קורס תכנות

שיעור 14: הסוף

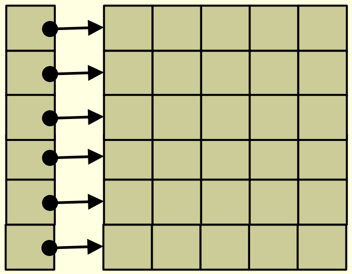
מה היום?

- חובות
- רשימות מקושרות
  - עוד דוגמאות + נושאים מתקדמים
- בחינה

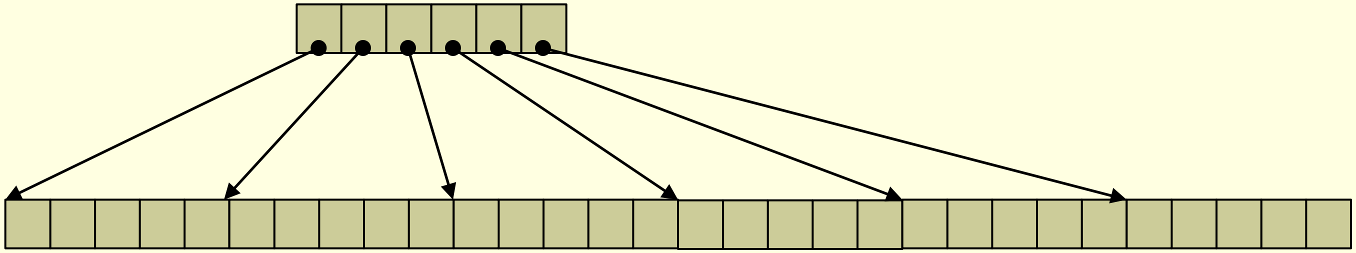
הקצאה דינאמית של מערכים דו-ממדיים

- שתי גישות אפשריות:

1. מערך של מערכים



2. מצביעים לתוך מערך "גדול"



מערך של מערכים

```
int main() {
    int rows, cols, i;
    int** table;

    scanf("%d%d", &rows, &cols);

    table = (int**) malloc(rows * sizeof(int*));
    /* incomplete, must check if failed */
    for (i = 0; i < rows; i++) {
        table[i] = (int*) malloc(cols * sizeof(int));
        /* incomplete, must check if failed */
    }

    init_table(table, rows, cols);
    print_table(table, rows, cols);

    for (i = 0; i < rows; i++)
        free(table[i]);
    free(table);
    return EXIT_SUCCESS;
}
```

איתחול והדפסה

```
void init_table(int **table, int rows, int cols)
{
    int i, j;

    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            table[i][j] = (i + 1) * (j + 1);
}


void print_table(int **table, int rows, int cols)
{
    int i, j;

    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++)
            printf("%4d", table[i][j]);
        printf("\n");
    }
}
```

מצביעים לתוך מערך גדול

```
int main() {
    int rows, cols, i;
    int* data;
    int** table;

    scanf("%d%d", &rows, &cols);

    data = (int*) malloc(rows * cols * sizeof(int));
    /* incomplete, must check if failed */
    table = (int**) malloc(rows * sizeof(int*));
    /* incomplete, must check if failed */
    for (i = 0; i < rows; i++)
        table[i] = data + (cols * i);

    init_table(table, rows, cols);
    print_table(table, rows, cols);

    free(table);
    free(data);

    return EXIT_SUCCESS;
}
```

תזכורת הגדרת חוליה ברשימה

- כל חוליה בשרשרת תמומש בעזרת מבנה המכיל
  - את המידע
  - מצביע לחוליה הבאה בשרשרת

- דוגמא: איבר ברשימה של מספרים שלמים

```
typedef struct node
{
    int data;
    struct node *next;
} node_t;
```

תזכורת: הוספת איבר בראש הרשימה

```
node_t* add_first(node_t *head, int data)
{
    node_t *new_node = create_node(data);
    if (new_node == NULL)
        return NULL;

    new_node->next = head;
    return new_node;
}
```

מימוש מחיקת איבר מרשימה (1)

```
node_t *delete(node_t *head, int value)
{
    node_t *curr = head, prev = head;

    if (head == NULL)
        return head;

    if (head->data == value)
    {
        curr = head->next;
        free(head);
        return curr;
    }
...
```

רשימה ריקה

האיבר הראשון ברשימה

מימוש מחיקת איבר מרשימה (המשך)

```
...
    prev = curr;
    curr = curr->next;
    while (curr != NULL) {
        if (curr->data == value)
        {
            prev->next = curr->next;
            free(curr);
            break;
        }
        prev = curr;
        curr = curr->next;
    }
    return head;
}
```

הוספת איבר במקום ה-n

```
node_t *add_nth(node_t *head, int n, int value)
{
    node_t *ptr, *new_node;
    int i;

    // index 0 is a special case
    if (n == 0)
        return add_first(head, value);

    ptr = head;
    for (i = 0; i < n - 1; ++i)
        ptr = ptr->next;

    new_node = create_node(value);
    // incomplete, must check for failure
    new_node->next = ptr->next;
    ptr->next = new_node;
    return head;
}
```

הפיכת רשימה (Reverse)

```
node_t* reverseList(node_t *head)
{
    node_t *iter = NULL, *next = NULL, *result = NULL;

    for (iter = head; iter != NULL; iter = next)
    {
        next = iter->next;
        iter->next = result;
        result = iter;
    }

    return result;
}
```

מיון רשימות מקושרות

- ראינו בנייה של רשימה ממוינת

- בהינתן רשימה קיימת כיצד נמיין?
- שתי אפשרויות:
  1. שימוש באחד מאלגוריתמי המיון שלמדנו
  2. בניית רשימה חדשה ממוינת מאברי הרשימה הישנה

אלגוריתם merge-sort

- נזכר באלגוריתם:
  - נחצה את הרשימה
  - נמיין כל מחצית
  - נמזג את הרשימות הממוינות

merge sort

```
node_t* mergeSort(node_t *head)
{
    node_t *other;

    // Base case -- length 0 or 1
    if ( (head == NULL) || (head->next == NULL) )
        return head;

    other = split(head);  // Split the list

    // Recursively sort the sublists
    other = mergeSort(other);
    head  = mergeSort(head);

    // answer = merge the two sorted lists together
    return sortedMerge(head, other);
}
```

split

```
node_t* split(node_t* source)
{
    int len = length(source);
    int i;
    node_t *other;

    if (len < 2)
        return NULL;

    for (i = 0; i < (len-1)/2; i++)
        source = source->next;

    // Now cut at current
    other = source->next;
    source->next = NULL;
    return other;
}
```

sorted merge

```
node_t* sortedMerge(node_t *a, node_t *b) {
    node_t dummy, *tail = &dummy;
    dummy.next = NULL;

    while (a != NULL && b != NULL) {
        if (a->data <= b->data) {
            tail->next = a;
            a = a->next;
        } else {
            tail->next = b;
            b = b->next;
        }
        tail = tail->next;
    }
    if (a == NULL) {
        tail->next = b;
    } else { // b == NULL
        tail->next = a;
    }
    return dummy.next;
}
```

רשימה כמבנה

- במקום לשמור מצביע לאיבר הראשון נחזיק מבנה המתאר רשימה
- המבנה יכיל את הרשימה עצמה ומידע נוסף על הרשימה שיעזור לנו במימוש חלק מהפעולות
  - מספר האברים ברשימה
  - מצביע לאיבר האחרון
  - ...

דוגמא

- מבנה של חוליה ברשימה - ללא שינוי

```
typedef struct node
{
    int data;
    struct node *next;
} node_t;
```

- מבנה נוסף המחזיק מצביע לתחילת/סוף הרשימה ואת גודלה

```
typedef struct
{
    node_t *head, *tail;
    int length;
} List;
```

כיצד נממש את add_first?

```
void add_first(List *list, int data)
{
    node_t *new_node = create_node(data);
    /* incomplete, must check for failure */

    new_node->next = list->head;
    list->head = new_node;

    if (list->tail == NULL)
        list->tail = new_node;

    list->length++;
}
```

כיצד נממש את add_last?

```c
void add_last(List *list, int data)
{
    node_t *new_node = create_node(data);
    /* incomplete, must check for failure */

    if (list->head == NULL)
    {
        list->head = new_node;
        list->tail = new_node;
        list->length = 1;
        return;
    }
    list->tail->next = new_node;
    list->tail = new_node;
    list->length++;
}
```

- אין צורך לרוץ על כל הרשימה

כיצד נשתמש

```
int main()
{
    int value = 0;
    List list;

    list.head = NULL;
    list.tail = NULL;
    list.length = 0;

    scanf("%d", &value);
    while (value >= 0) {
        add_first(&list, value);
        scanf("%d", &value);
    }
    ...
}
```

העלילה מסתבכת

- עד עכשיו – אם פונקציה יכולה לשנות את ראש הרשימה היא תחזיר את ראש הרשימה החדש

- כיצד נשנה משתנה שהוגדר בפונקציה אחרת?
  - בעזרת מצביע למשתנה

דוגמא: add_first

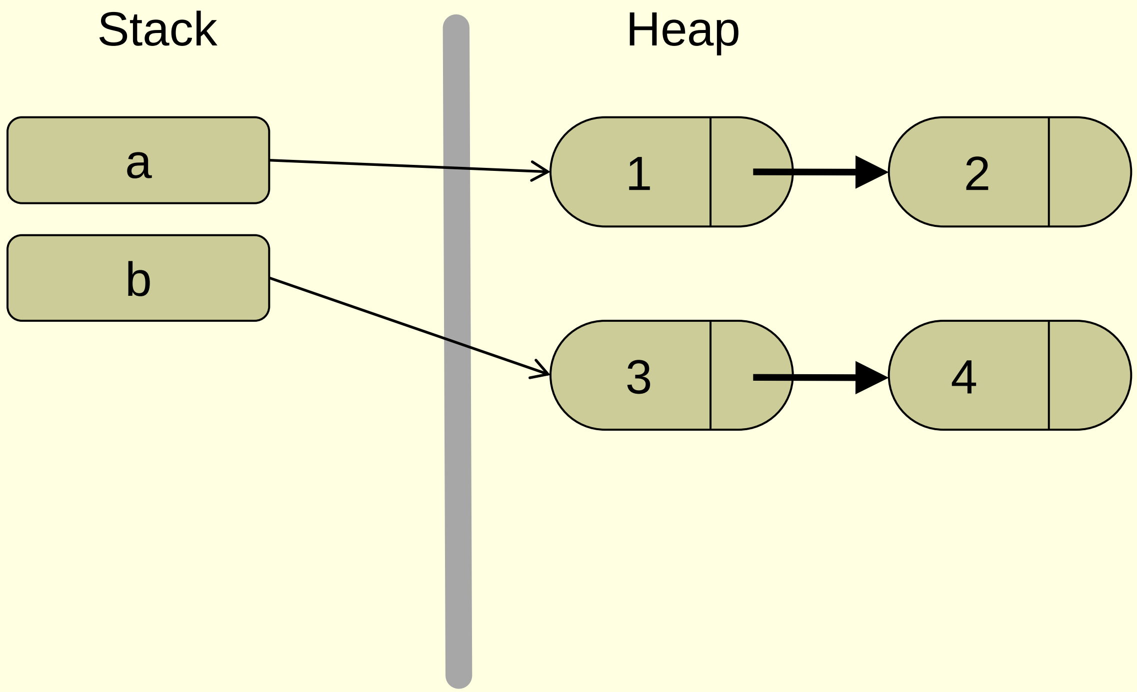
```
void add_first(node_t **headRef, int data)
{
    node_t *new_node = create_node(data);
    if (new_node == NULL)
    {
        printf("Fatal error: Unable to allocate memory!");
        exit(EXIT_FAILURE);
    }

    new_node->next = *headRef;
    *headRef = new_node;
}
```

שימוש

```
int main()
{
    int i;
    node_t *list = NULL;

    for (i = 0; i < 6; i++)
        add_first(&list, i);
    // list == {5, 4, 3, 2, 1, 0}

    print_list(list);
    free_list(list);

    return 0;
}
```

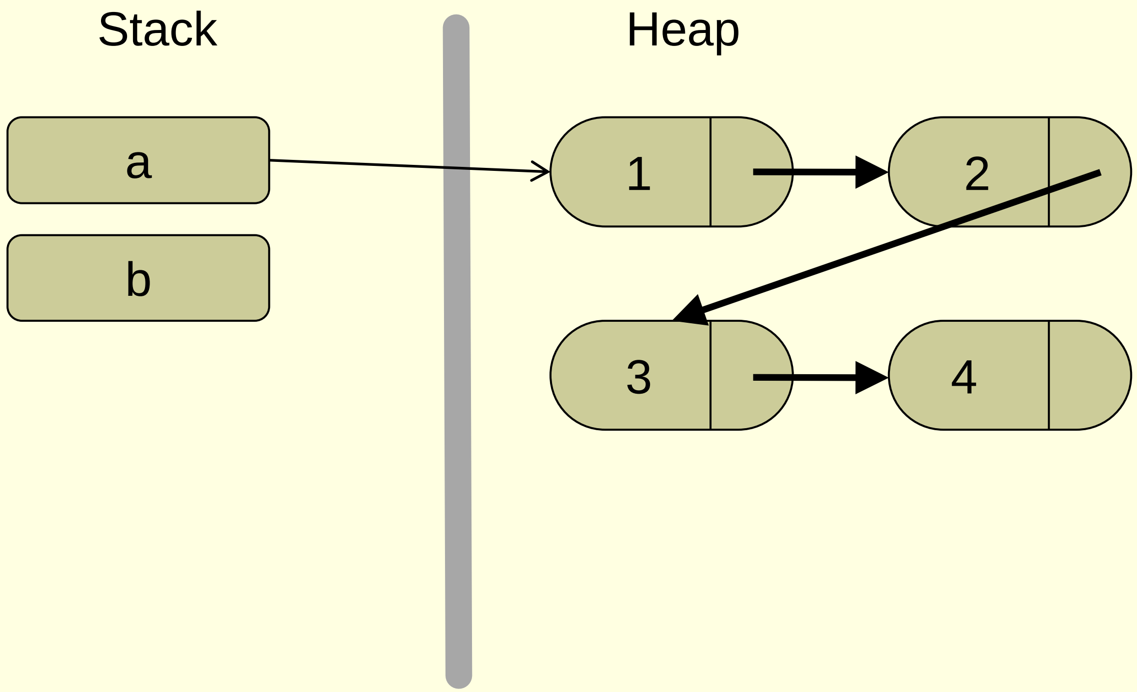
עוד דוגמא: append

- הפונקציה append מחברת רשימה מקושרת אחת בסוף של רשימה מקושרת שנייה
  - דומה לשרשור מחרוזות



לאחר הקריאה ל append

- a מצביע לרשימה המשורשרת
- הערך של b הוא NULL



מימוש

```
void append(node_t** aRef, node_t** bRef) {
    node_t* ptr;

    if (*aRef == NULL) { // special case if a is empty
        *aRef = *bRef;
    } else { // Otherwise, find the end of a, and append b there
        ptr = *aRef;
        while (ptr->next != NULL) { // find the last node
            ptr = ptr->next;
        }
        ptr->next = *bRef; // hang the b list off the last node
    }
    *bRef = NULL; // NULL the original b, since it has been
                  //appended above
}
```

דוגמת שימוש

```
int main()
{
    int i;
    node_t *a = NULL, *b = NULL;

    for (i = 0; i < 5; i++)
        add_first(&a, i);
    // a == {5, 4, 3, 2, 1, 0}

    for (i = 5; i < 10; i++)
        add_first(&b, i);
    // b == {9, 8, 7, 6, 5}

    append(&a, &b);
    // a == {5, 4, 3, 2, 1, 0, 9, 8, 7, 6, 5}
    // b == NULL
    print_list(a);

    free_list(a);
    return 0;
}
```

תרגיל בית***

- נכתוב פונקציה **רקורסיבית** שהופכת רשימה מקושרת
- נשתמש במצביע למצביע

- הדרכה:
  - כדי להבין את הקוד שרטטו!

תרגיל בית***

```
void RecursiveReverse(node_t** headRef) {
    node_t* first, *rest;

    // empty list base case
    if (*headRef == NULL)
        return;

    first = *headRef;     // suppose first = {1, 2, 3}
    rest = first->next;   // rest = {2, 3}

    if (rest == NULL)    // empty rest base case
        return;

    RecursiveReverse(&rest);

    first->next->next = first;
    first->next = NULL;   // tricky step -- make a drawing

    *headRef = rest; // fix the head pointer
}
```

שאלות?

מה למדנו בקורס

- מבוא: מה זה בעצם מחשב ומה זה תכנות

- הכרנו את שפת התכנות C, בפרט:
  - פקודות קיימות בשפה (כולל פונקציות-ספריה נפוצות)
  - אופן יצירת "פקודות חדשות" (פונקציות).
  - דרכים קיימות לייצוג ועיבוד מידע (טיפוסי-משתנים, מערכים, מחרוזות, מצביעים)
  - יצירת טיפוסים חדשים לייצוג מידע (מבנים, רשימות מקושרות).

- ראינו איך להשתמש בתכנות ב- C לפתרון בעיות שונות.

רשימת נושאי הקורס

- משתנים וסוגיהם, קלט/פלט ופעולות חשבון
- משפטי if ו- switch ופעולות השוואה
- לולאות while, for, do-while
- פונקציות
- מערכים ומערכים דו-ממדיים
- מחרוזות
- רקורסיה
- מצביעים
- מבנים
- הקצאת זיכרון דינאמית
- רשימות מקושרות
- אלגוריתמי חיפוש ומיון

הבחינה

- הבחינה תערך ביום שני, 13.2.11, בשעה 9:00
- משך הבחינה שלוש שעות (כלומר עד 12:00)
  - אין הארכות!
- הבחינה בכתב (אין קומפיילר)
- דף עזר ובו הגדרות של פונקציות ספרייה יצורף לטופס הבחינה
  - הדף כבר מופיע באתר
  - זהו חומר העזר היחיד המותר לשימוש

החומר למבחן

החומר כולל את **<u>כל</u>** מה שלמדנו בהרצאות, בתרגולים ובתרגילי הבית

דגשים להכנה למבחן

- חזרה על המצגות והתוכניות, במטרה להבין את כל הכללים, הדוגמאות, והפתרונות לתרגילים
- בחינות קודמות
  - מופיעות באתר הקורס בחינות משנים קודמות, לחלקן מצורף פתרון
- המבחן ללא מחשב, נסו לדמות את התנאים
  - פתרו על דף ורק לאחר מכן בידקו את עצמכם במחשב

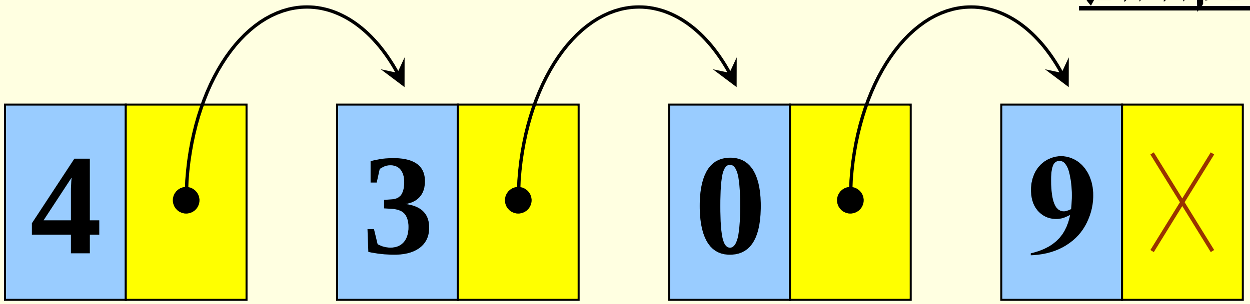
עצות כלליות למבחן עצמו

- זה רק מבחן, קחו אותו בפרופורציות
- קראו בעיון את השאלות
  - אם תהיה אי-בהירות כלשהי לגבי ניסוח אפשר לשאול את המרצה או את המתרגלים (שיעברו בין חדרי המבחן)
- חלקו את הזמן
  - 3 שאלות ב-3 שעות = בערך שעה לשאלה
- אם יש זמן אז כדאי לעבור על התשובות שוב בסוף המבחן ולוודא שהכול נכון

דוגמאות לשאלות ממבחן

שאלה 5 ממועד א' תשס"ד

שאלה 5 (25 נקודות)



סעיף א' (5 נקודות):

הגדירו מבנה (structure) אשר ישמש לייצוג רשימה מקושרת אשר תכיל מספרים שלמים (כמו בציור).

שאלה 5 ממועד א' תשס"ד

<u>סעיף א' (5 נקודות)</u>:

הגדירו <u>מבנה</u> (structure) אשר ישמש לייצוג רשימה מקושרת אשר תכיל מספרים שלמים (כמו בציור).

<u>פתרון</u>:

```
struct item
{
    int data;
    struct item *next;
};
```

שאלה 5 ממועד א' תשס"ד

<u>סעיף א' (5 נקודות)</u>:

הגדירו <u>מבנה</u> (structure) אשר ישמש לייצוג רשימה מקושרת אשר תכיל מספרים שלמים (כמו בציור).

<u>פתרון עם כתיב מקוצר</u>:

```
typedef struct item_t
{
    int data;
    struct item_t *next;
} item;
```

שאלה 5 מועד א' תשס"ד

סעיף ב' (10 נקודות):

כתבו פונקציה המקבלת מצביע לראש הרשימה ומחזירה את סכום המספרים ברשימה.

שאלה 5 ממועד א' תשס"ד

<u>סעיף ב' (10 נקודות)</u>:

כתבו <u>פונקציה</u> המקבלת מצביע לראש הרשימה ומחזירה את סכום המספרים ברשימה.

<u>פתרון:</u>

```
int sum_list(item *list)
{
    int sum=0;
    while (list != NULL)
    {
        sum=sum+list->data;
        list=list->next;
    }
    return sum;
}
```

מתקדמים על איברי הרשימה וסוכמים את הערכים שנמצאים בהם

שאלה 5 ממועד א' תשס"ד

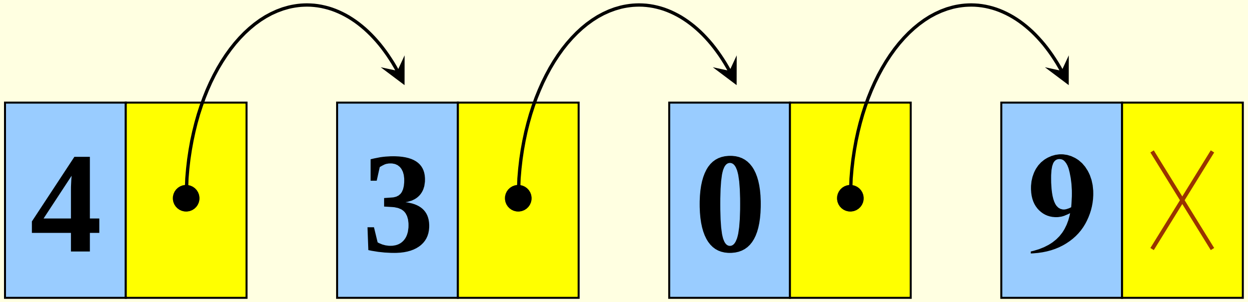
סעיף ב' (10 נקודות):

כתבו פונקציה המקבלת מצביע לראש הרשימה ומחזירה את סכום המספרים ברשימה.

איך ניתן היה לפתור את סעיף זה בצורה רקורסיבית?
(לא היה בשאלה המקורית – אבל היה יכול להיות...)

סכום רשימה – פתרון רקורסיבי

- תנאי עצירה:
  - אם הרשימה ריקה נחזיר 0.
- קידום:
  - נחשב (רקורסיבית) את סכום הרשימה שמתחילה מהאיבר השני (אורכה קטן ב-1 מהרשימה המקורית)
  - נוסיף לסכום שקיבלנו את ערך האיבר הראשון ונחזיר את התוצאה.



סכום רשימה – פתרון רקורסיבי

```
int sum_list(item *list)
{
     int sum;
     if (list == NULL)
       return 0;
     sum = sum_list(list->next);
     sum += list->data;
     return sum;
}
```

תנאי עצירה

פתרון בעיה "קטנה"

פתרון הבעיה "הגדולה"

סכום רשימה – פתרון רקורסיבי

או בקיצור...

```
int sum_list(item *list)
{
    if (list == NULL)
      return 0;

    return list->data + sum_list(list->next);
}
```

שאלה 5 ממועד א' תשס"ד

סעיף ג' (10 נקודות) :

כתבו פונקציה אשר תקבל מצביע לראש הרשימה, תשכפל את הרשימה ותחזיר מצביע לתחילת הרשימה המשוכפלת.

שאלה 5 ממועד א' תשס"ד

<u>סעיף ג' (10 נקודות)</u> :

כתבו <u>פונקציה</u> אשר תקבל מצביע לראש הרשימה, תשכפל את הרשימה ותחזיר מצביע לתחילת הרשימה המשוכפלת.

<u>שלבי הפתרון:</u>

- נתקדם על הרשימה המקורית עד שנגיע ל- NULL.
- בכל פעם נשכפל את האיבר הנוכחי, ונוסיף אותו בסוף הרשימה החדשה

<u>משתנים שנצטרך</u> :

- מצביע לאיבר החדש (new_item)
- מצביע לראש הרשימה החדשה (new_head)
- מצביע לאיבר האחרון שהוספנו לרשימה החדשה (new_last).

פתרון סעיף ג' (10 נקודות): שיכפול רשימה

```
item *duplicate(item *head)
{    item *new_item, *new_last=NULL, *new_head=NULL;
     while (head != NULL)
     {
          new_item=(item *) malloc(sizeof(item));
          if (new_item== NULL)
          {
               printf("memory allocation error\n");
               exit(1);
          }
          new_item->data=head->data;
          new_item->next=NULL;
          if (new_head == NULL)
                new_head=new_item;
          else
                new_last->next=new_item;
          new_last=new_item;
          head=head->next;
     }
     return new_head;
}
```

מתקדמים על הרשימה המקורית

שכפול האיבר הנוכחי

אם אין עדיין איברים ברשימה החדשה, אז זה ראש הרשימה

אחרת האחרון יצביע עליו

האיבר שהוספנו הוא עכשיו האחרון

מחזירים את ראש הרשימה החדשה

שאלה 1 ממועד א' תשס"ח

<u>סעיף א' (7 נקודות):</u>

כתבו פונקציה בעלת המפרט (prototype) הבא:

```
void arr_copy(int a[], int b[], int n)
```

- הפונקציה מקבלת שני מערכים המכילים מספרים שלמים,
- ומספר שלם אי-שלילי n (הניחו שזה אכן מה שהיא מקבלת).
- על הפונקציה להעתיק את n הערכים הראשונים במערך a ל-n המקומות הראשונים במערך b.
- הניחו ששני המערכים הם בגודל n לפחות.

שאלה 1 ממועד א' תשס"ח

סעיף א' (7 נקודות):

```
void arr_copy(int a[], int b[], int n){
    int i;
    for (i = 0; i < n; i++){
       b[i] = a[i];
      }
}
```

שאלה 1 ממועד א' תשס"ח

סעיף ב' (15 נקודות):

כתבו פונקציה בעלת המפרט (prototype) הבא :

```
void merge( int a[], int size_a, int int b[],
            int size_b, int *dest );
```

- הפונקציה מקבלת שני מערכים של מספרים שלמים שממויינים בסדר לא-יורד (כלומר מקטן לגדול). כמו-כן, היא מקבלת את גדלי המערכים, ומערך נוסף dest.
- על הפונקציה להעתיק בצורה ממויינת את הערכים משני המערכים, a ו- b, לתוך המערך dest.
- כלומר, בסיום הפונקציה המערך dest צריך להכיל את כל size_a + size_b הערכים הללו, והוא צריך להיות ממויין בסדר לא-יורד (מקטן לגדול).
- הניחו כי יש במערך dest מספיק מקום עבור כל הערכים המועתקים.

שאלה 1 ממועד א' תשס"ח

```
void merge( int a[], int size_a, int b[], int size_b,
            int *dest ){
      int ia = 0, ib = 0, id = 0;
      while (ia < size_a && ib < size_b){
            if (a[ia] > b[ib]){
                  dest[id] = b[ib];
                  ib++;
            }
            else{
                  dest[id] = a[ia];
                  ia++;
            }
            id++;
      }
      ......
}
```

נשמור אינדקס נפרד לכל מערך

כל עוד לא הגענו לסוף אחד המערכים נשווה בין שני התאים ונעתיק את הערך הקטן יותר

נותר לנו להעתיק את התאים שנשארו אחרי השלב הקודם...

שאלה 1 ממועד א' תשס"ח

```
void merge( int a[], int size_a, int b[], int size_b,
            int *dest ){
      ...

      while (ia < size_a){
            dest[id] = a[ia];
            ia++;
            id++;
      {
      while (ib < size_b){
            dest[id] = b[ib];
            ib++;
            id++;
      {
{
```

נותר לנו להעתיק את התאים
שנשארו אחרי השלב הקודם...

שאלה 1 ממועד א' תשס"ח

<u>סעיף ג' (12 נקודות)</u>:

- לפניכם הפונקציה הרקורסיבית **merge_sort**, שממיינת את המערך **a**, שגודלו **size**.
- הפונקציה משתמשת בפונקציות מסעיפים א' ו- ב'.
- עליכם **להשלים** את הקטעים החסרים בפונקציה, בתוך המסגרות (כל מסגרת היא שורת קוד).

- <u>הערה:</u> הניחו שהקצאת הזיכרון הצליחה.

תנאי עצירה

רקורסיה!

```
#include <stdlib.h>
void merge_sort(int a[], int size)
{
     int *left=NULL, *right=NULL, *temp=NULL;

     if ( size<= 1 )
          return;

     left  = a;
     right = a + size/2;

     merge_sort( left, size/2 );

     merge_sort( right, size - size/2 );

     temp = (int*) malloc( size*sizeof(int) );

     merge( left, size/2, right, size - size/2, temp );

     arr_copy( temp, a, size );
     free(temp);
}
```

That's all Folks!