

סמסטר "ב" תשס"ו

תרגולים בקורס

C++

מ-1 עד 7





- (1) כניסוס, רצ צורתיות - Encapsulation. (המימוש הפנימי נכרך לחילת מוחבא ע"ה זכרון לא משמש בו.) התנהגות האובייקט תלויה באינסוס.
- (2) ירושה - עקריות מחדש קיים ומיקצויה. נקרא "ע"ג של".
- (3) הסה - המוצר מכיל אובייקט אחר.

תוכנית "סוג"

```
#include <iostream>
```

⊖ קרובל iosream האבלים שמש

```
int main() {
```

⊖ בקור ופלא. אין צורך לכתוב 'h'.

```
std::cout << "Hello, world!\n";
```

⊖ האבסאור < - העקבת "שמת".

```
return 0;
```

⊖ אין צורך להגדיר את האבסאור של

```
}
```

מה שישפסם.

⊖ std::cout - אובייקט שמגדיר צגצרייה.

הגדרת אובייקט נקראת מתוקה - 'class'. בתוך ה-class אין מבדדים חלת הפס' של האובייקט ולמחנים פנימיים (צווגה ל-struct בלמת א).
בכתיבת : private - מפורט במימוש פנימי.

חס לא להצטרף יום מפורט ב- public א ב- private, ב-class חלומטת התמק היה
private וב- struct יהיה אטומטת - public.

בונ push ו- pop

pop - מוציאה מחדש מהמחסנית ולכן היא מתנהגת int -

push - מכניסה לאחסנית ולכן היא מקבלת int .

⊖ אונתני מקפריים: (1) מוסך (2) מצבים (3) קרובל המחסנית.
⊖ קיים מיפ דרך מתחר
⊖ מודל מן אונתני המצבים.

⊖ '::' - המלפנה שמיני נמשאלו - בתוך המתקה לשמשאל'.

⊖ האוסאור 'new' צפי הקצויה ניכרון ציונמית.

```
int * p = new int;
```

```
int * p_arr = new int[size];
```

delete;
⊖ מתחר צף ניכרון (destructor):

```
delete[] p_arr;   
⊖ ארס ח להפסם בלמחר צכרון ל int מוסך int-p.
```

if (-top-index == -size) - אם גודל הסטק הוא 0 (⊗)
 #include "stack.h" - הוספת הריבוי של הסטק (⊖)
 using namespace std - std - שם המרחב -
 cout << std::cout << endl



תרגיל 2

שק" 4

הפונ' ApplyToAll() יודעת לדפדף על הפונקציה ולתקן את הפונקציה אפ' שניתנו נמצך ב פנס לתקן איהם.

שק" 5: מציג אפונ': זהו לאו נהיה מקובץ אפונ' מסוימת, נטור פנייה אפונ'.
כך נוס להקדיר פנ' אפונ', כלש פנס נש לאורו אוריה פונ' צורת צר הארומט'ים.
ApplyToAll (func)

הצגת מצביע אפונ'

אמל, פנ' פנ' (int f), ה מצביע יהיה: int (*p)()
ארתול/השמה: זנעקוי (int idler), p = idler
* לא ניתן ליש פנ' שמתיה א double, אמל למצביע אפונ' שמתיה int
כ מצבוי קפי סוגם פנ' מצביעם.

שק" 8:

הקבלה ילטנו pow-p = pow

ותן התיבה: pow-p (1.2, 2.3) שנה אכתוב pow(1.2, 2.3)

פנ' שמתיה אפונ' אפונ'

הינו צריכים אמתוב:

int (*f)();

f היא פונ' שמתיה אפונ' אפונ' (או מקבלת פנ', מתיה int).

התיבה מסוגם ומסורת. לכן, נלטנו ה- typedef.

typedef int (*int_p)(); 2/3:

פנ' שמתיה מצביע אפנ': int_p f();
(int_p)

שק" 12

הפונ' ApplyToAll תפכין חן אכה א ב הפונקציה א פנס נוס לתקן א
להיג נתן אנתם פנרה פנ'. הימנוש יראה פחות מו יתרי:
פנ' אחיל.

for ()
grade = f(grade);

14 שנה

בתוך הספרייה הסטנדרטית יש class שנקרא String והוא יכול לבצע את
כל הפעולות שהספרייה שלמה עושה - אחרונות (יש קצת שינויים בגישה)
יש משינוי אחרון - כפי שהתוונת מוגדרת (יש לו, אין לו) ..
כיצד מלמדים בנק?

15 שנה

הבסיסים > #include <string>

- איש מ: string s1 - המחרוזת היא ריקה.

- ניתן לבצע שינוי של מחרוזת, לדוגמה: s2 = s2 + " " + s3;

3. נניח נוסדה של שרשרת: "learned" = s1 + " - s1 תשוקה בסוף המחרוזת learned.

18 שנה

בהתחלה אנו לא מודעים ל constructor שם בסוף ונקרא לה - default constructor

הפונקציה Widget()

- הפונקציה הראשונה בתוכנית היא main. אין צורך לקרוא לו.

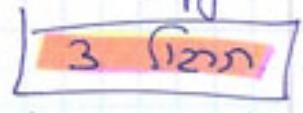
* יש למנוע מוסבר זאת תמיד - #include, #ifdef



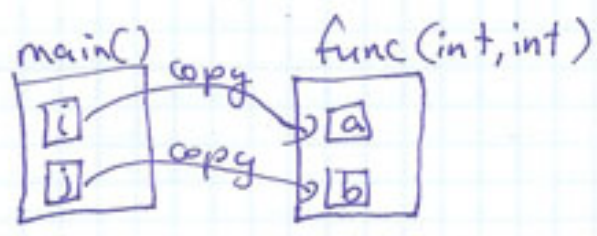
```
string s1, s2; *
s1 = s2;
```

File
Print, Save, Open : menu *
ניתן לפלוס item
למרות זאת המשתמש קורא - item למיין פונקציה,
* את המשתמש בקצה. אם המשתמש יזכר לפרטור ^{menu} (וסר, "פתח קובץ)

90) menu
Edit
Copy



העברה by value



ההבדל: ההעברה לעומת זה נטן. הסיבות שהיא קר: הנתון פנימי.
ההבדל בין ++C: בזמן שם שלמה לאלו משתנה. הדבר נקרא Reference

```
int &a = ...  
double &b = ...
```

אנו חייבים לעמוד בתנאים - reference. האתחול חייב להיות עם משתנה
מאז C++ למעשה ה-reference הוא גם נתון למשתנה.

הסיבה של a הוא reference ...
למעשה שם נלקח בפונקציה אם שנינו קיים:

- (1) לא ניתן לשנות את ההעברה.
- (2) Syntax שונה.
- (3) אין reference להיות NULL.

Const

```
void func(const string& s);
```

העברה מבלי להעביר את הפונקציה אליו
(ללא שם) נוסף להחזרת אליו (אם).



inline

- יש פסולת עם define
- בדרך כלל לא מומלץ
- קודי קריאה - מנוחה
- debugging עם פסולת

* עדיף להשתמש ב- #define במקום פונקציה קטנה.
 - היתרון: inline. ביצוע החלפה ישירות ב- compiler ולא ב-preprocessor.
 (כמו #define)

היתרון של inline הוא שיש לו גודל קטן יותר, אך זהו ה- compiler שצריך להחליט.
 לרוב, וכן אין סכסוך את הקריאה אפוא.
מחיר לא נמוך מדי?

inline היא קטנה. מחיר לא נמוך מדי?
 * ~~יש~~ אם הפונקציה היא קטנה יחסית.

* פונקציה קטנה

inline int A() : Syntax

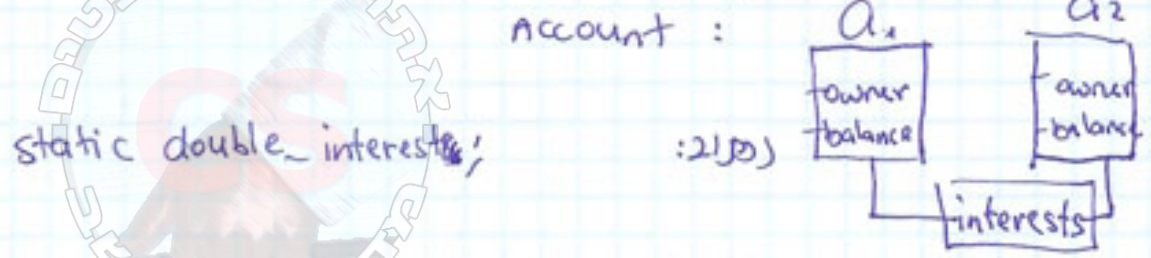
למשל - את הפונקציה inline ~~יש~~ Header - פונקציה
 * בדרך כלל רצוי להשתמש ב- constructors - destructors → inline.
Syntax (ולו) - פונקציה - inline:

&public:

```
string name() { return _name; }
```

Static Members

static member - כל מה שהאובייקטים יודעים עליו הוא שיש לו את המשתנה static member זהו - 23 47e



את כמות static, ולו רק ההפרדה והחלוקה. אומרים שיש להגדיר

double account::_interests; איתו, לכן אם צריך להגדיר אותו:
 אתר הסטודנטים - <http://cs.haifa.ac.il/students/> (class - static)

- אם לא מפרטים את השמות, הודא נא לתת מילויים 0-5.
- זהו שמות לב פה, אך יש לך נוספת לזמניו:

```
Account:: _ interest;
```

כי היא שייכת ל-class ויש לה אובייקט אחד.

~~הזמנה: הודא נא לתת מילויים~~

```
cout << a._ interest;
cout << Account::_ interest;
```

Default Arguments Values

ההצעה להפוך ל'אובייקט' של ה-`is_leap` הפכה ל-`bool` a.cpp
`bool is_leap(int year, int base = 10, char caldr = 'G');`
 הפסקת הקריאה של ה-`main` היא 2, כלומר `main` יומן ל-`is_leap`:

main.cpp
~~`is_leap(int y)`~~

לכן, רצוי לכתוב את ההצהרה ב-`Header`.

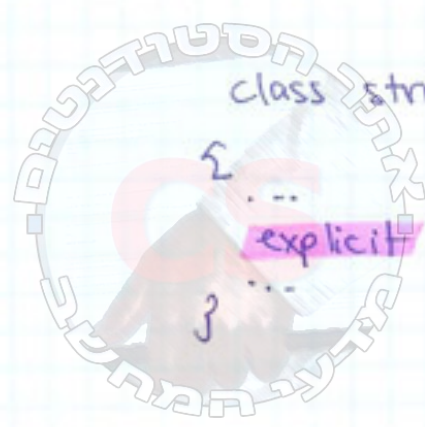
משתנים דינמיים - הקצאת זיכרון

משתנה דינמי הינו משתנה אשר מקצה לו מקום בזיכרון בזמן ריצת התוכנית.
 הצורה להקצות מקום ולשחררו היא `delete + new`

```
int * dynamic = new int[5]; // 2/3
```

ישנונה - constructor

- (1) מנתח משתנים
 - (2) הקצאת זיכרון דינמי
 - (3) העדת אירוס - 3/2:
- ישנונה השמורה: `explicit`:



```
class string
{
...
explicit String(const char*s)
...
}
```


(4) בג'י המספרה.

```
String::String(const String &other)
{
    if(other.str)
    {
        str = new char[strlen(s)+1]; //actual allocation
        strcpy(str, other.str);
    }
    else
        str = 0;
}
```

המפעלה תמיד באופן הסא:

```
String S2(S1);
String S3 (String ("Goodbye world"));
String S4 = S3;
```

Const

המשתנה בג'י כתיבת Const לא שונה מאותה [חיות קטנה] ואם לא שונה
ג'י או 2017 (const object) ג'י לנעלה (רק באופן שונה) 2017
קצרות.

```
const char* get-name() const; //2/3
```

using namespace std;

ג'י/ג'י standard library -> ג'י/ג'י

ג'י/ג'י

```
#ifndef STACK_H
class Stack
public:
    Stack(int size);
    ~Stack();
    int pop();
    bool push(int el);
#endif
```

```
private:
    int* _contents;
    int _size;
    int _pop_index;
};
```

* this - יזנה לב האובייקט הנתון שלו לא לשדה מסוים.

לדוג': אם אנו רוצים לקחת אובייקט אחר ולהתייחס אליו כאובייקט (וכתיב):

* ה-const מאמתה rhs. { string::copy (const string & rhs)

א ים אבדוק שאנחנו לא מנסים delete [] contents;

להתייחס אל שדה (string s) זהו זה.

להתייחס אל this - זהו זה. ~~להתייחס אל~~ ~~להתייחס אל~~ ~~להתייחס אל~~ ~~להתייחס אל~~

ה- this הוא למעשה מופיע ל- class.

- ניתן לבצע delete [] רק על משתנים שהוקצו באופן דינמי.

Constant Values

- ב-C ה-const לא כולל הפירוט בתוכנית. ב-C++ יש משתנה ל- reference const.

לדוג': להוסיף משתנה מה- compiler לא יאפשר לנו לשנות אותו.

const int i=5;

i=10; ← אסור

* const pointer - pointer לא ניתן לשנות או מצביע לשדה שלו ניתן לשנות.

איתנו. הנופשים p הוא קבוע - char* const p

char* const

מה של p מצביע עליו הוא קבוע, - const char* p

למען, זה ניתן לשנות את מהתוכן שעליו מופיע p בקוד p.

* לא ניתן לבצע השגה של פונקצי const אפילו רב.

* כשליצרים const תיבנים למעלה איתנו.

ניתן לקחת זכרים ותחתונים למשתנים של ה-class בצורה כזו:

Car:Car (int serial int color): serial (serial), color (color) {}

מקום constructor + שם, ורק constructor.

* אם אנחנו משתנה זקוקי const, תיבנים למעלה איתנו. ה-class לא תיבנים

למעלה איתנו. מהו יאמתה ב- initialization list בconstructor.

* יש חשיבות רבה למשתנים ב- initialization list.

* האמתה של משתנה static const יתבצע באמתה ל- static.

const car c, serial=0;

standard conversion - הפיכת פרמטרים של הקריאה לפרמטרים אחרים.

User-defined conversions: $7 \rightarrow 7 + 0.2i$ (הפיכת מספר שלם למרוכב).

Conversions כיצד נוסף למדריך?

```
class Complex {
    Complex(double re);
}
```

(1) ע"י מתנה. לוגי:

Best match

הפעם התקתה לא תהיה יותר זרימה אחרת ולפי חתם ~~סיני אתר שמוא~~ ~~סיני פדוסה~~ יורר C וכל מה שמהיר.

המרת C יפוס: אם אנו מבירים המרת C יפוס מתוך class C? פונ' operator... ההמרה ההמרה תתבצע בצורה לפונקציה. אחת ההמרה תתבצע כימה של המושל C יפוס, וההמרה לתתבצע תהיה לוא דנוא זה שרצינו.

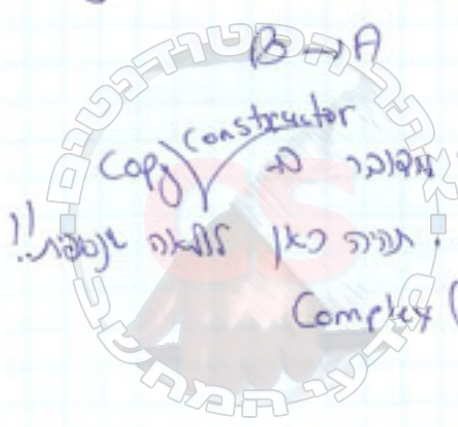
ללא המרה ~~המרה~~ $(double) a$

```
operator double(const); // המרה
(double) a - האויקל שטקטנטנה
```

למי אפשרות זה המרת C יפוס לא ניתן למעט את לשפון יתה

```
class A {
    A(B b)
}
class B {
    operator A();
}
```

$B \rightarrow A$



Complex(complex c) - אסור לבצע

Complex(const Complex &c); ופ"ה לפסקי אתר C Complex (copy constructor) תהיה כאן וללא תוספת! (באשר אין reference) לך, נכונג.

Complex - כשאנו רוצים לתת גישה private לפרטי של class

החברים החברים תחביר בקוד ה-class (היא לא public ולא private).

ניתן להכריז class עם שמה "חבר".

החברה היא לא אובייקט ולא סימבול.

```
void stop() {
```

```
    A a;
```

```
    a.real;
```

:213

}

Compiler-generated Methods

Copy Constructor

יחסי עם class - class

int* הפעולה: (א)בנייה ויש בניינים. עם זה.

תחביר by-value

Assignment operator

הענה מ-copy constructor, היא רצום צריך לעדכן את הפרטים והכלים שלהם.

יש שאלות בהם עם בניינים.



בדגם נראה להבדיל Copy constructor - Assignment Operator

למשל.

בנייה - Copy constructor :

```
string(const string & rhs){
```

```
    cnts = new char[rhs.len];
```

```
    len = rhs.len;
```

```
    strcpy(cnts, rhs.cnts);
```



= Assignment Operator -8 1023

```
String& operator = (const string& rhs) {  
    delete [] cnts;  
    cnts = new char [rhs.len];  
    len = rhs.len;  
    strcpy(cnts, rhs.cnts);  
}
```

if (this == &rhs) } (זהו קפני)
return; } self-Assignment

```
class String {  
    private:  
        char* cnts;  
        int len;  
}
```



תרגול 6 - אופרטורים

- אופרטור זוגי - אופרטור שמקבל שני אופרנדים.
- אופרטור יחיד - אופרטור שמקבל אופרנד אחד.

- overloadable - מיושם - אופרטורים לא ניתנים לכתוב מחדש. מקדים

- האופרטור יורש מכללם.

שק"ב

מרתמת האופרטורים חייבים להיות בתוך class.

אם באופן צפוי, קיים רק אופרטור השמה, והוא מוכנס השמה של כן את מהגברים -2 private באופן קודמים:

אם אתנו כמתבים אופרטור בעת class, אני מאגדים (ומאסי'ל) ורק אופרטור כדאי להגיד את האופרטור האופרטור שלנו:

אם אבאר אסמלת casting האופרטור שלנו, וזאת בתוך class.

שק"ב 20 - אופרטור <c()

- רצוי להאופרטור א יפיה פונ' אפול' (אחרת נקבל סיגור: cout << 1).

- cout הוא אבייט קיים בספרייה הליבר, (נמו או פוס ostream).

המתחה לostream לא מכלי את האופרטור <<, ורק היא יורש את האופרטור זכרה לטובת.

- יש לרשור - אומנו מונימים חתמה לרצוני.

שק"ב 23

- ניתן לסמל; $++i$ אם כי מסופו של דבר, רק תתבצע השמה של i למק i .

אומנו מ, לא ניתן לסמל $++i$ א

שק"ב 25

- אופרטור השמה חייב להיות - גורם אחר, (ובי לחצור reference).

- מתי נעמד אופרטור השמה? ליש לנו פוינטרים.

שק"ב 27

- "size_t" - ספס ומספרייה הספיקות אחר אינפוס.

- הפע' הספיקות תקרא לו באבייט'ים לאינס const להתחבר, אבייט'ים להס const.

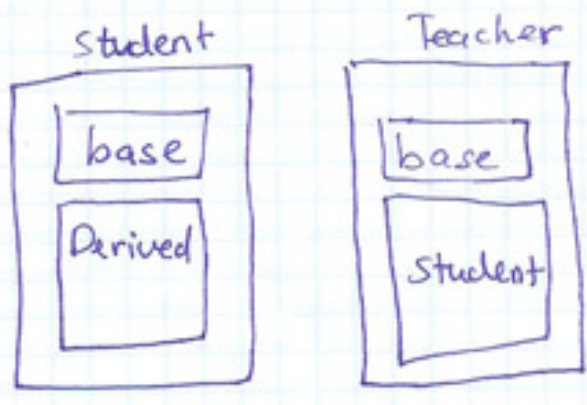
Inheritance

Syntax

```
Class B {
public:...
protected:...
private:...
};
```

```
class D:public B {
public:
private:
};
```

. B le protected ~~base~~ ^{class} B-^{to} D-^{to} protected -
 . D le protected ~~class~~ ^{class} D-^{to} protected -
 . B-^{to} D ^{to} private le private -
 . ^{to} D

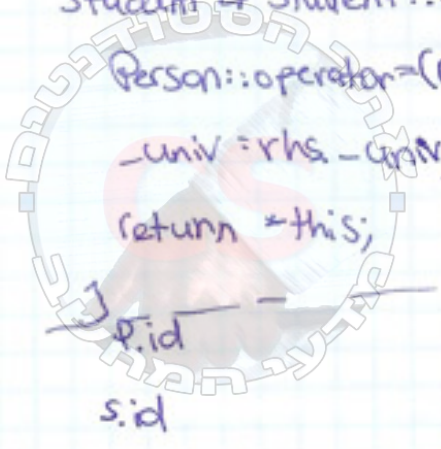


ly qe

Person ^{to} Student ^{to} Person ^{to} Student
 * ^{to} Student ^{to} Person ^{to} Student ^{to} Person
 base (^{to} data members) ^{to} Student ^{to} Person
 . ^{to} Student ^{to} Person
 - ^{to} Student ^{to} Person ^{to} Student ^{to} Person
 : (20 qe)

```
Student & Student::operator=(const Student & rhs) {
```

```
Person::operator=(rhs);
    _univ = rhs._univ;
    return *this;
```



- ^{to} Student ^{to} Person ^{to} Student ^{to} Person
 Student ^{to} Person ^{to} Student ^{to} Person