

היום

- מהי רשימה
- חלופות מימוש: יתרונות וחסרונות
- רשימה מקושרת
- מימוש רשימה מקושרת

2

קורס תכנות

שיעור 13: רשימות מקושרות

1

מעריך הוא רשימה?

- לא! מעריך הוא מימוש אפשרי אחד של רשימה

יתרונות:

- גישה נוחה לערך במקום ה-i
- הוספה/מחיקה של ערך בסוף הרשימה

חסרונות:

- גודל קבוע
- הוספה/מחיקה של ערך שלא בסוף הרשימה
- ייתכן בזבוז זיכרון

4

מהי רשימה

- רשימה היא אוסף סדור של ערכים

פעולות:

- הוספה של ערך
- מחיקה של ערך
- גישה לערך במקום ה-i
- ...

3

רשימה מקושרת

- מימוש אפשרי נוסף

יתרונות

- צריכת זיכרון פרופורציונאלית למספר האברים ברשימה
- הוספה/מחיקה נוחה של אברים משני קצוות הרשימה
- הוספה/מחיקה נוחה לאחר אבר ספציפי ברשימה

חסרונות

- אין גישה ישירה לאבר ה-i

6

מעריך דינאמי

- גם מעריך דינאמי הוא מימוש אפשרי של רשימה

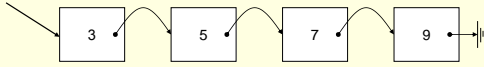
- פותר את בעיית הגודל הקבוע

- הוספת/מחיקת ערך עדיין לא יעיל. עדיין ייתכן בזבוז זיכרון

5

אילוסטרציה

רשימה מקושרת



מערך



7

רשימות מקושרות ב-C

- חוליה בשרשרת תצביע לחוליה הבאה
- שימוש במצביעים
- חוליה בשרשרת תכיל מידע ומצביע על החוליה הבאה
- שימוש במבנים
- נשתמש בזיכרון בהתאם לצורך
- שימוש בהקצאת זיכרון דינאמית

8

דוגמה: רשימה של מספרים שלמים

- כל חוליה בשרשרת תמומש בעזרת מבנה המכיל
 - את המידע (מספר שלם)
 - מצביע לחוליה הבאה בשרשרת
- נשתמש במבנה לצורך הגדרת חוליה ברשימה

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

9

החזקת רשימה

- נחזיק רשימה על ידי שמירת מצביע לחוליה הראשונה

```
int main()
{
    node_t *head = ...;
    ...
}
```

מצביע ל"ראש" הרשימה (המבנה הראשון)

10

פעולות על רשימות

- מעבר על הרשימה
- מספר איברים, חיפוש, ...
- הוספת איבר
- בתחילת הרשימה, בסופה, במקום כלשהו
- מחיקת איבר
- מחיקת רשימה שלמה

11

רשימה של מספרים

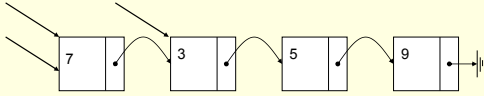
- נקלוט מהמשתמש רשימה של מספרים אי-שליליים
- נאחסן את הערכים ברשימה מקושרת
- בכל שלב נקלוט ערך מהמשתמש ונוסיף אבר לרשימה המכיל ערך זה
- בתחילה הרשימה ריקה

12

הוספת איבר בהתחלה*

1. ניצור איבר חדש
2. נדאג לכך ש

- האיבר החדש יצביע לראש הרשימה המקורית
- ראש הרשימה יהיה האיבר החדש



* הוספה באמצע ובסוף בהמשך

13

הוספת איבר לרשימה

```
int main()
{
    int value = 0;
    node_t *list = NULL, *new_node = NULL;
    printf("enter values (negative to stop)\n");
    scanf("%d", &value);
    while (value >= 0) {
        new_node = (node_t*) malloc(sizeof(node_t));
        if (new_node == NULL) {
            printf("Fatal error: memory allocation failed!\n");
            return EXIT_FAILURE;
        }
        new_node->data = value;
        new_node->next = list;
        list = new_node;
        scanf("%d", &value);
    }
    /* print the number of input values */
    /* free list */
    return EXIT_SUCCESS;
}
```

יצירת אבר חדש

החוליה החדשה תצביע לראש הרשימה

ראש הרשימה הוא החוליה החדשה

14

נעשה סדר בעזרת פונקציות

```
node_t* create_node(int data)
{
    node_t *new_node = (node_t*) malloc(sizeof(node_t));
    if (new_node != NULL) {
        new_node->data = data;
        new_node->next = NULL;
    }
    return new_node;
}

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;
}
```

ראש הרשימה

הערך שנרצה להוסיף

ניצור חוליה חדשה

נצביע לראש הנוכחי

הראש החדש הוא הערך המוחזר

15

הוספת איבר לרשימה

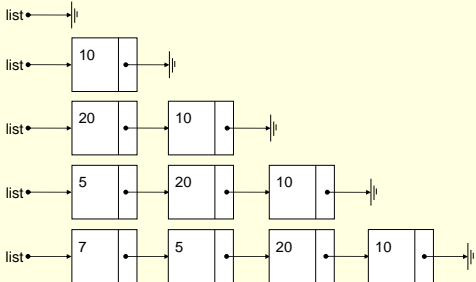
```
int main()
{
    int value = 0;
    node_t *list = NULL;
    printf("enter values (negative to stop)\n");
    scanf("%d", &value);
    while (value >= 0) {
        if ((list = add_first(list, value)) == NULL) {
            printf("Fatal error: memory allocation failed!\n");
            return EXIT_FAILURE;
        }
        scanf("%d", &value);
    }
    /* print the number of input values */
    /* free list */
    return EXIT_SUCCESS;
}
```

שימו לב: הערכים יישמרו ברשימה בסדר הפוך לסדר הכנסתם

16

שימוש ב add_first ליצירת רשימה

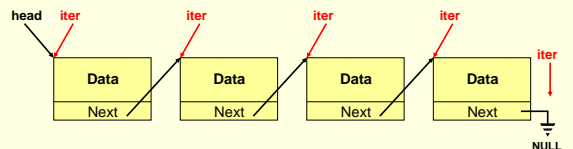
- עבור הקלט (משמאל לימין): 10 20 5 7



17

מעבר על רשימה

1. מתחילים בהתחלה (head)
2. נתקדם לאיבר הבא (iter->next)
3. עד שנגיע לסוף (iter == NULL)



18

חישוב אורך רשימה (מספר האיברים)

- נספור את מספר האיברים ברשימה

```
int length(const node_t *head)
{
    int count = 0;
    node_t *iter;

    for (iter = head; iter != NULL; iter = iter->next) {
        count++;
    }
    return count;
}
```

- רקורסיבי

```
int length(const node_t *head)
{
    if (head == NULL)
        return 0;
    return 1 + length(head->next);
}
```

19

מעבר על רשימה - תבנית

- מצביע לראש הרשימה
- כל זמן שלא הגענו לסופה
- בצע פעולה בעזרת האיבר הנוכחי
- התקדם לאיבר הבא

```
for (iter = head; iter != NULL; iter = iter->next) {
    // do something
}
```

20

חישוב אורך רשימה (מספר האיברים)

- נשתמש בפונקציה length

```
int main()
{
    int len;
    node_t *list = create_list();

    ...

    len = length(list);

    ...

    return 0;
}
```

21

הוספת איבר לרשימה

```
int main()
{
    int value = 0;
    node_t *list = NULL;

    printf("enter values (negative to stop)\n");
    scanf("%d", &value);
    while (value >= 0) {
        if ((list = add_first(list, value)) == NULL) {
            printf("Fatal error: memory allocation failed!\n");
            return EXIT_FAILURE;
        }
        scanf("%d", &value);
    }

    printf("read %d values", length(list));

    /* free list */
    return EXIT_SUCCESS;
}
```

22

דוגמה נוספת: חיפוש

- חיפוש ערך ברשימה

```
node_t* find(node_t *head, int val)
{
    while (head != NULL && head->data != val)
        head = head->next;
    return head;
}
```

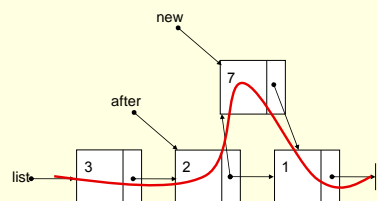
- רקורסיבי

```
node_t* find(node_t *head, int val)
{
    if (head == NULL)
        return NULL;
    return (head->data == val) ?
        head :
        find(head->next, val);
}
```

23

הוספת אברים שלא בהתחלה

- בהינתן החוליה שאחריה נרצה להוסיף, הוספת איבר היא קלה



24

מימוש הוספת איבר בסוף הרשימה

- שתי אפשרויות
 - הרשימה ריקה
 - הרשימה לא ריקה

- בכל מקרה צריך ליצור איבר חדש
- אם הרשימה ריקה, איבר זה יהיה הרשימה החדשה
- אחרת צריך להצביע על איבר זה מסוף הרשימה

25

מימוש הפונקציה add_last

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

    if (head == NULL)
        return new_node;

    while (tail->next != NULL)
        tail = tail->next;
    tail->next = new_node;

    return head;
}
```

הקצאת איבר חדש

הרשימה המקורית ריקה

נחץ לסוף הרשימה ונסוץ את האיבר החדש

ראש הרשימה לא השתנה

26

הוספת איבר לרשימה

```
int main()
{
    int value = 0;
    node_t *list = NULL;

    printf("enter values (negative to stop)\n");
    scanf("%d", &value);
    while (value >= 0) {
        if (list = add_last(list, value) == NULL) {
            printf("Fatal error: memory allocation failed!\n");
            return EXIT_FAILURE;
        }
        scanf("%d", &value);
    }

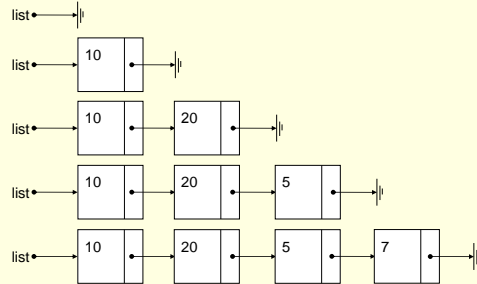
    printf("Read %d values\n", length(list));
    /* free list */
    return EXIT_SUCCESS;
}
```

שימו לב: הערכים יישמרו ברשימה לפי סדר הכנסתם

27

שימוש ב add_last ליצירת רשימה

- עבור הקלט (משמאל לימין): 10 20 5 7



28

רשימה ממוינת

- רשימה שבה האברים מסודרים לפי סדר כלשהו
 - מספרים – סדר עולה / יורד
 - סטודנטים – לפי שם (סדר לקסיקוגרפי), לפי מספר זהה
 - נקודות במרחב – לפי מרחק מראשית הצירים

שתי אפשרויות:

- נקלוט את כל הערכים ואז נמיין
- נשמור את הרשימה ממוינת על ידי הוספת ערך חדש במקום המתאים (אחרי האיבר שקטן ממנו ולפני זה שגדול ממנו)

29

הוספת איבר ברשימה ממוינת

```
node_t* add(node_t *head, int data) {
    node_t* iter, *prev = NULL;
    node_t* new_node = create_node(data);
    /* incomplete, must check for failure */
    if (head == NULL)
        return new_node;

    if (new_node->data < head->data) {
        new_node->next = head;
        return new_node;
    }

    iter = head;
    while (iter != NULL && new_node->data > iter->data) {
        prev = iter;
        iter = iter->next;
    }

    prev->next = new_node;
    new_node->next = iter;

    return head;
}
```

רשימה ריקה

מינימלי – ראש הרשימה

נמצא את המקום המתאים

הוספה במקום המתאים

30

הוספת איבר לרשימה

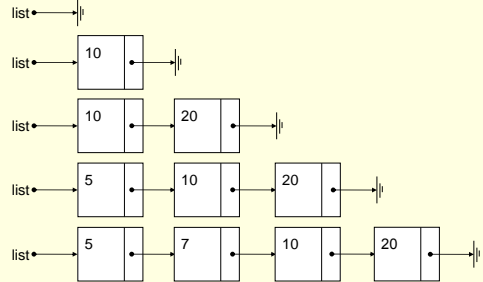
```
int main()
{
    int value = 0;
    node_t *list = NULL;

    printf("enter values (negative to stop)\n");
    scanf("%d", &value);
    while (value >= 0) {
        if ( (list = add(list, value)) == NULL) {
            printf("Fatal error: memory allocation failed!\n");
            return EXIT_FAILURE;
        }
        scanf("%d", &value);
    }
    printf("Read %d values\n", length(list));
    /* free list */
    return EXIT_SUCCESS;
}
```

31

שימוש ב add ליצירת רשימה

- עבור הקלט (משמאל לימין): 10 20 5 7



32

מחיקת איבר מרשימה

- הפונקציה **delete** מוחקת איבר ברשימה
 - מקבלת מצביע לראש הרשימה וערך למחיקה
 - מחזירה מצביע לראש הרשימה החדשה

אפשרויות:

- הרשימה ריקה
- הערך לא קיים ברשימה
- הערך נמצא באיבר הראשון
- הערך נמצא באיבר האחרון
- הערך נמצא בתוך הרשימה

33

מימוש מחיקת איבר מרשימה (1)

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

    if (head == NULL) // רשימה ריקה
        return head;

    if (head->data == value) // האיבר הראשון ברשימה
    {
        iter = head->next;
        free(head);
        return iter;
    }
    ...
}
```

34

מימוש מחיקת איבר מרשימה (המשך)

```
...
while (iter->next != NULL) // לא הגענו לסוף
{
    if (iter->data == value) // נמחק את האיבר
    {
        prev->next = iter->next;
        free(iter);
        break;
    }
    prev = iter;
    iter = iter->next;
}
return head;
```

35

שחרור רשימה מקושרת

- בסיום השימוש ברשימה נרצה לשחרר את הזיכרון של הרשימה
 - בשונה ממערך שהוקצה דינאמית, שחרור ראש הרשימה משחרר רק את האיבר הראשון ברשימה
 - יש לשחרר את איברי ברשימה אחד אחרי השני

נכתוב את הפונקציה **free_list**

- מקבלת מצביע לראש הרשימה
- עוברת ומשחררת את כל האיברים

36

שחרור רשימה מקושרת - מימוש

```
void free_list(node_t* head)
{
    node_t* temp;
    while (head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
}
```

כל עוד לא הגענו לסוף

נשמור מצביע לאיבר

נשחרר את האיבר ונתקדם הלאה

רקורסיבי

```
void free_list(node_t* head)
{
    if (head == NULL)
        return;
    free_list(head->next);
    free(head);
}
```

רשימה ריקה

שחרר את שאר הרשימה

שחרר את האיבר הנוכחי

37

גישה לאבר ה-i

- אין גישה ישירה
- נצטרך לספור אברים מההתחלה

```
/* assumes index >= 0 */
node_t* get(node_t* head, int index)
{
    while (index > 0 && head != NULL)
    {
        head = head->next;
        index--;
    }
    return head;
}
```

38

סיכום רשימות מקושרות

- נשתמש כשייתכנו מחיקות והוספות של נתונים
- מימוש – איברים המוקצים דינאמית
 - כל איבר כולל מידע ומצביע לאיבר הבא
- ההתקדמות לאורך הרשימה בעזרת המצביעים
- דוגמאות
 - הוספה, מחיקה, שיחזור, איטרציה

39