

# inline, Composition, Constructors & Destructors

הרצאה 2



# inline

# מהו inline?

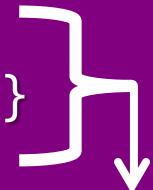
- ❖ קריאה לפונקציה זה דבר יקר overhead של מחסנית הקריאה).
- ❖ בפונקציות קצרות overhead גדול יותר מעלה הפונקציה עצמה – ב贊ני!
- ❖ inline – בקשה מהקומפיילר להעתיקת קוד הפונקציה במקום לקרוא לה ישירות.
- ❖ עולה יותר בזמן קומפיילציה ובגודל קוד!
- ❖ חוסך הרבה בזמן ריצה!
- ❖inline בקשה שתיענה רק אם הקומפיילר יחליט שזה סביר!
- ❖ פונקציות פשוטות וקצרות.
- ❖ בלי תנאים, לולאות, מבני בקרה, קוד אחר וכו' inline שלא נוענה – פשוט התעלמות! זה לא שגיאה!

# בקשת inline

- ❖ – משתמשים את המתודה בתוך הוצאה המחלקה.
- ❖ - המתודה ממומשת מחוץ להוצאה המחלקה, אך:
  - ❖ בקובץ ה-H.
- ❖ המילה השמורה inline מופיעה לפני מימוש הפונקציה.

# דוגמה ל-**inline**

```
//File: TwoDigitNumber.h
#ifndef TwoDigitsNumber_H
#define TwoDigitsNumber_H
#include <iostream>
class TwoDigitsNumber {
private:
    char m_firstD, m_secondD; //the digits
public:
    void setFirstDigit(char d1) {m_firstD = d1;}
    void setSecondDigit(char d2) {m_secondD = d2;}
    void show();
};
```



Implicit inline!

```
inline void TwoDigitsNumber::show() {
    cout<<"The number is: "<<m_secondD
        <<m_firstD<<endl;
}
```

```
#endif
```

→ Explicit inline!

# הכליה (Composition)

# Composition

- ❖ משתי מחלוקת לא חייבים להיות משתנים פרימיטיביים. הם יכולים בעצם להיות אובייקטים של מחלוקת כלשהי.

```
//File: Line.h
#ifndef LINE_H
#define LINE_H
#include "Point.h"
class Line {
private:
    Point m_p1, m_p2; //Composition!!
public:
    void setLine(int x1, int y1, int x2, int y2);
    void setLine(const Point& p1, const Point& p2);
    void show();
};
inline void Line::show() {
    cout<<"Line from: ";           m_p1.show();
    cout<<" To: ";               m_p2.show();
}
#endif //LINE_H
```

```
//File: Line.cpp
#include <iostream>
#include "Point.h"
#include "Line.h"

void Line::setLine(int x1, int y1, int x2, int y2)
{
    m_p1.set_x(x1);
    m_p1.set_y(y1);
    m_p2.set_x(x2);
    m_p2.set_y(y2);
}

void Line::setLine(const Point& p1, const Point& p2)
{
    m_p1 = p1;
    m_p2 = p2;
}
```

```
//main.cpp file:  
#include "Point.h"  
#include "Line.h"  
  
int main() {  
    Point p1, p2;  
    p1.set_x(15);      p1.set_y(10);  
    p2.set_x(0);       p2.set_y(0);  
  
    Line line1, line2;  
    line1.setLine(15,10,7,6);  
    line1.show();  
    line2.setLine(p1,p2);  
    line2.show();  
  
    return 0;  
}
```

### Output:

Line from: x=15 y=10

To: x=7 y=6

Line from: x=15 y=10

To: x=0 y=0

# אובייקטים כמשתני מחלקה - דוגמא

- ❖ בהינתן מחלקת `String` איך נגידיר את מחלקת `Client` השומרת עבור כל לקוח את שמו (פרטיו ומשפחה) וכן מספר לקוח (מספר סידורי בן 4 ספרות)?
- ❖ ממשו את מחלקת `משולש` (מוגדר על ידי 3 נקודות המהוות את 3 הקודקודים של המשולש).

# משתני מחלוקת לא פרימיטיביים

- ❖ **משתני מחלוקת יכולים להיות:**
  - ❖ אובייקטים (הכלה - composition)
  - ❖ מערך של אובייקטים (מוד נתונים של חברה המכיל את רשימת כל העובדים).
  - ❖ מצבים לאובייקטים (מחלקת מורה יכול מצביע לקורס שהוא מלמד, ולא את הקורס עצמו. מדוע?) – נקרא גם aggregation.

# מצביעים לאובייקטים

- ❖ בדיקן כמו שיש מצביעים למשתנים רגילים כך אנו יכולים להגדיר גם מצביעים לאובייקטים
  - ❖ `int* ptInt` ל-`int`:
  - ❖ `Point* ptPoint` ל-`Point`:
- ❖ חשוב לזכור: בדיקן כמו במצביעים לפרימיטיבים – גם פה מצביעים לא בהכרח מאותחלים או תקפים (מצביעים לאובייקט קיימים). אנו מוכרים:
  - ❖ לאותחל כל מצביע עם ערך אמיתי ותקף או `NULL`.
  - ❖ להשים לתוך המצביע כתובת של אובייקט קיים, או:
  - ❖ להקצת אובייקט חדש (על ה-`heap`) ולהשים את הכתובת לתוך המצביע.

# **בנאים - Constructors**

**&**

# **הורסרים - Destructors**

# אתחול – מетодת הבנאי (constructor)

## מוטיבציה:

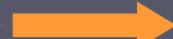
- ❖ כפי שכבר למדנו – משתנים נוצרים עם **ערך זבל** בתוכם.
- ❖ אותו הדבר תקף גם למשתני מחלוקת באובייקטים.
- ❖ ערכי זבל יכולים לגרום מספר בעיות קשות:
  - ❖ שימוש במשתני מחלוקת לפני שאותחלו.
  - ❖ **שימוש במצביים לפני שהוקצו!!!**
- ❖ למחרת שניתן לייצר מетодות `(init() או set(` מתאימות, אין שום הבטחה שהמתכנת המשתמש בחלוקת (`theUser` שלנו) ישתמש בהן לפני השימוש באובייקט וגם אין דרך לחיבר אותו בכור.

# METHOD הבנאי (Constructor)

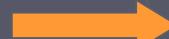
## בקדמה:

- ❖ METHOD הבנאי (constructor – C'tor) יכולה לפתר את כל הבעיות האלו.
- ❖ ה-Ctor נקראת **אוטומטית** בזמן שהאובייקט נוצר!

Object  
creation



Memory  
Allocation



C'tor  
Activation

- ❖ METHOD ה-Ctor יכולה:
  - ❖ לאותחל משתנים
  - ❖ להקצות זיכרון דינמי למשתני המחלוקת הecessary לכך.
  - ❖ וכן לבצע כל פעולה אחרת שאנו מעוניינים כי תבוצע לפני שמתחילים לעבוד עם האובייקט.

# ה-Ctor - אתחול

- ❖ ה- Ctor مبוטית לנו אתחול של כל אובייקט (מחלקה זאת).
- ❖ ביצירת אובייקט מחלוקת בעלת Ctor – הקומפיאר קורא לבניית אוטומטית.
- ❖ שם מתודת ה-Ctor הוא שם המחלוקת.
- ❖ המתודה צריכה להיות מוגדרת ב-public של המחלוקת.
- ❖ מדוע? מה יקרה אחרת?
- ❖ כמו כל פונקציה \ מתודה, גם מתודת ה-Ctor יכולה לקבל פרמטרים.
- ❖ ניתן לייצר כמה מתודות Ctor לאותה מחלוקת תוך שימוש בעוממת פונקציות - "Function name overloading".
- ❖ מתודת Ctor לא יכולה להחזיר ערכים. אסור לכתוב שהיא מחזירה void (שגיאת קומפיאציה!).

# ה-Ctor הדיפולטי

- ❖ **ה-Ctor הדיפולטי** הוא הבנאי שנקרא ב  - פרמטרים.
- ❖ הוא כל כך חשוב שם (וრק אם) אין Ctor (בכלל!) למחילה הקומפיאילר ייצור אחד אוטומטית למחילה הנ"ל.
  - ❖ ביצירת מערך של אובייקטים חייבים שייהי במבנה דיפולטי למחילה!

חוק אכבע:

זה קצת תמיד תכנות טוב לספק במבנה דיפולטי!

# מתנות הקומפיילר (מהדר)

- ❖ ישן 4 מетодות מאוד חשובות בכל מחלוקת.
- ❖ הן כל-כך חשובות וקריטיות בכל מחלוקת אפשרית, שאמם אנו לא ניצור אותן בחלוקת מסוימת – נקבל אותן "במתנה" מהקומפיילר.
- ❖ השימוש של הקומפיילר למетодות אלו הוא דיפולטי.
- ❖ אם נממש בעצמנו את המethodות הנ"ל – לא נקבל אותן מהקומפיילר (את אלו שלא נממש – כן נקבל).
- ❖ לרוב השימוש לא טוב לצרכינו – ורק אנו נממש את כל ה-4 בעצמנו בכל מחלוקת!
- ❖ **ה-*Ctor* הדיפולטי** הוא מתנת הקומפיילר הראשונה!
- ❖ מתנה לא טובה כי: שם ערכי זבל בתוך המשתנים!

# דוגמא של Ctor – לא דיפולטי!

```
#ifndef POINT_NO_DEFAULT_H
#define POINT_NO_DEFAULT_H
class PointNoDefault {
public:
    PointNoDefault (int valX, int valY)
        {m_x = valX; m_y = valY;}
    ...
    //The rest of the class declaration...
private:
    int m_x, m_y;
};
#endif //POINT_NO_DEFAULT_H
```

## דוגמא של Ctor – לא דיפולטי!

```
#include <iostream>
#include "PointNoDefault.h"
int main() {
    PointNoDefault pnd1(15,10);
    PointNoDefault p2; //Error!!!
}
```

יש פה שגיאת קומפילציה!!

למחלקה PointNoDefault אין בנאי דיפולטי

# דוגמה של שימוש ב-Ctor

```
#ifndef POINT_H
#define POINT_H
#include <iostream>

class Point {
public:
    Point() { m_x = 0; m_y = 0;} //default Ctor
    Point(int valX, int valY) {m_x = valX; m_y = valY;}
    void Set_x(int val) {m_x = val;}
    int Get_x() {return m_x;}
    void Set_y(int val) {m_y = val;}
    int Get_y() {return m_y;}
    void Show() {cout<<"x = "<<m_x<<, y = "<<m_y<<endl;}
private:
    int m_x, m_y;
};

#endif // POINT_H
```

## דוגמא של שימוש ב-Ctor

```
#include <iostream>
#include "Point.h"
int main() {
    Point p1(15,10);
    Point p2; //default point

    cout<<"The first point - ";
    p1.show();
    cout<<"The second point - ";
    p2.show();
    return 0;
}
```

The first point – x=15 y=10

The second point – x=0 y=0

# שורה איתחול - Initialization List

- ❖ בהכלה (composition) ה-Ctor נקראים בסדר הבא:
  - ❖ ה-Ctor של האובייקטים הפנימיים (חברי המחלקה).
  - ❖ ה-Ctor החיצוני של המחלקה.
- ❖ לדוגמה: Line מורכבת מ-2 Point. קודם כל נוצרת הנקודות הפנימיות ורק אחר כך הקו החיצוני!
  - ❖ מה קורה בהriseה?
- ❖ דבר זה נכון גם כאשר חברי המחלקה (data members) הם טיפוסים פרימיטיביים וגם כשהם בעצם אובייקטים.
- ❖ הקומפיאילר מודד את סדר היירה של אובייקט מהמחלקה:
  - ❖ דבר ראשון יש יירה של כל חברי המחלקה הפנימיים.
  - ❖ הקומפיאילר לא ידע איך לייצר אותם (אין לו פרמטרים) ולכן גורא  
לבנאים הדיפולטים שלהם!

# שורת אתחול - שאלות

❖  שאלה 1: מה קורה כשה-member של המחלקה הוא אובייקט לтиיפו המוגדר על ידי משתמש (מחלקה אחרת) שאין לו בנאי דיפולטי?

❖  שאלה 2: איך מתחלים `data member` מסוג `reference const` לערךם בעלי משמעות?

# שורט אתחול - התשובה

- ❖ תשובה: נשתמש באזור אתחול חברי המחלקה (שירות אתחול):
- ❖ חלק מה-Ctor שmagיע מיד לאחר שורת הפרמטרים, מתחיל ב ' : ' ומסתיים בפתח סוג מסולסל ' } ' (תחילתימוש הבנאי).
- ❖ מאד יעיל ולכן נחשב תכנות נכונות טוב להשתמש בו תמיד (כשאפשר) לאתחול משתנים ב-Ctor.
- ❖ קיימים אר ורק במתודת ה-Ctor.

```
class Point {  
public:  
    Point();  
    Point(int valX, int valY);  
    //the set & get methods as before...  
private:  
    int m_x, m_y;  
};
```

עקב חוסר במקום (בש侃פימ)  
משלב זה לא י Zion יותר ה-  
ifndef ifno' וכו' בקבצי ה-H.

```
#include <iostream>  
#include “Point.h”
```

```
Point::Point() {  
    cout<<“Creating a default point...\\n“;  
    m_x=0;  m_y=0;  
}
```

```
Point::Point(int x, int y) {  
    cout<<“Creating a point...\\n“;  
    m_x=x;  m_y=y;  
}
```

```
#include <iostream>
class Line {
public:
    Line() {cout<< "Creating a default line...\n";}
    Line(int x1, int y1, int x2, int y2);
private:
    Point m_p1, m_p2;
};
```

```
#include <iostream>
#include "Line.h"

Line::Line(int x1, int y1, int x2, int y2) {
    cout<<"Creating a line...\n";
    //set the x and y values of p1 and p2
    //with the arguments x1,y1,x2,y2.
}
```

```
#include "Line.h"
int main() {
    Line l1;
    Line l2(2,5,7,8);
}
```

Creating a default point...  
Creating a default point...  
Creating a default line...  
Creating a default point...  
Creating a default point...  
Creating a line

*A small change – init list*

```
#include <iostream>
class Line {
public:
    Line() {cout<< "Creating a default line...\n";}
    Line(int x1, int y1, int x2, int y2);

private:
    Point m_p1, m_p2;
};
```

```
#include <iostream>
#include "Point.h"
#include "Line.h"

Line::Line(int x1, int y1, int x2, int y2)
    : m_p1(x1,y1), m_p2(x2,y2) {
    cout<<"Creating a line...\n";
}
```

*A small change – init list*

```
#include "Line.h"
int main() {
    Line l1;
    Line l2(2,5,7,8);
}
```

Creating a default point...  
Creating a default point...  
Creating a default line...  
Creating a point...  
Creating a point...  
Creating a line

# שורת אתחול - דוגמא

- ❖ כיצד נשנה את מימוש ה-**Ctor** של מחלקה Point שראינו קודם  
קודם כך הייתה עם שורת אתחול?

```
Point::Point(int x, int y) {  
    cout<<“Creating a point...\\n“;  
    m_x=x;  m_y=y;  
}
```

- ❖ למה זה יותר יעיל?
- ❖ למחלקה זאת יש גם בנייתי-default, האם במקומם לבנות אותו  
יכלנו פשוט לשים ערכי-default במבנה זהה?

# שורה אתחול - הערות

- ❖ אם אין לנו מה לבצע בתוך ה-Ctor לאחר שורה אתחול –  
נרשום מימוש ריק (סוגרים מסולסלים ריקים {}).
- ❖ שורה האתחול נחשבת חלק **מיימוש** המתודה – ולכן צריכה  
להירשם במקומם מימוש המתודה – בקובץ ה-**CPP**.
- ❖ סדר הופעת ההשמות בשורה האתחול **לא** מחייב! הערכים  
**יאתחלו** (לרוב) לפי סדר הaczירה שלהם במחלקה.

# ההורס: The Dtor – Cleanup

- ❖ **ה-Dtor – Cleanup** הוא מетод הנקיוי המובטח של האובייקט.
- ❖ נקרא אוטומטית על ידי הקומpileר כ-**sh-lifetime** של האובייקט מסתiem.
- ❖ תחבר דומה לשול ה-Ctor, רק ששם נוסף ~ בהתחלה.
- ❖ ל-Dtor אין פרמטרים! משום שאין לו אופציות לבחור מהן.
- ❖ לכל מחלקה יש רק Dtor אחד(!) – אין העמתת פונקציות. למה?
- ❖ גם ל-Dtor (כמו ל-Ctor) אין ערך החזרה.

# מетодת ה-Destructor

## מטריבציה

- ❖ ה-Dtor היא מетодת שאחראית לבצע פעולות בדיק **לפני שאביקט נהרס סופית.**
  - ❖ שחרור הקצאות זיכרון דינמיות.
  - ❖ פעולות אחרות שאנו רוצים לבצע...
- 
- ❖ ה-Dtor (הורסים) יופעל בנסיבות הבאים:
    - ❖ הריסת משתנים מקומיים (כשיוצאים מהבלוק בו הוגדרו).
    - ❖ מהפעלת delete על אובייקט שהוקצתה דינמית.
    - ❖ הריסת משתנים גלובליים בסיום האפליקציה.

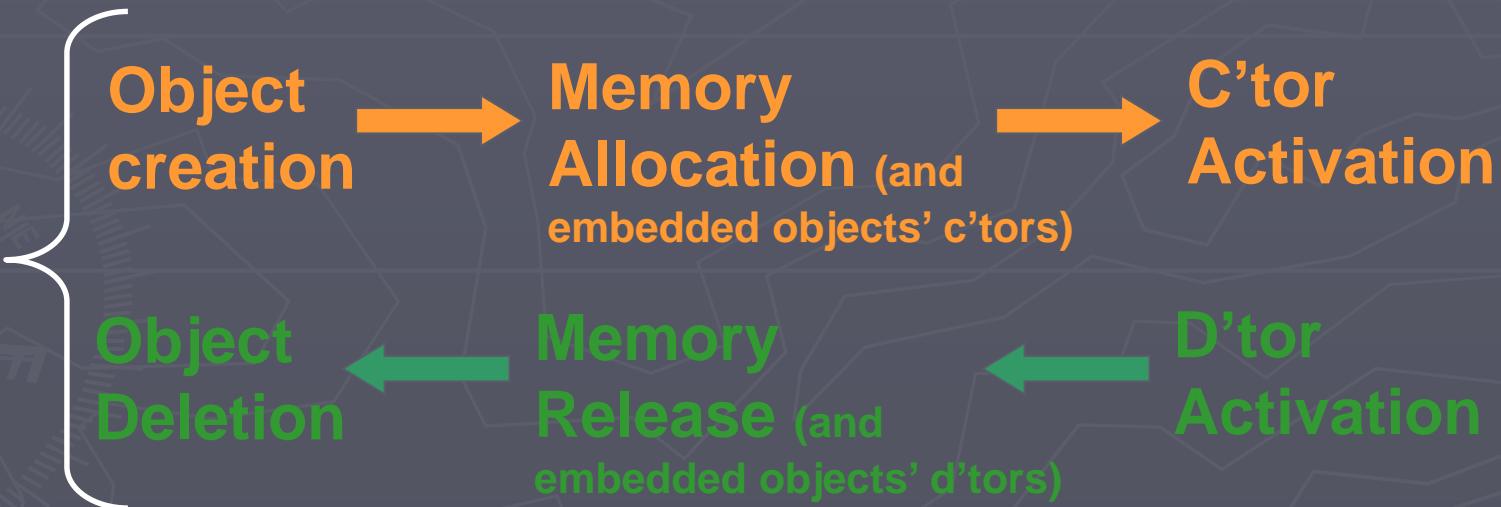
# (2) Destructor-הmethod

## הגדרה

`~ClassName`

- ❖ שם method הירושה (ה- Dtor) היא:
- ❖ method הירושה צריכה להיות ב-`public` של המחלקה.
- ❖ מה קורה אחרת?

Reverse  
order



```
#include <iostream>
class Point {
public:
    Point();
    Point(int valX, int valY);
    ~Point() {cout<<"Deleting a point...\\n";}
    //the set & get methods as before...
private:
    int m_x, m_y;
};
```



```
#include <iostream>
#include "Point.h"
Point::Point() : m_x=0, m_y=0 {
    cout<<"Creating a default point...\\n";
}
Point::Point(int x, int y) : m_x=x, m_y=y {
    cout<<"Creating a point...\\n";
}
```

```
#include <iostream>
#include "Point.h"
class Line {
private:
    Point m_p1, m_p2;
public:
    Line() {cout<< "Creating a default line...\n";}
    Line(int x1, int y1, int x2, int y2);
    ~Line();
};
```



```
#include <iostream>
#include "Point.h"
#include "Line.h"
Line::Line(int x1, int y1, int x2, int y2) : m_p1(x1,y1), m_p2(x2,y2) {
    cout<<"Creating a line...\n";
}
```

```
Line::~Line() {
    cout<<"Deleting a line...\n";
}
```



```
#include <iostream>
#include "Line.h"

int main() {
    Line l1;
    Line l2(2,5,7,8);
}
```

Creating a default point...  
Creating a default point...  
Creating a default line...  
Creating a point...  
Creating a point...  
Creating a line...  
Deleting a line...  
Deleting a point...  
Deleting a point...  
Deleting a line...  
Deleting a point...  
Deleting a point...

# ה-Dtor ניקוי המחסנית / הזיכרון

- ❖ ה-Dtor קרייטי לניקוי מוצלח של כל הזיכרון.
- ❖ מה שנוצר סטטיות – על המחסנית, משוחרר על ידי הקומפיילר, אך כל דבר מסוים יותר דורש הוספה קוד ל-Dtor שחרור זיכרון דינמי.
- ❖ סגירת קבצים (או משאבים אחרים: socket, semaphore וכו')
- ❖ ה-Dtor כרך כל חשוב שהוא **מתנה הקומפיילר השנייה**
- ❖ אם לא ניצור אותו בעצמו – הקומפיילר ייתן לנו למחוקה בעצמו.
- ❖ מתנה לא טובה בגלל שלא מנקה את מה שלא על המחסנית...

# הקצתה דינמית עם Dtor- Ctor

- ❖ Ctors שימושיים במיוחד לאובייקט יש שדות (חברי מחלקה) מסווג מוצבאים שמקצים דינמית.
- ❖ ב-Ctor אנו מקצים את המקום דינמית או משימים NULL לתוך המצביע (Ctor דיפולטי).
- ❖ הזכרן המקצתה דינמית ישוחרר (delete) ב-Dtor (כאשר האובייקט ימות)  
❖ וכך נמנעת דליפת זיכרון!
- ❖ מה קורה כאשר יש לנו כמה עותקים מאותו אובייקט?

# מבנה הזיכרון (זיכרון)

Stack

Heap

Global Data Segment

Code Segment

# בנאי העתקה - Copy Constructor

❖ האחראי על העברת והחזרת משתנים מעתיקות:  
בזמן הקריאה לפונקציות:

```
int f(int x, char c);  
  
int g = f(a,b);
```



```
push b  
push a  
call f()  
add sp, 4  
mov g, register a
```

Function  
Arguments

Return address

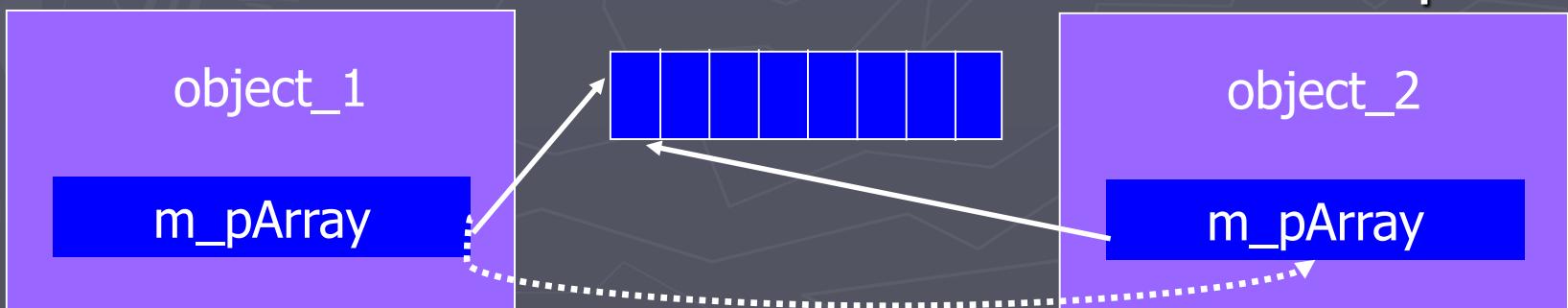
Local variables

# **בנייה העתקה – דרך להעתיק אובייקט**

- ❖ בשליחת אובייקט `value by` נוצר אובייקט חדש כהעתק של האובייקט המקורי.
- ❖ אובייקט חדש = משתנה מקומי המוכר רק בפונקציה וימותה בסופה.
- ❖ אובייקט המקורי = המשתנה החי מחוץ לפונקציה. לא מוכר בפונקציה (`scope`) אך יחיה לאחריה (`lifetime`).
- ❖ אותו דבר נכון גם בהחזרה `value by` של אובייקט!
- ❖ בניין העתקה (`Ctor copy`) הוא כל כר חשוב וקירותיו לתקן אובייקט שהוא מתנת הקומפיאילר השלישי!
- ❖ **מדוע זאת לא מתנה טובה?!**

# הבעיות עם Copy Constructor

- ❖ בהעברה by value הקומפイルר מniche שאנו רוצים העתקה רדודה של בית בית (bitcopy).
- ❖ לדוגמה: שדה מסוג int שמכיל 5 (0101...0) יועתק ביטים ולכן גם באובייקט העתק יהיה באותו מקום את הערך 5!
- ❖ הבעיה: מציבאים!
- ❖ מציבאים = משתנים שתוכנם הוא כתובות של משתנה אחר.
- ❖ בהעתקה bitcopy מועתק התוכן שלהם = הכתובת!
- ❖ גם האובייקט המקורי וגם העתק מציבאים על אותו מקום בדיק!



# The Copy Constructor (CC) Function

## מוטיבציה

- ❖ בהעתקה אובייקט – כל השדות מועתקים העתקה רדודה (shallow copy).
- ❖ כאשר חלק מהשדות הן מצביים – נקלט מצב של הצבעה כפולה (pointing dual):
- ❖ שינוי מידע באובייקט אחד – ישנה את המידע בכל האובייקטים המועתקים! (פוגע בכל הרעיון של העברת value by!).
- ❖ כאשר האובייקט מת – הם משחרר את השדות שלו (Dtor) – מה שגורם לכל העתקים האחרים לאבד את המידע!
- ❖ אם נממש בナイ העתקה (copy Ctor) משלנו שייתיק את האובייקט כפי שציריך – בהעתקה عمוקה!
- ❖ קר ונמנע מהקומפיילר להעתיק תור שימוש ב-bitcopy!
- ❖ כאשר אנו משתמשים את אחת ה"מתנות" עצמנו – הוקומפיילר ישתמש במימוש שלנו!

# METHOD ה-Ctor

- ❖ ה-CC (Copy Ctor) היא METHOD בינוי (Ctor) שמקבלת פרמטר אחד – אובייקט מסווג המחלקה (זה שהוא עתיק!).
- ❖ אסור להעביר את הפרמטר הזה !by value מדוע?
  - ❖ בהעברה value by נקרא ה-Copy Ctor.
  - ❖ שליחה value by של משתנה מסווג המחלקה לפני הכניסה ל-CC תגרום לקריאה ל-CC בשביל להעתיק את האובייקט.
  - ❖ תיזכר פה לולאה אינסוף!
- ❖ אם נשלח את הפרמטר תמיד !by const reference – כי אי אפשר use by value וזה יותר נכון מאשר by .address
- ❖ – כי לא נשנה את הפרמטר בזמן המethod!

# מתי משתמשים בMETHOD ה-Ctor?

.1. מתחלים אובייקט מאובייקט אחר

Point p1(3,4); //Ctor

Point p2 = p1; //Copy Ctor

Point p3(p1); //Copy Ctor

.2. מעבירים אובייקט by value

.3.מחזירים אובייקט by value

.4. אובייקטים זמינים (למשל בהמרות).

```
Class MyClass {  
    var1, var2, var3  
    ...  
}
```

```
MyClass(const MyClass& a) :  
    var1(a.var1), var2(a.var2), var3(a.var3)  
    ...  
}
```

❖ מספר נקודות נוספות:

❖ הבניי העתקה הדיפולטי:

❖ בניי העתקה פרטי.

```
class String {  
private:  
    char* m_str;  
    //...  
public:  
    String(const char* str=NULL); // How many Ctor?  
    String(const String& str); //copy Ctor  
    ~String(); //Dtor  
    //...  
};
```

```
#include “String.h”  
String::String(const String& str) {  
    m_str = new char[strlen(str.m_str)+1];  
    strcpy(m_str, str.m_str);  
}
```

# דוגמא מחלוקת מחסונית

```
typedef enum {Fail, Success} Result ; /* possible  
return values */  
  
class Stack {  
public: /* public methods - interface: */  
    Stack(int max_size=10);  
    ~Stack();  
    Result push(int elm) ;  
    Result pop() ;  
    Result top(int& elm) ;  
    int count();  
private:  
    int* array ;  
    int top ;  
    int maxCapacity ;  
};
```

# דוגמא מחלוקת מחסנית – חלוקת אחריות!

- ❖ שימו לב: במקרה של בעיה – הפניקציה מחזירה ערך המציין שגיאה (num) ולא מבצעת שום דבר נוסף!
- ❖ הדבר נובע מעיקרונו הפשטת הנ吐נים של OOP:  
**המחסנית היא רק מחסנית!** היא לא אמורה להתעסק עם טיפול בשגיאות!
- ❖ דבר זה נכון לגבי כל המחלוקות!
- ❖ מי שאמור להתמודד עם טיפול בשגיאות (למשל כתיבה למסר) היא המחלוקת המנהלת את המערכת או שמדובר על הדפסות הודעות שגיאה: המחלוקת שמנהלת את הקשר עם המשתמש בתוכנית!
- ❖ **חלוקת אחריות** נcona בין מחלוקות היא עקרון מפתח ל-  
!reusability

# דוגמא – רשימה מקו施רת

- ❖ כתה כיש לנו את מחלקת עובד, אנו רוצים ליצור את מוד הנתונים של החברה תוך שימוש ברשימה מקו施רת של עובדים.

- ❖ תזכורת רשימה מקו施רת:

LinkedList



# רשימה מקו施רת - תזכורת

```
class LinkedList
```

```
    int length;
```

```
    Node* list_head;
```

```
class Node
```

```
    Data* data;
```

```
    Node* next;
```

# רшиמה מקוشرת – Linkedlist.h

```
#include <iostream>
. . .
class Linklist {
    private:
        class Node {
            public:
                Data* data;
                Node* next;
        };
    Node* list_head;
public:
    linklist();
    ~linklist();
    Result insert (Data* data);      //insert a new element
    with the data 'data'
    Result deleteElement (Data* data); // delete the
    element with the data 'data'
    Result deleteNum (int num);      //delete the num'th
    element
    . . .
};
```



מחלקה  
פנימית!

## דוגמאות סיכום - עובד

- ❖ נניח שיש לנו חברת גודלה ואנחנו רוצים לתחזק את העובדים בה ומשכורתם שהם מקבלים. לכל עובד נרצה לשמר:
  - ❖ שם העובד.
  - ❖ משכורת העובד.
- ❖ איזה פעולות (METHODS) נצטרך לחלוקת זאת?
- ❖ כתבו אותה!
- ❖ כתבו מחלוקת רשימה מקושרת של עובדים.