

תכנות מונחה עצמים – מועד א'

- במבחן זה 3 שאלות. יש לענות על כולן.
- יש לתת שמות משמעותיים למשתנים ומחלקות.
- משך הבחינה – שעתיים וחצי.
- מותר להשתמש בכל חומר עזר פרט למחשב.
- כתבו בכתב יד קריא ומסודר.

שאלה 1 (20 נק'):

נתון הקוד הבא. מה יודפס?

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

class A{
public:
    A(int x=0) {cout<<"A"<<x<<endl;}
    void f() {cout<<"A.f"<<endl;}
};

class B : public A{
public:
    B() {cout<<"B"<<endl;}
    void f() {cout<<"B.f"<<endl;}
};

class C{
public:
    C():a(4) {cout<<"C"<<endl;}
    virtual void f() {cout<<"C.f"<<endl;}
private:
    A a;
    B b;
};

class D : public C{
public:
    D() {cout<<"D"<<endl;}
    virtual void f() {cout<<"D.f"<<endl;}
};

int main(){
    A *pA = new B();
    C *pC = new D();
    C c = *pC;

    pA->f();
    pC->f();
    c.f();

    delete pA;
    delete pC;

    return 0;
}
```

שאלה 2 (30 נק'):

אנו רוצים להגדיר מחלקה של שברים ואת האופרטורים עליהם. להלן הגרסה הראשונה של הגדרת המחלקה:

```
class Rational{
    int p, q; // represents p/q
public:
    Rational(int _p, int _q):p(_p),q(_q){}
};

int main(){
    Rational numbers[2];
    numbers[0] = Rational(1,2); // 1/2
    numbers[1] = Rational(2,3); // 2/3
    return 0;
}
```

א. תקנו את הקוד לעיל על מנת שיעבור הידור

ב. אילו מהטענות הבאות נכונות? הסבירו את תשובותיכם.

(233) את `operator+` חייבים להגדיר כ-`member` במחלקה `Rational`

(234) אם `operator+` מוגדר, לא חייבים להגדיר גם את `operator+=`, כי המהדר יעשה את זה אוטומטית

(235) לא חייבים להגדיר `operator=` עבור המחלקה `Rational`, כי המהדר יעשה את זה אוטומטית

(236) אם `operator+` ו `operator=` מוגדרים, אז המהדר יגדיר את `operator+=` אוטומטית

ג. הגדירו וממשו את `operator+`, `operator+=`, ו `operator=` עבור המחלקה `Rational`. אין צורך להגדיר אופרטורים שמוגדרים נכון על ידי המהדר.

ד. אנו רוצים לממש פונקציה (שאינה חברה במחלקה) שתקבל וקטור ארוך של שברים ותדפיס את השברים האלה בסדר עולה.

```
void printSorted(vector<Rational> v){/*...*/}
```

למזלנו, STL מספקת את האלגוריתם של מיון (`sort`). אילו אופרטורים יש להגדיר על מנת למיין את `v` ?

ה. איך אפשר ליעל את הקריאות ל `printSorted` ?

(407) להגדיר `default constructor` עבור `Rational`

(408) להגדיר `copy constructor` עבור `Rational`

(409) להגדיר את `v` כ `const vector<Rational>`

(410) להגדיר את `v` כ `vector<const Rational>`

(411) להגדיר את `v` כ `vector<Rational*>`

(412) להגדיר את `v` כ `vector<Rational>&`

(413) אף אחד מהנ"ל



שאלה 3 (50 נק'):

כתוב תוכנית שתנהל עבור עיריית חיפה בסיס נתונים, אשר יכיל, לכל חלקו, את הפרטים על מיקומה ומחירה. כל חלקה היא מאחד הסוגים הבאים: בניין, גן, ובית ספר. עבורם יש להגדיר את המחלקות הבאות:

```
* class Parcel: המחלקה הזו מתארת חלקה. אין לאפשר ליצור אובייקטים שהם מסוג Parcel אלא רק מאחת המחלקות היורשות מ Parcel.  
משתני המחלקה: char* street, int number, int area
```

```
* class Park: יורשת מ Parcel  
משתני מחלקה נוספים: char* name  
פרטים נוספים: הארנונה של הפארק היא 0  
מחיר הפארק הוא פי 10 משטחו
```

```
* class Building: יורשת מ Parcel.  
משתני מחלקה נוספים: int nFloors  
פרטים נוספים: הארנונה של הבניין שווה לשטחו  
המחיר של הבניין הוא פי 2 משטחו כפול מס' הקומות
```

```
* class School: יורשת מ Building.  
משתני מחלקה נוספים: char* name  
פרטים נוספים: הארנונה של ביה"ס היא שליש מזו של בניין אחר עם אותם מאפיינים.  
המחיר של ביה"ס שווה למחיר של בניין אחר עם אותם מאפיינים.
```

עבור כל מחלקה יש להגדיר את data constructor (מקבל את כל הנתונים) ו-copy constructor (משכפל אובייקטים קיימים). אין צורך להגדיר מתודות גישה set/get, אך ניתן להיעזר בהם ללא המימוש במידת הצורך.

בנוסף, יש להגדיר את המחלקה הבאה:

```
* class City: מחלקה זו מנהלת את בסיס הנתונים. במחלקה זו נחזיק את מאגר החלקות בעיר.
```

יש לממש את הפונקציות הבאות במחלקת City:

- פונק' המקבלת מצביע לחלקה חדשה ומוסיפה אותה למאגר (אין צורך להעתיק אותה). במידה וקיימת חלקה עם אותה הכתובת (שם הרחוב ומס' הבית), יש לזרוק DuplicateRecordException
- פונק' המחזירה את המצביע להחלקה היקרה בעיר. במידה ואין כזאת, יש לזרוק RecordNotFoundException
- פונק' המדפיסה לכל חלקה את הכתובת והארנונה שלה
- פונק' שמקבלת כתובת (שם הרחוב ומס' הבית) ומחזירה מצביע לחלקה בכתובת זו. במידה והחלקה לא נמצאה, יש לזרוק RecordNotFoundException
- פונק' המדפיסה, לכל חלקה, את כל הנתונים השמורים ברשומה שלה

אין צורך לכתוב main. יש להשתמש בשמות המחלקות כפי שהוגדרו. במידת הצורך ניתן להגדיר פונקציות חדשות. הקפידו על תכנות מונחה עצמים נכון!

בהצלחה!



#1 אספה אוניברסיטת חיפה

- 1) A 0 (-1)
 - 2) B
-
- 3) A 4
 - 4) B
 - 5) C
 - 6) D

אין איתחול
a fo
(-2)

- 1 15
- 2 30
- 3 57

מספר סידורי רץ
99
לשימוש משגיחה

חוג מ"מ

מחברת מס' 2

מתוך 2 מחברות

בחינה בקורס: OOP

תאריך הבחינה: מועד א' ב' / 24.07

שם המורה: מרינה ביבישצ'ין

לשימוש הבוחן בלבד

הערות הבוחן

הציון

חתימת הבוחן 100

- 7) A
 - 8) B
 - 9) C
-
- 10) A.f
 - 11) D.f
 - 12) C.f

אין איתחול
a fo
(-2)



שאלה #2

```
class Rational
```

```
{ int p, q;  
  public:
```

```
Rational ( int -p=0, int -q=1): p(-p),  
  q(q) {}
```

```
};
```

```
int main ()
```

```
{ Rational numbers[2]; // OK now  
  //  
}
```

(א)
6/6

(ב) לא נכון
1) את + operator אפשר להכיר מחוץ לחלקה כג שני יאפשר ביצוע

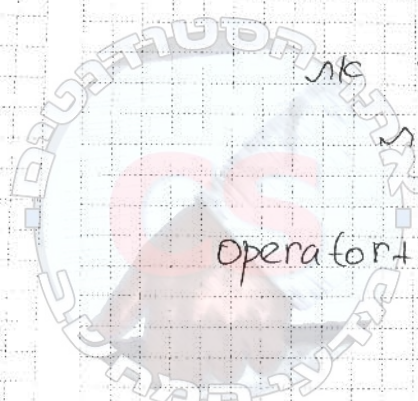
```
Rational q = 3 + q;  
Rational p, q;  
p = 3 + q;
```

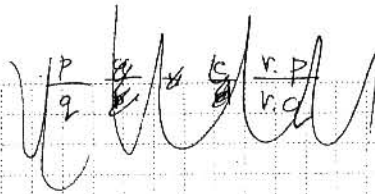
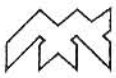
יש קוד הכאה :
לומר נקבל
אופרטור סימלי

(ג) לא נכון, חייבים להכיר
את operator = כי ~~אפשר~~
אופרטור שונה למחר

(ד) נכון, default operator =
צביכות לחלקה Rational בעזרת
שדה int,

(ה) לא נכון, חייבים להכיר את
operator = במפורש.





2
6/6

class Rational & ...

{ -- }

public:

Rational operator+(const Rational& r) const
{ return Rational(p*r.q + r.p*q, r.q*q); }

Rational& operator+=(const Rational& val)
{ return *this = *this + val; }

}

אם $\text{operator} =$ אין בורק להגדרה
 $\text{default operator} =$ אחרת הקבוצה היא

3
על מנת לבצע sort נצטרך

אופרטורים של השוואה

אם אין לנו אופרטור עבור sort ב-STL

נסתכן להגדיר אופרטור $\text{operator} <$

של המחלקה Rational שיוצג להשוואה בין

עצמים של Rational

עבור הגדרת נצטרך להגדיר גם

אופרטור $\text{operator} <<$

$\text{ostream}\& \text{operator} << (\text{ostream}\& \text{out},$
 $\text{const Rational}\& r);$

שאותו נגדיר מחוץ למחלקה Rational אך

נצטרך אותו כ- friend ב-Rational

4
6/6
להגדיר את

$\text{vector} < \text{Rational} > \& v$

כך נחסוך זמן חישוב

אשר ציטוט להצטרף של עצם

ונשתמש ב-reference שלו



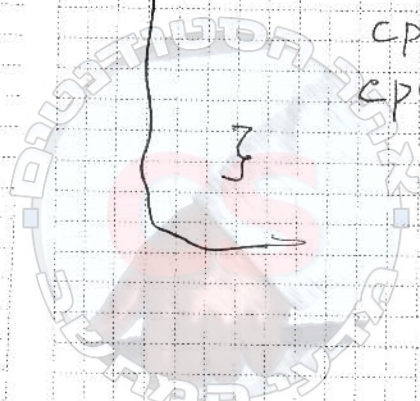


אבטו (Rational) - אבטו (Rational) - אבטו (Rational)
 אבטו (Rational) - אבטו (Rational) - אבטו (Rational)
 אבטו (Rational) - אבטו (Rational) - אבטו (Rational)
 אבטו (Rational) - אבטו (Rational) - אבטו (Rational)

if (this == &o) return *this;

```

... Park ( Park & o) // copy ctor
{
  area = o.area; number = o.number;
  street street = new char [ strlen(o.street) + 1 ];
  name = new char [ strlen(o.name) + 1 ];
  strcpy ( street, o.street );
  strcpy ( name, o.name );
}
  
```





#3 שאלה

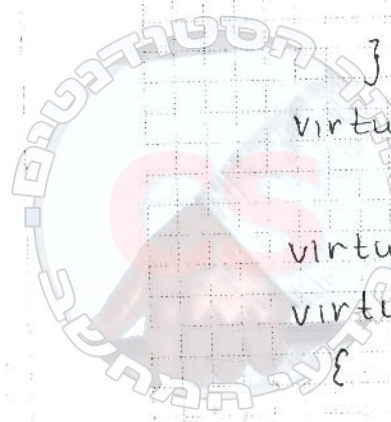
```
#include <iostream>
#include <cstring>
class Parcel ← using namespace std;
```

file: Parcel.h

```
{ protected:
  int area, number;
  char* street;
  friend City;
public:
  virtual int getArnona() = 0;
  virtual int getPrice() = 0;
  virtual void print() = 0;
  virtual ~Parcel() { delete[] street; }
```

```
class Park : public Parcel
{ protected:
  char* name;
public:
  Park ( int a, int n, char* st, char* nm )
  { area = a; number = n;
    street = new char [ strlen(st) + 1 ];
    name = new char [ strlen(nm) + 1 ];
    strcpy ( street, st );
    strcpy ( name, nm );
```

```
}
virtual ~Park()
{ delete[] name; }
virtual int getArnona()
{ return 0; }
virtual int getPrice() { return 10 * area; }
virtual void print()
{ cout << "Park" << " Area:" << area << " Address:"
  << street << number << " Name:" << name
  << "\n"; }
```




```

class Building : public Parcel
{
protected:
    int nFloors;
public:
    Building(int n, int a, char* street, int nF)
    {
        number = n; area = a;
        street = new char[ strlen(st) + 1 ];
        strcpy( street, st );
        nFloors = nF;
    }
    Building( Building& o )
    {
        if ( this == &o ) return *this;
        number = o.number;
        area = o.area;
        nFloors = o.nFloors;
        street = new char[ strlen(o.street) + 1 ];
        strcpy( street, o.street );
    }
    virtual ~Building()
    { delete street; }
    virtual int get Area()
    { return area; }
    virtual int get Price()
    { return area * 2 * nFloors; }
    virtual void print()
    {
        cout << "Building, Address: " << street <<
            << number << " Area: " << area
            << " Numb of Floors: " << nFloors
            << '\n';
    }
}

```

class friend City;



```
class School : public Building
{
protected:
    char * name ;
public:
    School (int n, int a, char * street, int nF,
            char * nm) : Building(n, a, st, nF)
    {
        name = new char [ strlen(nm) + 1 ];
        strcpy ( name, nm);
    }
    School ( School & o)
    {
Building::Building(o);
        name = new char [ strlen(nm + 1) ];
        strcpy ( name, nm);
    }
    virtual ~School ()
    { delete [] name; }
    virtual int getArnona ()
    { return  $\frac{1}{3}$  * Building::getArnona (); }
    virtual int getPrice ()
    { return Building::getPrice (); }
    virtual void print ()
    {
        cout << "School, Address: " : << street
            << number << "Name: " << name
            << "Area: " << area << '\n';
    }
    friend void city;
}
```





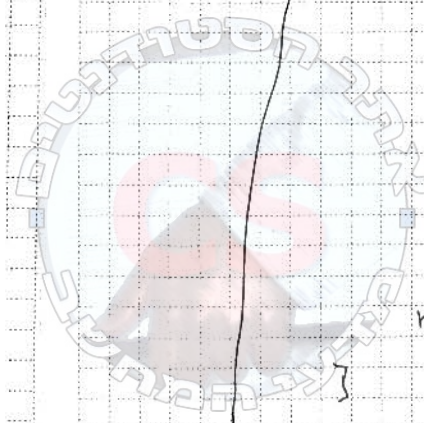
```
#include "Parcel.h"
#include <iostream>
#include <cstring>
#include <vector>
```

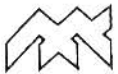
file: City.h

```
Class City
{ protected:
  vector< Parcel* > pV;
public:
  City() {}
  void add( Parcel * p)
  { vector< Parcel* > pV;
    vector< Parcel* >::iterator it;
    it = pV.begin();
    while ( pV it != pV.end())
    { if ( (*)it → number == p → number
      vector< Parcel* >
      && strcmp((*)it → street, p → street) == 0 )
        throw DuplicateRecordException(it);
      ++it; }
    pV.pushBack(*p);
  }
}
```

```
vector< Parcel* >
Parcel* BiggestPrice (void)
{ int max=0; Parcel* out=0;
  vector< Parcel* > vector< Parcel* >::iterator it;
  vector< Parcel* > it = pV.begin();
  while ( it != pV.end())
  { if ( (*it) → getPrice() > max )
    { max = (*it) → getPrice(); out = (*it);
    } ++it;
  }
  return out;
}
```

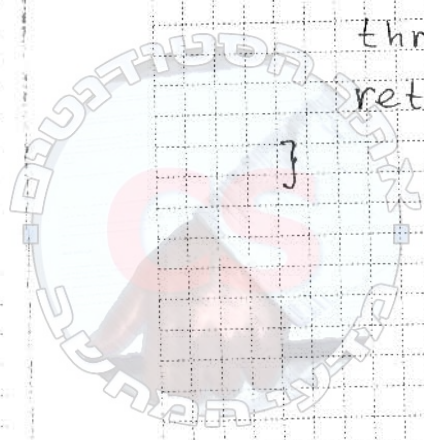
```
if ( pV.empty()) throw RecordNotFoundException;
```





```
void print AddressAndArnona (
{ vector < Parcel* > ; iterator it;
  it = pV.begin();
  while ( it != pV.end() )
  {
cout << "Address: " << (*it) -> street << "Arnona: " << (*it) -> getArnona() << '\n';
    cout << "Address:" << (*it) -> street << Arnona
    << "Arnona:" << (*it) -> getArnona() << '\n';
    ++it;
  }
}
```

```
Parcel* find By Address ( Parcel* char* st, int nm)
{ vector < Parcel* > :: iterator it;
  it = pV.begin();
  while ( it != pV.end() )
  {
    if ( (*it) -> number == nm
 && strcmp ( (*it) -> street, st) == 0 )
      && strcmp ( (*it) -> street, st) == 0 )
 return (*it);
      return (*it);
    ++it;
  }
  throw RecordNot FoundException ();
  return NULL; // dummy
}
```





```
void print()
{
    vector< Parcel * >::iterator it;
    it = pv.begin();
    while ( it != pv.end() )
    {
        (*it) -> print();
        ++it;
    }
}

} // end city class defs
```

```
class DuplicateRecordException: public std::exception
{
public:
    const char * what() const
    {
        return "Duplicate Record Exception";
    }
}
```

```
class RecordNotFoundException: public std::exception
{
public:
    const char * what() const
    {
        return "Record Not Found Exception";
    }
}
```

