רשימות מקושרת - המשך

<u>נושא למצגת זו :</u>

1. שימוש במבנה מנהל עבור רשימה מקושרת
2. עידכון מצביע לראש הרשימה
3. חלוקת התכנית ל-3 קבצים (יצירת פרויקט)
4. רשימה דו-כייונית

רשימות מקושרות

הוספת צומת לסוף הרשימה:

כדי להוסיף צומת לסוף הרשימה, **חובה** להשתמש במצביע לסוף הרשימה בנוסף למצביע לראש הרשימה כדי לבצע את הפעולה בצורה יעילה. **אין לסרוק את הרשימה כל פעם כדי להגיע לסוף.**

מגדירים את המצביעים בדרך כלל בפונקציה ראשית בצורה הבאה :

struct node*head = NULL,*tail = NULL

```
void add(struct node *head, struct node *tail, struct node
    *new_node )
{
  new_node->next = NULL;          /if not done before*/
  if( head == NULL )
     head = new_node;
  else
     tail->next = new_node;
  tail = new_node;
}
```

האם בחזרה לפונקציה ראשית head ו-tail המקוריים התעדכנו? התשובה - לא! מה עושים?

פתרון 1 – להשתמש במצביע כפול (כתובת של מצביע)

פתרון 2 – בדף הבא..

רשימות מקושרות

מבנה מנהל לניהול הרשימה:

רצוי להגדיר מבנה לניהול הרשימה שיכלול את הדברים שקשורים לרשימה כולה: מצביע לראש הרשימה, מצביע לסוף הרשימה (אם דרוש) ,מספר האיברים הנוכחי ברשימה (אם נדרש) וכו'.

מבנה מנהל יכול להראות כך:

```
typedef struct List
{
        struct Node   *head;
        struct Node   *tail;
        int count;
}  List;
```

<u>יתרונות :</u>

1. העברה נוחה יותר של רשימה לפונקציה.

2. המצביעים head ו-tail מתעדכנים ללא צורך במצביע כפול.

שימוש במבנה מנהל עבור רשימה מקושרת

```
#include <stdio.h>
#include <stdlib.h>

typedef struct item {
            int code;
            char name[20];
            int price;
            int stock;
            struct item *next;
          } Item;

typedef struct list
{
          Item * head;
          Item *tail;
          int size;          /*number of elements in list*/
} List,*PList;
```

זהו מבנה מנהל. מכיל דברים שקשורים לכל הרשימה ולא לכל צומת בנפרד.

הגדרת טיפוס של מצביע למבנה.
ניתן להגדיר משתנה בצורה כזאת :
PList p במקום List* p

שימוש במבנה מנהל עבור רשימה מקושרת

```
/*Insert the nodes to the head of linked list*/
void BuildListFromHead( PList s_l )
{
    Item *temp;
    int j;
    for (j = 0 ; j < s_l->size; j++)
    {
      if (( temp = (Item*) malloc (sizeof(Item) ) ) == NULL)
     {
        printf("\nNot enough memory for the allocation");
        Delete (s_l);
        exit(1);
     }
      printf("\nEnter Item(code name price stock):");
     if(scanf("%d %s %d %d",&(temp->code),
       temp->name, &(temp->price), &(temp->stock))<4)
      {
         printf("Input error\n");
         free(temp);
         Delete (s_l);
         exit(1);
     }
```

שימוש במבנה מנהל עבור רשימה מקושרת

```
  temp->next = s_l->head;
  s_l->head = temp;            /*update the head*/
 }
}
```

```
/*Insert the nodes to the tail of linked list*/
void BuildListFromTail(PList s_l)
{
    Item *temp;
    int j;
    for (j = 0; j<s_l->size; j++)
    {
      if (( temp = (Item*) malloc (sizeof(Item) ) ) == NULL)
            {
                printf("\n Not enough memory for the
allocation");
                Delete (s_l);
                exit(1);
            }
```

שימוש במבנה מנהל עבור רשימה מקושרת

```
 printf("\nEnter Item(code name price stock):");
if (scanf("%d %s %d %d",&(temp->code),temp->name,
  &(temp->price),&(temp->stock))<4)
     {
            printf("Input error\n");
            free(temp);
            Delete (s_l);
            exit(1);
     }
    temp->next = NULL;
    if (s_l->head == NULL)      /* s_l->tail = NULL */
       s_l->head = temp;
   else
      s_l->tail->next = temp;
   s_l->tail = temp;                /*update the tail*/
   }
}
```

שימוש במבנה מנהל עבור רשימה מקושרת

```
/*Print all nodes information in linked list*/

void PrintItems(PList s_l)
{
    int i = 1;
    Item* temp = s_l->head;
    while (temp != NULL)
    {
       printf("\nItem no' %d code: %d name: %s ",i,
       temp->code, temp->name);
      printf ("\n price: %d stock: %d", temp->price,
      temp->stock);
      temp = temp->next;
      i++;
    }
}
```

שימוש במבנה מנהל עבור רשימה מקושרת

```
 /*Destroy the linked list */
void Delete (PList s_l)
{
    Item *temp;
    while (s_l->head != NULL)
    {
      temp = s_l->head;
      s_l->head = (s_l->head)->next;
      printf("\n Delete item code : %d ",temp->code);
      free(temp);
    }
}
```

שימוש במבנה מנהל עבור רשימה מקושרת

```
int main()
{
  List store_list;
  store_list.head = store_list.tail = NULL;
  store_list.size = 3;   /*Number of nodes in linked list*/
  BuildListFromHead(&store_list);
  PrintItems(&store_list) ;
  Delete (&store_list);

  BuildListFromTail(&store_list);
  PrintItems(&store_list) ;
  Delete (&store_list);
  return 0;
}
```

חלוקת התכנית ל-3 קבצים

כדי ליצור פרויקט, נחלק כל תכנית ל-3 קבצים בצורה הבאה:

example.h	example.c	main.c
#ifndef _example #define _example typedef, enum, define... ספריות,הכרזות על פונקציות, הגדרת מבנים #endif	#include ”example.h” מימוש כל הפונקציות	#include ”example.h” פונקציה main

בדף הבא ישנו הסבר על החלוקה. הסבר יותר מפורט יינתן בהרצאה.

חלוקת התכנית ל-3 קבצים

- נהוג לחלק תוכניות גדולות למספר קבצים, כאשר כל קובץ מטפל בנושא מסוים בתוכנית. **מודולריות** היא חלוקה של התוכנה למספר מודולים עפ"י נושאים.

- כל מודול כולל קובץ ממשק (example.h) וקבצי מקור - קובץ מימוש (example.c.) וקובץ main.c . בטבלה ניתן לראות מה כל קובץ אמור להכיל.

- קריאה לפונקציה ממודול אחד לשני מבוצעת ע"י הכללת (#include) קובץ הממשק שלו בשלב ההידור.

- בזמן הקישור, המקשר (linker) יוצר קובץ ביצוע (.exe) יחיד וכל קוד הקבצים משולב לתוכו.

- כדי למנוע הכללה מרובה של קבצי הממשק, משתמשים בטכניקה המשלבת הוראות תנאי של הקדם-מעבד :

#ifndef, #define, #endif.

אם עדיין לא הוגדר... if not define

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
#ifndef _LINKED_LIST_H
#define _LINKED_LIST_H
/* linked_list.h – header file.
   This header contains a basic linked-list library.
   Each list node contains an integer. The library
   supports both sorted and unsorted lists. */

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

/* declaration of node type */
typedef struct Node
{
    int val;
    struct Node *next;
} Node;

/* declaration of functions */

/* creating a new node with a given value */
Node* createNode (int value);

/* adding a value to an unsorted list */
Node* insertList (Node *head, int value);
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* adding a value to a sorted list */
Node* insertSortedList (Node *head, int value);

/* deleting a value from an unsorted list */
Node* deleteFromList (Node *head, int value);

/* printing the elements in a list */
void printList (Node *head);

/* freeing the memory of the list nodes */
void freeList (Node *head);

#endif        /* _LINKED_LIST_H */
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* linked_list.c - implementation file */

#include " linked_list.h"


/* creating a new node with a given value */
Node* createNode (int value)
{
        /* allocating the memory */
        Node *ptr = (Node*) malloc (sizeof(Node));

        if(ptr!=NULL)
        {           /* initializing */
            ptr->val = value;
            ptr->next = NULL;
        }
        return ptr;
}
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* adding a value to an unsorted list */
Node* insertList (Node *head, int value)
{
    /* creating a new node */
    Node *ptr = createNode (value);

    if(ptr == NULL)
    {
      freeList (head);
      printf("Allocation problem");
      exit(1);
    }
     /* adding the new node to the head of the list */
    ptr->next = head;
    head = ptr;

return head;
}
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* adding a value to a sorted list */
Node* insertSortedList (Node *head, int value)
{
  Node *ptr,*pCurrent,*pPrev;
  /* creating a new node */
  pCurrent = pPrev = head;

  ptr = createNode (value);
  if(ptr == NULL)
  {
     freeList (head);
     printf("Allocation problem");
     exit(1);
  }
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
    /* adding the new node to the correct place in the list */
    while ( pCurrent != NULL )
    {
       if(pCurrent->val > ptr->val)
          break;
      pPrev = pCurrent;
      pCurrent = pCurrent->next;
    }
    ptr->next = pCurrent;
    if(pCurrent == head)  /*first element or empty list*/
       head = ptr;
    else
       pPrev->next = ptr;

   return head;
}
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* deleting a value from an unsorted list */
Node* deleteFromList (Node *head, int value)
{
    Node *pCurrent ,*pPrev,*pNext;

    pCurrent = head;
    pPrev = NULL;
    /* finding the right node to delete */
    while ( pCurrent != NULL )
    {
      if(pCurrent->val == value)
      {
         pNext = pCurrent->next;
         free(pCurrent);                    /* delete node */
         if (pPrev == NULL)      /*It was a first element*/
            head = pNext;
         else
            pPrev->next = pNext;
        break;
      }
      else {
         pPrev = pCurrent;
         pCurrent = pCurrent->next;
      }
    }
      return head;  }
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* printing the elements in a list */
void printList (Node *head)
{
    /* going over the element and printing each one */
    while (head != NULL)
    {
        printf ("%d ", head->val);
        head = head->next;
    }
}

/* freeing the memory of the list nodes */
void freeList (Node *head)
{
    Node *temp;
    while (head != NULL)
    {
        temp =head;
        head = head->next;
        free (temp);
    }
}
```

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* main.c*/

#include " linked_list.h"

int main()
{
 int array[10] = {23, 2, 78, 901, 1001, 3, 80, 23452, 5, 67};
 Node *unsorted_head = NULL, *sorted_head = NULL;
 int i;

    /* initializing the lists with the elements in the array */
 for (i = 0; i < 10; i++)
 {
    unsorted_head = insertList (unsorted_head, array[i]);
    sorted_head = insertSortedList (sorted_head, array[i]);
}

    /* printing */
    printList (unsorted_head);
    printf ("\n");
    printList (sorted_head);
    printf ("\n");
```

בגלל שפונקציות מחזירות ערך, המצביעים האלה מתעדכנים והשינוי שנעשה בפונקציות למצביעים, נשמר.

תכנית מהרצאה הקודמת מחולקת ל-3 קבצים

```
/* deleting some elements from unsorted list */
 for (i = 0; i < 10; i++)
   if( (i%3) == 0)
      unsorted_head =deleteFromList (unsorted_head,
       array[i]);
   /* printing */
  printList (unsorted_head);
 printf ("\n");
 printList (sorted_head);
 printf ("\n");

/* freeing the lists' dynamically allocated memory */
freeList (unsorted_head);
freeList (sorted_head);

return 0;
}
```

רשימות מקושרות דו-כיוונית

רשימה מקושרת – מבנה של צומת - תזכורת

- **רשימה מקושרת חד-כיוונית**

```
typedef struct Node
 {
        any_type  data;
        struct Node   *next;
}  node;
```

- **רשימה מקושרת דו-כיוונית**

```
typedef struct Node
    {
        any_type  data;
        struct Node   *prev, *next;
    } node;
```

רשימה מקושרת דו -כיוונית - דוגמא

לצורך פשטות, כל הפונקציות חולקו לשתי קבוצות – אלה שמעדכנות מצביע לראש הרשימה ולכן נדרש שימוש במצביע כפול ואלה שלא.

```
typedef struct Dlst
    {
        char    name[30];
        struct Dlst  *next;        /*forward*/
        struct Dlst  *prev;        /*backward*/
    } Dlist;

/*functions declarations*/

int InsertAfter ( Dlist  *afterptr, char newname[ ]);
void HeadInsert ( Dlist **headptr, char newname[ ]);
void DeleteHead ( Dlist  **headptr);
void DeleteItem ( Dlist  *item);
void DeleteAll ( Dlist  **headptr);
void DisplayBackwards ( Dlist *head);
```

שיטה נוספת (חוץ משימוש ב-return) לעידכון מצביע לראש הרשימה - שימוש במצביע כפול.
ב-main יש לשלוח לפונקציה את המצביע לראש הרשימה בתוספת &.

רשימה מקושרת דו כיוונית - דוגמא

```
/*this function inserts the new node to the list after the
 node pointered by afterptr*/

int InsertAfter( Dlist *afterptr, char newname[ ])
{
    Dlist *temp;
    temp = ( Dlist *)malloc(sizeof( Dlist));
    if (temp==NULL)
    {
        printf("Allocation Failed!");
        return 0;        /*need to free the list before exit*/
     }
    strcpy(temp->name, newname);  /*copy the data*/
    temp->next = afterptr->next;
    temp->prev = afterptr;

    if (afterptr->next != NULL)      /*if afterptr isn't last*/
        afterptr->next->prev = temp;
    afterptr->next = temp;
    return 1;
}
```

רשימה מקושרת דו כיוונית - דוגמא

```
/*this function inserts the new node to head of list */

void HeadInsert( Dlist **headptr, char newname[ ])
{
    Dlist *temp;
    temp = (Dlist *)malloc(sizeof( Dlist));
    if (temp==NULL)
     {
      printf("Allocation Failed!");
      DeleteAll (headptr);
      exit(1);
      }
   strcpy(temp->name, newname); /*copy the data*/
   temp->next = *headptr;
   temp->prev = NULL;
   if ( *headptr != NULL)       /*if it is a head of list*/
      (*headptr)->prev = temp;

   *headptr = temp;            /*update the head of list*/
}
```

רשימה מקושרת דו כיוונית - דוגמא

```
/*this function prints  the list information by reverse order */

void DisplayBackwards(Dlist *head)
{
   while (head!=NULL && head->next!=NULL)
        head = head->next;
   while (head!=NULL)
     {
        puts(head->name);
        head = head->prev;        /*go backward*/
      }
}
/*this function deletes  the node pointered by item. It can't
be a  head of list  */

void DeleteItem( Dlist *item)
{
  if (item->next != NULL)            /* there is next */
            item->next->prev = item->prev;
    item->prev->next = item->next;
    free(item);
}
```

רשימה מקושרת דו כיוונית - דוגמא

```
 /*this function deletes a  head of list  */

void DeleteHead( Dlist **headptr)
{
     Dlist *todel;
     todel = *headptr;                    /*pointer to head*/
    *headptr = (*headptr)->next;    /*update the head*/
      if ( *headptr != NULL)
        (*headptr)->prev = NULL;
      free(todel);
}

/*this function deletes a  list  */
void DeleteAll( Dlist **headptr)
{
   while (*headptr != NULL)
         DeleteHead(headptr);
}
```

רשימה מקושרת דו כיוונית - דוגמא

```
int main( )
{
    Dlist *head = NULL;
    int flag;

    HeadInsert( &head , "Yoav");
    HeadInsert( &head , "Yael");

    flag = InsertAfter(head, "Moshe");
    if(flag == 0)
    {
       printf("Allocation failed");
       DeleteAll(&head);
       return 1;
     }
    DeleteHead(&head);

    DisplayBackwards(head);

    DeleteAll(&head);
    return 0;
}
```