מבוא למדעי המחשב – כתה יא-1 תאריך: 12.12.2025

דף עבודה – מספר #3 בנושא רשימות מקושרות (Linked Lists)

I. מבוא לרשימות מקושרות

כאשר אנו מתכנתים, אנו רוצים **לאחסן נתונים** בצורה המאפשרת לנו לבצע עליהם פעולות כגון **גישה נוחה, הוספה, מחיקה ושינוי.**

זהו הרעיון המרכזי מאחורי כל *מבנה נתונים* - לבחור דרך לארגן מידע כך שיהיה לנו קל לעבוד איתו.

1. ארגון נתונים באמצעות מערך

האפשרות הראשונה והמוכרת היא שימוש במערך.

- אנו מגדירים גודל מראש.
- מקצים זיכרון לכל האיברים (new).
- ולאחר מכן מאכלסים כל תא במידע המתאים.

היתרון: גישה אקראית ומהירה (Random Access).
החיסרון: גודל המערך חייב להיות ידוע מראש, והוספה/מחיקה "באמצע" מסורבלות ויקרות (מה זה אומר "יקרות"?).

2. רשומות מקושרות – הגישה הראשונית

בשיעורים הקודמים הכרנו דרך אחרת לארגן מידע:
רשימה מקושרת (Linked List).

העיקרון היה פשוט:

- יש לנו משתנה שמצביע לראש הרשימה (head).
- כל איבר (אובייקט) ברשימה מכיל את המידע הרלבנטי לגבי אותו אובייקט – לדוגמא, ערך נתון כגון מספר או תו, אם שם תלמיד והציון שלו.
- בנוסף: כל איבר מכיל הפניה לאיבר שאחריו – next.
- כך יצרנו מחלקות כמו CharNode, CellNode ו- IntNode, שבהן כל אובייקט הכיל גם מידע וגם את הקישור לרכיב הבא.

זוהי דרך טובה להתחיל להבין רשימות מקושרות — אבל יש בה בעיה מהותית.

3. מה הבעיה בגישה הזאת?

כאשר אותו אובייקט מכיל ***גם*** מידע ***וגם*** את הקישור (next), יש בלבול בין שני סוגים שונים של "אחריות":

- **המידע –** במקרה שלנו value, special.
- **מבנה הרשימה** – כלומר מי הבא בתור.

אם נרצה להשתמש באותו מידע, באותו סוג של אובייקט – למשל במערך — שדה `next` אינו רלוונטי.
אם נרצה לשנות את מבנה הרשימה — לדוגמא, נרצה לבנות דבר שנקרא רשימה דו-כיוונית, שבה כל איבר יודע להגיד מי אחרי אבל גם מי לפניי (הקודם לי) – נצטרך לשנות את הגדרת המחלקה.

במילים אחרות:
ערבבנו בין שני סוגי אחריות – (א) המידע עצמו (ב) איך הוא מאורגן. זה פוגע בניקיון ובגמישות של הקוד.

זו בדיוק הסיבה שאנו מתקדמים היום צעד אחד קדימה, לתפיסה נכונה ומקצועית יותר.

שימו לב למילה "**אחריות**" – עקרון חשוב בתכנה – למחלקה מסוימת, או לפונקציה מסוימת – נרצה שתהיה אחריות אחת. נרחיב על זה בהמשך השנה, בפרט כשנלמד תכנות מונחה עצמים.

4. הגישה החדשה: הפרדת אחריויות – המחלקות IntData + IntNode

החל מהשיעור הזה אנו מפרידים בין:

- המידע עצמו – במחלקה **IntData**
  - מכילה רק את מה שקשור לתוכן הנתון:
    - `int value`
    - `bool special`
    - פעולות כמו `Print()` או `Mul(int n)`
- **המחלקה** `IntNode` – מכילה את **מבנה הרשימה**
  - מכיל:
    - `IntData data` – המידע
    - `IntNode next` – הקישור לאיבר הבא

הפרדה זו מאפשרת לנו:

- להשתמש באובייקטים מסוג - `IntData` גם ברשימה וגם במערך.
- לנהל את מבנה הרשימה בלי קשר למידע.
- להתקדם בשיעורים הבאים לרשימות גנריות (`Node<T>`) – שזה מה שאנחנו צריכים לדעת לבגרות, - וגם נוכל להבין את ההיגיון ואת היתרון – למה זה בא לעולם.

5. מה נלמד ונתרגל כעת

בתרגול הבא נעבוד עם המחלקות החדשות ונכתוב פעולות קלאסיות על רשימות:

- יצירת רשימה
- הוספה לראש/זנב
- הדפסה
- חיפוש
- מחיקה
- סימון איברים מיוחדים
- פעולות על המידע (כמו `Mul(n)`)

ובתרגול או בשיעור שיעור הבא נלמד איך לתרגם בין:

- **מערך ← רשימה**
- **רשימה ← מערך**

וזה ייצור גשר חשוב לקראת נושא נוסף: רשימות גנריות ומבני נתונים מורכבים יותר.

II. עבודה עם חוליה מקשרת, הפרדה בין החוליה לבין ה data

עד עכשיו עבדנו עם רשימות שבהן כל חוליה הכילה גם את המידע וגם את ההפניה next.
כאמור, החל מהתרגיל הזה אנו מפרידים בין:

- **IntData –** שמכילה את המידע ואת הפעולות
- **IntNode –** מחלקה שהיא החוליה המקשרת (data + next)

זוהי חלוקה מקצועית יותר, שתלווה אותנו בהמשך השנה ותשמש בסיס להבנה של מבני נתונים גנריים (Node<T>) כפי שנדרש בבגרות.

הקבצים שאתם עובדים איתם

הקבצים הדרושים נמצאים ב GDRIVE. תוכלו להגיע אליהם בקלות דרך הקישור הבא:
https://tzvimelamed.com/lab .

בתיקייה שאתם מקבלים על ה GDRIVE יש את הקבצים:

- **Program.cs** – מכיל את ה- Main ואת התפריט
- **Utils.cs** – הדפסת צבעים וצלילים (ללא צורך לשנות)
- **ListTasks_Student.cs** – כאן נמצאות המשימות שאתם משלימים

תחילה העבודה

1. *צרו פרוייקט חדש ב Visual Studio וצרפו אליו את הקבצים הנ"ל.*
2. *בנו שתי מחלקות (כל אחת בקובץ משלה לפי הנחיות הבאות.*
3. *כעת נסו לקמפל את התכנית. אתם אמורים לקבל שגיאת קומפילציה – מיד נתעכב על זה.*
4. *לאחר מכן – בצעו את המשימות השונות בתרגיל הזה.*

יצירת המחלקות החדשות:

משימה 1 – יצירת המחלקה IntData

```
public class IntData
{
    public int value;
    public bool special;

    public IntData(int value)
    {
        this.value = value;
        special = false;
    }

    public void Print()
    {
        Utils.PrintIntAndBeep(value, special);
    }

    public void Mul(int n)
    {
        value *= n;
    }
}
```

משימה 2 – יצירת המחלקה IntNode

```
public class IntNode
{
    public IntData data;
    public IntNode next;

    public IntNode(IntData data, IntNode next = null)
    {
        this.data = data;
        this.next = next;
    }
}
```

משימה 3 – תיקון שגיאת הקומפילציה בפונקציה PrintList

שימו לב:

ב־ Utils.csקיימת פונקציית PrintList *שאינה מתאימה למבנה החדש* ולכן התוכנית **לא תתקמפל.** עליכם לתקן את הפונקציה כך שתעבור קומפילציה. עיון בקוד של שתי המחלקות IntData, IntNode יבהיר מה צריך לתקן.

זו משימה חשובה — להבין איפה נעשה שינוי במבנה הנתונים, ואיך יש להתאים את הקוד.

בצעו את המשימות הבאות:

(הוראות לכך מופיעות כשאתם מריצים את התכנית, כפי שהיה בתרגילים הקודמים)

1. **יצירת רשימה ידנית(Fixed list)**

צרו 3–4 חוליות IntNode המחוברות ביניהן ידנית.

2. **יצירת רשימה אקראית של 4 איברים**

בכל פעם צרו ערך חדש באמצעות הקריאה לפונקציה Utils.GetRandom

()Utils.GetRandom

3. **הוספת איבר אקראי לראש הרשימה**

יצירת IntNode חדש + חיבור כ־ headהחדש.

4. **הוספת איבר אקראי לסוף הרשימה**

חצו את הרשימה עד הסוף והוסיפו חוליה חדשה.

5. **ספירת האיברים ברשימה**

חצו את הרשימה וחזרו את מספר החוליות.

6. **העבירו** את האיבר הכי קטן לסוף הרשימה

במשימה זו עליכם לבצע מעבר בודד של "מיון בועות", בדגש על מציאת הערך הקטן ביותר ודחיפתו לסוף הרשימה.

הנחיות:

a) התחילו מה־head.
b) בכל צעד בדקו את הצמד (current, next).
c) אם next.value קטן יותר מ־current.value, החליפו ביניהם (הזזה של חוליה).
d) המשיכו עד סוף הרשימה.
e) בסוף המעבר, האיבר הכי קטן "יזחול" ויגיע לסוף.

<u>הנחיות חשובות נוספות:</u>

א. יש להשתמש בציורים בשביל להמחיש את הרשימה והפעולה שמצבעים עליה.
ב. תמיד לקחת בחשבון: רשימה ריקה (כלומר, למשל head == null) היא רשימה חוקית! להתייחס למקרה הזה.
ג. אסור לגשת לשדה או לפונקציה של אובייקט, אם האובייקט (כלומר: המשתנה שמצביע עליו) הוא null.