

תכנות מונחה עצמים

הרצאה 1 – הקדמה לתוכנות מונחה
עצמים

ADMINISTRATIVE

בחינה ותרגילים

- ❖ כל החומר הנלמד בהרצאה ובמעבדות הוא למחhn.
- ❖ בחינה סופית - 80% מציון הקורס.
- ❖ תרגילי בית – 20% מציון הקורס
- בסביבות ה-5-6 תרגילי תכנות בשפת C++.
- בדיקת התרגילים תווידא קודם שהם מתקמפלים ורצים ורק במידה וכן תהיה בדיקה של איצ'ות הקוד.
- תרגיל שאינו מתקמפל יקבל אפס.
- תרגיל שעף בראיצה – ציונו יהל מ-60.
- **העתיקות, עבודות משותפות או שימוש בקוד של אחרים - אסור!**

ADMINISTRATIVE

mood

❖ עדכוניים וחומרים לימוד

- המציגות והדוגמאות מהכיתה (מלבד אלו שיכתבו על הלוח וחובתכם להעתיק!)
- חומרי המעבדה
- עדכוניים חשובים
- ועוד דברים נוספים...

svetar@sce.ac.il

❖ דוא"ל :

- בשורת הנושא יחד עם נושא הודעה חייב להופיע שם הקורס!
- בתוך האימייל חובה לכתוב שם מלא + ת"ז וקמפוס
- אימיילים לא לפני ההנחיות אלו – לא יענו.

שעות קבלה

- א' 17:00-18:00 בחדר G02 ב', בניין קציר במחלקה
ג' 17:00-18:00 בחדר G02 ב', בניין קציר במחלקה

ספרות

- ▶ The C++ Programming Language / Bjarne Stroustrup (Third edition).
- ▶ Thinking in C++ / Bruce Eckel (Second edition) - available online.
- ▶ Effective C++ / Scott Meyers, Addison-Wesley, 1997 (Second edition).

כל ספר טוב על שפת ++ C יסייע לך בקורס זה.

חומר רקע לקורס זה

- ❖ **מושאים בסיסיים:** טיפוסים בסיסיים, קלט ופלט
define, const, enum
- ❖ **תנאים:** if, else, switch, operator ?:
- ❖ **לולאות:** while, do-while, for, break, continue
- ❖ **פונקציות:** הזרות, הגדרות (מיושן) ערך החזרה,
העברת פרמטרים by value ובהברה by address
- ❖ **BITS:** בוליניים ומספריים, אופרטורים בוליניים
וזה++ vs. ++זה

חומר רקע לקורס זה (2)

- ❖ **מערכות ומחריזות:** איתחול, השמה, מעבר על המבנה, שליחה כפרמטר ומטריצות.
- ❖ **מצביים:** הצהרה, איתחול, השמה אופרטור * - &, עבודה נכונה (ובתווחה), מצביע ל-void, מצביע לפונקציה
- ❖ **הקצתה זיכרון דינמית:** עבודה בטווחה, הקצאה, שחרור, בדיקת הצלחה
- ❖ **קורסיה**
- ❖ **מבנים:** הצהרה, שימוש.

חומר רקע לקורס זה (3)

❖ לסייע:

כל החומר של הקורס הקודם!

וודאו כי אתם שולטים בכלו באופן
שותף! לפני תחילת קורס זה!

איכות תוכנה - Software Quality

- ❖ הנדסת תוכנה היא יצרת תוכנה איכותית!
- ❖ "איכות" מתוארת על ידי מספר מדדים.
- ❖ מדדים פנימיים (Internal factors) – פנימיים לתוכנה (קריאות, מודולריות וכו'...)
- ❖ מדדים חיצוניים (External factors) – חיצוניים לתוכנה (מהירות פעולה המערכת, קלות השימוש במערכת וכו'...)

למשתמש אכפת רק ממדדים חיצוניים

لمתחזק רקוד זה כבר סיפור אחר...

מדדדי איזות קוד

1. **Correctness** – נכונות. יכולת התוכנה לבצע את המטלה שהוגדרה בש晖לה במפרט בדיקון מוחלט.
2. **Robustness** – עמידות למקרי קיצון. יכולת המערכת להגיב בצורה נכונה לתנאים חריגיים.
3. **Extendibility** – יכולת הרחבה. היכולת להתאים את המערכת לשינויים בדרישות המפרט.
4. **Reusability** – שימוש מחדש. היכולת לשימוש ברכיבי המערכת על מנת לבנות aplikציות אחרות.

מדדי איות קוד משניים

- ❖ **תאימות קוד** – Compatibility. יכולת השילוב בין חלקי הקוד השונים עם אלמנטים אחרים.
- ❖ **יעילות** – Efficiency. מיעוט (עד כמה שניתן) בדרישות משאבי החומרה של המערכת.
- ❖ **ניידות** – Portability. קלות העברת חלקי קוד המערכת לסביבות חומרה או תוכנה אחרות.
- ❖ **הקלות בה שימוש** – Ease of use. יכולם לerneוד להשתמש במערכת.

הבדלים בין שפת C ושפת C++

מצגת נפרדת*

תכנות מונחה עצמים

מהו/a את ההבדל העיקרי בין שפת C
 לשפת +C++!

תכנות פרוצדורלי – שפת C

- ❖ כל מערכת ממוחשבת מכילה עם שני גורמים – **מידע** ו- **אלגוריתם** (תכנות דרך פעולה).
- ❖ שפה פרוצדורלית היא שפה שמדגישה את החלק **האלגוריתמי** בתוכנות.
- ❖ כל תוכנית היא סט של פרוצדורות (או פונקציות) שעלה המחשב לבצע.
- ❖ בכללי, תוכנה מתוכננת, מתוארת ונבנתה תוך שימוש במונחי הפעולות שהיא מבצעת.
- ❖ מדובר בתוכנות **Top-down**.

תכנות מונחה עצמים – שפת C++

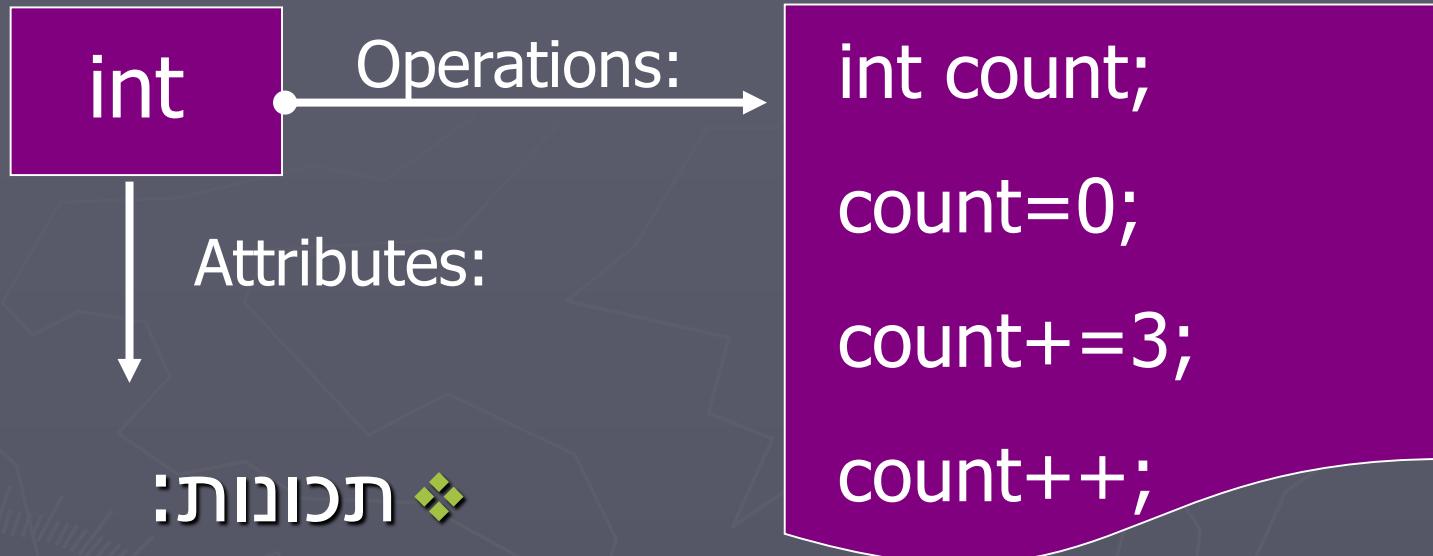
- ❖ **תכנות מונחה עצמים (Object Oriented Programming) –** מתקד בחלק **המידע** במערכת.
- ❖ הרעיון העיקרי הוא לתקן מבני / מחלקות מידע (classes) שייצגו את התכונות העיקריות של הבעייה.
- ❖ **מחלקה (class)** מגדירה בעזרת איזה **מידע מייצגים אובייקט** בתוכנית ואיזה **פעולות** ניתן לבצע על אותו מידע.
- ❖ מדובר בתכנות **Bottom-up**.

תכונות התכנות המונחה עצמים

- ❖ ה**תכונות החשובות ביותר בתכנות מונחה עצמים** הן:

1. Abstraction – (Abstract Data Type) – הפשטה.
2. Encapsulation and Data hiding – **כימוי וסתירת מידע.**
3. Inheritance – ירושא.
4. Polymorphism – **רב צורותיות.**
5. Reusability of code – **שימושיות מחודשת.**

Abstract Data Type



❖ **תכונות:**

- גודל: 4 bytes
- סוג: מספרשלם.

❖ **פעולות:**

- אופרטורים אריתמטיים
- אופרטורים לוגיים
- bitwise
- פעולות קלט / פלט

Abstract Data Type (cont...)

❖ **כיצד נתראר מכונית?**



Window

Door

Wheel

Abstract Data Type (cont...)

```
char*      sCarName;  
int        iNumOfDoors;  
bool       bAutomatic;  
//...
```

❖ בדרכו ה"ישנה":

❖ שאלות:

- ומה אם אנחנו צריכים 100 מכוניות?
- כיצד נתאר את הפעולות: "פתיחה דלת במכונית",
"הנעת מכונית" וכו'

הגדרה - Abstract Data Type

- ❖ ADT הוא מפרט של סט מידע וסט הפעולות שניתן לביצוע על אותו מידע (לדוגמא, מספרים ממשיים, מחסנית, מכונית...).
- ❖ ADT מהויה ומכליל:
 - ❖ 1. טיפוס נתונים חדש.
 - ❖ 2. סט שלפעולות. סט הפעולות זהה מהויה ממשק (interface) לטייפוס החדש.
 - ❖ 3. פעולות המשק הם המנגנון האחד הייחיד המאפשר גישה למבנה טיפוס הנתונים.

מכונית C - ADT

Car

Num_of_Doors;

Year_Of_Manufacturing;

Is_Automatic;

Open(Door);

Open(Window);

TurnOnTheCar();

מidea

פועלות

ADT
=
Car

- ❖ **היכולת לאגד ביחד מידע עם פעולות** – מאפשרת - ליצור טיפוס חדש של נתונים.
 - זה את בדיקת ההגדרה של כימום.
- ❖ **דוגמאות נפוצות ל-ADT:**
 - טיפוסים פרימיטיביים (בנויים בשפה)
 - **Stack, Queue, Tree, Car, Student, Text-editor.**
- ❖ ADT מאפשר להגדיר מאפיינים חיוניים מבלתי להיכנו יותר מיידי לעומק השימוש הפרטני.
- ❖ **המשמעות ב-ADT מספק** רשימת פעולות אפשריות ("**מה אפשר לעשות?**") ולא הגדרותימוש ("**איך אפשר לעשות זאת?**")

הגדרה - Abstract Data Type

- ❖ ADT הוא מפרט של סט מידע וסט הפעולות שנitin לבצע על אותו מידע (לדוגמא, מספרים ממשיים, מחסנית, מכונית...).
- ❖ ADT מהוּה ומכיל:
 - ❖ טיפוס נתונים חדש.
 - ❖ סט שלפעולות. סט הפעולות זהה מהוּה ממשק (interface) לטייפוס החדש.
 - ❖ פעולות המשק הם המנגנון האחד הייחיד המאפשר גישה למבנה טיפוס הנתונים.

כינוח (Encapsulation)



תכנות מונחה עצמים, סמי שמעון, סבטלנה רוזין תשע"ז

מחלקות - Classes

- ❖ בשפת ++ C (וגם Java ו- Python ועוד..) אנו **משתמשים** במחלקות (**classes**) על מנת להגדיר **ADTs**.
- ❖ מחלוקת בשפת ++ C היא היצוג של ישות מידע (טיפוס מוגדר על ידי המשתנה – **user defined type**).
- ❖ Class היא שילוב של יציג מידע ומетодות (**methods**) אשר פונקציות פנימיות של המחלוקת המשמשות לפעול המידע.
- ❖ **אובייקטים (Objects)** הם **מופעים של מחלקות**. כלומר, הקשר בין אובייקטים למחלקות זהה לקשר בין משתנים לטיפוסים.
- ❖ הגדרת מחלוקת לא מקaza זיכרון או אחסון לפחות אובייקט!

מחלקה - Class

- ▶ The syntax:

```
class ClassName
```

```
{
```

```
//attributes and operations
```

```
};
```

Classes & Objects

הקדמה

- ❖ כל מחלקה מכילה:
 - – המידע המאופן. **Data Members**
 - – הפעולות שניתן לבצע על אותו מידע מאופן. **Methods**
- ❖ המוכנית תגדיר / תיצור מגוון **אובייקטים** מהמחלקות הקיימות. ואז היא תוכל להשים ערכיהם ל- **data member** של האובייקטים ולהפעיל את המתודות שלהם.

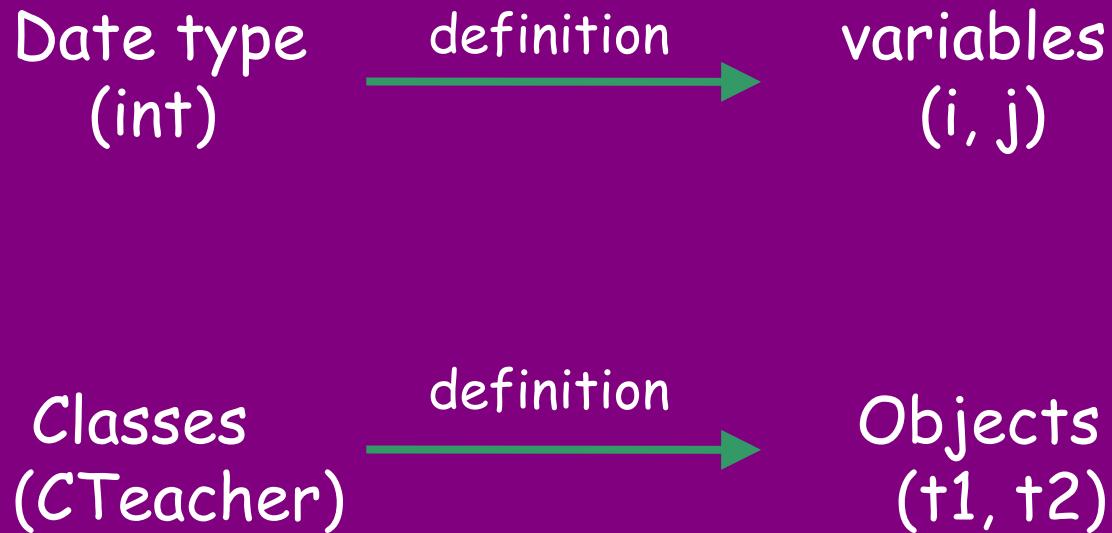
Classes & Objects

- ❖ **מחלקות הם טיפוסים מיידע חדשים!** וכך יש להתייחס אליהם.
- ❖ **מחלקה מסוימת** למכנת ליצור טיפוס חדש ויש להתייחס לטיפוס זה בדיק כפי שמתיחסים לטיפוסים הprimיטיביים הקיימים בשפה: int, float, char, etc.

מה ניתן לבצע עם הטיפוסים הprimיטיביים?



האנלוגיה הבסיסית



Classes & Objects

תזכורת קצרה:

הגדרה ושימוש של מבנים בשפת C:

```
...  
struct CPoint  
{  
    int x;  
    int y;  
};
```

```
int main()  
{  
    struct CPoint P1;  
    P1.x = 5;  
    P1.y = 7;  
  
    return 0;  
}
```

Classes & Objects

הגדלה:

```
class MyClass {  
[private:]  
    variables (data members)  
    ...  
    functions (methods)  
    ...  
  
public:  
    variables (data members)  
    ...  
    functions (methods)  
    ...  
};
```

```
int main() {  
    // define objects of type  
    // class_name  
    MyClass MyObject1;  
    MyClass MyObject2;  
  
    // call a member function  
    MyObject1.func1(...);  
  
    // assign value to data members  
    MyObject1.Index = 12;  
    return 0;
```

Classes & Objects

דוגמא - מחלקה Point

מחלקה Point מייצגת נקודה דו מימדית ♦

```
class Point {  
public:  
    int m_x, m_y;  
  
void show() {  
    cout<<"x = "<<m_x<<", y = "<<m_y<<endl;  
}  
};
```

```
#include <iostream>  
int main() {  
    Point p1, p2;  
    p1.m_x = 15;  
    p1.m_y = 10;  
    p1.show();  
    cout<<"Please enter x & y values: "  
    cin>>p2.m_x>>p2.m_y;  
    p2.show();  
  
    return 0;  
}
```

Classes & Objects - דוגמא

- ❖ כיצד נכתבת מחלקה המייצגת סטודנט?
 - ❖ איזה מידע אנו שומרים עבור כל סטודנט?
 - ❖ איך פועלות נctrar לבצע על המידע זהה?
-
- ❖ כתבו את המחלקה.

משתני מחלוקת – Data member

משתני מחלוקת או משתנים מקומיים

- ❖ מתודות מכירות ו"יודעת" את כל משתני המחלוקת של האובייקט שהפעיל אותם.
- ❖ כמספריים מתודה על אובייקט מסוים יש למתודה יכולת לגשת ולשנות את כל משתני המחלוקת של אותו אובייקט (בלי שהם נשלחים כפרמטר!)
- ❖ בנגד למשתנים מקומיים (שחינם רק עד סוף הבלוק בו הוצהרו!) משתני מחלוקת חיים כל עוד האובייקט שבו הם קיימים חי.

כלל אבעז 1:

אם אתם כל הזמן שולחים את אותו משתנה לכל (או רוב) המתודות של המחלוקת, זה היא אינדיקציה טובה שיתכן כי המשתנה צריך להיות **משתנה מחלוקת!**

הסתרת מידע - Information Hiding

- ❖ משתמשי המחלקה יכולים לראות בבחירה במאה הם יכולים להשתמש וממה להתעלם.
- ❖ נועד להבטיח כי משתמשי המחלקה לא יהיו תלויים בימוש ספציפי של המחקה העולן בעתיד להשתנות!
- ❖ **public** = כל חברי המחלקה (משתנים וمتודות) שוגדים כר **נגישים שלכולם!** מדובר במקרה (interface) המחלקה.
- ❖ **private** = אף אחד לא יכול לגשת או להפעיל את חברי המחלקה שוגדים כר מלבד חברי המחלקה האחרים! ז"א נגישים רק **לחבריו המחלקה עצמה**. מהו זה מحسום (קיר חוץ) בין האובייקט והמשתמש (**כימוס**).

משתני מחלוקת - Data Members

Private and Public Permissions

- ❖ **חברי מחלוקת (משתנים ומетодות)** יכולים להיות **public** או **private**.
- ❖ **לחברי מחלוקת **public** (פומביים)** יש נגישות בכל חלקו הקוד (דרך האובייקט שבו הם מוכלים, ובתנאי שיש גישה אליו!).
- ❖ **לחברי מחלוקת **private** (פרטיים)** ניתן לגשת אך ורק דרך **METHODS** של אותה מחלוקת!
- ❖ **public** ו- **private** הן מילים שמורות שברגע שכותבים אחת מהן בהגדרת המחלוקת כל חברה המחלוקת שמופיעים אחרת מקבלים את הרשאות הגישה שהיא קובעת.
- ❖ **ההרשאה הדיפולטית** במחלוקת (אם לא כתבנו אף מילה) היא **private**.
- ❖ **ניתן לשנות את הרשאות כמו פעים שרצוים במחלוקת, אך נהוג רק פעם אחת (וקן נפהג).**

Private vs. Public

Example - Point2

```
#include <iostream>
int main() {
    Point p1;
    //p1.m_x = 15; ----- ERROR
    p1.set_x(15);
    //p1.m_y = 10; ----- ERROR
    p1.set_y(10);
    p1.show();
    p1.set_x(17);
    p1.set_y(5);

    cout<<"x= "<<p1.get_x()<<" y= "<<p1.get_y()<<endl;

    return 0;
}
```

```
class Point  {
public:
    void set_x(int x) {m_x = x;}
    void set_y(int y) {m_y = y;}
    int get_x() {return m_x;}
    int get_y() {return m_y;}
    void show() {
        cout<<"x = "<<m_x<<, y = "<<m_y<<endl;
    }
private:
    int m_x, m_y;
};
```

חזרה לדוגמת הסטודנט

- ❖ איך נשנה את המימוש הקודם שלנו לחלוקת סטודנט כרך שתתאים למידע החדש של מדנו?
- ❖ כיצד נבעץ פה הסתרת מידע?
- ❖ אילו פעולות (METHODS) נוספות יש להוסיף לחלוקת?

משתני מחלוקת פרטיים

מודע לשימוש ב-private

- ❖ מונע גישה ישירה למשתני המחלוקת על ידי מתכנתים המשתמשים בחלוקת.
- ❖ חשוב לשמר על כימום, מודולריות, ובטיחות!
- ❖ חוק חשוב:
 - ❖ חייב להיות סיבה ממש טובה על מנת להגיד משתנה מחלוקת כ-
public.
 - ❖ בדרך כלל אין זאת סיבה!!!
- ❖ אז, איך ניגש למשתני מחלוקת פרטיים?
 - ❖ методות get ו-set

Classes vs. Structs

Formal

- ▶ A struct is a class whose default permission is **public** (reminder: the default permission in a class is private).
- ▶ Thus:
 - struct { private: ... } is equivalent to class { ... }
 - class { public: ... } is equivalent to struct { ... }

In Practice

- ▶ The use of structs in C++ is scarce.
- ▶ They will be mainly used to define data collections with no methods (very similar to their usage in C).

Methods (Member Functions)

METHODS VS FUNCTIONS (GLOBALE)

- ❖ **METHODS (member functions)** מייצגות את השירותים שהמחלקה מציעה למשתמשים בה. המethodות פועלות על משתני המחלקה של האובייקט ומופעלות אף דרך האובייקט!
 - ❖ **INITIALISATION**.
 - ❖ **MODIFICATIONS**.
- ❖ **FUNCTIONS (GLOBALE)** מיעדות למטרות כלליות ולא נמצאות תחת השירותים של אף מחלקה.
 - ❖ **EXAMPLE?**

~~void Increment(Point P, int j)~~

~~{ ... }~~

GENERAL: אם אתם חייבים לשלוח אובייקט לפונקציה גלובלית והוא צריכה גישה כמעט לכל האובייקט, ופעלת על חלקים ממנו, זהה אינדיקציה טובה שפונקציה זאת בעצם צריכה להראות מетодה של המחלקה!

שימוש חיצוני של מתחדשות

Example - Point2

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

```
void Point::set_x(int x) {m_x = x;}
void Point::set_y(int y) {m_y = y;}
```

```
int Point::get_x() {return m_x;}
int Point::get_y() {return m_y;}
```

```
void Point::show() {
    cout<<"x = "<<m_x<<, y = "<<m_y<<endl;
}
```

```
class Point  {
public:
    void set_x(int x);
    void set_y(int y);
    int get_x();
    int get_y();
    void show();
};
```

```
private:
    int m_x, m_y;
```

Methods (Member Functions)

שימוש חיצוני של מетодות

- ❖ מетодות שמשומשות בתור הגדרת המחלקה מוגדרת אוטומטית כ-**inline functions**.
- ❖ רק פונקציות פשוטות ולא מסובכות.
- ❖ ללא לולאות, `switch`, קריאות לפונקציות אחרות וכדומה!
- ❖ רוב המethodות ימודשו חיצונית להגדרת המחלקה. בהגדרת המחלקה יצינו רק הצהרות המethodות.
- ❖ מmethodה שמשומשת חיצונית חייבת להופיע בשם המלא. כלומר – חייב להופיע גם שם המחלקה שזה היה ממשמשת אופרטור השיכות ":".

Methods (Member Functions)

עובדת עם מספר קבצים

- ❖ כל מחלקה תחומרש בשני קבצים:
- ❖ קובץ כותרות: (.h) בו יופיעו הגדרת המחלקה, משתני המחלקה והצהרות המתודות.
- ❖ קובץ מימוש: (.cpp) בו ימומשו כל המתודות

point.h

```
class Point
{
    int m_x, m_y;

public:
    void Init();
    bool Set(int ax, int ay);
    void Print();

    int GetX() { return m_x; }
    int GetY() { return m_y; }
};
```

point.cpp

```
#include "point.h"

void Point::Init()
{
    ...
}

bool Point::Set(int ax, int ay)
{
    ...
}

...
```

main.cpp

```
#include "point.h"
#include "crectangle.h"
...

int main()
{
    Point P1;
    ...
}
```

תכנות מונחה עצמים, סמי שמעון, בבלונה רוסין תשע"ז

שאלות?

תכנות מונחה עצמים, סמי שמעון , סבטלנה רוסין תשע"ז