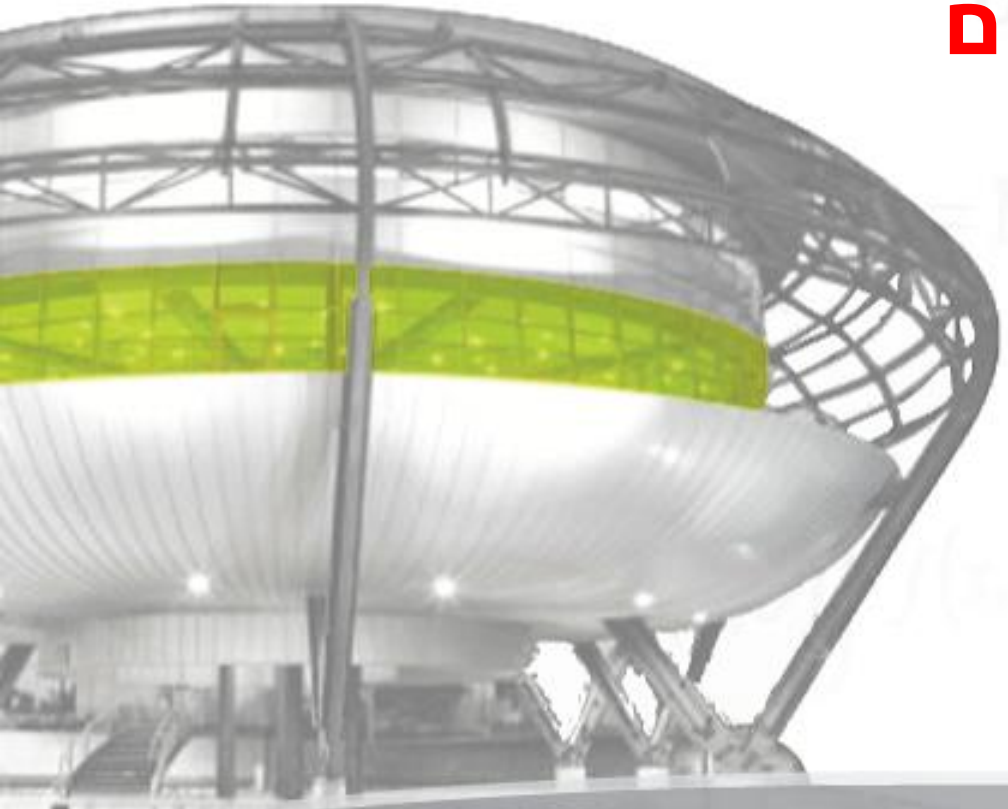

תכנות מונחה עצמים

סמסטר קיץ

מעבדה 2

sce
המכללה האקדמית להנדסה סמי שמעון

תתחבר לרעיונות גדולים
באר שבע | אשדוד | www.sce.ac.il | מהנדס

Class(מחלקה)

- מחלקה היא אבטיפוס עבור טיפוס נתונים.
- מחלקה מכילה:

Data Members – המידע המאופסן.

Methods – הפעולות שניתן לבצע על אותו מידע מאופסן.

- אובייקטים הם המופעים של המחלקות כלומר הקשר בין אובייקטים למחלקות זהה לקשר בין משתנים לטיפוסים.

דוגמא - (מחלקה)Class

```
#include <iostream>
using namespace std;

class Point {
public:
    int x, y;
    void print() {
        cout << "x = " << x << ", y = " << y << endl;
    }
};

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

עבודה עם מספר קבצים

1. מתודות שממומשות בתוך הגדרת המחלקה מוגדרות אוטומטית כ-inline functions.

- רק פונקציות פשוטות ולא מסובכות.
- ללא לולאות, switch, קריאות לפונקציות אחרות וכדומה!

2. רוב המתודות ימומשו חיצונית להגדרת המחלקה. בהגדרת המחלקה יצוינו רק הצהרות המתודות.

3. מתודה שממומשת חיצונית חייבת להופיע בשמה המלא. כלומר – חייב להופיע שם המחלקה ואופרטור השייכות "::".

עבודה עם מספר קבצים

- כל מחלקה תמומש בשני קבצים:
- קובץ כותרות: (.h)header file בו יופיעו הגדרת המחלקה, משתני המחלקה והצהרות המתודות.
- קובץ מימוש: (.cpp)code file בו ימומשו כל המתודות

עבודה עם מספר קבצים

Point.h

```
class Point {
public:
    void setX(int x);
    void setY(int y);
    int getX();
    int getY();
    void print();
private:
    int x, y;
};
```

Point.cpp

```
#include <iostream>
#include "Point.h"
using namespace std;

void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
int Point::getX() { return x; }
int Point::getY() { return y; }
void Point::print() {
    cout << "x = " << x << ", y = " << y << endl;
}
```

main.cpp

```
#include "point.h"

int main()
{
    Point p1;
    p1.print();
    return 0;
}
```

פעולת ה-include

- פעולת ה- include היא פקודת קדם-מעבד (preprocessor) אשר מבצעת השמה של תוכן הקובץ שאותו כללנו בפקודה במקום כל פקודת include.
- במידה ונבצע include לקובץ מסוים יותר מפעם אחת נקבל שגיאה של redefinition מאחר והקומפיילר רואה את ההצהרה יותר מפעם אחת.

הידור מותנה

- ראינו בעבר את הפקודה #define לצורך הגדרת קבוע מסוים
- פקודה זו מוסיפה את הקבוע שהוגדר לטבלת סימולים של התוכנית במידה וטרם הוגדר. במידה וכבר הוגדר דורסת את ערכו.
- ניתן גם לכתוב פקודת define ללא ערך, רק כדי להכניס קבוע מסוים לטבלת הסימולים
- ניתן לבדוק האם קבוע מסוים הוגדר בטבלת הסימולים בעזרת הפקודה #ifdef או אם לא הוגדר בעזרת הפקודה #ifndef

  – במידה והתנאי מתקיים, הקופיילר יהדר את קטע הקוד הבא עד אשר יתקל ב- #endif

Data hiding

- הסתרת נתונים היא אחד המאפיינים החשובים של תכנות מונחה עצמים המאפשר למנוע לגשת ישירות לייצוג הפנימי של המחלקה.
- הגבלת הגישה לחברי המחלקה (משתנים ומתודות) נקבעת על ידי הקטעים הציבוריים, הפרטיים והמוגנים שבתוך גוף המחלקה.
- חברי מחלקה(משתנים ומתודות) יכולים להיות public או private.
- public – חברי מחלקה המוגדרים כך נגישים מכל מקום.
- private – חברי מחלקה המוגדרים כך נגישים רק בתוך המחלקה עצמה. אף אחד לא יכול לגשת או להפעיל את חברי המחלקה המוגדרים כך מלבד חברי מחלקה אחרים.
- הרשאת ברירת המחדל במחלקה היא private.

Private attributes

- מונע גישה ישירה למשתני המחלקה על ידי מתכנתים המשתמשים במחלקה.
- חשוב לשמירה על כימוס, מודולריות, ובטיחות!
- הגישה למשתני המחלקה תתבצע ע"י setters ו-getters.

constructor

- מתודת הבנאי(constructor) היא מתודה המאתחלת את ה-attributes של הבנאי בעת יצירתו.
- ה-C’tor נקראת אוטומטית בזמן שהאובייקט נוצר!
- ה-C’tor מבטיח אתחול של כל אובייקט (ממחלקה זאת).
- שם מתודת ה-C’tor הוא שם המחלקה.
- מתודת ה-C’tor יכולה לקבל פרמטרים.
- מתודת ה-C’tor לא יכולה להחזיר ערכים. ואסור לכתוב שהיא מחזירה void(שגיאת קומפילציה).

Default Constructor

- C’tor ברירת המחדל הוא הבנאי שנקרא בלי פרמטרים.
- אם ורק אם אין C’tor למחלקה הקומפיילר ייצר אחד אוטומטית למחלקה אשר יאתחל את תכונות האובייקט עם זבל.
- ביצירת מערך של אובייקטים חייבים שיהיה בנאי ברירת מחדל למחלקה.

Default Constructor

Point.h

```cpp
#ifndef __POINT_H
#define __POINT_H

class Point {
public:
    void setX(int x);
    void setY(int y);
    int getX();
    int getY();
    Point();
    void print();
private:
    int x, y;
};
#endif //__POINT_H
```

Point.cpp

```cpp
#include "Point.h"
using namespace std;

void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
Point::Point() {
    x = 0;
    y = 0;
}
void Point::print() {
    cout << "x = " << x << ", y = " << y << endl;
}
```

main.cpp

```cpp
#include "point.h"

int main()
{
    Point p1;
    p1.print();
    return 0;
}
```

Constructor

- מאחר וה-C’tor הוא מתודה ניתן לייצר כמה C’tor תוך שימוש בהעמסת אופרטורים

Point.h

```
#ifndef __POINT_H
#define __POINT_H

class Point {
public:
    void setX(int x);
    void setY(int y);
    int getX();
    int getY();
    Point();
    void print();
private:
    int x, y;
};
#endif //__POINT_H
```

Point.cpp

```
#include <iostream>
#include "Point.h"
using namespace std;

void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
Point::Point() {
    x = 0;
    y = 0;
}
Point::Point(int m_x, int m_y) {
    x = m_y;
    y = m_y;
}
void Point::print() {
    cout << "x = " << x << ", y = " << y << endl;
}
```

main.cpp

```
#include "point.h"

int main()
{
    Point p1;
    p1.print();
    return 0;
}
```

Copy Constructor

- copy C’tor הוא מקרה פרטי של בנאי שהפרמטר שהוא מקבל הוא אובייקט אחר מאותו הטיפוס.
- מטרתו לייצר אובייקט נוסף זהה לאובייקט שהתקבל כפרמטר
- הקומפיילר מספק לנו copy C’tor אשר מבצע shallow copy(העתקה רדודה).

Copy Constructor

- בשליחת אובייקט by value נוצר אובייקט חדש כהעתק של האובייקט המקורי.
- בהעברה by value הקומפיילר מניח שאנחנו רוצים העתקה רדודה של bit bit(bitcopy).
- הבעיה הינה מצביעים כיוון שתוכנם הוא כתובת של משתנה אחר.
- בהעתקת bitcopy מועתק התוכן שלהם = הכתובת.
- גם האובייקט המקורי וגם ההעתק מצביעים על אותו מקום בדיוק!

Copy Constructor

כלומר בהעתקת אובייקט המכיל שדות שהן מצביעים נקבל מצב של הצבעה כפולה(dual pointing):

- שינוי מידע באובייקט אחד – ישנה את המידע בכל האובייקטים המועתקים(פוגע בכל הרעיון של העברה by value).
- כאשר האובייקט מת – השדות שלו משתחררים(D’tor) מה שגורם לכל ההעתקים האחרים לאבד את המידע.
- לכן יש צורך לממש בנאי העתקה(copy C’tor) שיעתיק את האובייקט כפי שצריך בהעתקה עמוקה.
- כך נמנע מהקומפיילר להעתיק תוך שימוש ב-bitcopy.

Copy Constructor

Point.h

```
#ifndef __POINT_H
#define __POINT_H

class Point {
public:
    void setX(int x);
    void setY(int y);
    int getX();
    int getY();
    Point();
    Point(int m_x, int m_y);
    Point(const Point& other
    void print();
private:
    int x, y;
};
#endif //__POINT_H
```

Point.cpp

```
#include "Point.h"
using namespace std;

void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
Point::Point() : Point(0, 0) {
}
Point::Point(int m_x, int m_y) {
    x = m_x;
    y = m_y;
}
Point::Point(const Point& other) {
    x = other.x;
    y = other.y;
}
void Point::print() {
    cout << "x = " << x << ", y = " << y << endl;
}
```

main.cpp

```
#include "point.h"

int main()
{
    Point p1;
    p1.print();
    return 0;
}
```

Constructor Delegation

- ניתן לקרוא מבנאי אחד לבנאי אחר, ובכך לחסוך את שורות הקוד המאתחלות את שדות האובייקט ומרכזים את כל האתחולים.

Point.h

```
#ifndef __POINT_H
#define __POINT_H

class Point {
public:
    void setX(int x);
    void setY(int y);
    int getX();
    int getY();
    Point();
    void print();
private:
    int x, y;
};
#endif //__POINT_H
```

Point.cpp

```
#include <iostream>
#include "Point.h"
using namespace std;

void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
Point::Point() : Point(0, 0) {
}
Point::Point(int m_x, int m_y) {
    x = m_x;
    y = m_y;
}
void Point::print() {
    cout << "x = " << x << ", y = " << y << endl;
}
```

main.cpp

```
#include "point.h"

int main()
{
    Point p1;
    p1.print();
    return 0;
}
```

Destructor

- ה-D’tor הוא מתודה שאחראית לבצע פעולות בדיוק לפני שהאובייקט נהרס סופית.
- נקרא אוטומטית ע”י הקומפיילר כשה-lifetime של האובייקט מסתיים.
- קיים D’tor ברירת מחדל בכל מחלקה שאינו עושה דבר ואותו ניתן לדרוס.
- לכל מחלקה יש רק D’tor אחד! – אין העמסת פונקציות
- שם מתודת ההריסה(D’tor) היא ~ClassName()
- ל-D’tor אין פרמטרים! משום שאין לו אופציות לבחור מהן.
- ל- D’tor אין ערך מוחזר.

Destructor

Point.h

```cpp
#ifndef __POINT_H
#define __POINT_H

class Point {
public:
    void setX(int x);
    void setY(int y);
    int getX();
    int getY();
    Point();
    Point(int m_x, int m_y);
    ~Point();
    void print();
private:
    int x, y;
};
#endif //__POINT_H
```

Point.cpp

```cpp
#include <iostream>
#include "Point.h"
using namespace std;

void Point::setX(int m_x) { x = m_x; }
void Point::setY(int m_y) { y = m_y; }
Point::Point() : Point(0, 0) {
}
Point::Point(int m_x, int m_y) {
    x = m_x;
    y = m_y;
}
Point::~Point() {
    cout << "Deleting a point" << endl;
}
void Point::print() {
    cout << "x = " << x << ", y = " << y << endl;
}
```

main.cpp

```cpp
#include "point.h"

int main()
{
    Point p1;
    p1.print();
    return 0;
}
```

Destructor

- ה-D’tor קריטי לניקוי מוצלח של כל הזיכרון.
- מה שנוצר סטטית – על המחסנית, משוחרר על ידי הקומפיילר, אך כל דבר מסובך יותר דורש הוספת קוד ל-D’tor.
  - שחרור זיכרון דינמי
  - סגירת קבצים(או משאבים אחרים semaphore, socket וכו’).