

תכנות מונחה עצמים בשפת C++

אוהד ברזילי
אוניברסיטת תל אביב

מנהלות

- איפה? מתי?
 - שרייבר 007
 - ב' 15:00 – 18:00
 - ה' 17:00 – 20:00
- מבנה השיעור:
 - שעה ורבע שיעור
 - 20 דקות הפסקה
 - שעה ורבע שיעור
- בימי שני יוקדש השיעור השני לתרגול
- דרישות קדם:
 - תוכנה 1 בשפת C

מנהלות

ספרות: □

- *Object-Oriented Software Construction*, Second Edition
BERTRAND MEYER, ISE Inc., Santa Barbara Prentice Hall
- *The C++ Programming Language*, 3rd and Special Edition
Bjarne Stroustrup, Addison-Wesley



מונחה עצמים בשפת C++
אוניברסיטת תל אביב



מנהלות

תקשורת: □

- אתר קורס: www.cs.tau.ac.il/~ohadbr/oopcpp05c
- דוא"ל (באנגלית בלבד): ohadbr@tau.ac.il

מבחן סיום □

תרגילים תאורטיים ומעשיים □

על מודולריזציה ושימוש חוזר

ההרצאה מבוססת על מצגת של פרופ' עמירם יהודאי
ע"פ הספר *Object-Oriented Software Construction*,
.2nd edition, by Bertrand Meyer (Prentice Hall)
כל הזכויות שמורות למחברים

מבוא ומוטיבציה

□ גישת תכנות מונחה עצמים הוצגה כפתרון לבעיות שונות:

■ סימולציה (בשנות ה-60 ה-simula67)

■ ממשק משתמש (בשנות ה-70 ה-smaltalk80)

■ הנדסת תוכנה

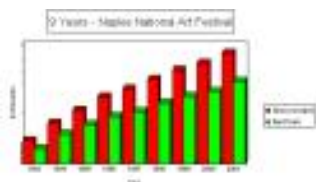
□ אותה נציג בקורס זה ע"פ סיפרו של B. Meyer,

Object Oriented Software Construction

לפי מה נמדדת איכות תוכנה?

□ מדדים חיצוניים

■ בעיני המשתמש



□ מדדים פנימיים

■ בעיני הגוף המפתח

תכנת מונחה עצמים בשפת C++
אוניברסיטת תל אביב

7

לפי מה נמדדת איכות תוכנה?

□ נכונות (correctness)

■ התוכנה ממלאת אחר הדרישות

□ יציבות (Robustness)

■ תפקוד בתנאי קצה

□ אמינות (Reliability) = נכונות + יציבות



תכנת מונחה עצמים בשפת C++
אוניברסיטת תל אביב

8

לפי מה נמדדת איכות תוכנה?

- יכולת הרחבה (Extendibility) – הגמישות לשינויים בתוכנה עקב שינויים במפרט (תכונת הרציפות)
- שימוש חוזר (Reusability)
- מודולריות (רכיביות, Modularity) – הפרדה לוגית של התוכנה למרכיבים
- מודולריות = שימוש חוזר + יכולת הרחבה

לפי מה נמדדת איכות תוכנה?

- תאימות (Compatibility) – הקלות שבה ניתן לשלבה במערכת גדולה יותר
- יעילות (Efficiency) – כתלות בחומרה
- ניידות (Portability) – כתלות בחומרה ובתוכנה
- קלות שימוש (Ease of use)
- תפקודיות (Functionality) – אפשרויות השימוש שהתוכנה מציעה
- עמידה בלוח זמנים, אימות, נוחות התיקון, חסכוניות, ועוד...

נושאים נלווים

□ תיעוד

- פנימי
- חיצוני
- ממשק מודול

□ חלק מהמדדים לאיכות תוכנה סותרים

- לדוגמא: יעילות לעומת ניידות

תחזוקת תוכנה

□ 70% מעלות הפיתוח הכוללת

- שינוי בדרישות המשתמש 41.8%
- שינוי במבני הנתונים 17.4%
- תיקוני חרום 12.4%
- תיקוני שגרה 9.0%
- שינויים בחומרה 6.2%
- תיעוד 5.5%
- שיפורי יעילות 4.0%



קריטריונים לתוכנה מודולרית

- **עיצוב (תכן) מודולרי**
 - **פירוק מודולרי (Modular decomposability)**
 - פרקי את הבעיה ופתרי כל תת בעיה בנפרד
 - **הרכבה מודולרית (Modular composability)**
 - הרכיבי רכיבי תוכנה ליצירת מערכת חדשה
- **הבנה מודולרית – האם ניתן להבין כל רכיב בנפרד**
- **רציפות מודולרית – שינוי קטן במפרט משפיע על מספר מצומצם של מודולים**
- **הגנה מודולרית – בעיות ריצה נשארות תחומות במספר מצומצם של מודולים**

כללים ועקרונות לתוכנה מודולרית

- מיפוי ישיר בין עולם הבעיה ועולם הפתרון
- ממשקים: מועטים, קטנים, מפורשים
- הסתרת מידע (information hiding)
- תמיכה תחבירית בשפה

כללים ועקרונות לתוכנה מודולרית

- תיעוד עצמי
- רכיב סגור-פתוח:
 - פתוח להרחבות
 - סגור לשימושים המוגדרים בלבד
- בחירה יחידה – כאשר מערכת צריכה לתמוך בכמה אלטרנטיבות, רכיב אחד בלבד מחזיק את התמונה המלאה

שימוש חוזר

- מה ממחזרים?
 - כח אדם, מפרט ועיצוב, תבניות עיצוב (design patterns), קוד.
- לא רק בעיה טכנית
 - מנטלית, שיווקית...
- אין שתי בעיות זהות
- מאידך יש המון חזרות ('קלישאות') בעולם התוכנה



דוגמא לשימוש חוזר חיפוש בטבלה

□ ללא תלות במימוש ואפילו לא בייצוג:

- טבלה \ מערך \ עץ
- ממוין \ ללא ממוין
- טבלת גיבוב (hash table)

□ יש 'רק' להחליף את ה `UPPERCASE` בדוגמא
שבשקף הבא

דוגמא לשימוש חוזר חיפוש בטבלה

```
/*  
 * Is there an occurrence of x in t?  
 */  
bool has (TABLE t, ELEMENT x)  
{  
    POSITION pos;  
  
    pos = INITIAL_POSITION (x, t);  
    while (!EXHAUSTED (pos, t) && !FOUND (pos, x, t))  
    {  
        pos = NEXT (pos, x, t);  
    }  
    return !EXHAUSTED (pos, t);  
}
```

מה נדרש מרכיב כדי שניתן יהיה למחזר אותו?

- גמישות בטיפוסים
 - ELEMENT
- ריכוז שגרות
 - has צריך להיות חלק מספרייה המכילה גם את insert, remove וכיו"ב
- גמישות במימוש
 - מבני נתונים
 - אלגוריתם

מה נדרש מרכיב כדי שניתן יהיה למחזר אותו?

- חוסר תלות בייצוג
 - המשתמש אמור להיות אדיש למבנה הנתונים הספציפי לדוגמא בקריאה: $present=has(t,x)$
 - הייצוג עשוי להשתנות במהלך הפיתוח
 - הייצוג עשוי להשתנות במהלך הריצה!
 - מי מחליט על הייצוג?
- ביטוי של ההתנהגות הכללית
 - איך נמנע מחזרה על קוד?
 - דוגמא: has עבור טבלה סדרתית (מערך, רשימה, קובץ)

חיפוש בטבלה סדרתית

```
/*  
 * Is there an occurrence of x in t?  
 */  
bool has (SEQUENTIAL_TABLE t , ELEMENT x)  
{  
    start  
  
    while (!after && !found (x))  
    {  
        forth  
    }  
    return !after;  
}
```

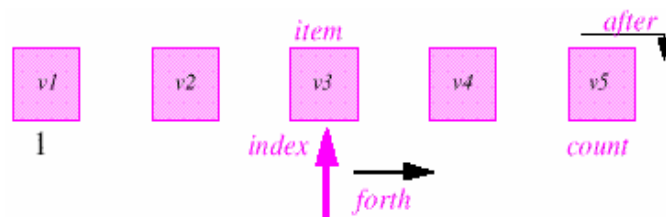


תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

21

פעולות בטבלה סדרתית

- start*
- forth*
- after*
- found(x)*



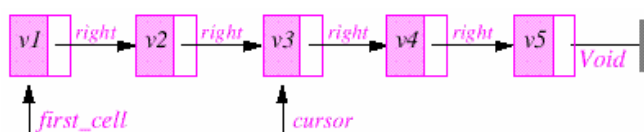
אוניברסיטת תל אביב

22

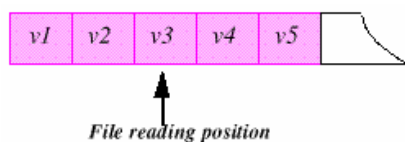
מימושים של טבלה סדרתית



מערך



רשימה
מקושרת



קובץ

תכנת מונחה ענן
אוניברסיטת תל אביב

23

טבלאות סדרתיות - סיכום

	<i>start</i>	<i>forth</i>	<i>after</i>	<i>found(x)</i>
Array	$i = 0$	$i = i + 1$	$i > \text{count}$	$t[i] == x$
Linked List	$c = \text{first_cell}$	$c = c.\text{right}$	$c = \text{NULL}$	$c.\text{item} == x$
File	<i>rewind</i>	<i>read</i>	<i>end_of_file</i>	$f \uparrow = x$

תכנת מונחה עצמים בשפת C++
אוניברסיטת תל אביב

24

מרכיבי שפה המאפשרים שימוש חוזר

- שגרות (Routines)
- חבילות (Packages)
- העמסה (Overloading)
- הכללה (Genericity)



מרכיבי שפה המאפשרים שימוש חוזר

- שגרות (Routines)
 - עובדות היטב כאשר ניתן לאפיין את הבעיה ע"י מספר מצומצם של פרמטרים
 - שגרות פותרות בעיות נפרדות זו מזו – שימוש חוזר בתכן אבל לא בקוד
 - שגרות אינן מתאימות לטיפול במגוון רחב של מבני נתונים

מרכיבי שפה המאפשרים שימוש חוזר

□ חבילות (Packages)

- בשפת C++ נקראות מרחב שמות (namespace)
- יחידה תחבירית בשפה הכוללת הגדרות של שגרות ונתונים בעלי מאפיינים משותפים
- תמיכה בהסתרת מידע ובריכוז שגרות
- חוסר תמיכה בייצוג משתנה
- על namespaces בשפת C++ - בתרגול



מרכיבי שפה המאפשרים שימוש חוזר

□ העמסה תחבירית (syntactic overloading)

- רב צורתיות זוטא (ad-hoc polymorphism)
- `int square (int x) {...}`
- `float square (float x) {...}`
- `double square (double x) {...}`
- `complex square (complex x) {...}`
- הלקוח רושם את אותו הקוד `square(x)` ובפועל מתבצעות פונקציות שונות בעלות מאפיינים משותפים
- עוד על העמסה תחבירית בתרגול

מרכיבי שפה המאפשרים שימוש חוזר

□ העמסת משמעות (semantic overloading)

- רב צורתיות אמיתית
- קישוריות דינאמית
- תומכת בעצמאות הייצוג

□ על פולימורפיזם נדבר בהמשך הקורס



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

29

מרכיבי שפה המאפשרים שימוש חוזר

□ הכללה (Genericity)

□ נתמכת ב C++ ע"י תבניות (templates)

□ הכללה מאפשרת לספק להביע קונספט כללי ולממש אותו פעם אחת עבור מגוון ישויות

□ במקום לתחזק 3 מודולים:

- INTEGER_TABLE_HANDLING
- ELECTRON_TABLE_HANDLING
- ACCOUNT_TABLE_HANDLING

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

30

הכללה (Genericity)

□ נתחזק מודול יחיד:

■ TABLE_HANDLING <T>

□ וממנו יגזור הלקוח:

■ TABLE_HANDLING <int>

■ TABLE_HANDLING <ELECTRON>

■ TABLE_HANDLING <ACCOUNT>

הכללה (Genericity)

□ שגרות מוכללות (Generic Functions)

□ במקום שהלקוח יממש בנפרד את:

■ `int square (int x) {...}`

■ `float square (float x) {...}`

■ `double square (double x) {...}`

■ `complex square (complex x) {...}`

□ הלקוח יממש רק את:

■ `T square (T x) {...}`

הפסקה
