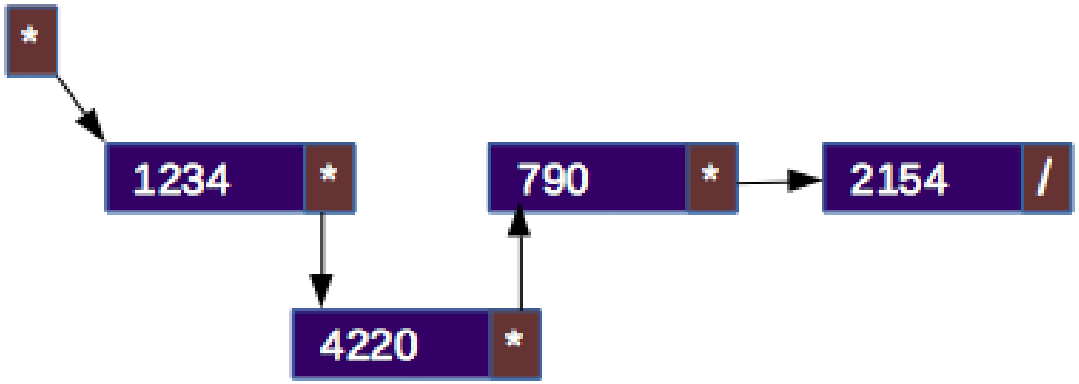
קורס יסודות התכנות בשפת C

פרק 13

מבני נתונים -רשימות מקושרות

Linked - Lists



ד"ר שייקה בילו

יועץ ומרצה בכיר למדעי המחשב וטכנולוגית מידע

מומחה למערכות מידע חינוכיות, אקדמיות ומנהליות

חזרה - מבנים - הגדרת מבנה חדש

- נקודה במישור מיוצגת על-ידי שתי קואורדינטות, X ו- Y.
- הגדרת מבנה שייצג נקודה תיראה למשל כך:

```
struct point
{
        double x;
        double y;
};
```

- ההגדרה הזו לא תופיע בתוך פונקציה, אלא בתחילת הקובץ (בד"כ מיד אחרי שורות ה- #include וה- #define).
- כעת ניתן להגדיר בתוכנית משתנים מהסוג הזה, כפי שנבחן בהמשך.

חזרה - מבנים - הגדרת מבנה חדש

- ניתן לגשת לכל אחד משדות המבנה ולכתוב/לקרוא אליו/ממנו, למשל:

```
int main()
{
    struct point P1,P2;
    P1.x = 6;
    P1.y = 7;
    P2.x = 4;
    P2.y = 2;
    printf("%.2lf\n", P1.y);
    return 0;
}
```

	P2
x	4
y	2

	P1
x	6
y	7

(יודפס 7)

חזרה - מבנים - הגדרת מבנה חדש


```
#include <stdio.h>
struct point {
   double x;
   double y;
};
int main()
{
  struct point P1,P2;
  P1.x = 6;
  P1.y = 7;
  P2.x = 4;
  P2.y = 2;
  printf("%.2lf\n", P1.y);
  return 0;
}
```

	P2
x	4
y	2

	P1
x	6
y	7

חזרה - מבנים – כתיבה מקוצרת

- שמו של הטיפוס החדש שהגדרנו הוא struct point
- אפשר לחסוך את כתיבת המילה struct באמצעות הפקודה typedef, שמאפשרת לתת שם חדש לטיפוס-משתנה:

```
struct point
{
        double x;
        double y;
};
typedef  struct point   point_t;
```

טיפוס קיים שם חדש

חזרה - עוד על typedef


```
struct complex
{
   double real;
   double image;
   };
int main()
{
   struct complex c={5,7};
   struct complex *pc;
}
```

```
struct complex
{
     double real;
     double image;
     };
typedef struct complex
    complex_t;
int main()
{
    complex_t c={5,7};
    complex_t *pc;
}
```

חזרה - מצביעים ומבנים

- בדוגמא זו, כדי להגיע לשדות של P דרך ptr שמצביע על P, אפשר להשתמש ב- * כרגיל:

P.x = 3; שקול ל- (*ptr).x = 3;

P.y = 7; שקול ל- (*ptr).y = 7;

- יש צורך בסוגריים כי אחרת לנקודה יש קדימות על פני ה- *.

חזרה -מצביעים ומבנים – גישה לשדות המבנה

- בדוגמא זו, כדי להגיע לשדות של P דרך ptr שמצביע על P, אפשר להשתמש ב- * כרגיל:

P.x = 5; שקול ל- (*ptr).x = 5;

P.y = 8; שקול ל- (*ptr).y = 8;

- אבל בדרך-כלל נשתמש לצורך זה בסימון מקוצר: חץ ->

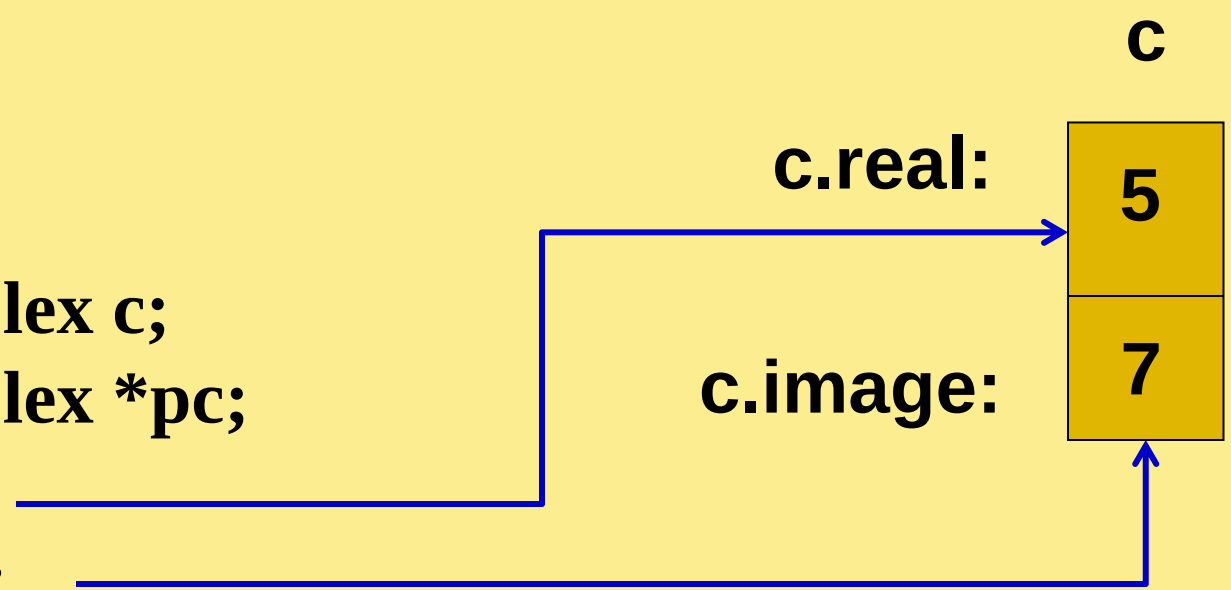
P.x = 5; שקול ל- ptr->x = 5;

P.y = 8; שקול ל- ptr->y = 8;

- כלומר משמעות החץ היא גישה לשדה במבנה שמצביעים עליו.

חזרה - מבנים Syntax - שימוש במצביעים


```
struct complex
{
    double real;
    double image;
};
int main()
{
   struct complex c;
   struct complex *pc;
   c.real = 5;
   c.image = 7;
}
```



חזרה - מבנים Syntax - כתובות ומצביעים


```
typedef struct {
    double real, image;
}comples;
int main()
{
    complex c;
    complex *pc;
    c.real = 5;      c.image = 7;
    pc = &c;
    pc->real = 3;
    pc->image = 4;
}
```

c → 0x7fff9255c05c

c.real: 5

c.image: 7

pc → 0x7fff9255c05c

pc.real: 3

pc.image: 4

pc → 0x7fff9255c05c

חזרה - מבנה בתוך מבנה

- אפשר להגדיר את זה על-ידי הקואורדינטות של שני הקודקודים:

```
struct rect
{
        double xl, xh, yl, yh;
};
```

- ברור יותר להגדיר ע"י 2 נקודות.
- נדגים כאן גם את הרישום המקוצר עם typedef:

```
struct rect {
        point_t p;
        point_t q;
};
typedef struct rect rect_t;
```

חזרה - מבנים – כתיבה מקוצרת

- שמו של הטיפוס החדש שהגדרנו היה struct point
- אפשר לחסוך את כתיבת המילה struct באמצעות הפקודה typedef, שמאפשרת לתת שם חדש לטיפוס-משתנה:

```
struct point
{
   double x;
   double y;
};
typedef struct point point_t;
```

שם חדש טיפוס קיים

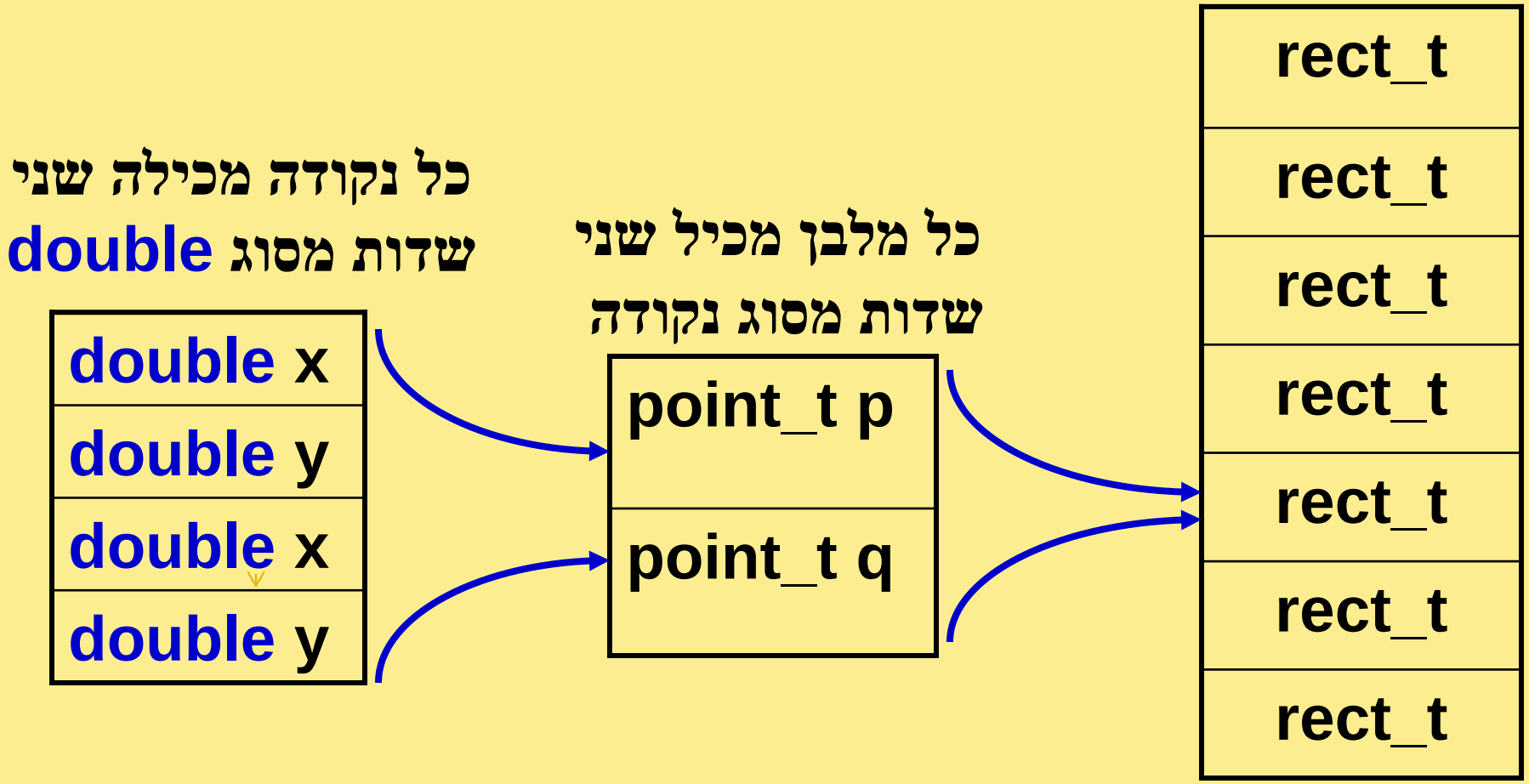
חזרה - מערך של מבנים

- כמו טיפוסי משתנים אחרים, אפשר להגדיר מערך של מבנים.
- מערך זה יוגדר ויכיל בתוכו שדות שהם מבנים הכוללים תתי שדות.
- למשל, אפשר להגדיר מערך של נקודות:

```
int main()
{
   point_t   arr[20];
   …
   …
   return 0;
}
```

חזרה - מערך של מבנים בזיכרון

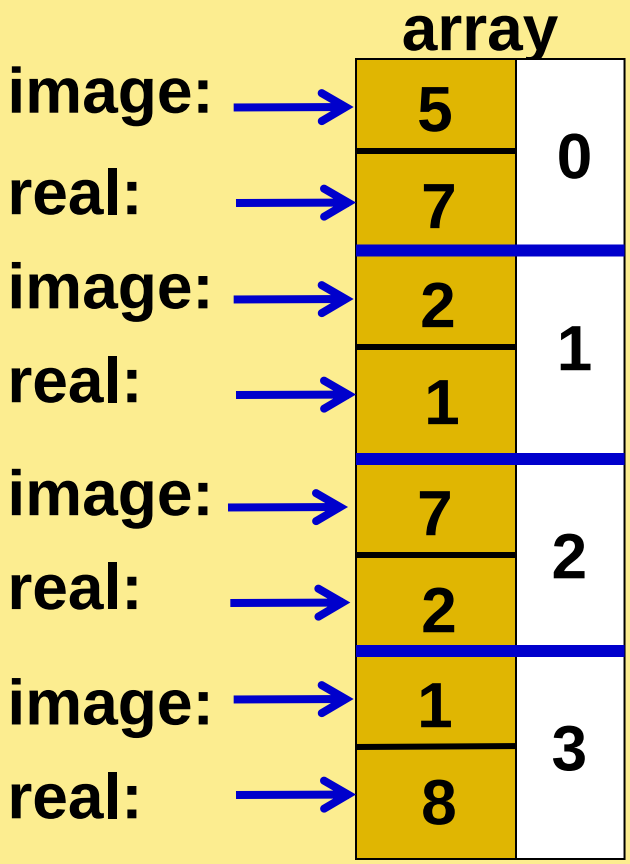
המבנים נשמרים ברצף בזיכרון:



חזרה - מערכים ומבנים

- מאחר ומבנים מגדירים סוג חדש של משתנים הרי שכמו לסוגים רגילים ניתן ליצור מערכים עבור סוגים אלו.

```
complex_t arr[SIZE]=
{{5,7},{2,1},{7,2},{1,8}}
arr[1].image = 9;
arr[1].real = 2;
arr[3].image = 8;
arr[3].real = 3;
```



חזרה -מערכים ומבנים אחרי ביצוע הפקודות


```
complex_t arr[SIZE]=
{{5,7},{2,1},{7,2},{1,8}}
arr[1].image = 9;
arr[1].real = 2;
arr[3].image = 8;
arr[3].real = 3;
```

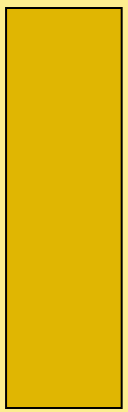
תזכורת - הקצאת זיכרון דינאמית

- הזיכרון המקסימאלי ששימש את התוכנית שלנו עד היום היה קבוע מראש על פי הגדרת טיפוסי וסוגי המשתנים בתוכנית.
- לפני הרצת התוכנית, לאחר שהתוכנית עברה את שלב התרגום, כל המערכים שבתוכנית גודלם נקבע ואינו ניתן לשינוי במהלך ריצת התוכנית!
- <u>בעיות אפשריות:</u>

1. אם הגדרנו מערך בגודל 100 לשמור את ציוני כל הסטודנטים וכעת מגיע סטודנט נוסף מה נעשה?
2. התוכנית כבר כתובה ולא ניתן לשנות את גודל המערך!
3. נצטרך פתרון יצירתי המסוגל לייצר עוד תאים על פי דרישה.

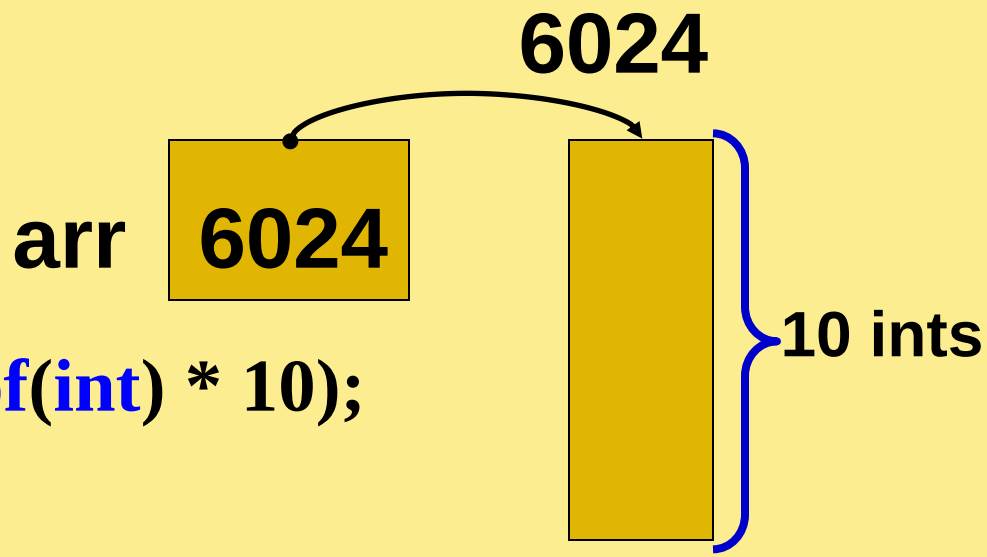
תזכורת - הקצאת זיכרון דינאמית


```
int main()
{
    int arr[10];
    arr[6] = 7;
}
```



שפת C מאפשרת לנו להקצות זיכרון תוך כדי ריצת התוכנית וכך להגדיר מערכים ומבנים אחרים בצורה דינאמית: 10 ints

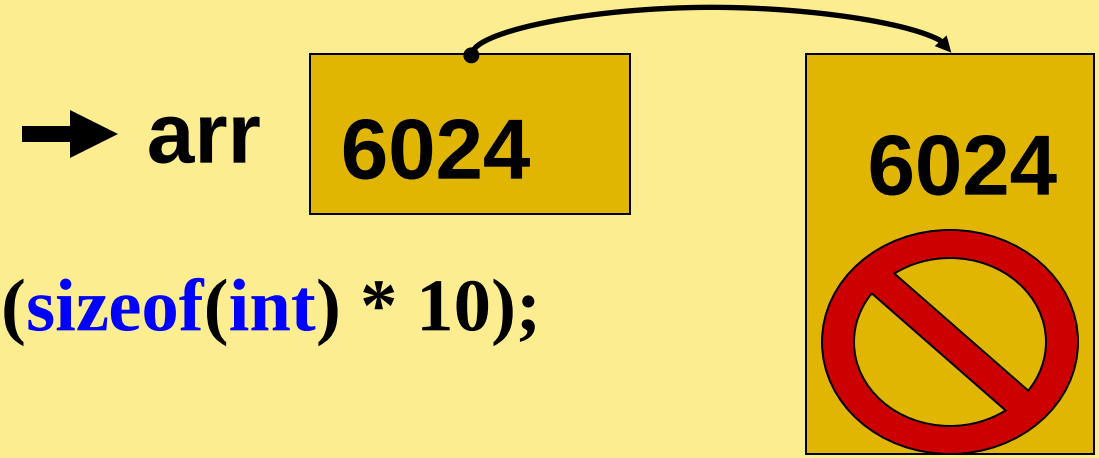
```
int main()
{
    int *arr;
    arr = (int*) malloc(sizeof(int) * 10);
    arr[6] = 7;
}
```



תזכורת - שחרור זיכרון שהוקצה דינאמית

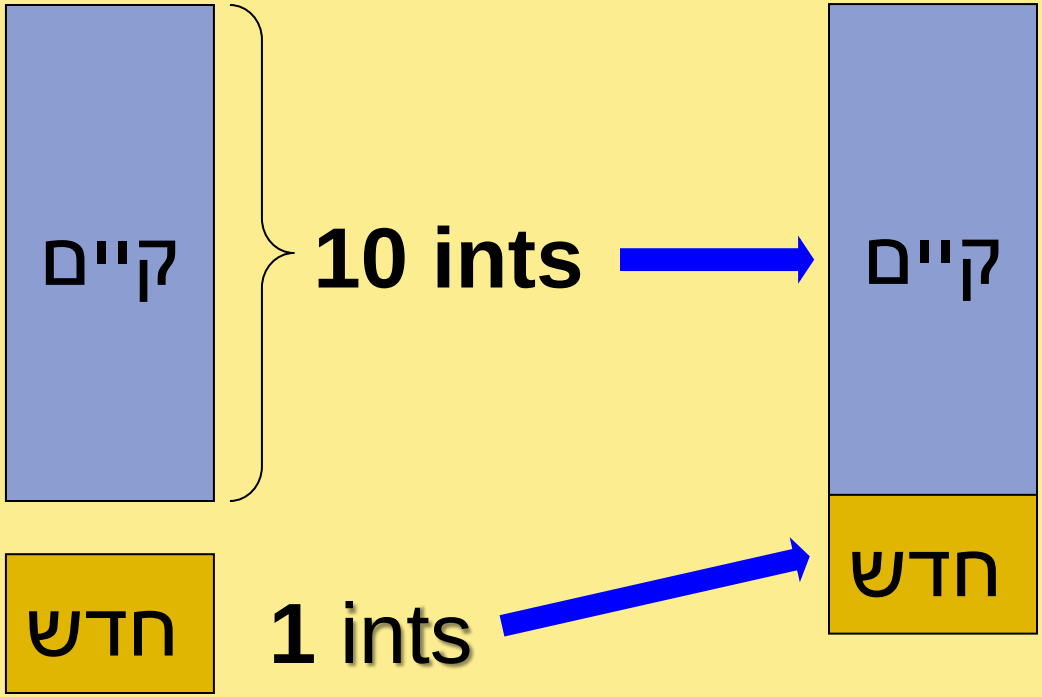
- כל שטח זיכרון שהוקצה ע"י המתכנת צריך להיות משוחרר בסיום השימוש בו.
- לכן חייבים תמיד להחזיק את כתובת האזור שהוקצה על מנת שנוכל לשחרר אותו בסיום השימוש.

```
int main()
{
    int *arr;
    arr = (int*) malloc(sizeof(int) * 10);
    arr[6] = 7;
    free(arr);
}
```



תזכורת - הקצאת זיכרון דינאמית

- במערך אם נרצה להוסיף איבר חדש נוסף נצטרך להעתיק את כל האיברים הישנים למערך גדול יותר אשר הקצנו.
- נקצה מערך חדש ונעתיק אליו את המערך הישן.



תזכורת - הקצאת זיכרון דינאמית


```
#define SIZE 10
int main()
{
    int *arr,*brr,i;
    arr = (int *) malloc(sizeof(int) * SIZE);
    for (i=0;i<SIZE;i++)
        arr[i] = i;
```

תזכורת - הקצאת זיכרון דינאמית


```
    brr = (int *) malloc(sizeof(int) *( SIZE + 1))
    for (i=0;i<SIZE;i++)
        brr[i] = arr[i];
    brr[i] = i;
    free(arr);
    free(brr);
}
```

תזכורת - הקצאת זיכרון דינאמית - פונקציות

הפונקציות נמצאות ב- stdlib.h:

```
void *malloc(size_t * size);
```

- הקצאת בתים בגודל נדרש בזיכרון.
- קריאה מוצלחת מחזירה כתובת בסיס חדשה.
- אחרת מוחזרת כתובת NULL.

```
void *calloc(size_t n,size_t size_el)
```

- הקצאת מערך של אלמנטים כל אחד בגודל של size_el
- כל תא שהוקצה בזיכרון מאותחל לאפס.
- קריאה מוצלחת מחזירה כתובת בסיס חדשה.
- אחרת יוחזר NULL.

תזכורת - הקצאת זיכרון דינאמית - פונקציות

הפונקציות נמצאות ב- stdlib.h:

```
void free(void *ptr)
```

- מבטל את הקצאת הזיכרון.
- מקיים מצביע על המיקום בו הוקצה הזיכרון הקודם.
- משחרר/מוחק את הכתובת הקודמת הנמצאת במצביע.

```
void *realloc(void *ptr,new_size)
```

- מקצה מחדשה מקום בזיכרון.
- המקום החדש בזיכרון הוא על פי הגודל של new_size.

תזכורת - הקצאת זיכרון דינאמית

ספריות והצהרות על פונקציות

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

void FillArray(int arr[ ], int n);

void PrintArray(int arr[ ], int n);
```

תזכורת - הקצאת זיכרון דינאמית

תוכנית ראשית

```
int main()
{
    int *arr, n;
    scanf("%d", &n);
    while( n< 0 || n > 1000)
                scanf("%d", &n);
    arr = (int *)calloc(n, sizeof(int));
    FillArray(arr, n);
    PrintArray(arr, n);
    free(arr);
    return 0;
}
```

תזכורת - הקצאת זיכרון דינאמית


```
void FillArray(int arr[ ], int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        arr[i] = rand()%100 + 1;
    }
}
```

מילוי המערך החד ממדי
במספרים אקראיים

תזכורת - הקצאת זיכרון דינאמית


```
void PrintArray(int arr[ ], int n)
{
    int i;
    for(i = 0; i < n; i++)
    {
        printf("%3d ", arr[i ]);
        if (i % 10 == 0)
            printf("\n");
    }
}
```

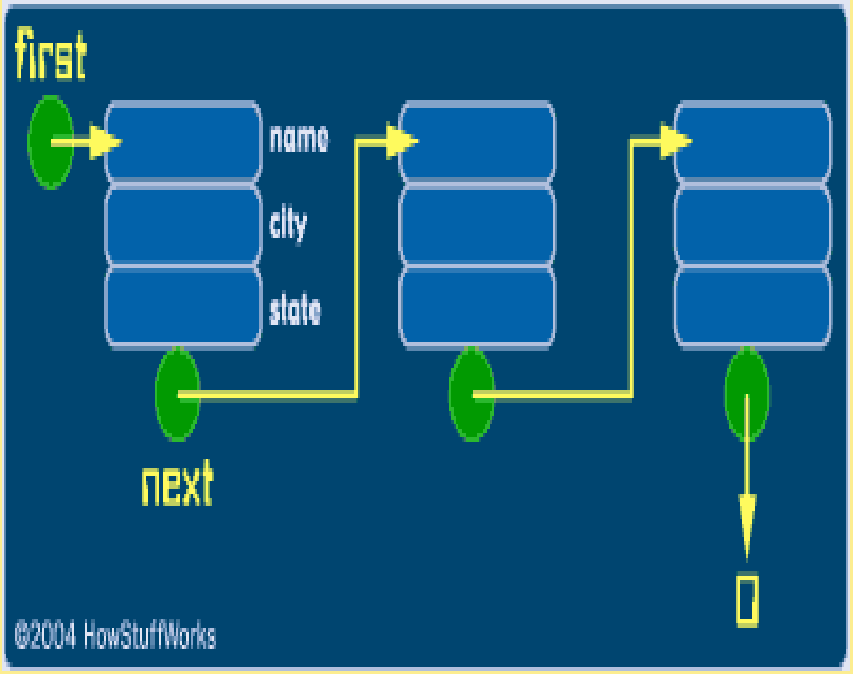
הדפסת המערך

החד ממדי

שאלות על השעור הקודם?

רשימות מקושרות - נושאים

- רשימות מקושרות צורך, הגדרה, הסבר והכנה למימוש
- מימוש רשימה מקושרת בשפת C.
- מימוש פעולות על רשימה מקושרת ב- C:
  - הוספת איבר לרשימה מקושרת.
  - מציאת איבר ברשימה מקושרת.
  - מחיקת איבר ברשימה מקושרת.
  - מחיקת כל הרשימה המקושרת.



רשימות מקושרות

- ידוע לכל מתכנת כי במערכות רבות קיימת תחלופה תכופה של אובייקטים (מחיקה והוספה בקצב מהיר).

<u>למשל במערכות הבאות:</u>

- מערכת ממוחשבת לניהול ספרים בספריה.
- מערכת לרישום סטודנטים באוניברסיטה.
- מערכות בקרת תעבורה, טיסה, שייט וכו'
- מערכת מעקב במצלמות כביש 6, מערכות ניווט וכו'.

רשימות מקושרות

- במידה ומייצגים את הנתונים באמצעות מערך, יש להעתיק את כל המערך מחדש כל פעם כשצריך להוסיף משהו, ו-"לצופף" את כל המערך כשצריך למחוק משהו מהאמצע.
  - זה לא יעיל.
  - זה לא נוח.
  - זה לא סביר.
  - זה דורש גישות מרובות לזיכרון.

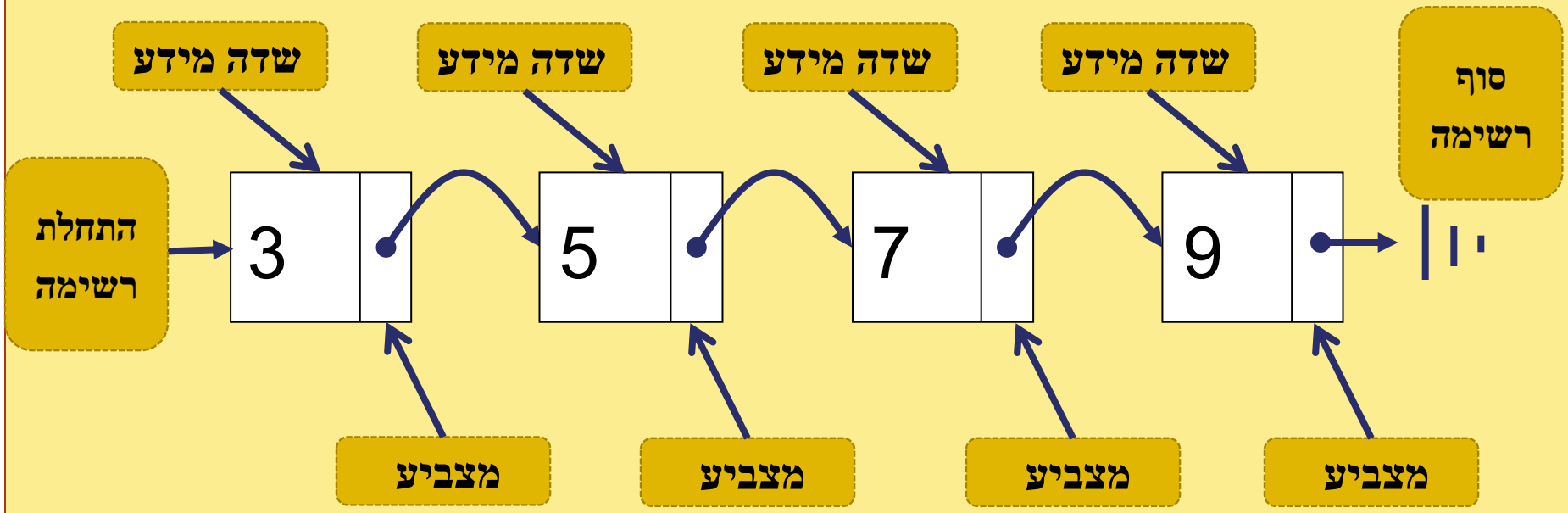
רשימות מקושרות

- לעבודה עם רשימות גדולות שיש בהן שינויים תכופים משתמשים ברשימה מקושרת במקום להשתמש במערך.
- בניגוד למערך, שהוא אוסף של תאים רצופים, רשימה מקושרת היא אוסף של תאים לא רצופים, שכל אחד מהם כולל גם מצביע לתא שאחריו ברשימה.
- כפי שאמרנו, זה מקל מאוד על ביצוע תוספות ומחיקות, בלי צורך לשנות את שאר תאי הרשימה.
- כדי לעבור על הרשימה מספיק להחזיק מצביע לתא הראשון שלה, וממנו להתקדם לתא הבא, וכן הלאה.

רשימות מקושרות

- המוסכמה היא שהתא האחרון ברשימה תמיד מצביע על סוף הרשימה המצוין תמיד ע"י הערך NULL.
- רשימה מקושרת היא סוג של מבנה נתונים דינאמי אשר יכול לצמוח לפי דרישת המנגנון אותו הוא בא לשרת.
- רשימות מקושרות יכולות ונועדו גם לצמוח וגם להצטמצם באותה מידה בכל זמן לפי הצורך.
- רשימות מקושרות יש צורך לשחרר בסיום השימוש בהן כמו שחרור הקצאת זיכרון דינאמית למצביע.

דוגמא: רשימה מקושרת של מספרים

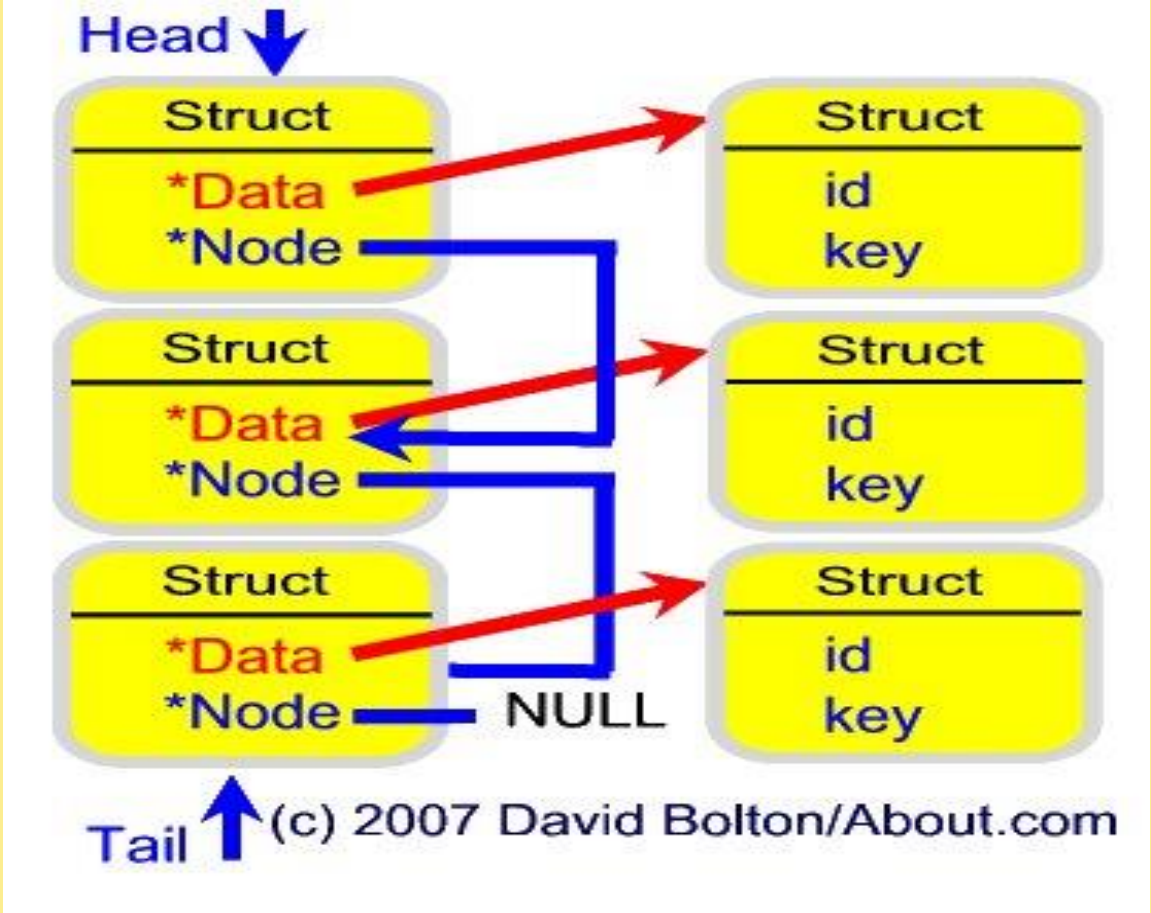


לעומת מערך:

3	5	7	9

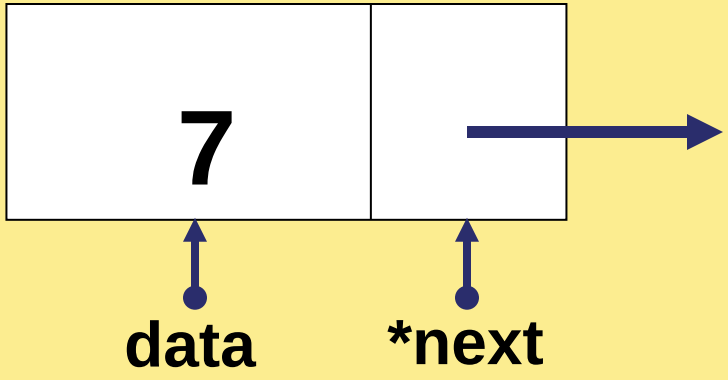
רשימות מקושרות - נושאים

- מימוש וביצוע פעולות על רשימה מקושרת ב- C:
  - הוספת איבר.
  - מציאת איבר.
  - מחיקת איבר.
  - מחיקת כל הרשימה.



הגדרה של מבנה עבור איבר ברשימה מקושרת


```
typedef struct Item
{
    int data;
    struct Item *next;
}Item;
```



- השדה data מכיל את הערך שהאיבר מכיל.
- השדה next* מכיל מצביע לאיבר הבא ברשימה.
- אם אין איבר נוסף, כלומר זה האיבר האחרון, אז ערך המצביע הזה יהיה NULL.

הוספת איברים לרשימה

- מעצם טבעה של רשימה מקושרת (שיש בה תוספות ומחיקות), מספר האיברים בה לא ידוע מראש ואין עליו חסם.
- אנחנו נדרשים להוסיף איברים במהלך ריצת התוכנית.
- תוספת איבר לרשימה מקושרת תבוצע ע"י הקצאה דינאמית שלו. הקצאה דינאמית של איבר אחד נעשית במקרה זה ע"י:

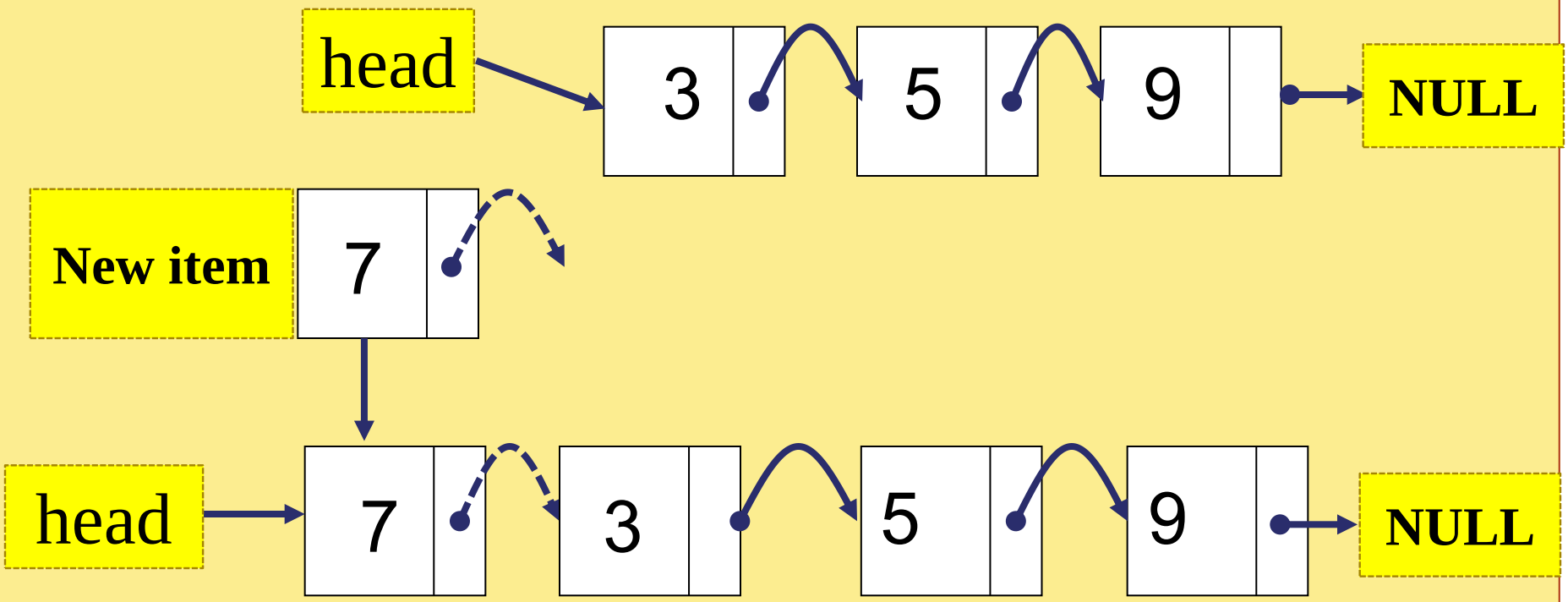
```
Item *head;
head = (Item *) malloc (sizeof(Item));
```

- בהמשך השעור נתאר פונקציה להוספת איבר כזה לרשימה.

הוספת איבר לתחילת רשימה

- אמרנו שתמיד נשמור מצביע לאיבר הראשון ברשימה, וממנו נוכל להתקדם לאחרים.
- נממש פונקציה בשם Add, שתקבל מצביע לראש הרשימה וערך להוספה, ותוסיף את הערך בתחילת הרשימה.
- הפונקציה תחזיר מצביע לראש הרשימה (החדש).
- נוכל להשתמש בפונקציה הזאת לבניית רשימה מקושרת חדשה אם נתחיל עם רשימה ריקה, כלומר עם מצביע שערכו NULL.

דוגמא: הוספת איבר בהתחלה



במימוש על-ידי מערך היינו צריכים להזיז קדימה את כל הערכים שלו,
כדי להכניס את 7 בהתחלה וזה בהנחה שהיה מקום עבורו.

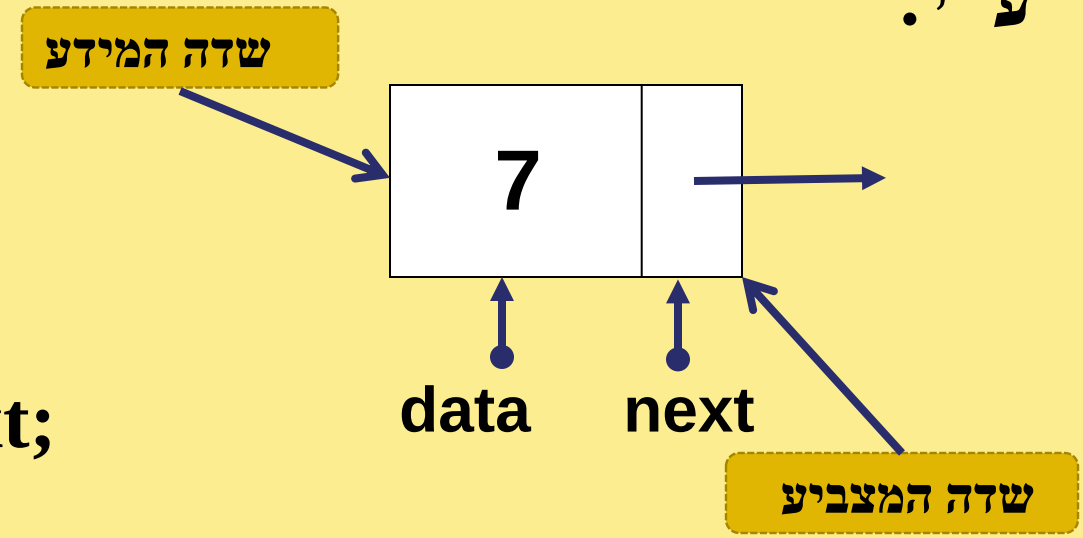
מימוש פונקציית ההוספה Add

- פונקציית ההוספה צריכה לבצע שלוש משימות:
  1. ליצור איבר עבור המידע החדש.
  2. לדאוג שהוא, האיבר החדש, יצביע על תחילת הרשימה הקיימת (או על NULL אם היא ריקה).
  3. להחזיר מצביע על האיבר החדש (תחילת הרשימה החדשה).
- האיבר החדש צריך להיות מוקצה דינאמית ע"י הפונקציה.
- משתנה מקומי רגיל נעלם בסוף הפונקציה.

מימוש רשימות מקושרות ב-C

- אז עבור הדוגמא של רשימת מספרים שלמים נגדיר איבר ע"י:

```
typedef struct Item
{
    int data;
    struct Item *next;
} Item;
```



- השדה data יכיל את המספר שהאיבר מכיל.
- השדה next יכיל מצביע לאיבר הבא ברשימה. (אם אין איבר נוסף, כלומר זה האיבר האחרון, אז ערך המצביע הזה יהיה NULL)

פונקציה להוספת איבר בהתחלה

מקבלים מצביע לראש רשימה וערך להוספה

```
Item *Add(Item *head, int value)
{
    Item *new_item;
    new_item = (Item*) malloc(sizeof (Item));
    new_item->data = value;
    new_item->next = head;
    return new_item;
}
```

יוצרים איבר חדש עם הערך הדרוש:

האיבר החדש יצביע לראש הרשימה הקודם, ויהיה ראש הרשימה החדש.

מחזירים את כתובות של האיבר החדש

הערה: לצורך ההדגמה אנחנו מניחים שהקצאת הזיכרון הצליחה.

רשימות מקושרות ב-C – שימוש ב- Add

מתחילים מרשימה ריקה וכל פעם
מוסיפים לה ערך בהתחלה

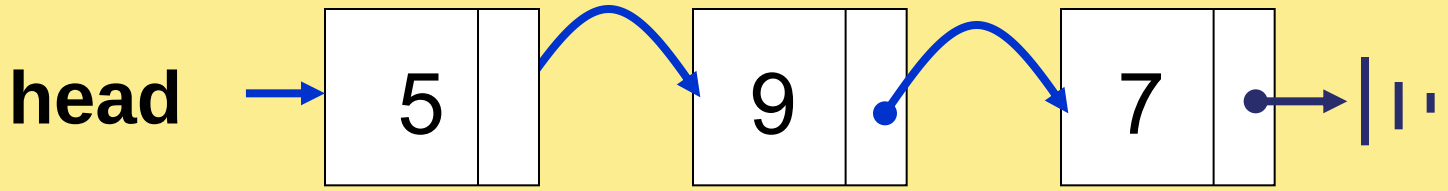
```
....
int main()
{
Item *head = NULL;
head = Add(head,7);
head = Add(head,9);
head = Add(head,5);

....
}
```

הוספת איבר ראשון להתחלת הרשימה

הוספת איבר שני לרשימה, אחרי הראשון

הוספת איבר שלישי לרשימה, אחרי השני

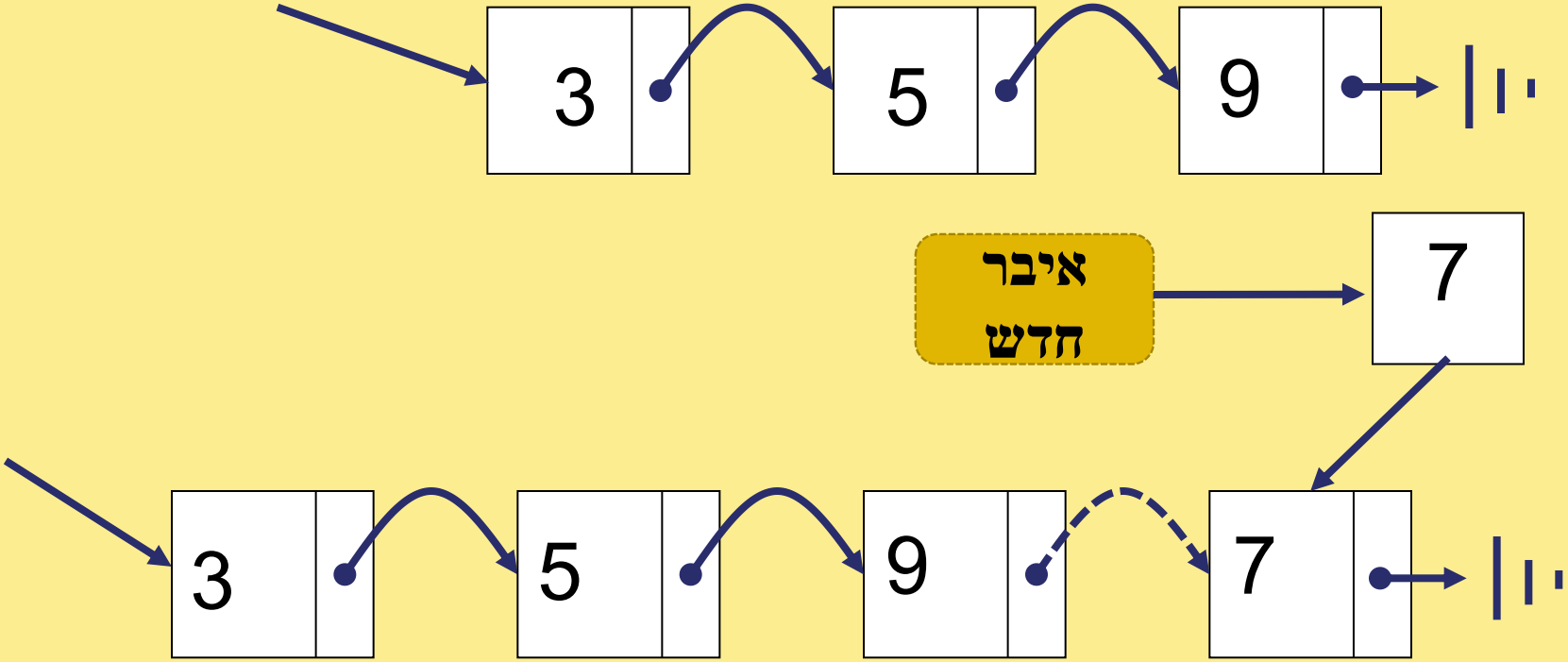


שאלות?

הוספת איבר בסוף רשימה

- נממש פונקציה בשם Add_End, שתקבל מצביע לראש הרשימה וערך להוספה, ותוסיף את האיבר בסוף הרשימה.
- הפונקציה תחזיר מצביע לראש הרשימה (החדש).
  - אם הרשימה הייתה ריקה אז עכשיו ראש הרשימה הוא האיבר שהוספנו.
  - כל סיבוב בודקת הפונקציה האם אנו נמצאים באיבר האחרון.
  - עמידה על האיבר האחרון מצוינת ע"י הצבעה ל- NULL.
  - לאחר איתור האחרון ניתן להסית ממנו את המצביע ל- NULL להצביע על האיבר החדש שהוכנס.

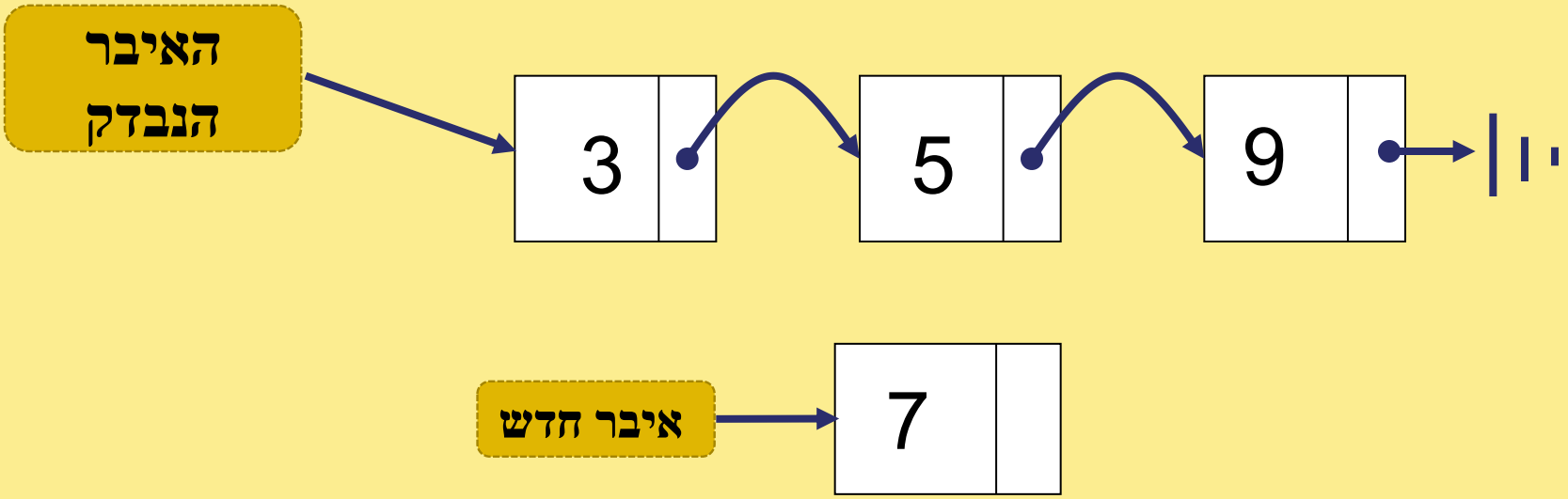
דוגמא: הוספת איבר בסוף



במימוש על-ידי מערך היינו צריכים להזיז אחורה את כל הערכים שלו, כדי להכניס את 7 בסוף וזה בהנחה שהיה מקום עבורו.

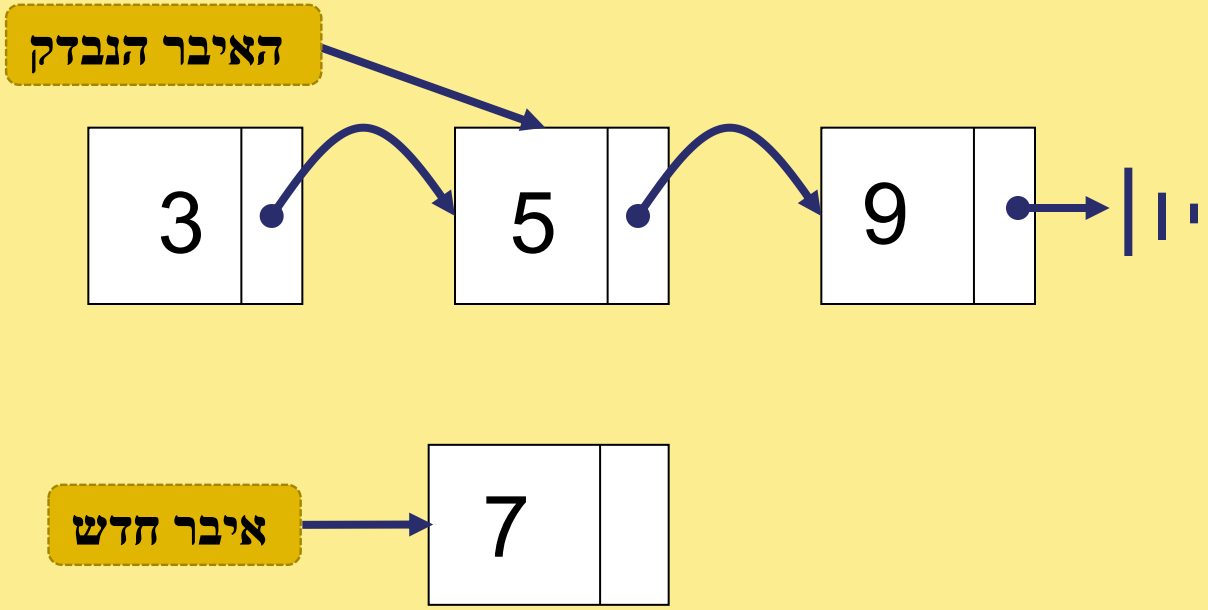
הוספת איבר בסוף – שלב א'

- נחפש את האיבר האחרון ברשימה, וכאשר נמצא אותו נגרום לו להצביע על 7, ו-7 יהפוך להיות האיבר האחרון.
- שלב א' בדיקת האיבר הראשון:



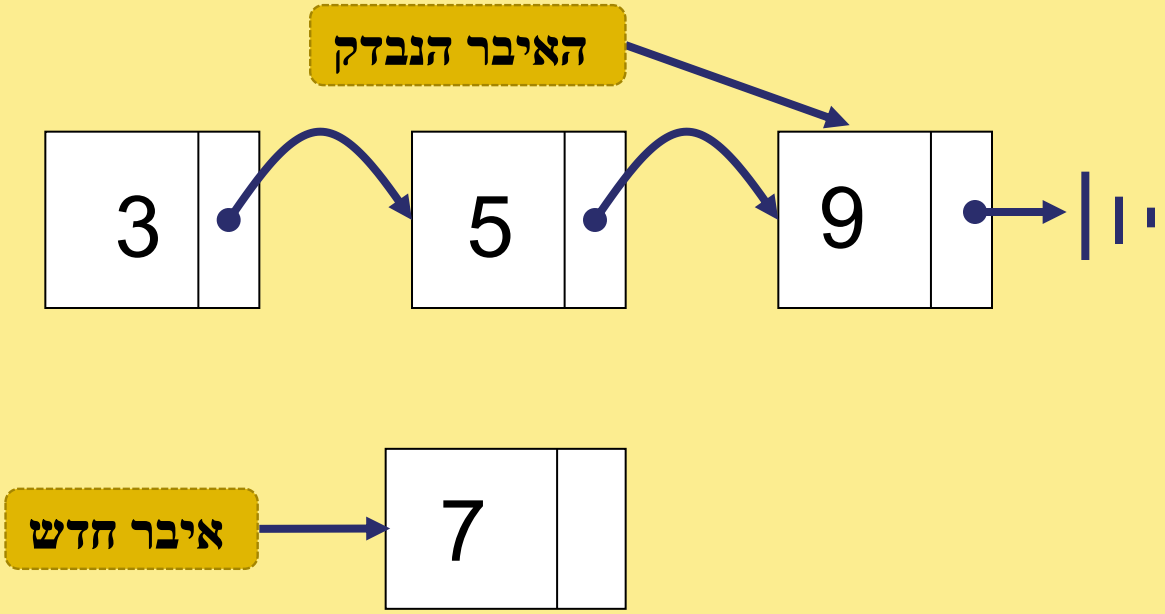
הוספת איבר בסוף – שלב ב'

- נחפש את האיבר האחרון ברשימה, וכאשר נמצא אותו נגרום לו להצביע על 7, ו-7 יהפוך להיות האיבר האחרון.
- שלב ב' בדיקת האיבר השני:



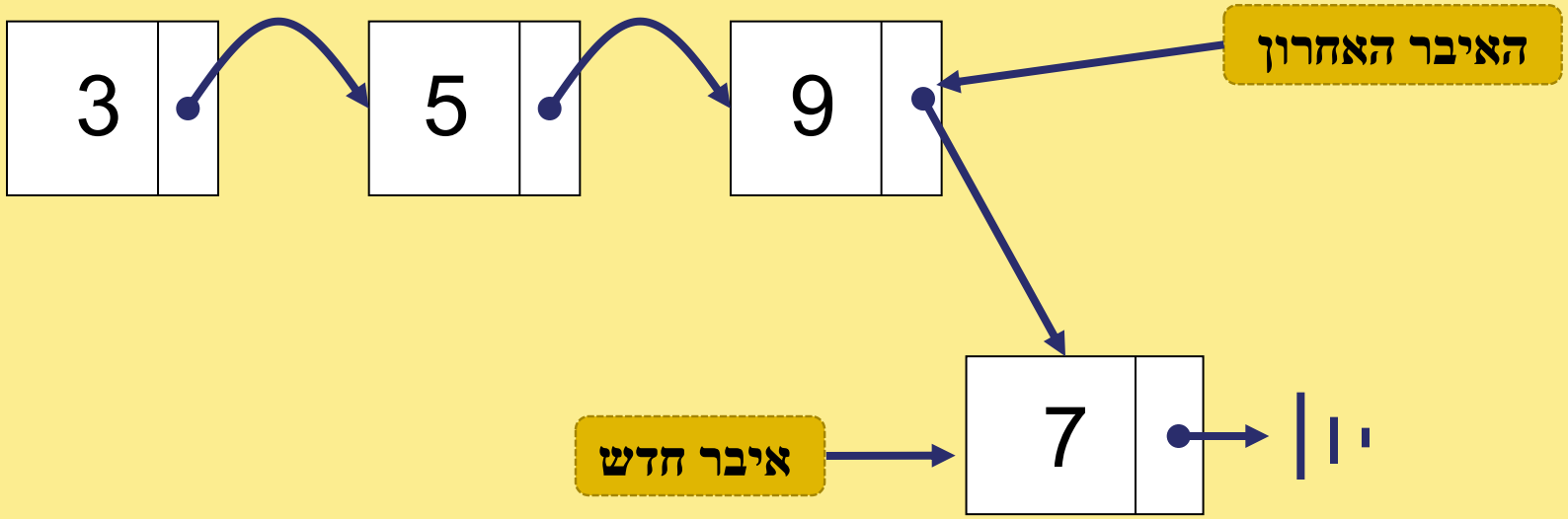
הוספת איבר בסוף – שלב ג'

- נחפש את האיבר האחרון ברשימה, וכאשר נמצא אותו נגרום לו להצביע על 7, ו-7 יהפוך להיות האיבר האחרון.
- שלב ג' בדיקת האיבר השלישי:



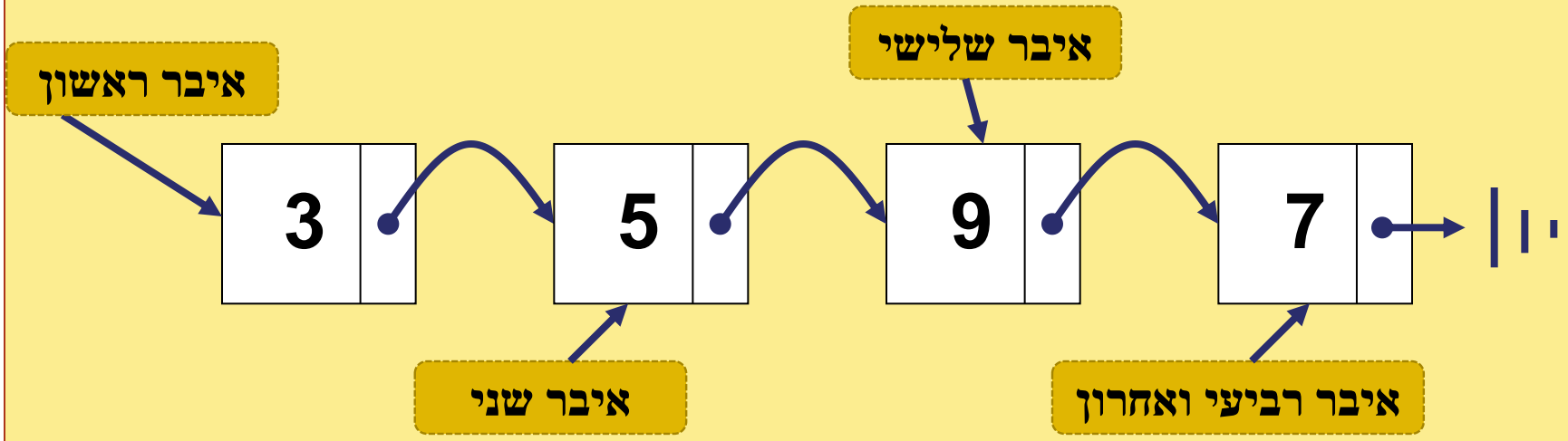
הוספת איבר בסוף – שלב ד'

- נחפש את האיבר האחרון ברשימה, וכאשר נמצא אותו נגרום לו להצביע על 7, ו-7 יהפוך להיות האיבר האחרון.
- שלב ד' איתור האחרון וחיבורו ל- 7:



הוספת איבר בסוף - סיום

- כעת משמצאנו את האיבר האחרון וגרמנו לו להצביע על ה- 7, 7 הפך להיות האיבר האחרון ברשימה.
- הרשימה החדשה לאחר הוספת ה- 7:



פונקציה להוספת איבר בסוף רשימה


```
Item *Add_End(Item *head, int value)
{
   Item *tail=head;
   Item *new_item=(Item*) malloc(sizeof (Item));
   new_item->data = value;
   new_item->next = NULL;
   if(tail == NULL)
         return new_item;
   while (tail->next != NULL)
            tail= tail->next;
   tail->next=new_item;
   return head;
}
```

מקבלים מצביע לראש רשימה וערך

יוצרים איבר חדש עם הערך הדרוש

האיבר החדש יצביע לסוף הרשימה ל- NULL.

מבצעים תנועה על הרשימה עד להגעה לסופה, רק אז מבצעים הכנסת איבר חדש לסוף הרשימה

מחזירים את כתובת התא הראשון

הערה: לצורך ההדגמה אנחנו מניחים כי הקצאת הזיכרון הצליחה.

פונקציה להדפסת רשימה מקושרת (for)

מקבלים מצביע לראש רשימה וערך להוספה

```
void print_list(Item *head)
{
  Item *ptr;
  for (ptr=head; ptr!=NULL; ptr=ptr->next)
  {
    printf ("%4d", ptr-> data);
  }
}
```

מבצעים בדיקה שיש איברים ברשימה והיא לא ריקה

מדפיסים את תוכן האיברים עד סוף הרשימה

פונקציה להדפסת רשימה מקושרת (while)


```
void print_list(Item *head)
{
    Item *ptr=head;
    while (ptr!=NULL)
    {
      printf ("%4d", ptr-> data);
      ptr=ptr->next;
    }
 }
```

מקבלים מצביע לראש רשימה וערך להוספה

מבצעים בדיקה שיש איברים ברשימה והיא לא ריקה

מדפיסים את תוכן האיברים עד סוף הרשימה

פונקציה להדפסת רשימה מקושרת (do while)


```
void print_list(Item *head)
{
    Item *ptr=head;
    do
    {
      printf ("%4d", ptr-> data);
      ptr=ptr->next;
    } while (ptr!=NULL);
 }
```

מקבלים מצביע לראש רשימה וערך להוספה

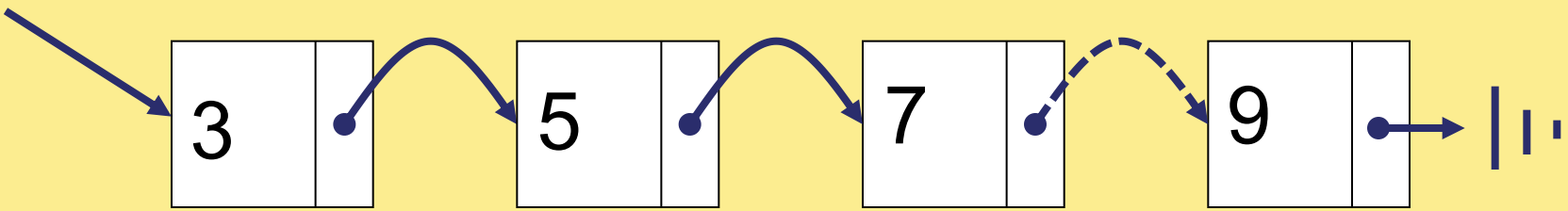
מבצעים בדיקה שיש איברים ברשימה והיא לא ריקה

מדפיסים את תוכן האיברים עד סוף הרשימה

שאלות?

עוד דוגמא: מחיקת איבר מרשימה מקושרת

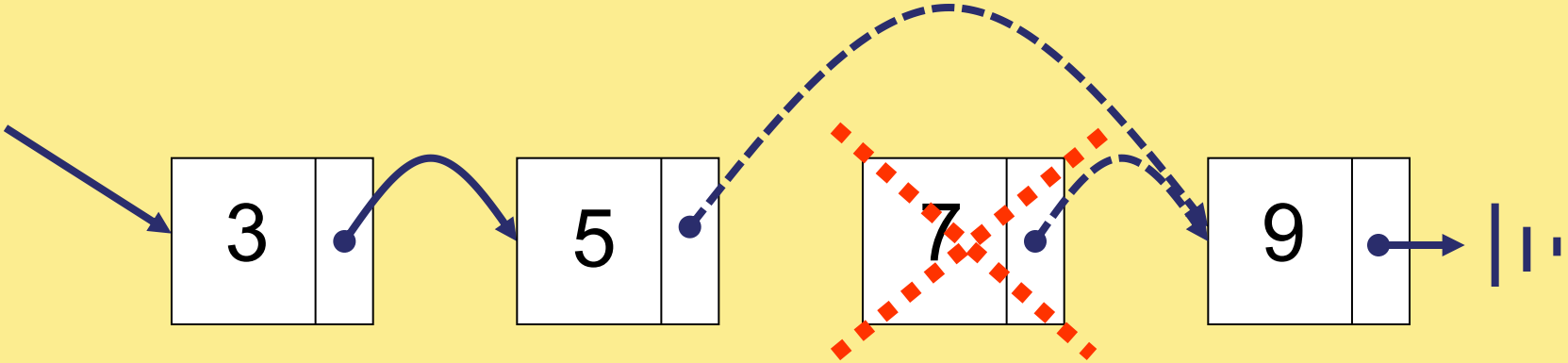
- נתונה הרשימה הבאה של מספרים שלמים:



- רוצים למחוק את האיבר שבו נמצא הערך 7.
- יש לעבור על הרשימה ולאתר את מי שמצביע על האיבר אותו רוצים למחוק (האיבר שערכו 5), ולגרום לו להצביע על מי שאחרי האיבר שערכו 7 (כלומר האיבר שערכו 9).

עוד דוגמא: מחיקת איבר מרשימה מקושרת

- נתונה הרשימה הבאה של מספרים שלמים:



- יוצרים קישור שעוקף את 7, ומוחקים את האיבר עם הערך 7.
- הקישור, המעקף מחבר כעת בין האיבר עם הערך 5 לאיבר עם הערך 9.
- בשימוש במערכים היינו צריכים לצמצם את כל הסדרה.

פונקציה למחיקת איבר ברשימה


```
void Del_Item(Item *head, int value)
{
Item *curr,*prev;
curr=head;
prev= NULL;
while (curr !=NULL)
{
if(curr-> data == value)
        if (prev==NULL)
        {
                head = curr->next;
                free(curr);
                return;
        }
```

מקבלים מצביע לראש רשימה וערך למחיקה

מבצעים חיפוש עד להגעה לסוף הרשימה במידה והאיבר אינו נמצא מציגים הודעה מתאימה

פונקציה למחיקת איבר ברשימה


```
        else
        {
            prev->next = curr->next;
            free(curr);
            printf("The Item %d is deleted !!!\n",value);
         return;
        } //if (prev==NULL)
} //if(curr-> data == value)
prev=curr;
curr=curr->next;
}  // while(curr->next !=NULL)
 printf("No such entry (%d)to delete !!!\n",value);
}
```

מבצעים חיפוש עד להגעה לסוף הרשימה במידה והאיבר נמצא מציגים הודעה מתאימה

מבצעים חיפוש עד להגעה לסוף הרשימה במידה והאיבר אינו נמצא מציגים הודעה מתאימה

רשימות לעומת מערכים

- הוספה ומחיקה של איברים מרשימה ידרשו תמיד רק שינוי מקומי ולא שינוי מערכתי על כל הרשימה (הטיפול יעשה לכל היותר באיבר אחד או שניים), בעוד שבמערכים עשוי להידרש שינוי בכל המערך.
- אין אפשרות של גישה ישירה לכל איבר (למשל כדי לגשת לאיבר המיליון נצטרך לעבור את כל האיברים שלפניו כדי לדעת איפה הוא).
- כשמוסיפים איבר נצטרך להקצות אותו דינאמית, מה שדורש יותר זמן לעומת גישה למערך שהוקצה מראש.

רשימות לעומת מערכים

- בסיום הטיפול ברשימות מקושרות חובה לשחרר את כל הזיכרון איבר אחרי איבר עד סוף הרשימה. במערכים מספיק היה לשחרר את המצביע לראש המערך וכל המערך הדינאמי היה משוחרר.
- נעדיף רשימה מקושרת רק כשיש דרישה לפעילויות של הוספה ומחיקה תכופות.
- נעדיף מערכים כאשר מספר האיברים הינו קבוע ומוגדר מראש ללא שינויים ואין צורך בהקצאה דינאמית של הגדלה והקטנת גודל המערך לעיתים תכופות.

רשימות לעומת מערכים

- בניגוד למערך, רשימה מקושרת איננה טיפוס משתנה שקיים בשפת C והיא מוגדרת ע"י המשתמש ברמה הגלובלית בכל תוכנית.
- כדי לשמור נתונים ברשימה מקושרת, עלינו לבנות אותה בעצמנו, ולממש את הפעולות אותן נבקש לבצע על איברי הרשימה (כמו הוספת איבר, מחיקת איבר, חיפוש איבר, הדפסת הרשימה וכו').
- עלינו להקפיד כי האיבר הראשון ברשימה מקבל מצביע לראש הרשימה והוא עצמו מצביע תמיד לאיבר השני, אם קיים.

מימוש רשימות מקושרות ב-C

- עלינו להקפיד כי האיבר האחרון ברשימה המקושרת מצביע תמיד ל- NULL ואילו האיבר הראשון מצביע לשני אחרת אין לנו רשימה מקושרת.
- רשימה מקושרת היא למעשה אוסף האיברים שלה שהמבנה שלהם זהה מבחינת השדות הפנימיים והם כולם משורשרים ע"י קישור של מצביע ביניהם.
- כדי לעבוד עם רשימה מקושרת יספיק לשמור את כתובת האיבר הראשון שלה, כי ממנו אפשר להתקדם לאחרים.

מימוש רשימות מקושרות ב-C

- הוספה ומחיקה של איברים מרשימה ידרשו תמיד רק שינוי מקומי ולא שינוי מערכתי על כל הרשימה (הטיפול יעשה לכל היותר באיבר אחד או שניים), בעוד שבמערכים עשוי להידרש שינוי בכל המערך.
- אין אפשרות של גישה ישירה לכל איבר (למשל כדי לגשת לאיבר המיליון נצטרך לעבור את כל האיברים שלפניו כדי לדעת איפה הוא).
- כשמוסיפים איבר נצטרך להקצות אותו דינאמית, מה שדורש יותר זמן לעומת גישה למערך שהוקצה מראש.

רשימות לעומת מערכים

- נעדיף מערכים כאשר מספר האיברים הינו קבוע ומוגדר מראש ללא שינויים ואין צורך בהקצאה דינאמית של הגדלה והקטנת גודל המערך לעיתים תכופות.
- בניגוד למערך, רשימה מקושרת איננה טיפוס משתנה שקיים בשפת C והיא מוגדרת ע"י המשתמש ברמה הגלובלית בכל תוכנית.
- כדי לשמור נתונים ברשימה מקושרת, עלינו לבנות אותה בעצמנו, ולממש את הפעולות אותן נבקש לבצע על איברי הרשימה (כמו הוספת איבר, מחיקת איבר, חיפוש איבר, הדפסת הרשימה וכו').

מימוש רשימות מקושרות ב-C

- עלינו להקפיד כי האיבר הראשון ברשימה מקבל מצביע לראש הרשימה והוא עצמו מצביע תמיד לאיבר השני, אם קיים.
- עלינו להקפיד כי האיבר האחרון ברשימה המקושרת מצביע תמיד ל- NULL ואילו האיבר הראשון מצביע לשני אחרת אין לנו רשימה מקושרת.
- רשימה מקושרת היא למעשה אוסף האיברים שלה שהמבנה שלהם זהה מבחינת השדות הפנימיים והם כולם משורשרים ע"י קישור של מצביע ביניהם.

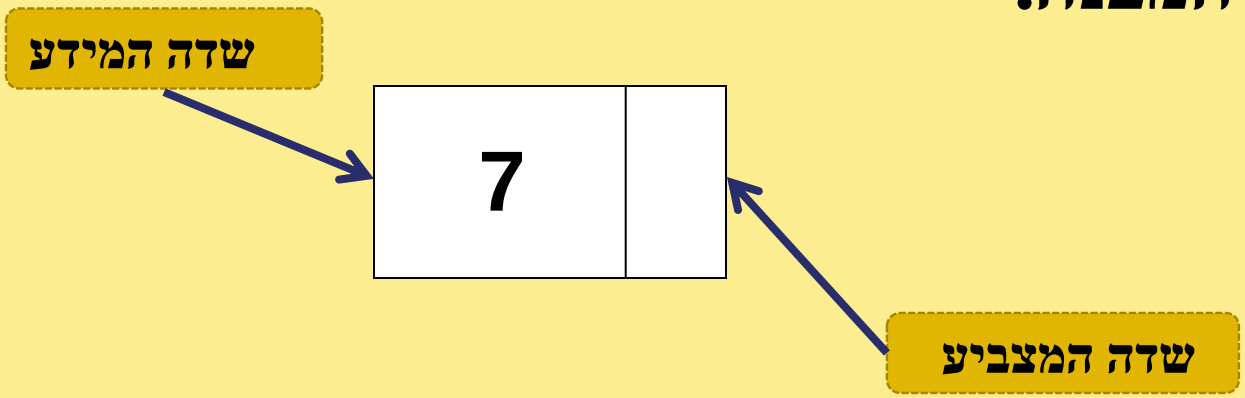
מימוש רשימות מקושרות ב-C

- כדי לעבוד עם רשימה מקושרת יספיק לשמור את כתובת האיבר הראשון שלה, כי ממנו אפשר להתקדם לאחרים.
- קיימות רשימות מקושרות דו כיווניות וסיבוביות בהן האיבר האחרון מצביע על הראשון וכך נוצר מעגל איברים.
- ניתן להוסיף שדות למבנה לפי הצורך.
- כל שדה המוגדר במבנה הראשי של איבר ברשימה מקושרת יכול להיות מכל אחד מטיפוסי הנתונים הידועים לנו.

מימוש רשימות מקושרות ב-C

- איבר ברשימה ייוצג תמיד רק על-ידי מבנה שיכיל לפחות שני שדות:

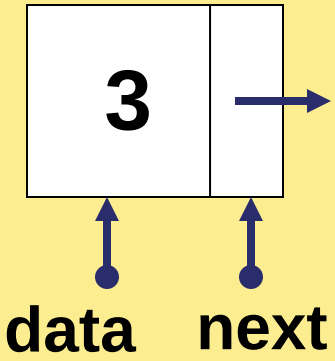
1. שדה אחד יכיל מידע.
2. שדה שני יכיל מצביע לאיבר הבא, חייב להיות מצביע מטיפוס המבנה.



מימוש רשימות מקושרות ב-C

- אז עבור הדוגמא של רשימת מספרים שלמים נגדיר איבר ע"י:

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

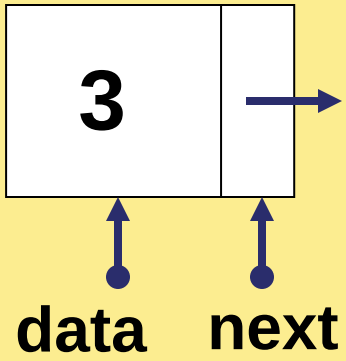


- השדה data יכיל את המספר שהאיבר מכיל.
- השדה next יכיל מצביע לאיבר הבא ברשימה.
- (אם אין איבר נוסף, כלומר זה האיבר האחרון, אז ערך המצביע הזה יהיה NULL)

מימוש רשימות מקושרות ב-C

- נוח להשתמש בריישום המקוצר:

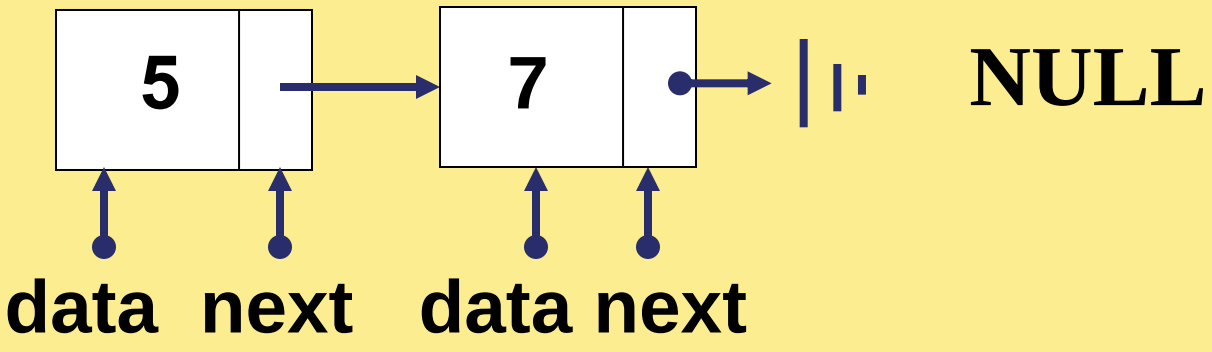
```
typedef struct Item
{
    int data;
    struct Item *next;
} Item;
```



- השדה data יכיל את המספר שהאיבר מכיל.
- השדה next יכיל מצביע לאיבר הבא ברשימה (אם אין איבר נוסף אז הוא יכיל את הערך NULL).
- בקובץ stdlib.h מוגדר #define NULL 0 והמוסכמה היא לסמן שמצביע לא מצביע על כלום ע"י הערך NULL (אפס).

מימוש רשימות מקושרות ב-C

- שימו לב שהמצביע next מצביע על האיבר הבא, שהוא מאותו סוג כמו האיבר הנוכחי (כי כל איברי הרשימה הם מאותו הסוג).
- האחרון מצביע ל סוף הרשימה על NULL.



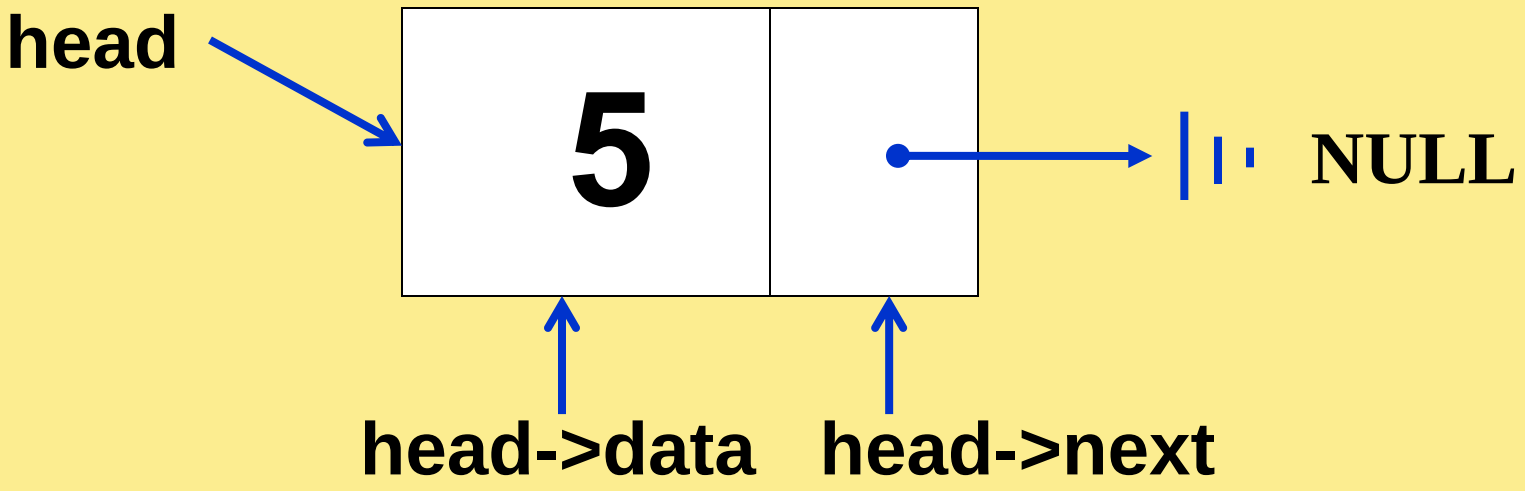
מימוש רשימות מקושרות ב-C

- בהגדרת מבנה ניתן לכלול מצביע למבנה מאותו סוג:
  - הקומפיילר מסתדר עם זה והקומפילציה תעבור בהצלחה.
  - מבחינת ההגדרה מספיק לו, לקומפיילר, לדעת שתאוחסן שם כתובת.
- הגדרנו טיפוס-משתנה שמייצג איבר ברשימה. במקרים רבים נעבוד עם מצביעים לאיברים ברשימה.
  - למשל אמרנו שתמיד נשמור מצביע לאיבר הראשון.
  - כל איבר מכיל מצביע לאיבר שאחריו.

מימוש רשימות מקושרות ב-C

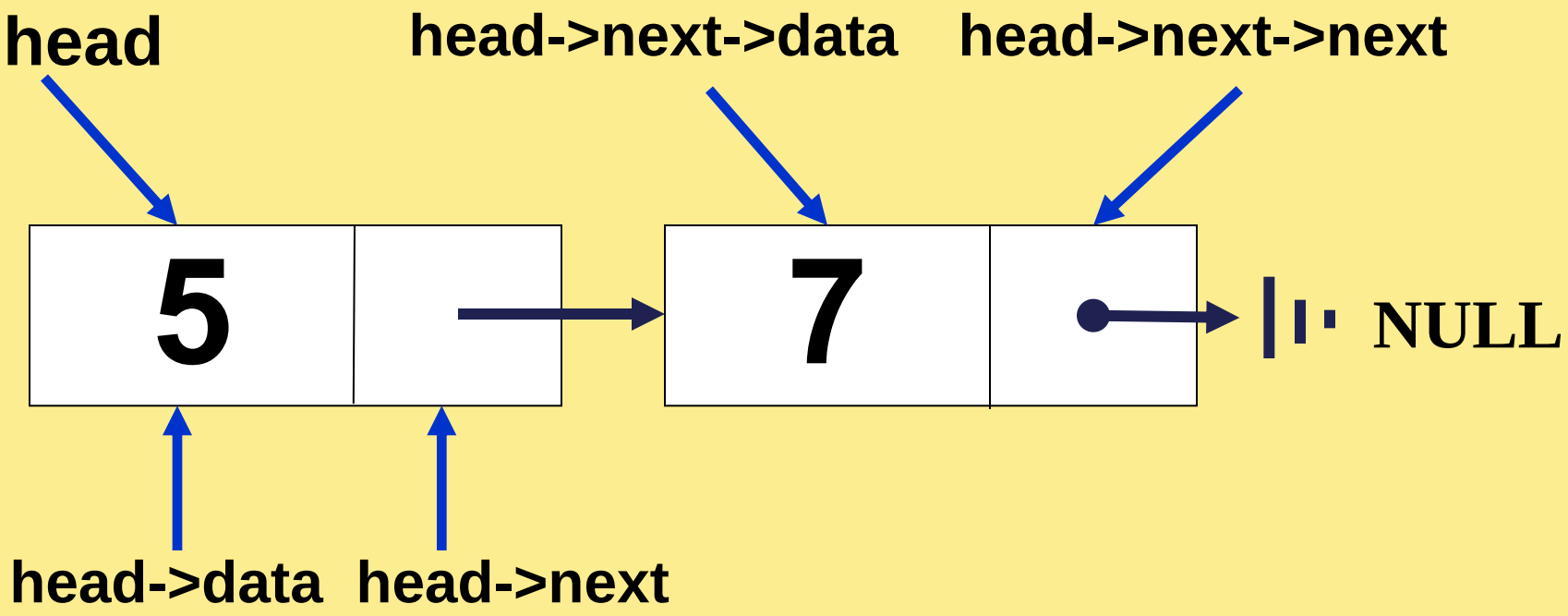
- נזכיר שאפשר להתייחס לשדות ע"י מצביע בעזרת <-

לדוגמא:



מימוש רשימות מקושרות ב-C

- כמו-כן, אפשר להתייחס באופן דומה גם לשדות של האיבר הבא ולדוגמא:



מימוש רשימות מקושרות ב-C

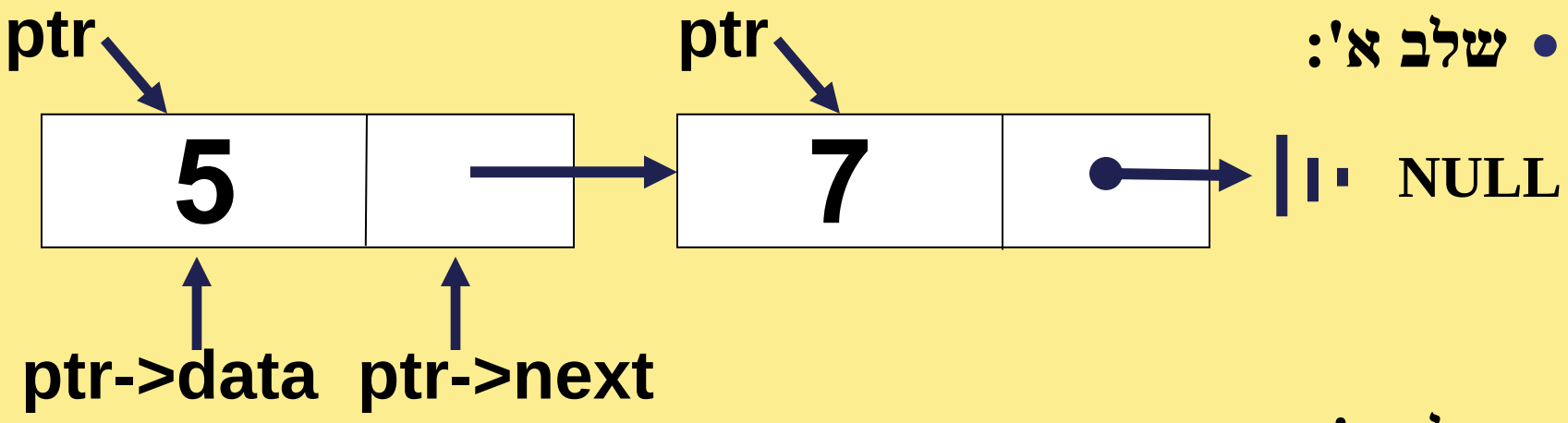
- כשיש לנו מצביע בשם ptr לאיבר ברשימה, ונשאף להצביע על האיבר הבא ברשימה, נתקדם ע"י הפקודה:

```
ptr = ptr -> next;
```

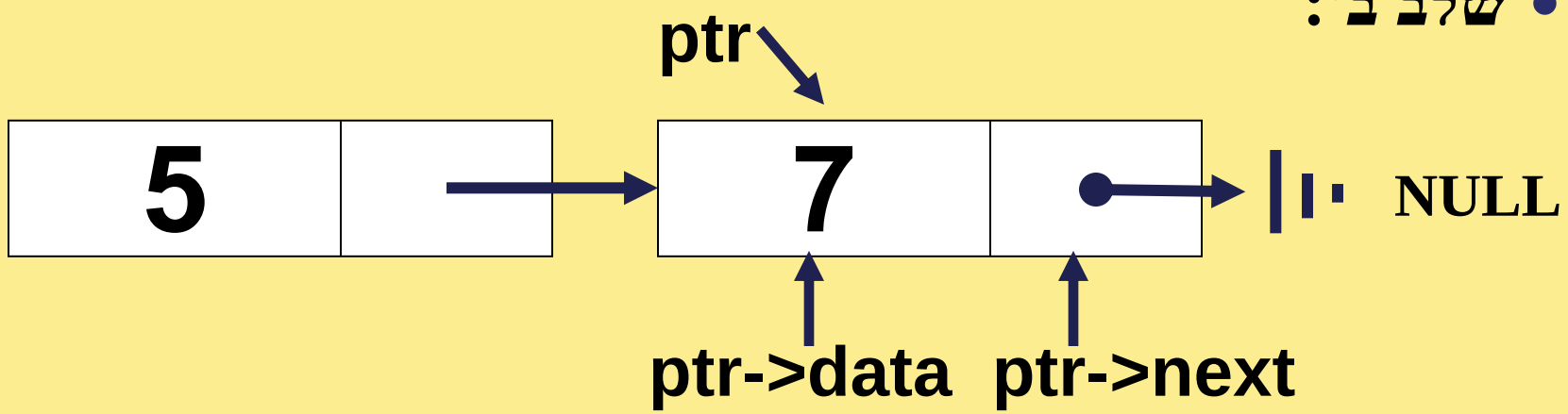
- עכשיו המצביע ptr יכיל את כתובת האיבר הבא ברשימה.
- זה מה שמקביל ברשימות מקושרות לקידום האינדקס בלולאה במעבר על תאי מערך.
- סוף הרשימה הוא כשמגיעים לכתובת NULL.

מימוש רשימות מקושרות ב-C

- מהלך ההתקדמות על פני הרשימה המקושרת הוא כדלהלן:
- שלב א':



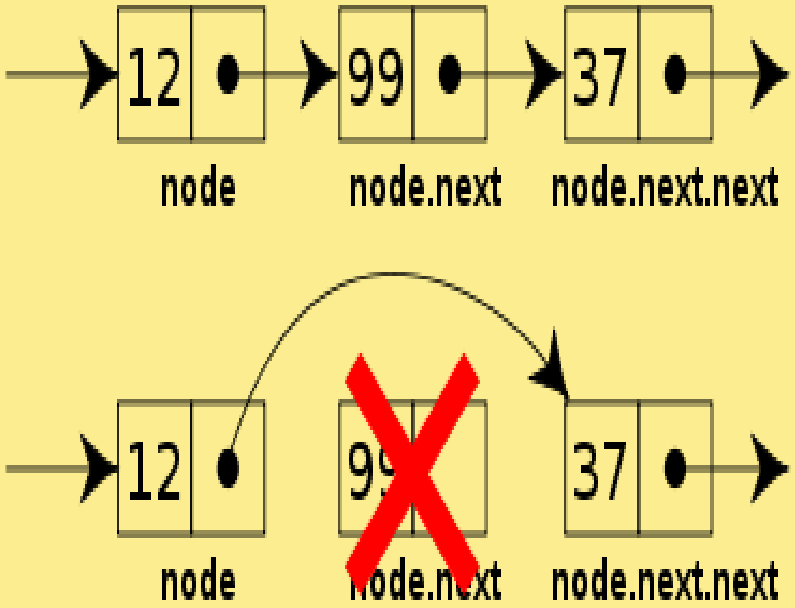
- שלב ב':



שאלות?

מציאת איבר ברשימה

- פעולה נפוצה נוספת הניתנת למימוש היא חיפוש איבר ברשימה והחזרת מצביע אליו.
- הפרמטרים לפונקציה יהיו:
  - מצביע לראש הרשימה
  - הערך שאותו מחפשים
- הערך המוחזר יהיה:
  - מצביע לאיבר עם הערך המבוקש.
  - יוחזר NULL אם האיבר אינו ברשימה.



מציאת איבר ברשימה - מימוש

- הפונקציה מתקדמת על איברי הרשימה לפי המצביעים, אחד אחרי האחר, עד שהערך המבוקש נמצא או עד שמגיעים לסוף הרשימה כלומר מצביעים על NULL.
- ברגע שמתקיים אחד מהתנאים האלה, הפונקציה מחזירה את המקום, שהוא או מצביע לאיבר המבוקש או סוף הרשימה, כלומר ל- NULL.

מציאת איבר ברשימה


```
Item *Find(Item *head, int value)
{
   Item *check = head;
   while ((check  != NULL) && (check->data != value))
         check = check->next;
   return check;
}
```

הפונקציה מקבלת מצביע לראש הרשימה וערך מבוקש

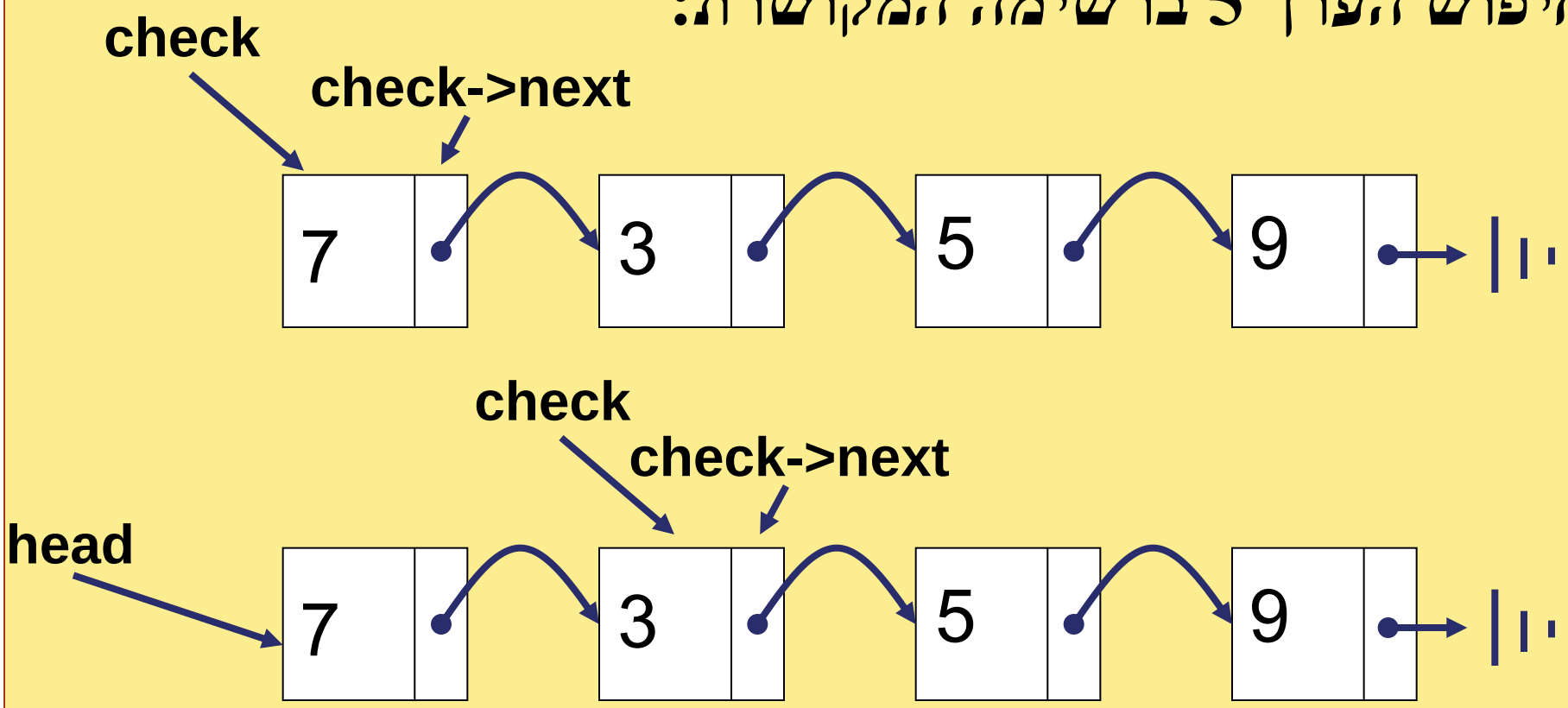
כל עוד לא הגענו לסוף ולא מצאנו את הערך המבוקש – מתקדמים ברשימה.

כשהגענו לערך המבוקש או לסוף, מחזירים מצביע למקום שהגענו אליו

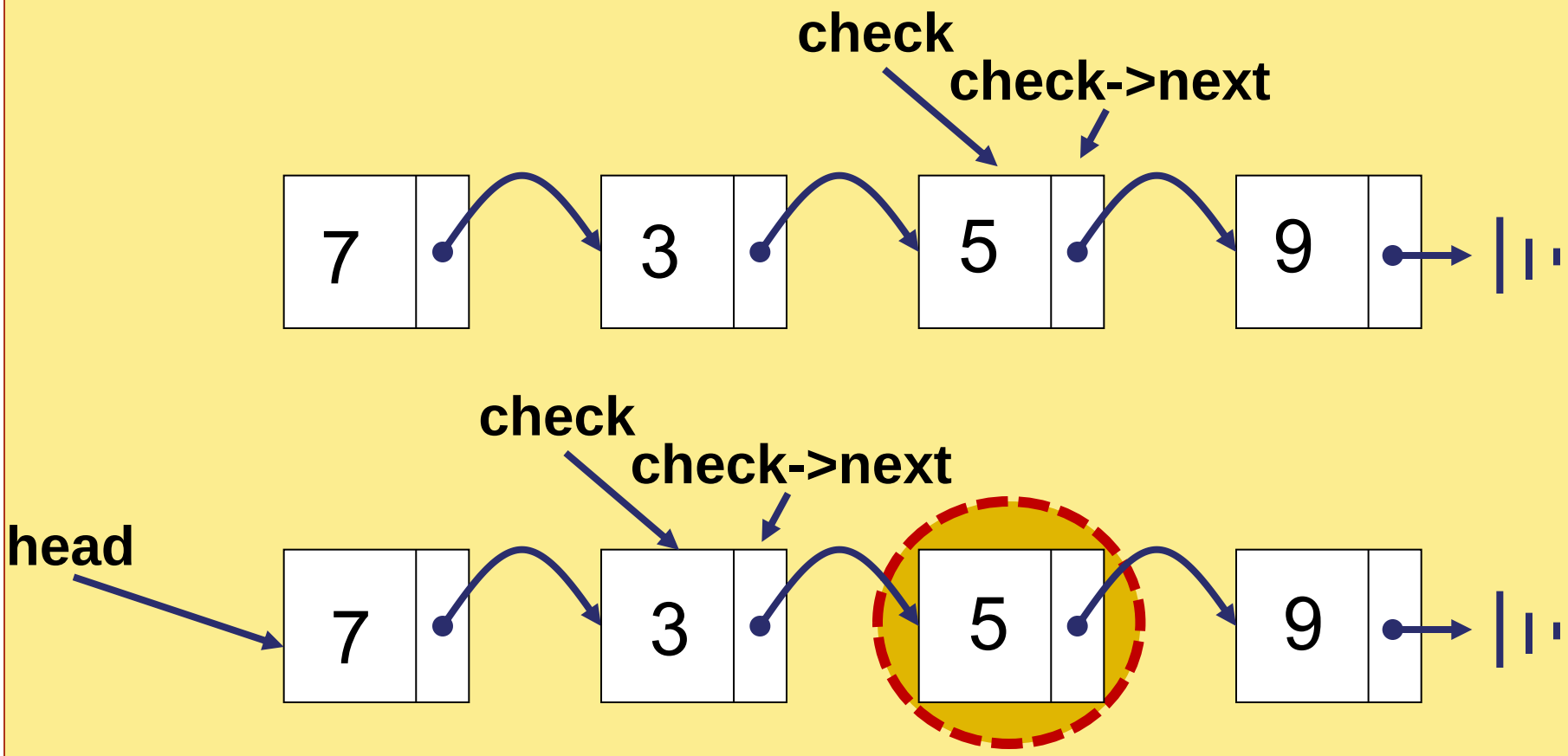
(באופן דומה יכולנו לחפש את האיבר הראשון שמקיים תנאי כלשהו)

רשימות מקושרות-מציאת איבר

חיפוש הערך 5 ברשימה המקושרת:



רשימות מקושרות-מציאת איבר



נקודה לתשומת-לב לגבי תנאים ב-C

- כשיש AND בין כמה תנאים, אם אחד מהם לא מתקיים אז כבר ברור שהערך הוא false, ולכן שאר התנאים ב-AND לא נבדקים.
- למשל אם check==NULL אז התנאי השני לא ייבדק (ולא תהיה תעופה):

```
while ((check != NULL)  &&
        (check->data != value))
```

נקודה לתשומת-לב לגבי תנאים ב-C

- בדרך-כלל לא הסתמכנו על התכונה הזאת בקורס (בדקנו קודם בנפרד תנאי שעלול לגרום לתעופה), אבל לפעמים נוח להשתמש בה.
- באופן דומה, כשיש OR, תנאי או, בין כמה תנאים, אם אחד מהם מתקיים אז השאר לא נבדקים, כי כבר ברור שהערך הוא true (הבדיקה נעשית משמאל לימין).
- למשל כאן אם n==0 אז התנאי השני לא ייבדק, ולכן אין חשש לתעופה:

```
if ((n == 0) || (k/n < 0.5))
```

שחרור רשימה מקושרת

- בסיום השימוש ברשימה נרצה לשחרר את כל הזיכרון שהיא משתמשת בו.
  - בשונה ממערך שהוקצה דינאמית, free של ראש הרשימה משחרר רק את האיבר הראשון.
  - כל אחד מאיברי הרשימה הוקצה ע"י malloc נפרד, ו- free משחרר את מה שהוקצה בפקודת malloc אחת בדיוק.
- נכתוב פונקציה שמקבלת מצביע לראש הרשימה ועוברת ומשחררת את כל האיברים שהוקצו על ידי פונקצית ההוספה.

שחרור רשימה מקושרת - מימוש


```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

מקבלים מצביע לראש רשימה מקושרת

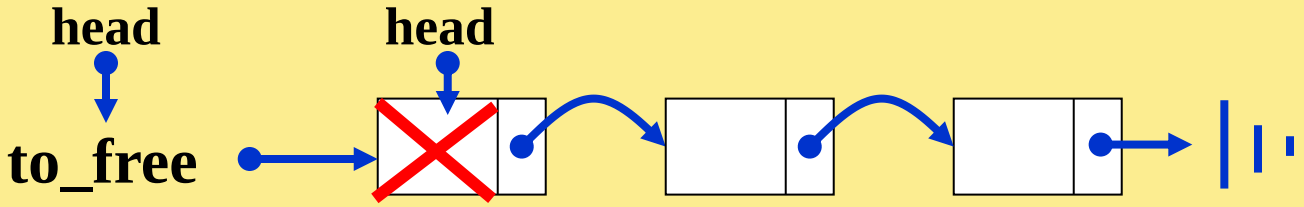
מתחילים לשחרר מראש הרשימה.

כל פעם משחררים את האיבר הראשון ברשימה שנשארה, עד שכולה משוחררת.

ראש הרשימה מתקדם לאיבר הבא.

משחררים את ראש הרשימה הקודם.

באיטרציה הבאה ישוחרר ראש הרשימה הבא.



שחרור רשימה - מימוש


```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

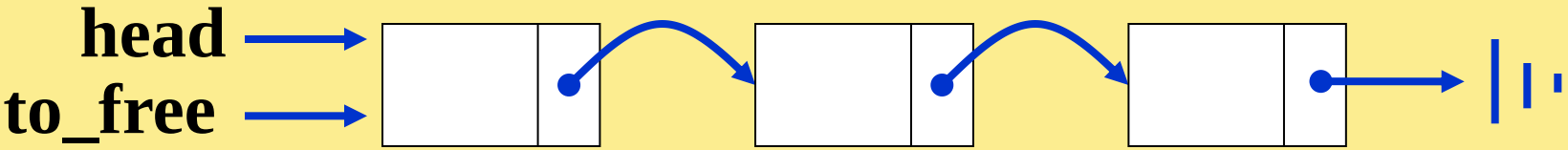
כל פעם מוחקים את
האיבר הראשון
שנשאר ברשימה,
עד שהיא מתרוקנת.

שחרור רשימה - מימוש


```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

מצביע נוסף על האיבר הראשון, שישוחרר

ממשיכים כל עוד לא הגענו לסוף הרשימה



שחרור רשימה - מימוש

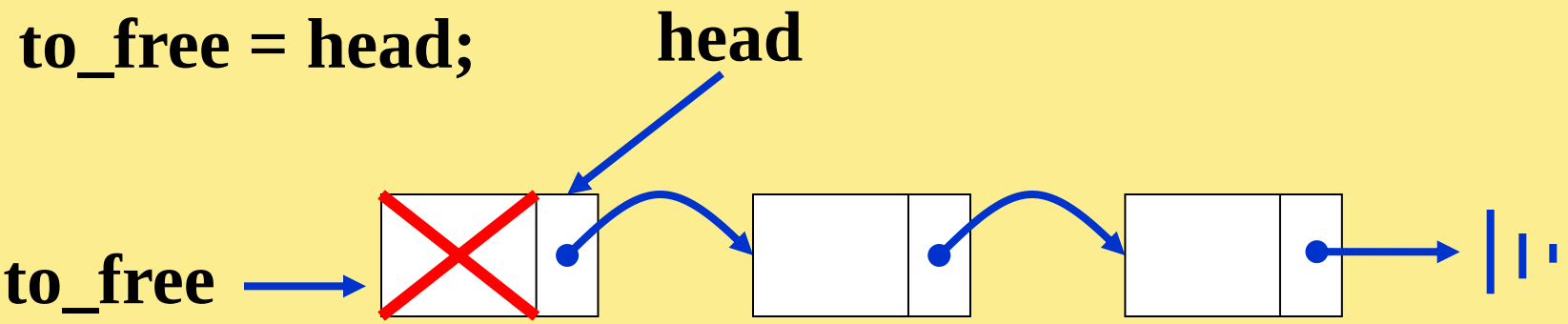

```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

מצביע נוסף על האיבר הראשון, שישוחרר

ממשיכים כל עוד לא הגענו לסוף הרשימה

ראש הרשימה מתקדם

האיבר הראשון משוחרר



שחרור רשימה - מימוש


```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

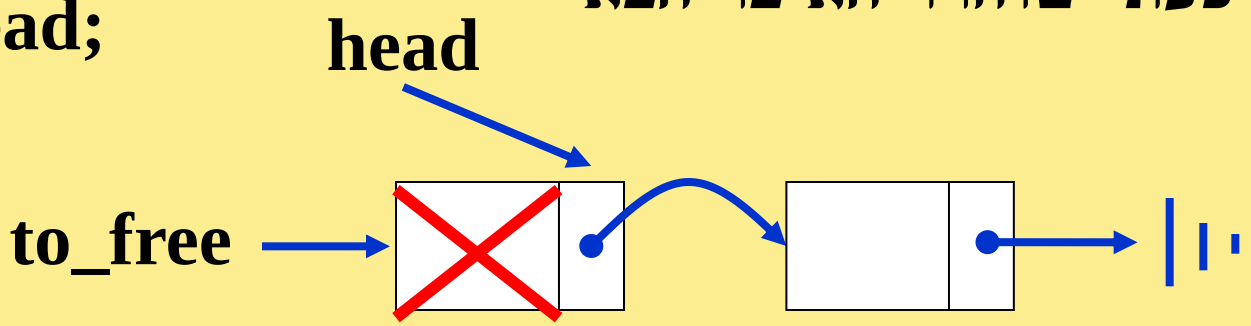
מצביע נוסף על האיבר הראשון, שישוחרר

ממשיכים כל עוד לא הגענו לסוף הרשימה

ראש הרשימה מתקדם

האיבר הראשון משוחרר

כעת ישוחרר האיבר הבא



שחרור רשימה - מימוש


```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

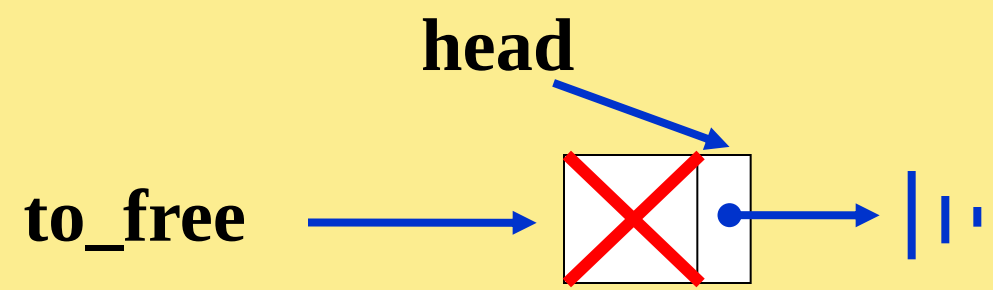
מצביע נוסף על האיבר הראשון, שישוחרר

ממשיכים כל עוד לא הגענו לסוף הרשימה

ראש הרשימה מתקדם

האיבר הראשון משוחרר

כעת ישוחרר האיבר הבא



שחרור רשימה - מימוש


```
void FreeList (Item *head)
{
   Item *to_free = head;
   while (to_free != NULL)
   {
         head = head->next;
         free(to_free);
         to_free = head;
   }
}
```

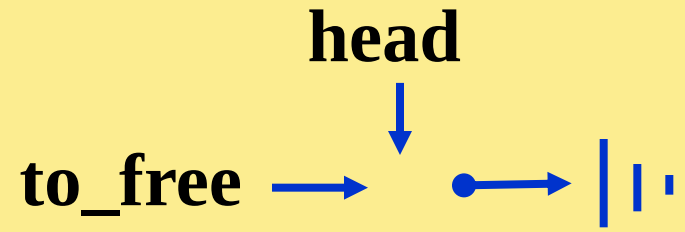
מצביע נוסף על האיבר הראשון, שישוחרר

ממשיכים כל עוד לא הגענו לסוף הרשימה

ראש הרשימה מתקדם

האיבר הראשון משוחרר

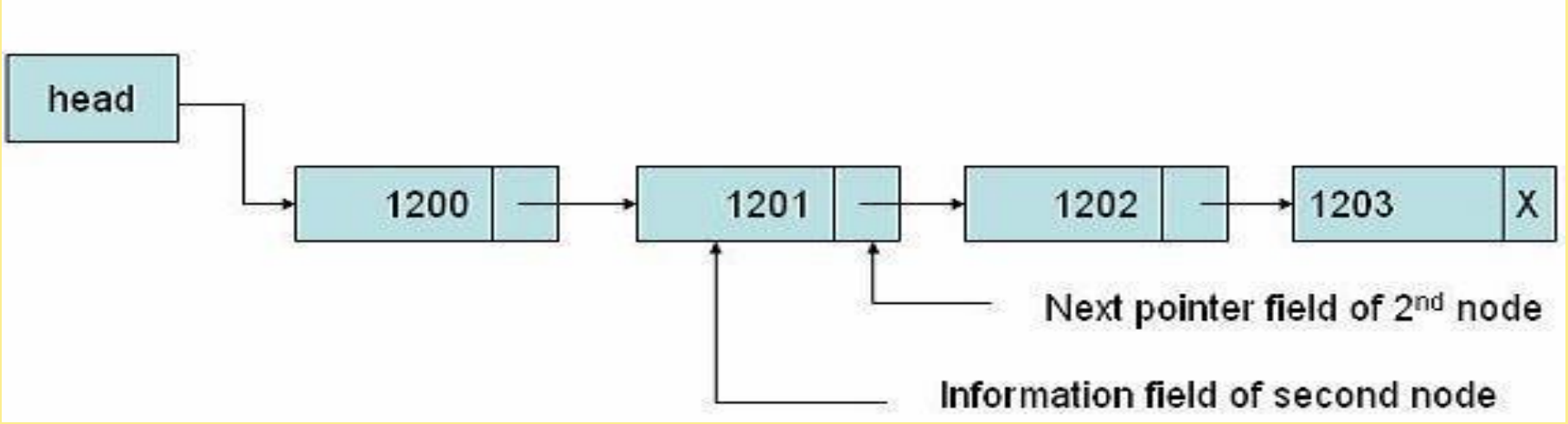
כעת ישוחרר האיבר הבא



כשהגענו ל- NULL מסיימים

רשימות מקושרות ב-C – סיכום ביניים

- הוספה איבר בהתחלת רשימה מקושרת
- הוספת איבר בסוף רשימה מקושרת
- מציאת איבר ברשימה מקושרת
- שחרור רשימה מקושרת



שאלות?

רשימות מקושרות דוגמאות

הדפסת רקורסיבית של רשימה מקושרת:

```
void PrintList(Item *head)
{
    if (head == NULL){
            printf("\n");
        return;}
    else
    {
        printf("%d ",head-> data);
        PrintList(head->next);return;
    }
}
```

רשימות מקושרות דוגמאות

פונקציה לספירת איברי רשימה מקושרת:

```
int Count(Item * head)
{
        int counter=0;
        while (head != NULL)
        {
                counter++;
                head=head->next;
        }
        return counter;
}
```

רשימות מקושרות דוגמאות - רקורסיבית

פונקציה רקורסיבית לספירת איברי רשימה מקושרת:

```
int CountR(Item * head)
{
        if (head == NULL)
            return 0;
        return 1 + CountR(head->next);
}
```

רשימות מקושרות דוגמאות

פונקציה למחיקת רשימה מקושרת:

```
void DeleteList(Item *head)
{
        Item *curr=head;
        while (curr != NULL)
        {
                head=head->next;
                free(curr);
                curr=head;
        }
        return;
}
```

רשימות מקושרות דוגמאות-רקורסיבית

פונקציה רקורסיבית למחיקת רשימה מקושרת:

```
void DeleteList(Item *head)
{
        if (head != NULL)
        {
                DeleteList(head->next);
                free(head);
        }
        return;
}
```

רשימות מקושרות דוגמאות

פונקציה המקבלת מצביעים לשתי רשימות מקושרות ומחברת אותן לרשימה אחת.

```
Item *Meld(Item *List1,Item *List2)
{
        Item *List1_head = List1;
        if (List1 == NULL)                  return List2;
        if (List2 == NULL)                  return List1;
        while (List1->next != NULL)
                List1=List1->next;
        List1->next = List2;
        return List1_head;
}
```

שאלות?

רשימות מקושרות דוגמאות

- הכנסה ממוינת של נתונים לאיברים:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
    int data;
    struct node *next;
 } Node;

    Node *insert (int input,Node *list);
    void free_list (Node *list);
    void print_list (Node *list);
```

רשימות מקושרות דוגמאות


```
int main()
{
    int input;
    Node *list=NULL;
    while (scanf("%d", &input) == 1)
    {
        list=insert(input,list);
    }
    print_list(list);
    free_list(list);
    return 0;
}
```

רשימות מקושרות דוגמאות


```
Node *insert (int input , Node *list)
{
    Node *p,*q;
    if ((p=(Node *)malloc(sizeof (Node)))==NULL)
    {
        printf ("Cannot allocate memory\n");
        exit (1);
    }
```

רשימות מקושרות דוגמאות


```
p-> data =input;
if (list==NULL || list-> data > input)
{
        p->next=list;
        return p;
}
else
```

רשימות מקושרות דוגמאות


```
{
   q=list;
   while(q->next!=NULL&&(q->next->data< input))
   {
        q=q->next;
   }
   p->next=q->next;
   q->next=p;
   return list;
    }
}
```

רשימות מקושרות דוגמאות

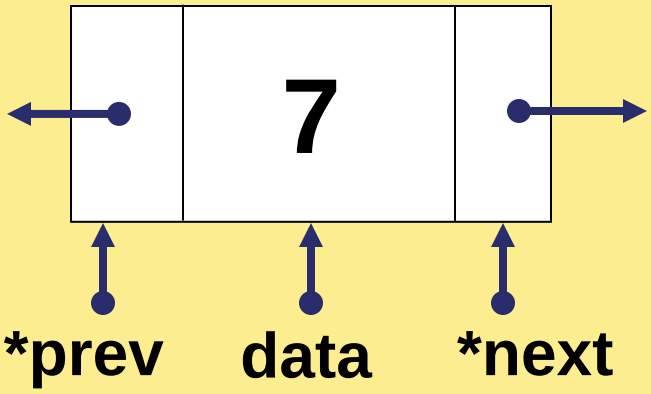

```
void free_list(Node *list)
{
    Node *p;
    while (list!=NULL)
    {
        p=list->next;
        free(list);
        list=p;
    }
}
```

רשימות מקושרות דוגמאות


```
void print_list(Node *list)
{
    Node *p;
    for (p=list; p!=NULL; p=p->next)
    {
        printf ("%4d", p-> data);
    }
    printf ("\n");
}
```

מבנה איבר ראשון ברשימה מקושרת דו כיוונית


```
typedef struct Item
{
    int data;
    struct Item *next;
    struct Item *prev;
}Item;
```



- השדה data מכיל את הערך שהאיבר מכיל.
- השדה next* מכיל מצביע לאיבר הבא ברשימה.
- השדה prev* מכיל מצביע לאיבר הקודם ברשימה.
- אם אין איבר נוסף, כלומר זה האיבר האחרון, אז ערך המצביע next* יהיה NULL.

הדפסת רשימה מקושרת דו כיוונית


```
void print_list(Item *list)
{
    Item *tmp;
    printf("\nThe list from start to end\n");
    for (tmp=list ; tmp != NULL ; tmp=tmp->next)
        printf("%3d ",tmp -> data);
}
```

הדפסת רשימה מקושרת דו כיווניות -הפוך


```
void print_rev(Item *list)
{
    Item *tmp;
    printf("\nThe list from end to start\n");
    for(tmp=list ; tmp != NULL ; tmp=tmp->prev)
        printf("%3d ",tmp -> data);
}
```

הוספת איבר לרשימות מקושרת דו כיוונית


```
Item  insert_first (Item *list, int  input)
{
        Item *tmp;
        if (!(tmp=(Item*)malloc(sizeof( Item)))) exit(1);
        tmp->data = input;
        if (list == NULL){ /* if the list is empty */
                tmp->next = NULL;
                tmp->prev = NULL;
                list = tmp;
                return list;}
        tmp-> next = list;
        list-> prev = tmp;
        tmp-> prev = NULL;
        list = tmp;
        return list;
}
```

מחיקת איבר ברשימה מקושרת דו כיווניות


```
Item remove_first (Item list, int * input)
{
        Item tmp;
        if (list==NULL) /* if the list is empty */
                return NULL;
        if (list->next == NULL) { /* if the list contains one element */
                *input = list-> data;
                free(list);
                return NULL;}
        tmp = list->next;
        tmp -> prev = NULL;
        *input = list-> data;
        free(list);
        return tmp;
}
```

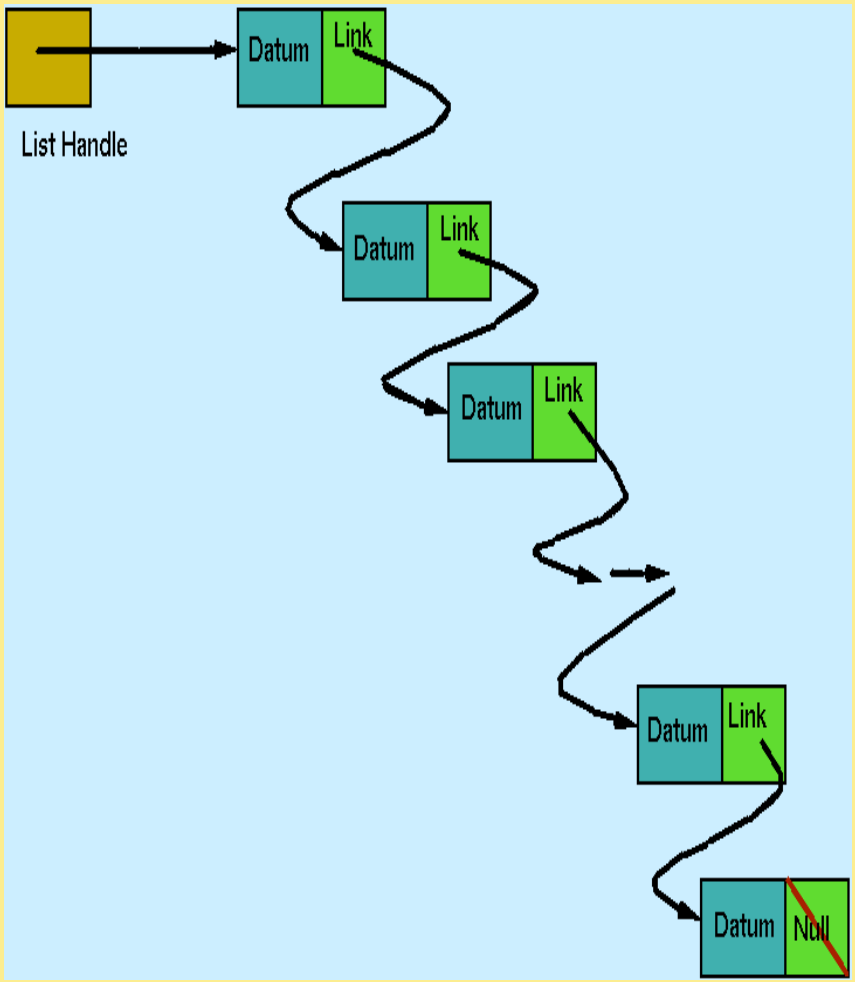
שרשור רשימות מקושרות דוגמא


```
Item *Meld(Item *List1,Item *List2)
{
   Item *List1_head = List1;
   if (List1 == NULL)
         return List2;
   if (List2 == NULL)
         return List1;
   while (List1->next != NULL)
         List1=List1->next;
   List1->next = List2;
   return List1_head;
}
```

סיכום: מה ראינו לגבי רשימות מקושרות

- כשיתכנו מחיקות ותוספות של נתונים, שימוש במערכים אינו יעיל.
- משתמשים ברשימה מקושרת, שבה כל איבר כולל גם מצביע לאיבר הבא, ושומרים מצביע לאיבר הראשון.
- ראינו איך לממש הוספה בהתחלה, חיפוש, ושחרור.
  - ההתקדמות לאורך הרשימה היא בעזרת המצביעים.
  - להוספה מקצים דינמית איבר חדש ומחברים אותו עם המצביעים.
  - יהיו דוגמאות נוספות בתרגול, וגם בשיעור הבא.
- כשאין תוספות ומחיקות – נשתמש במערכים.

רשימות מקושרות סיכום



- צורך ברשימה מקושרת.
- יתרונות חסרונות.
- הוספת איבר בהתחלת הרשימה.
- הוספת איבר בסוף הרשימה.
- מחיקת איבר.
- חיפוש איבר.
- מחיקת רשימה מקושרת.
- רשימה מקושרת ורקורסיה.
- רשימה מקושרת דו כיוונית.

שאלות?

תרגילי כיתה - 1

- כתוב תוכנית היוצרת רשימה מקושרת של איברים בעלי ערך רנדומלי עד 100.
- התוכנית תבצע את המשימות הבאות ע"י שימוש בפונקציות.

1. פונקציה ראשונה תקלוט מספרים רנדומליים לרשימה המקושרת.
2. פונקציה שניה תבצע הדפסה רקורסיבית של ערכי הרשימה המקושרת.
3. פונקציה שלישית תחשב ותדפיס את ממוצע הרשימה הרנדומלית.
4. הפונקציה הרביעית מוחקת את כל איברי הרשימה המקושרת באופן רקורסיבי.

תרגילי כיתה - 2

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות לגבי רשימה מקושרת של פרטי סטודנטים בהם שם וגיל:

1. קליטת איברים ויצירת רשימה מקושרת. ניתן לביצוע ע"י הוספת איבר בהתחלת הרשימה או בסופה.
2. הדפסת רשימה מקושרת.
3. חיפוש איבר ברשימה מקושרת.
4. מחיקת איבר ברשימה מקושרת.
5. מחיקת רשימה מקושרת.

תרגילי כיתה - 3

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:
  1. קליטת איברים ויצירת שתי רשימות מקושרות.
  2. הדפסת כל רשימה בנפרד.
  3. ביצוע שרשור של שתי הרשימות לרשימה אחת משולבת.
  4. הדפסת הרשימה המקושרת החדשה.
  5. חיפוש איבר ברשימה המקושרת החדשה.
  6. מחיקת איבר ברשימה המקושרת החדשה.
  7. מחיקת הרשימה המקושרת החדשה.

תרגילי כיתה - 4

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:

1. קליטת איברים ויצירת רשימה מקושרת. ניתן לביצוע ע"י הוספת איבר בהתחלת הרשימה או בסופה או קלט רנדומלי.
2. הדפסת הרשימה המקושרת.
3. מיון הרשימה המקושרת, ניתן לביצוע במהלך קליטת הנתונים או בפונקציית מיון נפרדת.
4. הדפסת הרשימה המקושרת הממוינת.
5. מחיקת הרשימה המקושרת הממוינת.

תרגילי כיתה - 5

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:

1. קליטת איברים ויצירת רשימה מקושרת דו כיוונית. ניתן לביצוע ע"י הוספת איבר בהתחלת הרשימה או בסופה.
2. הפיכת הרשימה המקושרת האיבר הראשון לאחרון וכדומה.
3. הדפסת הרשימה המקושרת ההפוכה.
4. חיפוש איבר ברשימה המקושרת החדשה, ההפוכה.
5. מחיקת איבר ברשימה המקושרת החדשה, ההפוכה.
6. מחיקת הרשימה המקושרת ההפוכה.

תרגילי כיתה - 6

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:

1. קליטת מספרים שלמים, עד 10 לכל רשימה, לשתי רשימות שונות ויצירת שתי רשימות מקושרות.
2. הדפסת כל רשימה בנפרד.
3. ביצוע פעולת חיבור מתמטי של שתי הרשימות לרשימה חדשה שלישית במספר.
4. הדפסת הרשימה המקושרת החדשה הכוללת את תוצאת החיבור של שתי הרשימות המקוריות.

תרגילי כיתה - 7

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:

1. קליטת מספרים שלמים, לרשימה מקושרת דו כיוונית.
2. חיפוש על פני הרשימה מי המספר הגדול ביותר שנקלט.
3. חיפוש על פני הרשימה מי המספר הקטן ביותר שנקלט.
4. חיפוש על פני הרשימה מי המספר האמצעי ביותר שנקלט.
5. הדפסת המספר הגדול ביותר שנקלט, הקטן ביותר והאמצעי.
6. שחרור הרשימה המקושרת.

תרגילי כיתה - 8

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:

1. קליטת מספרים שלמים, לרשימה מקושרת רגילה.
2. הדפסת הרשימה המקושרת שנקלטה.
3. חיפוש על פני הרשימה האם יש מספרים המופיעים יותר מפעם אחת ברשימה המקושרת ומחיקתם מהרשימה.
4. הדפסת הרשימה המקושרת החדשה ללא המספרים שהופיעו בה יותר מפעם אחת.
5. שחרור הרשימה המקושרת.

תרגילי כיתה - 9

- כתוב תוכנית המשתמשת בפונקציות שנלמדו ואחרות ומבצעת באמצעותן את המשימות הבאות:

1. קליטת מספרים שלמים, לרשימה מקושרת רגילה.
2. הדפסת כל איברי הרשימה המקושרת שנקלטה בסדר הפוך לקליטה.
3. הדפסת חלק מאיברי הרשימה המקושרת שנקלטה בסדר הפוך לפי בקשת המשתמש.
4. שחרור הרשימה המקושרת.

תרגילי כיתה - 10

- כתוב תוכנית המדגימה שימוש ברשימה מקושרת סיבובית המאפשרת ביצוע המשימות הבאות ע"י שימוש בפונקציות:
  1. יצירת מבנה של איבר ברשימה מקושרת סיבובית.
  2. קליטת איברים לתחילת הרשימה המקושרת.
  3. מחיקת איברים לסוף הרשימה המקושרת.
  4. הדפסת הרשימה המקושרת הפוכה לסדר הקליטה.
  5. חיפוש איבר ברשימה המקושרת הסיבובית.
  6. מיון הרשימה המקושרת הסיבובית.
  7. עדכון הרשימה המקושרת הסיבובית.
  8. שחרור הרשימה המקושרת.