==temp){
                if (temp->pNext!=NULL){

                     pHead=pHead->pNext;
                     free(temp);
                }else{
                     free(temp);
                     pHead=NULL;
                }
           }else{
                prev->pNext=temp->pNext;
                free(temp);
```

```
          }
      }
    prev=temp;
    temp=temp->pNext;
    }

    return;
}
```

CONCLUSION

The linked list library is built and adhered to conventions established in the starter code.

APPENDIX

Header file link.h

```
/*
 * ll.h
 *
 *  Created on: Jun. 29, 2020
 *   Author: takis
 */

#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_ */
```

Source code linked.c

```
#include <stdlib.h>
#include "link.h"
#include <stdio.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 *temp,*prev,*newn;
    temp=pHead;
    prev=pHead;
    newn=createNode();
    setPayload(newn,payload);

    if(pHead==NULL){
      pHead=newn;
      return pHead;
    }
    while(temp!=NULL){
```

```
    if(insertionPoint==temp->payload.key){

      // head special case
           if (prev==temp){
           newn->pNext=pHead;
           pHead=newn;
      }else{
           newn->pNext=temp;
           prev->pNext=newn;
      }
    }
    prev=temp;
    temp=temp->pNext;
    }

    return pHead;
}

/*
 * deleteNode():
 *
 */
void deleteNode(ll_t *pHead, data_key_t nodeToDeleteKey)
{
    ll_t *temp,*prev;

    temp=pHead;
    prev=pHead;

    if(pHead==NULL){
      printf("the node is not found");
      return ;
    }

    while(temp!=NULL){
      if(nodeToDeleteKey==temp->payload.key){

           // head special case
           if (prev==temp){
               if (temp->pNext!=NULL){

                    pHead=pHead->pNext;
                    free(temp);
               }else{
                    free(temp);
                    pHead=NULL;
               }
           }else{
               prev->pNext=temp->pNext;
               free(temp);
           }
      }
    prev=temp;
    temp=temp->pNext;
    }

    return;
}
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);
    while (token.X != 9999 && token.Y != 9999 && token.key
!= 9999)
    {
      scanf("%lf %lf %lu", &token.X, &token.Y, &token.key);
      addNode(pLLHead, token);
    }
    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");
    return 0;
}
```