

סדנת תכנות ב- C++

67317

מועד ב'.

מרצה: מוטי פריימן

17.03.06

הנחיות:

- משך הבחינה: שעתיים.
- יש לענות על כל שאלות הבחינה.
- ניתן להשתמש בכל חומר עזר כתוב. אין להשתמש בכלי חישוב מכל סוג שהוא.
- כתיבת התשובות תתבצע אך ורק על דפי הבחינה. ניתן להעזר במחברות כטיוטה, אולם הם לא יבדקו כלל.
- שאלות סגורות: יש להקיף בעיגול את האות בתחילת השורה בה נמצאת התשובה הנכונה (קיימת תשובה נכונה יחידה בכל שאלה סגורה).
- שאלות הדורשות כתיבה: כתבו רק בשורות המיועדות לכתיבה, התשובה תיחשב תקינה במידה של חריגה בשל גודל כתב או סגנון כתיבה.
- יש להקפיד על רישום ברור של מס' תז.

שאלות:

1. נתונה התכנית הבאה:

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

struct WhiteHouse
{
    int *p;
    WhiteHouse(int n) { p = new int; *p = n; }
    ~WhiteHouse() { delete p; }
};

void f(WhiteHouse other)
{
    WhiteHouse fdr(1932);
    fdr = other;
}

int main()
{
    WhiteHouse ww(1912);
    f(ww);
    cout << *ww.p;
    return 0;
}
```

א. (4%) תארו את ניהול הזכרון בתכנית?

במחסנית של ה main נוצר אובייקט מסוג WhiteHouse המכיל מצביע למשתנה

מסוג int המכיל את המספר 1912. בקריאה ל f נוצר אובייקט במחסנית של f בשם other המכיל מצביע שמצביע לאותו איזור בזיכרון שמוצבע על ידי המצביע באובייקט ww. לאחר מכן נוצר אובייקט חדש בשם fdr שמקצה מקום חדש בזכרון המכיל את הערך 1932. המצביע שהצביע לאיזור המיכל את 1932, מקבל את הכתובת של האיזור שמכיל את הערך 1912 ע"י הפעלת אופרטור ההשמה. עם היציאה מהמחסנית של f מופעל ה dest של האובייקט fdr, והאיזור המכיל את הערך 1912) שכעת מוצבע ע"י המצביע ב fdr) משוחרר. האיזור שמכיל את הערך 1932 אינו מוצבע ע"י שום משתנה כעת, ולכן אין שום דרך לשחררו – דליפת זכרון. לאחר מכן ב main מנסים לשת לאיזור שהכיל בתוכו את הערך 1912, ושחרר.

ב. (5%) תקנו את כל הבאגים בתכנית, כך שהתכנית תרוץ באופן היעיל והבטוח ביותר, ללא שינוי של data members של WhiteHouse וללא שינוי של המימוש של f:

הפונקציה f צריכה להיות מוגדרת:

```
void f (WhiteHouse & other)
```

יש להוסיף מימוש של אופרטור השמה למחלקה WhiteHouse:

```
WhiteHouse & operator= (const WhiteHouse & other)
{
    delete p;
    p = new int (*other.p);
    return *this;
}
```

2

נתונה התכנית הבאה:

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

class C
{
public:
    C(int i) : _p(new int (i)) {}
    ~C() {delete _p;}
    int *_p;
};

class C2 : public C
{
public:
    C2(int i) : C (i), _i(i) {}
    ~C2 (){}
    int _i;
};

int main()
{
    C2 o1 (5);
    C2 o2 (o1);
}
```

```
C2 o3 (o2);
```

```
cout << o1._i << ", " << *o1._p << endl;  
cout << o2._i << ", " << *o2._p << endl;  
cout << o3._i << ", " << *o3._p << endl;
```

```
}
```

א. (3%) מה יהיה הפלט של התוכנית?

5, 5

5, 5

5, 5

ב. (4%) הסבירו מהו ה bug הקיים בתכנית ותקנו אותו?

מאחר ולמחלקה C לא מוגדר copy constructor, נעשה שימוש ב copy constructor הדיפולטיבי, ולכן המצביע _p מועתק באופן רדוד (ז.א. מועתק רק הערך של הכתובת) ולכן שלושת האובייקטים שנוצרו בתכנית, מכילים מצביעים לאותו מקום בזכרון, וכשמופעלים ה dest של o2,o3 הם מנסים לשחרר מקום שכבר שוחרר. התיקון: להוסיף למחלקה C את ה cons הבא:

```
C (const C & other) : _p (new int (*other._p)) {}
```

ג. (12%)

נתונה התכנית הבאה:

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

```
class FloatArr  
{  
private:  
    int _size;  
    float *_arr;  
  
public:  
    FloatArr () : _size (0)  
    {  
        _arr = new float [_size];  
    }  
    FloatArr (int i, float init=0) : _size (i)  
    {  
        _arr = new float [_size];  
        for (int j=0;j<_size;j++)  
            _arr[j]=init;  
    }  
    ~FloatArr () { delete [] _arr; }  
};
```

```
int main()  
{  
    FloatArr arr1(5,3);  
    FloatArr arr2(5);
```

```

arr2 [3] = arr1 [1];
cout<<arr1+arr2<<endl;
}

```

ממשו את האופרטורים החסרים (כולל ה prototype שלהם), כך שהתכנית תעבוד היטב. מותר אך לא הכרחי להוסיף עוד מתודות למחלקה FloatArr (המשמעות של פעולת החיבור על האובייקטים היא של חיבור ווקטורי. ז.א. כל איבר מחובר לאיבר המקביל בוקטור השני).

הערה: היה bug מינורי בתכנית כפי שהוצגה במבחן, אך זה לא שינה את השאלה כלל. ה bug תוקן בגירסה זו.

```

float & operator[] (int i) {return _arr[i];}
const float & operator[] (int i) const {return _arr[i];}
friend FloatArr operator+ (const FloatArr & arr1, const FloatArr & arr2)
{
    FloatArr res (arr1._size);
    for (int i=0;i<arr1._size;i++)
        res [i] = arr1[i] + arr2[i];
    return res;
}
friend ostream & operator<< (ostream & os, const FloatArr & arr1)
{
    for (int i=0;i<arr1._size;i++)
        os<<arr1[i]<<" ";
    return os;
}

```

4 (6%)

נתונה התכנית הבאה:

```

struct A
{
    int *_p;
    A(int i) : _p (new int (i)) {}
};
struct B
{
    int _size;
    int *_arr;
    A ** _arrA;
    B (int size) : _size (size), _arr (new int [_size]), _arrA (new A* [_size])
    {
        for (int i=0;i<_size;i++)
        {
            _arr[i] = i;
            _arrA [i] = new A (_arr[i]);
        }
    }
};
int main ()
{
    B (5);
}

```

הוסיפו prototype ומימוש של destructor למחלקות A,B כך שלא תהיה כל דליפת זכרון בתכנית לעיל:

```
~A () {delete _p;}
~B ()
{
    for (int i=0;i<_size;i++)
    {
        delete _arrA [i];
    }
    delete [] _arrA;
    delete [] _arr;
}
```

5. (7%)

התכנית הבאה אינה מתקמפלת, כתבו את מספרי השורות שנעשות בהן פעולות לא חוקיות:

```
1. class A
2. {
3. protected:
4.     int _i;
5.     A () : _i (0) {}
6.     ~A() {}
7. };
8. class B: public A
9. {
10. public:
11.     A *_pa;
12.     B () : A(), _pa (new A()) {}
13.     ~B () {delete _pa;}
14. };
15. int main ()
16. {
17.     A a;
18.     B b;
19. }
```

12, 13, 17 .

6. (7%)

מתכנת רצה ליצר מחלקות כך שמחלקת האב תפעיל את פונקצית האתחול של כל בן. לצורך כך לכל בן מומשה פונקצית init אשר הוגדרה באב כ virtual, והופעלה על ידי ה constructor של האב.

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

```
class B
{
public:
    int _b;
    B() { init(); }
    virtual ~B(){}
    virtual void init() { _b = 10; }
```

```

virtual void print() { cout<<"b = "<<_b<<endl;}
};

class D : public B
{
public:
    int _d;
    virtual void init() { _d = 20; _b = 20; }
    virtual void print() { B::print();cout<<"d = "<<_d<<endl;}
    virtual ~D(){}
};

```

```

int main()
{
    B *x = new D();
    x->print ();
    delete x;
}

```

המתכנת כתב את ה main לעיל לצורך בדיקה וציפה שהפלט יהיה:

```

b = 20
d = 20

```

אולם להפתעתו הפלט היה:

```

b = 10
d = 0

```

הסבירו מהי השגיאה שגרמה לפלט הזה?

מאחר וסדר יצירת D שירש מ B הוא: בתחילה נוצר החלק של B, ורק לאחר מכן נוצר החלק של D. אזי בזמן יצירת B הפונקציה init היחידה שקיימת היא B::init ולכן הקריאה ב cons של B ל init יכולה לקרוא רק ל B::init וזאת למרות שבעקרון init היא פונקציה וירטואלית. הוירטואליות באה לידי ביטוי רק לאחר שהאובייקט D נוצר בשלמותו ושתי הפונקציות B::init, D::init קיימות.

7.

בהתייחס לתכנית הבאה:

```

template <typename T>
class A
{
    T* _t;
public:
    A () : _t ( new T ()){}
    ~A () {}
};

```

```
int main()
{
    A <int*> a1;
    A <int> a2;
    A <A<double>*> a3;
}
```

א. (3%) מהו מספר המחלקות השונות בקוד של התכנית: 4.

ב. (3%) מהו הטיפוס T בכל מופע של A.

a1: int*

a2: int

a3: A<double>*

8. (6%)

נתון קטע הקוד הבא:

```
int main ()
{
    const int *a = new int (3);
    int *p1 = &(*a);
    int * const p1 = a;
    const int * p1 = &(*a);
    int b = *a;

    a++;
    b++;
    p1++;
    (*p1)++;
}
```

כתבו את מספרי השורות שאינן עוברות קומפילציה מבין השורות הממוספרות (הניחו ביחס לכל שורה כי השורות שאינן תקינות ונמצאות מעל לה, נמחקו):

2, 3, 9

9. (9%)

ממשו (כולל כתיבת prototype) פונקציית template גלובלית Sum אשר תקבל טווח של input iterator ומשתנה סכום, אשר לתוכו יוספו המחוברים. ה-main הבא צריך לפעול כראוי:

```
int main() {
    vector<int> vec;
    for (int i=0; i<5; ++i) vec.push_back(i);

    double arr[]= {-2, 0.5, 4, 1.853};

    int s= 25;
```

```

Sum(vec.begin(),vec.end(),s);

double s2= 0;
Sum(arr,arr+3,s2);

cout << s << endl;
cout << s2 << endl;
}

```

הפלט הצפוי:

35
2.5

```

template <class IteratorType, class ValType>
ValType Sum (IteratorType begin, IteratorType end, ValType & acc)
{
    IteratorType iter = begin;
    while (iter != end)
    {
        acc += *iter;
        ++iter;
    }
    return acc;
}

```

10 (6%)

הסבירו מהי הבעיה בתכנית הבאה:

```

class Exp {};
class A
{
public:
    A()
    {
        _p1 = new int (0);
        throw Exp ();
        _p2 = new int (0);
    }
    ~A()
    {
        delete _p1;
        delete _p2;
    }
    int *_p1, *_p2;
};
int main ()
{
    A a;
}

```

מאחר והאובייקט a זורק exception באמצע ה cons, הוא לא נוצר באופן שלם, וכתוצאה מכך, לא מופעל ה dest שלו, ולכן אין שחרור של הזכרון שהוקצה ב cons לפני זריקת ה .excpetion.

התכנית הבאה אינה מתקמפלת:

```
#include <iostream>
using namespace std;
class A
{
public:
    void print () {cout<<"A"<<endl;}
};
class B: public A
{
public:
    void print () {A::print(); cout<<"B"<<endl;}
};
int main ()
{
    A *a = new B();
    B* b = dynamic_cast<B*>(a);
    b->print ();
}
```

א. (4%) הסבירו מהי הבעיה?

אי אפשר להפעיל `dynamic cast` על אובייקט שאינו פולימורפי (ז.א. אינו מכיל לפחות פונקציה וירטואלית אחת).

ב. (4%) תקנו את התכנית (ללא שינוי של ה `main`) כך שהיא תפעל והפלט יהיה:

A
B

ההצהרה של הפונקציה `print` במחלקה A צריכה להיות:

```
virtual void print () {cout<<"A"<<endl;}
```

12 (5%)

המחלקה הבאה בנויה באופן שגוי (אינה עוברת קומפילציה), תקנו אותה כך שתעבור קומפילציה. אין לשנות את טיפוסי המשתנים.

```
class A {
    int _i;
    const int _c_i;
    int& _r_i;
    A (int i) { _i = i; _c_i = _i; _r_i = _i;}
};
```

ה `cons` צריך להיות מוגדר באופן הבא:

```
A (int i) : _i (i), _c_i (_i), _r_i (_i) {}
```

א. (7%) מלצר כתב לעצמו תכנית שתאפשר לשמור מערך של הזמנות למשקים מסוגים שונים. בנוסף לאתחול של ההזמנות, המלצר בנה אפשרות לשנות מספר הזמנות של סוג משקה מסוים בעזרת הפונקציה `fix`. בכדי לבדוק את התכנית שלו הוא כתב את פונקציית ה `main`, והריץ את התכנית.

```
#include <iostream>
using namespace std;
class Drink
{
public:
    int *_arr;
    Drink(int arr[], int size) : _arr (new int [size])
    {
        for (int i=0;i<size;i++)
        {
            _arr[i] = arr[i];
        }
    }
    ~Drink () {delete []_arr;}
    int & get(int i) const { int t = _arr[i]; return t;}
};
void fix( int &index, int num) { index = num; }
int main() {
    int orders[] = {0,1,2,3,4};
    int orders_size = 5;
    Drink d (orders,orders_size);
    fix ( d.get(2), 4);
    for (int i=0;i<orders_size;i++)
    {
        cout<<d._arr[i]<<" ";
    }
    cout<<endl;
}
```

להפתעתו הוא גילה כי הפלט של התכנית הוא:

0, 1, 2, 3, 4,

תקנו את הטעון תיקון (ניתן לשנות רק את המחלקה `Drink`) כך שהתכנית תעבוד היטב והפלט יהיה:

0, 1, 4, 3, 4,

הפונקציה `get` צריכה להיות ממומשת באופן הבא:

```
int & get(int i) const { return _arr[i];}
```

ב. (5%) האם בתכנית הבאה יש דליפת זכרון?

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

class A
{
    int *_p;
public:
    A(const int& i): _p (new int (i)){}
    ~A () {delete _p;}
};
A& FactoryA (int i)
{
    A *t = new A (i);
    return *t;
}
int main()
{
    A& a = FactoryA (5);
}
```

כן / לא

תשובה: כן

בהצלחה!