

**פתרון מבחן בתכנות מונחה עצמים בשפת C++**  
אוניברסיטת תל אביב

סמסטר קיץ – תשס"ה, מועד א'  
תאריך הבחינה: 20.09.2005

מרצה: אוהד ברזילי

חומר עזר מותר בשימוש: כל חומר עזר כתוב ומשוכפל  
(אסורים בשימוש: ספרים, מחשבים)

משך הבחינה: 3 שעות

הנחיות כלליות: הפתרון כולל 12 עמודים  
4 שאלות ללא בחירה

יש לענות על כל השאלות במחברות המצורפות

המבחן מנוסח בלשון נקבה אולם הוא פונה לנשים וגברים כאחד

בהצלחה

## שאלה מספר 1 (40 נקודות)

עבור קטעי הקוד הבאים רשמי מה יודפס כפלט:

א. (10 נקודות)

```
class Base1 {
public:
    Base1() { cout << "Base1::Base1()" << endl; }
    ~Base1() { cout << "Base1::~~Base1()" << endl; }
};

class Derived1 : public Base1 {
public:
    Derived1() { cout << "Derived1::Derived1()" << endl; }
    ~Derived1() { cout << "Derived1::~~Derived1()" << endl; }
};

class Base2 {
public:
    Base2() { cout << "Base2::Base2()" << endl; }
    ~Base2() { cout << "Base2::~~Base2()" << endl; }
};

class Derived2 : public Base2 {
public:
    Derived2() { cout << "Derived2::Derived2()" << endl; }
    ~Derived2() { cout << "Derived2::~~Derived2()" << endl; }
};

class Base {
    Base2 b2;
public:
    Base() { cout << "Base::Base()" << endl; }
    ~Base() { cout << "Base::~~Base()" << endl; }
};

class Derived : public Base {
    Derived1 d1;
public:
    Derived() { cout << "Derived::Derived()" << endl; }
    ~Derived() { cout << "Derived::~~Derived()" << endl; }
};

int main()
{
    Derived d;
    cout << "main()" << endl;
}
```

תשובה (סעיף א.):

```
Base2::Base2()
Base::Base()
Base1::Base1()
Derived1::Derived1()
Derived::Derived()
main()
Derived::~~Derived()
Derived1::~~Derived1()
Base1::~~Base1()
Base::~~Base()
Base2::~~Base2()
```

```

class Base1 {
public:
    Base1() { cout << "Base1::Base1()" << endl; }
    ~Base1() { cout << "Base1::~~Base1()" << endl; }
};

class Derived1 : public Base1 {
public:
    Derived1() { cout << "Derived1::Derived1()" << endl; }
    ~Derived1() { cout << "Derived1::~~Derived1()" << endl; }
};

class Base2 {
public:
    Base2() { cout << "Base2::Base2()" << endl; }
    ~Base2() { cout << "Base2::~~Base2()" << endl; }
};

class Derived2 : public Base2 {
public:
    Derived2() { cout << "Derived2::Derived2()" << endl; }
    ~Derived2() { cout << "Derived2::~~Derived2()" << endl; }
};

class Base {
    Base2 *b2;
public:
    Base() { cout << "Base::Base()" << endl; }
    ~Base() { cout << "Base::~~Base()" << endl; }
};

class Derived : public Base {
    Derived1 d1;
public:
    Derived() { cout << "Derived::Derived()" << endl; }
    ~Derived() { cout << "Derived::~~Derived()" << endl; }
};

int main()
{
    Base *b = new Derived();
    cout << "main()" << endl;
    delete b;
}

```

תשובה (סעיף ב.):

```

Base::Base()
Base1::Base1()
Derived1::Derived1()
Derived::Derived()
main()
Base::~~Base()

```

```

class Base1 {
public:
    Base1() { f(); }
    ~Base1() { f(); }
    void f() { cout << "Base1::f()" << endl; }
};

class Derived1 : public Base1 {
public:
    Derived1() { f(); }
    ~Derived1() { f(); }
    void f() { cout << "Derived1::f()" << endl; }
};

class Base2 {
public:
    Base2() { f(); }
    ~Base2() { f(); }
    void f() { cout << "Base2::f()" << endl; }
};

class Derived2 : public Base2 {
public:
    Derived2() { f(); }
    ~Derived2() { f(); }
    void f() { cout << "Derived2::f()" << endl; }
};

class Base {
public:
    Base2 *b2;
    Base() { f(); }
    ~Base() { f(); }
    void f() { cout << "Base::f()" << endl; }
};

class Derived : public Base {
public:
    Derived1 d1;
    Derived() { f(); }
    ~Derived() { f(); }
    void f() { cout << "Derived::f()" << endl; }
};

int main()
{
    Base *bd = new Derived();
    cout << "main()" << endl;
    bd->f();
    delete bd;
}

```

תשובה (ג.):

Base::f()  
Base1::f()  
Derived1::f()  
Derived::f()  
main()  
Base::f()  
Base::f()

```

class Base1 {
public:
    Base1() { f(); }
    virtual ~Base1() { f(); }
    virtual void f() { cout << "Base1::f()" << endl; }
};

class Derived1 : public Base1 {
public:
    Derived1() { f(); }
    virtual ~Derived1() { f(); }
    virtual void f() { cout << "Derived1::f()" << endl; }
};

class Base2 {
public:
    Base2() { f(); }
    virtual ~Base2() { f(); }
    virtual void f() { cout << "Base2::f()" << endl; }
};

class Derived2 : public Base2 {
public:
    Derived2() { f(); }
    virtual ~Derived2() { f(); }
    virtual void f() { cout << "Derived2::f()" << endl; }
};

class Base {
    Base2 *b2;
public:
    Base() { f(); }
    virtual ~Base() { f(); }
    virtual void f() { cout << "Base::f()" << endl; }
};

class Derived : public Base {
    Derived1 d1;
public:
    Derived() { f(); }
    virtual ~Derived() { f(); }
    virtual void f() { cout << "Derived::f()" << endl; }
};

int main()
{
    Base *bd = new Derived();
    cout << "main()" << endl;
    bd->f();
    delete bd;
}

```

תשובה (ד.):

Base::f()  
Base1::f()  
Derived1::f()  
Derived::f()  
main()  
Derived::f()  
Derived::f()  
Derived1::f()  
Base1::f()  
Base::f()



## שאלה מספר 2 (20 נקודות)

נתונה המחלקה Date :

```
class Date {
    int d , m , y ;
public:
    int day() const { return d ; }
    int month() const { return m ; }
    int year() const { return y ; }
    // ...
};
```

נרצה להוסיף למחלקה את המתודה string\_rep() המחזירה את התאריך כמחרוזת (לצורכי הדפסה למשל).

א. האם מתודה זו היא שאילתה או פקודה? נמקי (2 נקודות)

זוהי שאילתה מכיוון שהיא מחזירה ערך

ב. מהי החתימה של מתודה זו? (2 נקודות)

**string string\_rep() const**

ג. לאחר עבודה עם המחלקה התגלה כי המתודה string\_rep() לא יעילה מספיק, ויוצרת צוואר בקבוק בביצועי המחלקה. כדי לייעל את פעולת המחלקה נוסיף לה מנגנון של הטמנה (caching) כמתואר בקטע הקוד הבא :

```
class Date {
    mutable bool cache_valid;
    mutable string cache;
    void compute_cache_value() const;
    // ...
public:
    // ...
};
```

ממשי את string\_rep() לפי החתימה מהסעיף הקודם. אם ברצונך לשנות את החתימה בעקבות הנתונים החדשים – נמקי זאת (8 נקודות).

```
string Date::string_rep() const
{
    if(!cache_valid) {
        compute_cache_value();
        cache_valid = true;
    }
    return cache;
}
```

ד. cache ו-cache\_valid הוגדרו כ-mutable. רישמי חוזה למתודה string\_rep() שיסביר כי המתודה אינה משנה את המצב המופשט של Date. (8 נקודות).

מצב מופשט משתנה כאשר ערך של אחת המתודות הציבוריות משתנה. כדי להראות ש string\_rep אינה משנה את המצב המופשט נראה כי היא אינה משפיעה על אף אחת מהמתודות הציבוריות:

```
/** @post: day() == $prev(day())
 * @post: month() == $prev(month())
 * @post: year() == $prev(year())
 */
string string_rep() const
```

### שאלה מספר 3 (20 נקודות)

ביישום המטפל בצורות הנדסיות רבות (כגון: חרוט, פירמידה-משולשת, פירמידה-מרובעת, מלבן, ריבוע, משולש, מעגל ועוד...) נמצאו קטעי הקוד הבאים (חרוט == CONE):

```
// CONE.h
```

```
class CONE {
    double r;
    double h;
public:
    static const double PI;
    double volume() const;
    //...
};
```

```
// CONE.cpp
```

```
double const CONE::PI = 3.1415926535897932384626433832795;

double CONE::volume() const
{
    return (PI * r * r) * h / 3;
}
```

א. הסבירי במונחים של הנדסת תוכנה מדוע מימוש המתודה volume גרוע? אם יש יתרונות למימוש המוצג צייני גם אותם. (5 נקודות)

בשאלה זוין במפורש כי היישום מטפל במספר רב של צורות גיאומטריות. תחת הנחה זו ברור שיש שכפול קוד במימוש המתודה volume() אשר קיימת גם בפירמידות משולשות ומרובעות. בכל שלושת המקרים ניתן היה להכליל את חישוב הנפח לפי התכונה הגיאומטרית:  $volume() == baseArea() * height() / 3$ . זהו אלגוריתם כללי לחישוב נפח בכל הצורות המחוודות הישרות. יתרון מסוים שיש למימוש המוצג הוא היעילות.

ב. תקני את המימוש. אם לצורך המימוש יש להגדיר מחלקות \ מתודות \ שדות נוספים הגדירי אותם. ניתן להשתמש בתעתיק הלועזי של מונחים באנגלית שאינך יודעת את פירושם (לדוגמא HAROOT במקום חרוט). אין צורך לממש אף מתודה פרט ל `volume()`. הסבירי בקצרה את הפתרון, אם השתמשת בשיטות כלליות של הנדסת תוכנה (כגון: תבנית עיצוב קלאסית) צייני את שם השיטה. (15 נקודות)

נבנה היררכיית מחלקות ששורשה `SHAPE` והיא מכילה מחלקות דו-מימדיות (`2DSHAPE`) ותלת מימדיות (`3DSHAPE`). למחלקות הדו-מימדיות תהיה מתודה (וירטואלית) של שטח. למחלקות התלת-מימדיות תהיה מתודה (וירטואלית) של נפח. תת-משפחה של המחלקות התלת-מימדיות תהיה משפחת הצורות המחודדות שתיוצג ע"י בסיס דו מימדי וגובה, ותגדיר את המתודות הבאות:

```
virtual double PointedShape::volume() const
{
    return baseArea() * height() / 3;
}

virtual double PointedShape::baseArea() const
{
    return base()->area();
}
```

`volume()` מכונה אלגוריתם כללי (template method design pattern). כדי לחשב נפח של חרוט עלינו רק לגזור את חרוט ממחלקה מחודדת ולממש את ה `hooks` שמחלקה מחודדת מגדירה. אין צורך להגדיר מחדש את `volume` ב-`CONE` נדגיש, כי זו אינה תבנית התיכון `STRATEGY`. התבנית `STRATEGY` מדברת על החלפה דינאמית של התנהגות של עצם מסוים, בעוד שהרעיון שהצגנו כאן מתאר תבנית התנהגות של משפחה של מחלקות.

## שאלה מספר 4 (20 נקודות)

א. בשפת C++ מותר להמיר `Derived*` ל- `Base*`. (למחלקה מותר להצביע בפועל לצאצאיה), ואולם ההמרה `Derived**` ל- `Base**` אסורה. מדוע? (5 נקודות)

מצביע ל `Base *` שקול למיכל של `Base *`. בדומה, מצביע ל `Derived *` שקול למיכל של `Derived *`. לפי כללי הרב צורתיות (פולימורפיזם) `Derived` ו- `Base` מקיימים יחס is-a ולכן ההמרה `Derived` ל- `Base` מותרת (slicing) ואף ההמרה `Derived*` ל- `Base*` (פולימורפיזם). ואולם, מיכל של `Derived *` ומיכל של `Base *` אינם מקיימים את היחס is-a ולכן אין הצדקה מתודולוגית לאפשרו ההמרה. נימוק חלופי - מצביע ל `Derived` אינו מקיים יחס is-a עם מצביע ל `Base` ולכן ההמרה `Derived**` ל- `Base**` אסורה.

ב. תני דוגמת קוד ב C++ המדגימה את ההשלכות הלא רצויות של המרה שכזו. (15 נקודות)

אם השמה כזו הייתה אפשרית, ניתן היה לבנות דוגמא של שתי מחלקות היורשות מאותה מחלקה, ואז מצביע לאחת מהן היה מקבל בפועל מצביע לעצם מהמחלקה האחרת. ההשמה תתאפשר ע"י שימוש במצביע למצביע של מחלקת הבסיס (מיכל של מחלקת הבסיס)

דוגמא:

```
class Vehicle {
public:
    virtual ~Vehicle() { }
    virtual void startEngine() = 0;
};

class Car : public Vehicle {
public:
    virtual void startEngine();
    virtual void openGasCap();
};

class NuclearSubmarine : public Vehicle {
public:
    virtual void startEngine();
    virtual void fireNuclearMissile();
};

int main()
{
    Car car;
    Car* carPtr = &car;
    Car** carPtrPtr = &carPtr;

    NuclearSubmarine sub;
    NuclearSubmarine* subPtr = &sub;

    Vehicle** vehiclePtrPtr = carPtrPtr; // This is an error in C++
    *vehiclePtrPtr = subPtr; // This would have caused carPtr to point to sub !

    carPtr->openGasCap(); // This might call fireNuclearMissile()!
    ...
}
```