

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

סמסטר קיץ – תשס"ה, מועד ב'
תאריך הבחינה: 15.11.2005

מרצה: אוהד ברזילי

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

משך הבחינה: 3 שעות

הנחיות כלליות: פתרון זה כולל 9 עמודים (כולל עמוד זה)
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 (30 נקודות)

לגבי כל אחד מקטעי הקוד הבאים עני – האם הוא עובר קומפילציה? אם לא – פרטי מדוע, אם כן – פרטי מה קורה בזמן ריצה.

א. (10 נקודות)

```
1.  #include <iostream>
2.  #include <typeinfo>
3.
4.  class A
5.  {
6.      int a;
7.      int *pa;
8.  public:
9.      A(int i):a(i),pa(&a) { cout << a << endl; }
10.
11.     ~A() { cout << *pa << endl; }
12.
13.     int *get() { return pa; }
14. };
15.
16. class B : public A
17. {
18.     int *pb;
19. public:
20.     B (A obj):A(obj),pb(obj.get()+1)
21.     {
22.         cout << *(pb-1) << endl;
23.     }
24.     ~B() { cout << *(pb-1) << endl; }
25. };
26.
27. void print_me( A *p)
28. {
29.     cout << typeid(*p).name() << endl;
30. }
31.
32. int main()
33. {
34.     A *p = new B(A(3));
35.     print_me(p);
36. }
```

```
1.  #include <iostream>
2.  class A
3.  {
4.      int i;
5.      static int j;
6.
7.  public:
8.      A ()
9.      {
10.         i=j;
11.     }
12.     void operator[] (int val)
13.     {
14.         j = val;
15.     }
16.     ~A()
17.     {
18.         cout << i << endl;
19.     }
19. };
20.
21. int A::j = 23;
22.
23. int main()
24. {
25.     A obj;
26.
27.     obj[0];
28.
29.     A obj1;
30. }
```

```
1. #include <iostream>

2. template <class S>
3. void print (S obj1 = 0, S obj2 =0)
4. {
5.     cout << obj1 + obj2 << "\n";
6. }

7. int main()
8. {
9.     print (3,5);
10.    print (3);
11.    print ();
12.}
```

פתרונות:

א. יודפס:

3
3
3
3
class A

ב. יודפס:

0
23

ג. טעות קומפילציה – הטעות נגרמת בגלל שורה 11. לא מוגדרת פונקציה חסרת ארגומנטים בשם `print`.
המהדר לא מנסה להריץ את פונקציית התבנית מכיוון שאין לו רמז לגבי הטיפוס הפורמלי `S`.

שאלה מספר 2 (20 נקודות)

בשאלה הבאה נדון ביחסי הירושה בין מלבן ובין ריבוע. הניחי כי המחלקות RECTANGLE ו-SQUARE מתארות מלבן וריבוע בהתאמה אשר צלעותיהם מקבילות לצירים.

א. הניחי כי ריבוע מיוצג ע"י הקודקוד השמאלי-תחתון שלו (מטיפוס POINT) ואורך צלעו (מטיפוס double). האם במקרה כזה, ניתן לרשת את המחלקה מלבן מהמחלקה ריבוע ע"י הוספת שדה רוחב (מטיפוס double) במחלקה הירושה? נמקי את תשובתך בעזרת דוגמת קוד או בעזרת שימוש בתכונות קבלנות משנה של "עיצוב ע"י חוזה".

מלבן לא יכול לרשת מריבוע מהסיבה שהיחס is-a הוא בכיוון ההפוך. כל ריבוע הוא מלבן אבל לא כל מלבן הוא ריבוע. האמור לעיל מתייחס לירושה ציבורית (public), ניתן לבצע ירושה מימוש (protected או private) כדי לעשות שימוש חוזר הקוד בלבד (ללא ממשק).

ניתן להציג את הסתירה בעזרת השמורה של המחלקות:

```
/** @inv: area() == length() * length() */
class RECTANGLE {...};

/** @inv: area() == length() * width() */
class SQUARE : public RECTANGLE {...};
```

שמורת המחלקה SQUARE לא מחזקת את של RECTANGLE אלא סותרת אותה.

ב. הניחי כי מלבן מיוצג ע"י הקודקודים השמאלי-תחתון והימני-עליון שלו (מטיפוס POINT), וכן כי יש לו את המתודה void setSize(double x, double y) המשאירה את הקודקוד השמאלי-תחתון כעוגן ומשנה את שאר הקודקודים בהתאם למימדים שהוכנסו. האם במקרה כזה ניתן להגדיר ריבוע ע"י ירושה מהמחלקה מלבן?
1. אם כן – נמקי את תשובתך בעזרת חוזה המתודה setSize.
2. אם לא – הציעי תיקון סביר לתנאי השאלה והסבירי איך תוגדר המחלקה ריבוע תחת האילוצים החדשים. הסבירי את האילוצים שהתיקון יוצר – דוני בין כמה חלופות.

בתנאי השאלה לא ניתן לרשת ישירות (ירושה ציבורית – public inheritance). הדבר מפתיע, כי היצורים המתמטיים הטהורים מלבן וריבוע מקיימים את יחס is-a "בכיוון הנכון". קיימות כמה "פשרות כואבות" בתנאי השאלה (חלקן שקולות) שניתן לשקול:

1. הסרת setSize מהמחלקה מלבן. חסרון: פגיעה בקוד קיים המטפל במלבנים שאינם ריבועים
2. חיזוק תנאי הקדם של setSize במחלקה מלבן. למשל ע"י בדיקה האם הצורה משוכללת (דגל שיתוסף למלבן) – חסרון: פגיעה בקוד קיים המטפל במלבנים שאינם ריבועים
3. שינוי יחס הירושה - למשל ע"י ויתור על המחלקה ריבוע. ריבוע אינה מחלקה חדשה אלא עצם מטיפוס מלבן. כלומר ריבוע הוא מצב (STATE) – חסרון: לא ניתן לממש מפורשות בקוד, פעולות ייחודיות לריבוע.
4. שינוי יחס הירושה - למשל ע"י הוספת מחלקת בסיס המשותפת גם לריבוע וגם למלבן. ריבוע ומלבן יהפכו בצורה זו למחלקות אחיות. חסרון: מלבן וריבוע לא מקיימים עוד יחס פולימורפי.

שאלה מספר 3 (25 נקודות)

ביישום תקשורת כלשהו מופיעה המחלקה המופשטת MODEM שתפקידה לקודד ולפענח ערוצי תקשורת (המינוח העברי נקרא "אפנון"). למחלקה שני מרכיבים, MODULATOR (מקודד) ו- DEMODULATOR (מפענח). המחלקה מכילה כמה אלגוריתמים כלליים (template methods) כגון: init, open, send, attach, close ואחרים.

א. (8 נקודות) תארי (מילולית – ללא קוד) איך נבנה בעזרת המחלקה MODEM את המחלקה CABLE_MODEM לתמיכה בערוצי תקשורת כבלים.

המחלקה CABLE_MODEM תירש ציבורית (public inheritance) מהמחלקה MODEM. מכיוון שיש למחלקת הבסיס אלגוריתמים כלליים נותר רק לממש את ה hooks – אבני הבניין – של המחלקה הנגזרת. מכיוון שהמחלקה המופשטת משתמשת בשני מרכיבים חיצוניים **סביר** כי גם להם קיימת היררכיה מקבילה (כגון CABLE_MODULATOR ו- CABLE_DEMODULATOR) ויש לשקול להסתיר את השדות המקוריים ע"י גרסאות עדכניות של המחלקות האקטואליות.

ב. (9 נקודות) תני דוגמת קוד לבעיה של בטיחות טיפוסים שעשויה להיגרם בעקבות הפתרון שהצגת בסעיף הקודם.

נניח כי למחלקה MODEM יש המתודה void attach (MODULATOR *m). מכיוון שזהו אלגוריתם כללי הוא אינו מציין במפורש את טיפוס הפרטי ביותר האפשרי וכך אפשר לייצר את הקוד:

```
ANALOG_MODULATOR am;  
CABLE_MODEM cm;  
  
cm.attach (&am);
```

הקוד לעיל מצרף מאפן שגוי (אנלוגי) למודם כבלים.

ג. (8 נקודות) הציעי דרך לפתור את הבעיה שבסעיף הקודם בשפת C++. תארי את היתרונות והחסרונות של פתרון זה.

ניתן לפתור את בעיית ה covariance ע"י הפיכת המחלקות MODEM ו- CABLE_MODEM לתבניות (templates) עם טיפוסים פורמלים עבור כל אחד מהטיפוסים שעשוי להשתנות במהלך היררכיית הירושה. ניתן לספק ערכי ברירת מחדל לטיפוסים אלו אשר ישתנו בכל אחת מהמחלקות היורשות.

חסרונות השיטה:

1. סרבול עד כדי חוסר קריאות בעיקר כאשר מספר ההיררכיות המקבילות גדל (במקרה שלנו יש 3 היררכיות מקבילות – זו של MODEM ואלו של MODULATOR ו- DEMODULATOR).

2. לא ניתן לקבוע כטיפוס פורמלי את טיפוס המחלקה עצמה. במקרה שלנו הדבר לא בא לידי ביטוי, אבל אם למודם היה, לדוגמא, מצביע למודם חלופי מאותו טיפוס – לא ניתן היה לממש זאת בשיטה זו.

שאלה מספר 4 (25 נקודות)

נתונה מבנה הנתונים NODE המייצג קודקוד בעץ בינארי, המכיל ערכים מטיפוס שלם:

```
struct NODE {
    NODE *left;
    NODE *right;
    NODE *parent;
    int val;
};
```

א. (13 נקודות) בני מחלקה בשם tree המממשת מיכל (container) של שלמים המוחזקים בעץ בינארי. על המחלקה לתמוך בשרותי אצן (איטרטור) סטנדרטיים בלבד. אין צורך לתמוך בכמה שיטות לסריקה של העץ, ואין צורך להגדיר const_iterator. הגדירי אצן-קידום בלבד (forward-iterator) לפי שיטת הסריקה שבחרת. יש צורך לממש רק את המתודות של tree אין צורך לממש את הממשק של האצן.

ניתן ומומלץ להגדיר את האצן כחלק מהמחלקה tree (כמחלקה פנימית). המחלקה tree תגדיר לפחות את המתודות begin() ו-end() המחזירות אצן מתאים. מימוש האצן חייב לכלול לפחות 3 מתודות (אופרטורים): ++, ==, * (שאין צורך לממשן במבחן!).

נשים לב שסריקה של עץ בינארי מאוד נוחה בצורה רקורסיבית ואולם הפעולה ++ הממומשת באצן מרמזת על סריקה איטרטיבית ומחייבת אותו לשמור משתנה נוסף שיעזור לו למצוא את האיבר הבא בעץ (בהתאם לשיטת הסריקה של העץ). בדוגמא אנו סורקים עץ בשיטת LDR (Left-Data-Right) ולכן יש להחזיק דגל המציין האם אנו בדרכנו למעלה או למטה.

```

class tree {
    NODE *root;

public:
    tree(NODE *r) : root(r) {}

    class iterator {
        NODE *cur;
        enum DIR {UP, DOWN} direction;

public:
        iterator(NODE *pos, DIR d = UP) : cur(pos), direction(d)
        {}

        iterator operator++();

        int operator*() { return cur->val; }

        bool operator==(iterator other)
        { return cur == other.cur; }

        bool operator!=(iterator other)
        { return cur != other.cur; }

        void go_left() {
            while (cur->left)
                cur = cur->left;
            direction = UP;
        }

};

iterator begin()
{
    iterator i(root);
    i.go_left();
    return i;
}

iterator end()
{
    return iterator(NULL);
}

};

```

ב. (6 נקודות) ממשי את הפונקציה הגלובלית `void print_tree(tree& t)` המדפיסה את תוכנו של `t` מבלי לחשוף את המבנה הפנימי שלו. המימוש צריך להשתמש בתכונות שמומשו בסעיפים א. ו-ב.

```
void print_tree(tree& t)
{
    for (tree::iterator iter = t.begin() ;
         iter != t.end() ; iter++)
        cout << *iter << endl;
}
```

ג. (6 נקודות) הציגי מימוש חלופי ל- `print_tree` מהסעיף הקודם, המשתמש באחד האלגוריתמים הסטנדרטים המופיעים בכותרת `<algorithm>` וכן בפונקציה `.void print_int(int)`.

```
void print_tree(tree& t)
{
    for_each(t.begin(), t.end(), print_int);
}
```

101, בהצלחה