



עצמים

תכנות מונחה עצמים בשפת C++
אוהד ברזילי
אוניברסיטת תל אביב



עצמים

המצגת מכילה קטעים מתוך מצגת של פרופ' עמירם יהודאי ע"פ הספר:
Object-Oriented Software Construction, 2nd edition,
by Bertrand Meyer (Prentice Hall) .

וכן מתוך הספר:

The C++ Programming Language 3rd Edition
by Bjarne Stroustrup

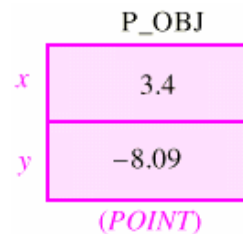
כל הזכויות שמורות למחברים

עצמים

- עצם הוא מופע זמן-ריצה (instance) של מחלקה
- עצם הוא מופע ישיר של בדיוק מחלקה אחת, המחלקה היוצרת, ואולי מופע של מחלקות נוספות כתוצאה מירושה
- מבנה הנתונים המייצג עצם מכיל שדות נתונים בלבד ללא פונקציות (מתודות)

POINT Class

```
class POINT
{
    float x, y;
    // Routines...
};
```



Simple Book

```
class BOOK1
{
    string title;
    int date, page_count;
};
```

<i>title</i>	"The Red and the Black"
<i>date</i>	1830
<i>page_count</i>	341

(BOOK1)

שפת C++

Writer Class

```
class WRITER
{
    string name, real_name;
    int birth_year, death_year;
};
```

<i>name</i>	"Stendhal"
<i>real_name</i>	"Henri Beyle"
<i>birth_year</i>	1783
<i>death_year</i>	1842

(WRITER)

שפת C++
כתיבה



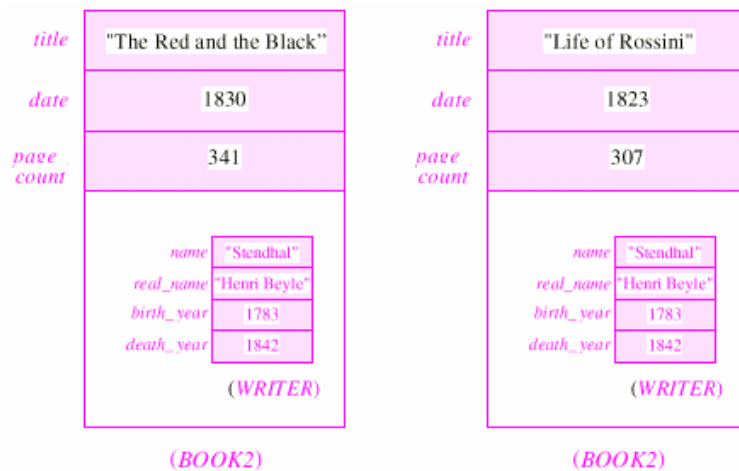
עצמים

- נסווג את השדות לסוגים לפי טיפוסיהם:
 - טיפוסים יסודיים (פרימיטיבים): חלק משפת התכנות. כגון: int, char, float, bool (אבל לא string)
 - טיפוסים מורכבים (user defined types)
- שדה מורכב: עצם מוכל או עצם מוצבע?

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

7

תת עצמים



8

עצמים, מצביעים והתייחסויות

```
Point p;
```

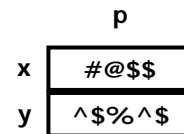
```
Point *pp1, *pp2;
```

```
pp1 = &p;
```

```
pp2 = pp1;
```

```
Point &rp1 = p;
```

```
Point &rp2;
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

9

עצמים, מצביעים והתייחסויות

```
Point p;
```

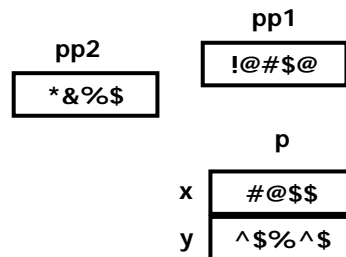
```
Point *pp1, *pp2;
```

```
pp1 = &p;
```

```
pp2 = pp1;
```

```
Point &rp1 = p;
```

```
Point &rp2;
```

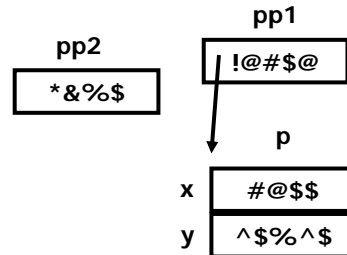


תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

10

עצמים, מצביעים והתייחסויות

```
Point p;  
Point *pp1, *pp2;  
pp1 = &p;  
pp2 = pp1;  
Point &rp1 = p;  
Point &rp2;
```

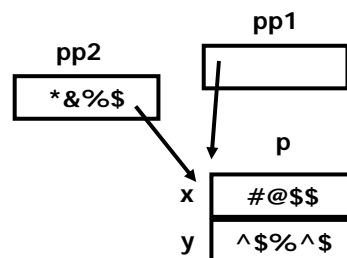


תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

11

עצמים, מצביעים והתייחסויות

```
Point p;  
Point *pp1, *pp2;  
pp1 = &p;  
pp2 = pp1;  
Point &rp1 = p;  
Point &rp2;
```

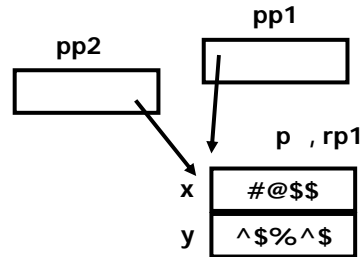


תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

12

עצמים, מצביעים והתייחסויות

```
Point p;  
Point *pp1, *pp2;  
pp1 = &p;  
pp2 = pp1;  
Point &rp1 = p;  
Point &rp2;
```

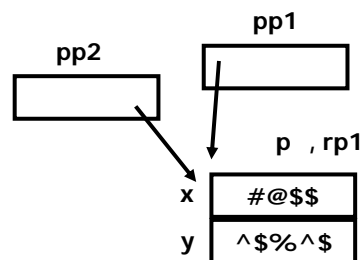


תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

13

עצמים, מצביעים והתייחסויות

```
Point p;  
Point *pp1, *pp2;  
pp1 = &p;  
pp2 = pp1;  
Point &rp1 = p;  
Point &rp2; // ERROR. Unattached reference
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

14



עצמים, מצביעים והתייחסויות

- נשים לב להבדלים בין מצביע (pointer) להפנייה (reference) בשפת ++C:
 - הפנייה הינה שם (לפעמים נוסף) לאובייקט קיים. לא ניתן לשנות את הקישור בין הפנייה ובין העצם שעליו היא מצביעה
 - מצביע הוא ישות עצמאית, עצם בפני עצמו, שעשויה להכיל כתובת של אובייקט קיים. ניתן לעדכן את ערכו של מצביע ובכך להחליף את העצם המוצבע.



עצמים, מצביעים והתייחסויות

- בעת יצירת עצם מקומי (על המחסנית) אנו בעצם יוצרים גם את העצם וגם את ההפניה עבורו, השם שדרכו ניתן יהיה לגשת אליו
- בעת יצירת הפנייה בלבד אנו נותנים שם נוסף לעצם קיים
- לא נתבלבל עם העברת נתונים לפונקציה by reference. העתקה או אי העתקת העצם לא תלויה בעצם עצמו אלא בהגדרת שיטת ההעברה בפונקציה

שפת Java

- בשפת Java הפנייה היא יישות נפרדת מהעצם המוצבע (ממש כמו מצביעים ב C++)
- בשפת Java לא ניתן ליצור עצמים מקומיים כלל. עצמים נוצרים רק ע"י הקצאת זיכרון מפורשת בפקודת new
- ניתן לראות זאת כאילו כל המשתנים הלא פרימיטיביים ב java הם מצביעים עם גישה לשדותיהם ע"י נקודה (.) במקום ->

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

17

עצם מוכל או מוצבע?

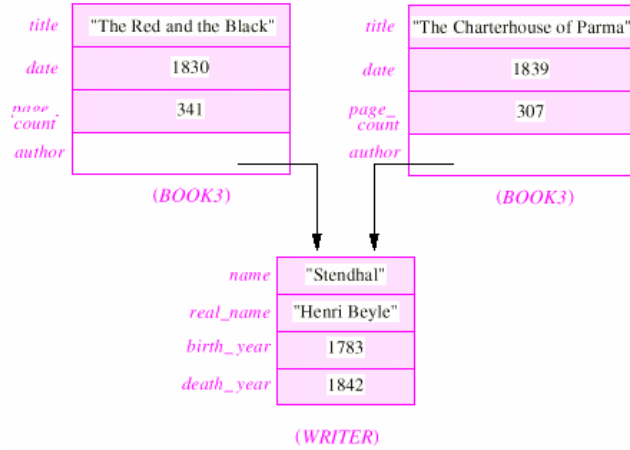
- תת עצמים תופסים יותר זכרון
- עצמים מוצבעים קל יותר לשתף בין מספר ישויות
- ב C++ לא כל המצביעים מאותחלים אוטומטית (ל- null) כך שקיימות 3 אפשרויות:
 - המצביע מתייחס לעצם אחד מסוים
 - מצביע לכתובת 'זבל'
 - המצביע הוא null כלומר לא משויך לעצם מסוים



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

18

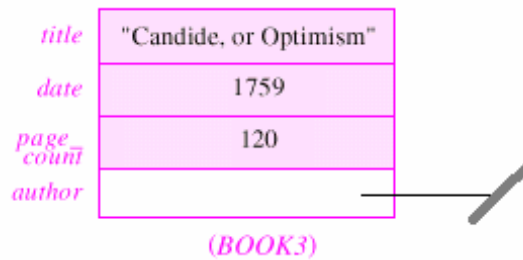
מוצבע



אוניברסיטת תל אביב

19

לא משויך לאף עצם



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

20

זהות של עצמים

- שני עצמים נפרדים עשויים להיות זהים
- שוויון שדות אינו תנאי מספיק לזהותם של שני עצמים

```
class WRITER {
    string name;
    int birth_year, death_year;
};

class BOOK {
    string title;
    int date;
    WRITER *author;
};
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

21

מעגלים ביחס הלקוחות

- מחלקה עשויה להיות לקוחה של עצמה
- נבדיל בין מעגל של עצמים ומעגל של מחלקות

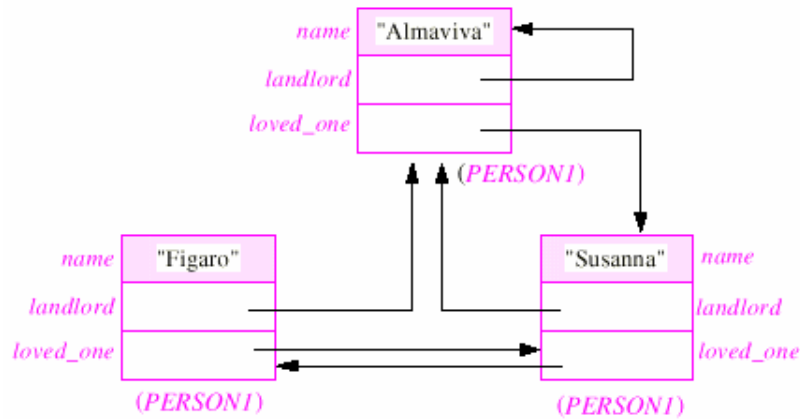
```
class PERSON1
{
    string name;
    PERSON1 *loved_one, *landlord;
};
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

22

מצביע לעצמו



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

23

פונקציה יוצרת

- יצירת ואתחול עצמים מפורשת
- כאשר בפונקציה כלשהי מגיעים להצהרה $T \text{ obj}$ קורים הדברים הבאים:
 1. מוקצה מקום על המחסנית עבור עצם מטיפוס T (עבור כל אחד משדותיו)
 2. שדותיו מאותחלים לפי כללי אתחול ברירת המחדל (בהמשך)
 3. השם obj משויך לעצם שנוצר

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

24



פונקציה יוצרת

- יצירת ואתחול עצמים מפורשת
- כאשר בפונקציה כלשהי מגיעים לשורה
 $T *x = new T$ קורים הדברים הבאים:
 1. מוקצה מקום עבור עצם מטיפוס T (עבור כל אחד משדותיו)
 2. x מקבל את כתובתו של העצם שהוקצה



מצביעים ומצבם

- מצביע יכול להיות קשור או לא קשור לעצם
- מצביע p יקשר לעצם ע"י:
 - $p = new SomeClass(...)$
 - או ע"י $p = p1$, כאשר p1 הוא מצביע קשור
- p יהפך ללא קשור ע"י:
 - $p = null$
 - או ע"י $p = p1$ כאשר p1 הוא null
- הפעלת מתודה דרך מצביע שאינו קשור תגרום לטעות זמן ריצה



פעולות על מצביעים

- השמה של מצביעים אינה מבצעת העתקה של האובייקט המוצבע והיא מותרת אם המצביע אינו מוגדר כ `const` (ראה שיעור 3)
- ההשוואה בין מצביעים `p1==p2` תחזיר `true` רק אם הם מצביעים על אותו עצם עצמו או ששניהם `null`
- ההשוואה בין עצמים `obj1==obj2` תעבוד רק אם הוגדר עבור המחלקה `T` של `obj1` אופרטור השוואה
- ההשמה `p=null` שימושית כדי לבדוק בהמשך האם `p!=null` (שכן למצביע עלול להיות גם ערך 'זיבלי')



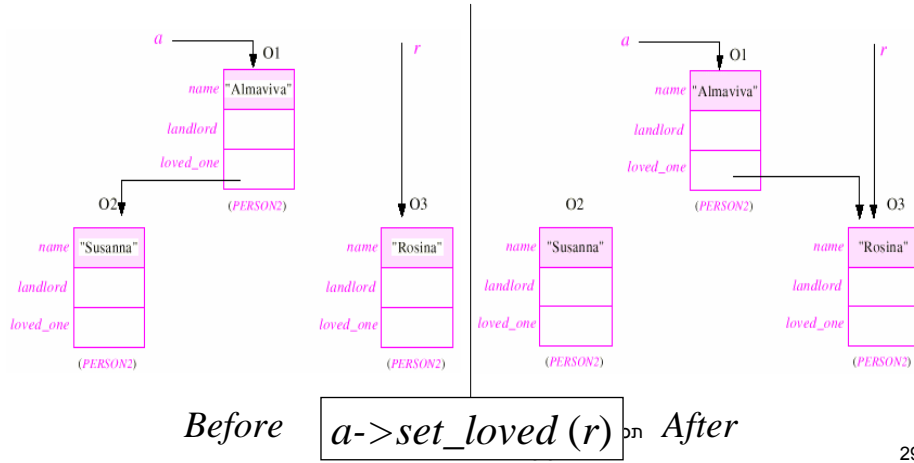
קשירת מצביע

```
class PERSON2
{
    string name;
    PERSON2 *loved_one, *landlord;

    /** Attach the loved_one field of current
        object to l. */
    void set_loved (PERSON2 *l)

    {
        loved_one = l;
    }
}
```

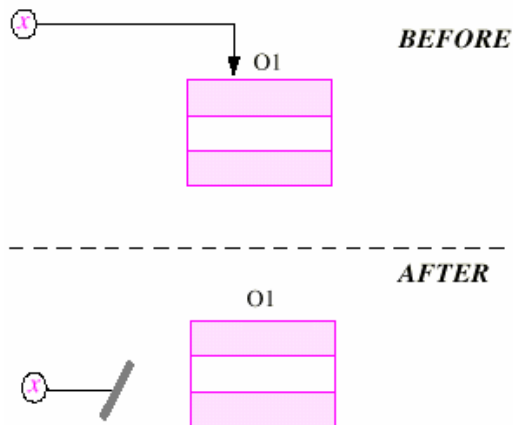
קשירת מצביע



29

איפוס מצביע (ללא שחרור!)

$x = null$



30



אתחול, השמה והריסה ב C++



אתחול ברירת מחדל

- אם הוגדר משתנה (עצם) עם ערך אתחול אזי זה יהיה ערכו הראשוני.

```
int i = 5;    לדוגמא:
```

- עצמים גלובלים, עצמים שהוגדרו בתוך מרחב שמות ועצמים סטטים מקומיים יאותחלו ל-0 אם לא הוגדר אחרת

לדוגמא:

```
int a; // means "int a = 0;"  
double d; // means "double d = 0.0;"
```




אתחול ברירת מחדל

- לעצמים מקומיים (משתני מחסנית, משתנים אוטומטיים), ועצמים שהוקצו דינאמית אין אתחול ברירת מחדל

דוגמא:

```
void f()
{
    int x; // x does not have a well-defined value
    // ...
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

33



פונקציה יוצרת

- במחלקות שנכתוב ניתן (ורצוי!) להחליף את אתחול ברירת המחדל בפונקציה
- פונקציה זו נקראת הבנאי (constructor) של המחלקה
- ניתן להעמיס אותה
- ניתן להעביר לה פרמטרים
- שמה כשם המחלקה שאותה היא מאתחלת ואין לה ערך מוחזר

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

34

בנאים ל CPoint

```
class CPoint
{
//...
public:
    CPoint()
    {
        m_x = 0;
        m_y = 0;
    }
    CPoint(int x, int y)
    {
        m_x = x;
        m_y = y;
    }
//...
}
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

35

בנאי יותר טוב ל CPoint

```
class CPoint
{
//...
public:
    CPoint(int x=0, int y=0)
    {
        m_x = x;
        m_y = y;
    }
//...
}
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

36



עוד על בנאים

- פונקציות אתחול שהוגדרו כ `private` לא תאפשר ליצור באמצעותה עצמים
- למחלקה שלא נכתבו עבורה בנאים כלל מספק המהדר (הקומפיילר) בנאי ברירת מחדל (`default constructor`) זהו בנאי חסר פרמטרים שאינו עושה דבר
- מרגע שנכתב בנאי כלשהו למחלקה לא ניתן עוד להשתמש בבנאי ברירת המחדל
- יש מקרים שבהם נהיה חייבים להשתמש בבנאי חסר פרמטרים ולכן נגדיר כזה לכל מחלקה

מפרקים (destructors)

- בכל פעם שעצם מסיים את תפקידו, יש לבצע פעולת הריסה מסודרת
- לדוגמא: בסיום חייו של עצם שלאחד משדותיו הוקצה זכרון דינמי יש צורך לשחרר הן את הזיכרון של העצם עצמו והן את הזכרון שהוקצה עבור שדותיו
- לכל עצם נגדיר פונקציית פירוק ששמה כשם המחלקה בצרוף הסימן טילדה (~)
- המפרק הוא יחיד, הוא יקרא אוטומטית לא ניתן לקרוא לו מפורשות ולא ניתן להעביר לו פרמטרים


מפרקים (destructors)

```
class Name {
    const char *s;
    // ...
};

class Table {
    Name *p;
    size_t sz;
public :
    Table (size_t s = 15)           // constructor
    { p = new Name[sz = s]; }

    ~Table() { delete[] p ;}      // destructor

    Name *lookup(const char *);
    bool insert (Name *);
};
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

39

דוגמא

- כל בנאי (גם בנאי ברירת המחדל) קורא לבנאים של כל שדותיו

```
struct Tables {
    int i;
    int vi[10];
    Table t1;
    Table vt [10];
};

Tables tt; // calls t1(15)
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

40

קבועים והפניות

- משתני const ומשתני reference יש לאתחל

```
struct X {  
    const int a;  
    int &r;  
};  
X x ;    // error: no default  
         // constructor for X
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

41

זמני יצירה והריסה של עצמים

- עצמים מקומיים (אוטומטיים, מחסנית)
 - נוצרים בכל פעם שמגיעים לשורת ההצהרה עליהם
 - נהרסים בכל פעם שיוצאים מהבלוק שבו הוגדרו
- עצמים שהוקצו דינמית
 - נוצרים בפקודת new
 - נהרסים בפקודת delete
- עצמים שהם שדות של עצם מכיל
 - נוצרים כאשר העצם המכיל נוצר
 - נהרסים כאשר העצם המכיל נהרס



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

42

זמני יצירה והריסה של עצמים

- אברים במערך
 - נוצרים בכל פעם שהמערך נוצר
 - נהרסים בכל פעם שהמערך נהרס
- משתנה סטטי מקומי (בפונקציה)
 - נוצר כאשר מגיעים לשורת ההצהרה עליו בפעם הראשונה
 - נהרס בסיום התוכנית כולה
- משתנה גלובלי, משתנה מרחב שמות ומשתנה סטטי של מחלקה
 - נוצר ב"תחילת התוכנית"
 - נהרס בסוף התוכנית



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

43

זמני יצירה והריסה של עצמים

- עצם זמני
 - נוצר במהלך חישוב הביטוי שבו הוא מופיע
 - נהרס בסיום חישוב הביטוי העוטף
- לא נדון בעצמים שמוקצים ע"י אופרטור הקצאה ובשדות של union (שאוילי אין להם כלל בנאי ומפרק)

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

44

זמני יצירה והריסה של משתנים מקומיים

```
void f(int i)
{
    Table aa;
    Table bb;
    if (i>0) {
        Table cc;
        // ...
    }

    Table dd;
    // ...
}
```

- aa, bb, ו- dd ייווצרו (בסדר זה) בכל קריאה ל f()
- dd, bb, ו- aa ייהרסו (בסדר זה) בכל יציאה מ f()
- אם $i > 0$ אז cc ייווצר לפני dd ויהרס לפני bb

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

45

העתקת עצמים

```
void h()
{
    Table t1;

    // copy initialization
    // same as Table t2(t1);
    Table t2 = t1;

    Table t3;

    // copy assignment
    t3 = t2;
}
```

- הבנאי יקרא פעמיים (עבור t1 ו-t3) ואילו המפרק 3 פעמים (עבור כולם)
- t2 ו-t1 חולקים את אותו מערך מכיוון שהאתחול ביצע העתקה נאיבית (שדה-שדה)
- לא ניתן לשחרר את המערך שהוקצה ביצירת t3 מכיוון שהוא נדרס ע"י ההשמה $t3=t2$
- יש צורך לשלוט הן על תהליך האתחול בעזרת העתקה והן על תהליך ההשמה בעזרת העתקה

התוכנית שגויה!

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

46



העתקת עצמים

- יצירת עצם תוך כדי אתחולו ע"י סימן "=" היא סוכר תחבירי לבנאי
- בנאי זה מכונה "בנאי העתקה" (copy constructor)
- השמה של עצם לתוך עצם קיים ע"י סימן "=" היא סוכר תחבירי לאופרטור ההשמה
- השמה ואתחול הן שתי פעולות שונות!

```
class Table {  
    // ...  
    // copy constructor  
    Table(const Table &);  
  
    // copy assignment  
    Table &operator=(const Table &);  
};
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

47



בנאי העתקה

- בנאי ההעתקה נקרא בעת יצירת עצם חדש ע"י ארגומנט מאותו טיפוס, העברה של נתון by value והחזרה של נתון by value

```
// copy constructor  
Table::Table(const Table& t) // by reference!  
{  
    p = new Name[sz=t.sz];  
    for(int i=0 ; i<sz ; i++)  
        p[i] = t.p[i];  
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

48



אופרטור השמה

1. הגני מפני השמה עצמית
2. שחררי את האובייקט הישן (על מרכיביו)
3. אתחלי אותו
4. העתיקי שדה-שדה את האובייקט החדש



אופרטור השמה

```
// assignment operator
Table &Table::operator=(const Table& t)
{
    if (this!= &t) // beware of t = t
    {
        delete []p;
        p = new Name[sz=t.sz];
        for(int i=0 ; i<sz ; i++)
            p[i] = t.p[i];
    }
    return *this;
}
```



תקציר הפרקים הקודמים...



עצמים מוכלים

- בתוך גוף הבנאי (אחרי הסוגריים המסולסלים) העצם כבר בנוי. לאתחל עכשיו את שדותיו יכול להיות מאוחר מדי או לחלופין לא יעיל

```
class Club {  
    string name;  
    Table members;  
    Table officers;  
    Date founded;  
    // ...  
    Club(const string& n, Date fd);  
};
```



עצמים מוכלים

- שורת האתחול מאפשרת לאתחל שדות לפני התחלת ביצוע הבנאי
- השדות יאותחלו לפי סדר הגדרתם במחלקה (ולא לפי סדר הופעתם בשורת האתחול)

```
Club::Club(const string& n , Date fd) : name(n) ,  
    members() , officers() , founded(fd)  
{  
    // ...  
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

53



עצמים מוכלים

- ניתן להשמיט שדות שהבנאי שלהם לא מקבל ארגומנטים

```
Club::Club(const string& n , Date fd)  
: name(n) , founded(fd)  
{  
    // ...  
}
```

- לאחר ביצוע ה destructor עבור עצם מסוים יקראו ה- destructors של כל אחד משדותיו המוכלים. המפרקים יקראו בסדר הפוך לסדר יצירת העצמים המוכלים

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

54



שורת האתחול

- עצמים ללא בנאי ברירת מחדל (ללא בנאי חסר פרמטרים), הפניות או קבועים חובה לאתחל בשורת האתחול

```
class X {
    const int i;
    Club c;
    Club& pc;
    // ...
    X(int ii, const string& n, Date d , Club& c)
    : i(ii), c(n,d), pc(c) { }
};
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

55



שימוש בעייתי בעצמים זמניים

```
void f(string& s1 , string& s2 , string& s3)
{
    const char *cs = (s1+s2).c_str();
    cout << cs;

    if(strlen(cs=(s2+s3).c_str())<8
        && cs[0]=='a ' )
    {
        // cs used here
    }
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

56



שימוש תקין בעצמים זמניים

```
void f(string& s1 , string& s2 , string& s3)
{
    cout << s1 + s2;
    string s = s2 + s3;
    if(s.length()<8 && s[0]=='a ' )
    {
        // use s here
    }
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

57



עוד שימוש תקין בעצמים זמניים

```
void g(const string&, const string &);

void h(string& s1 , string& s2)
{
    const string &s = s1 + s2;
    string ss = s1 + s2;
    g(s,ss); // we can use s and ss here
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

58



ועוד אחד

```
void f(Shape &s , int x , int y)
{
    // construct Point to pass to Shape::move()
    s.move(Point(x ,y));

    // ...
}
```



זה חנים !

- בשפת C++ עבור כל מחלקה מוגדר ע"י המהדר:
 - בנאי ברירת מחדל (בנאי ריק)
 - אלא אם הוגדר בנאי אחר כלשהו
 - בנאי העתקה
 - אופרטור השמה
- שלושתם בעלי הרשאת public
- את שלושתם ניתן לדרוס ע"י כתיבת גרסאות משלנו (בדר"כ רלוונטי עבור מחלקות המכילות מצביעים)



פעולות יסודיות על עצמים

בשפת C++ השפה אינה מספקת פעולות יסודיות נוספות פרט לאלו שפורטו בשקף הקודם



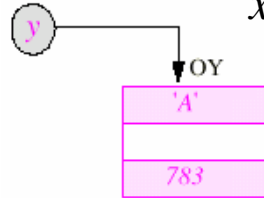
שיבוט העתקה והשוואה

- clone - הינה פעולה אשר יוצרת עותק זהה לזה של העצם המשובט ומחזירה מצביע אליו
- equal - השוואה בין שני עצמים שדה-שדה תמומש ב C++ בדר"כ ע"י האופרטור '=='. פעולה זו על המתכנת לממש בעצמו
- copy – העתקה שדה-שדה. ממומשת ב C++ ע"י אופרטור ההשמה
- ב copy לעומת clone, לא נוצר עצם חדש
- בהקשר הזה ניתן לדבר על deep_copy, deep_equal ו-deep_clone

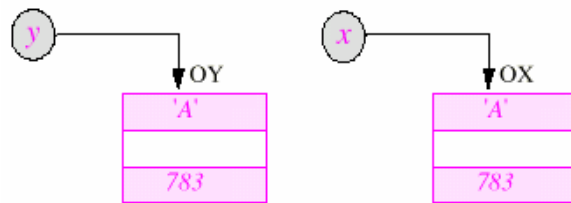
שיבוט עצמים



$x = \text{clone}(y)$

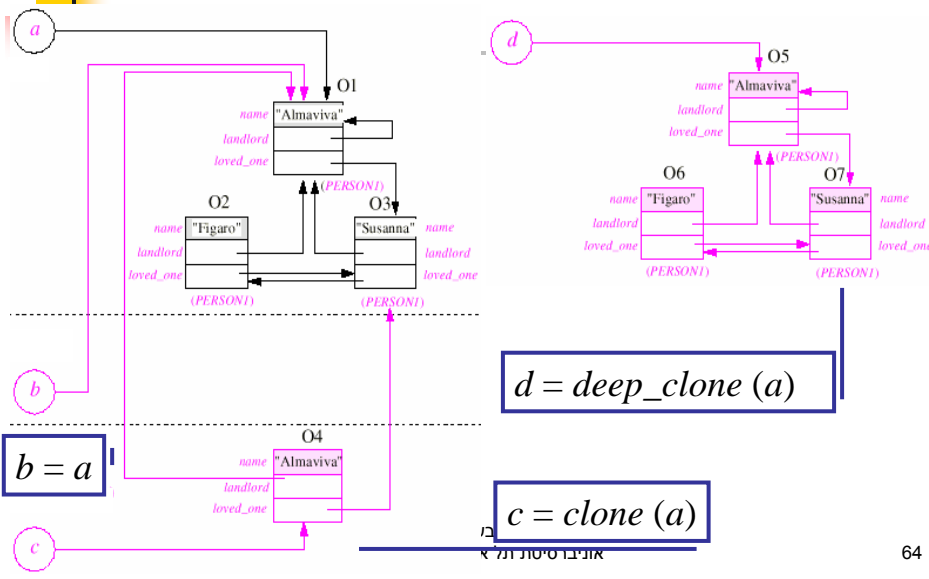


BEFORE



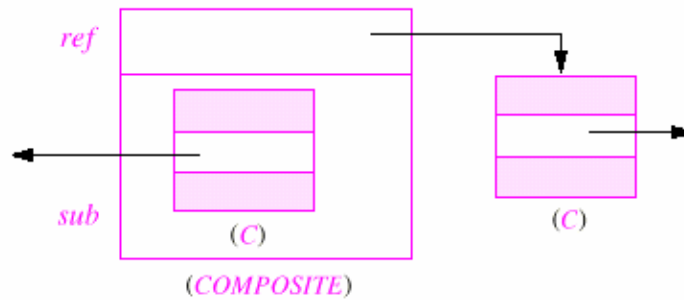
AFTER

שיבוט רדוד ושיבוט עמוק



עצם מוכל (expanded type)

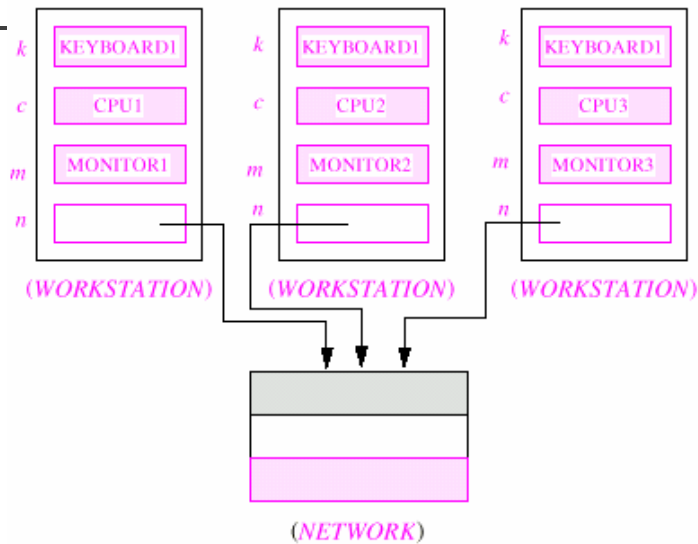
```
class COMPOSITE {  
    C *ref;  
    C sub;  
}
```



יתרונות העצם המוכל

- יעילות
 - גישה לשדות מוכלים שלא דרך dereference של מצביע
- מודל טוב יותר
 - מצביע למחלקה S פירושו שהלקוח "יודע על" S
 - עצם כעצם מוכל מעיד על כך שהלקוח מכיל S
 - בפרט, הכלה מרמזת על אי-שיתוף
- תמיכה אחידה בטיפוסים פרימיטיביים
 - עצמים מכילים את הטיפוסים היסודיים עצמם ולא מצביע אליהם

אובייקטים מורכבים



67

עצמים מוכלים

- לא יתכנו מעגלי ספק-לקוח כאשר הספק הוא עצם מוכל (expanded type)
- יצירה והריסה של האובייקט המוכל תלויה בזמני היצירה וההריסה של העצם המכיל
- בהשמת העצם העוטף לא יופעל בהכרח אופרטור ההשמה של כל אחד משדותיו המוכלים
- כאשר עצם מכיל מצביעים לעצמים אחרים יש לשקול מתי לשחרר את הזכרון של עצמים אלו (למשל אם הם משותפים לכמה עצמים)

68



מתי לשחרר

- לבעיה זו כמה גישות שונות:
- הקצת? תשחרר!
 - כל הקצאה בבנאי תשחרר במפרק.
 - כל העתקה של עצם אחר (בבנאי העתקה או ע"י אופרטור השמה) תשכפל מידע מוצבע
- ספירת הפניות
 - כל העתקה של עצם אחר המכיל מצביעים תעתיק רק את המצביע ותעדכן מונה.
 - כל שחרור של עצם המכיל מצביעים לא תשחרר את הזכרון המוצבע אלא רק תוריד את המונה ב-1
 - עצם שהוקצה דינמית ושמונה ההצבעות אליו ירד ל-0 ישחרר



קצת על הכללה

(templates)



הכללה (Generic Types)

- אחרי שתארנו ADT בעזרת טיפוס כללי אך טבעי לבנות מחלקה עם טיפוס כללי
- לכאורה, השימוש בטיפוסים סטטים מיותר – ואולם הוא לצורכי קריאות ופשטות
- הכללה מספקת מנגנון לשימוש חוזר בקוד (ע"י ביטול התלות בטיפוס)
- כתיבה נכונה לא מניחה דבר על הטיפוס הפרמטרי, ואולם הנחות מובלעות המופיעות בקוד, יגרמו לטעות קומפילציה עבור טיפוסים מסוימים



מחסנית של chars

```
class Stack {  
    char* v;  
    int max_size;  
    int top;  
public:  
    Stack(int s);           // constructor  
    ~Stack();              // destructor  
    void push(char);  
    void pop();  
    char top();  
};
```

מחסנית כללית

```
template <class T>
class Stack {
    T* v;
    int max_size;
    int top;
public:
    Stack(int s);           // constructor
    ~Stack();              // destructor
    void push(T);
    void pop();
    T top();
};
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

73

push כללי

```
template<class T>
void Stack<T>::push(T c)
{
    if(top == max_size)
        // Error Handling Code...
    v[top] = c;
    top = top + 1;
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

74



מחסנית לכל דורש

```
Stack<char> sc; // stack of characters
Stack<complex> scplx; // stack of complex numbers
Stack<list<int>> sli; // stack of list of integers

void f()
{
    sc.push('c');

    if(sc.pop() != 'c')
        // Logical Error!

    scplx.push(complex(1,2));

    if(scplx.pop() != complex(1,2))
        // Logical Error!
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

75



תבניות וטיפוסים

- עבור כל יצירה של מחלקה עם טיפוס אקטואלי (char, complex, list<int>) יוצר הקומפיילר עותק של התבנית (מחלקה) עבור טיפוס זה
- על המחלקה הנוצרת חלים כל כללי התאמת הטיפוסים הרגילים של C++:
- `sc.push(4.5)` טעות קומפילציה

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

76



המערך מהדור הישן

```
struct Entry{
    string name;
    int number;
};

Entry phone_book[1000];

void print_entry(int i) // simple use
{
    cout << phone_book[i].name << ' ' <<
        phone_book[i].number << '\n';
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

77



המערך של הדור החדש vector<T>

```
vector<Entry> phone_book(1000);

// simple use, exactly as for array
void print_entry(int i)
{
    cout << phone_book[i].name << ' ' <<
        << phone_book [i].number << '\n';
}

void add_entries(int n) // increase size by n
{
    phone_book.resize(phone_book.size()+n );
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

78



אופרטור ההעתקה של vector

מה עושה הקוד הבא?

```
void f(vector<Entry>& v)
{
    vector<Entry> v2 = phone_book;
    v = v2;
    // ...
}
```

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

79



vector

- השימוש ב vector פותר את בעיית ניהול הזכרון, אבל הוא לא פותר את בעיית הגלישה מגבולות המערך
- האופרטור [] לא מבצע בדיקה כי הכניסה המבוקשת קיימת במערך (מדוע?)
- המתודה at(int) מספקת שרות זה
- כדי לבצע בדיקה בכל גישה ניתן להגדיר מחלקה עוטפת (Vec) שבה יוגדר האופרטור [] ע"י קריאה ל – at(int). טכניקה זו נקראת האצלה (delegation)

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

80

האצלת סמכויות (delegation)

```
template<class T>
class Vec {
    vector<T> rep;
public:
    Vec() : rep() {}
    Vec(int s) : rep(s) {}
    int size() const {return rep.size();}
    //for all methods of vector<T>...

    T& operator[](int i)
    { return rep.at(i); } // rangechecked

    const T& operator[](int i) const
    { return rep.at(i); } // rangechecked
};
```



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

81

עוד על vector

- פעולות על המחלקה vector עם דוגמאות מצויינות באתר MSDN:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcstdlib/html/vclrfvectormembers.asp>

- או חפשו ב Google :msdn c++ vector

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

82