פקולטה: מדעי הטבע
מחלקה: מדעי המחשב
שם הקורס: מבנה זיכרון ושפת C++
קוד הקורס: 7027810-2 כל הקבוצות - קבוצות 1,2,3,4
מועד __א___ סמסטר ___ב___תאריך בחינה:__4/7/2018___ **פתרון**
משך הבחינה: 3 שעות
שם המרצים: אראל סגל-הלוי, חרות סטרמן

יש לענות על כל השאלות במחברת הבחינה.
אין צורך להעתיק את השאלון למחברת - השאלון יתפרסם במודל לאחר הבחינה.

בשאלות 1-5, כששואלים "מה סוג השגיאה", יש לכתוב אחת מהאפשרויות הבאות:

- שגיאת קומפילציה (compilation error)
- שגיאת קישור (link error)
- שגיאת זמן-ריצה (runtime error)
- שגיאה לוגית (logic error)

כששואלים "מה גורם לשגיאה" יש לפרט ולציין מספרי שורות בהתאם לקוד המובא בשאלה.
כששואלים "איך לתקן את השגיאה" יש לציין מספרי שורות ולהסביר מה צריך לשנות בכל שורה/שורות.
בחלק מהמקרים ישנם כמה תיקונים אפשריים. ניקוד מלא יינתן רק לתיקון הטוב ביותר מנקודת מבט של הנדסת תוכנה, כפי שנלמד בכיתה.

יש לענות תשובות מלאות אך ממוקדות. לא יינתנו נקודות על תשובות עם טקסט מיותר שאינו קשור לנושא.

ניתן לענות על כל שאלה במילים "לא יודע" או "לא יודעת". במקרה זה השאלה לא תיבדק כלל, ואתם תקבלו 20% מהניקוד על אותה שאלה (מעוגל כלפי מטה).

אסור להשתמש בכל חומר עזר.

בהצלחה!!

שאלה 1 [10 נק']

נתונה התוכנית:

```
01  #include <iostream>
02  using namespace std;
03
04  class triplet {
05      double x,y,z;
06  public:
07      triplet(double x, double y,double z): x(x), y(y), z(z) {}
08      friend ostream& operator<< (ostream& out, const triplet& t);
09  };
10
11  ostream& operator<< (ostream& out, const triplet& t) {
12      out << "(" << t.x << "," << t.y << "," << t.z << ")";
13  }
14
15  int main() {
16      triplet t(1,2,3);
17      cout << t << endl;
18  }
```

כשמריצים make, מתקבל הפלט הבא:

```
main.cpp:13:1: warning: control reaches end of non-void function [-
Wreturn-type]
}
^
1 warning generated.
./a.out
make: *** [all] Illegal instruction (core dumped)
```

א [5 נק']. מה סוג השגיאה (קומפילציה / קישור / זמן-ריצה / לוגית)? מה בדיוק גורם לשגיאה?

שגיאת זמן ריצה [1 נק'].

באופרטור הפלט (שורות 11-13) אין פקודת return בסוף [1 נק'].

כתוצאה מכך, האופרטור מחזיר זבל - רפרנס למקום לא חוקי [1 נק'].

בשורה 17, הפעולה cout << t מחזירה רפרנס-זבל [1 נק'].

מייד לאחר מכן מנסים לכתוב לתוכו endl - הכתיבה נעשית לכתובת לא חוקית ולכן נזרקת חריגה על הוראה לא חוקית [1 נק'].

ב [5 נק']. יש לתקן את הבאג ע"י שינוי/הוספה של שורה אחת. איזה שינוי יש לבצע כך שהתוכנית תרוץ בלי שגיאות ותדפיס (1,2,3)?

אחרי שורה 12 ולפני שורה 13, להוסיף return out.

שאלה 2 [10 נק']

נתון הקוד החלקי הבא (שלוש נקודות מציינות קוד שהושמט כי אינו רלבנטי לשאלה):

```
01  #include <iostream>
02  using namespace std;
03
04  class LargeMatrix {
05  public: // ...
06    LargeMatrix(int width, int height) {
07      cout << "standard ctor" << endl; // ...
08    }
09    LargeMatrix(const LargeMatrix& other) {
10      cout << "copy ctor" << endl; // ...
11    }
12  };
13
14  int sum_of_values(LargeMatrix m) {
15    cout << "sum_of_values" << endl;
16    int result = 0;
17    // do calculation ...
18    return result;
19  }
20
21  int main() {
22    LargeMatrix m1(1000,1000);
23      // ...
24      cout << sum_of_values(m1) << endl;
25  }
```

התוכנית מתקמפלת ורצה, אבל יש בה באג הגורם לכך שהתוכנית רצה מאד לאט.

הפלט המתקבל כולל את ארבע השורות הבאות:

```
standard ctor
copy ctor
sum_of_values
0
```

א [5 נק']. מהו הבאג? (ציינו מספרי שורות).

בשורה 14, המשתנה m מועבר by value [1 נק'].

כתוצאה מכך, כשקוראים לפונקציה sum_of_values, נקרא הבנאי המעתיק [2 נק'].

הבנאי המעתיק של מטריצה גדולה, כנראה, מעתיק את כל המטריצה וזה לוקח הרבה זמן [2 נק'].

ב [5 נק']. יש לתקן את הבאג ע"י שינוי/הוספה של שורה אחת. איזה שינוי יש לבצע כך שהתוכנית תרוץ מהר ובלי שגיאות?

בשורה 14, להעביר את המשתנה m כרפרנס: ***LargeMatrix& m***.

שאלה 3 [10 נק']

נתונה התוכנית הבאה:

```
class XX {
   int ixx;
};

class YY: public XX {
   int iyy;
};

class ZZ {
   YY oyy;
   int izz;
};

int main() {
   ZZ ozz;
}
```

התוכנית בשפת C++. התוכנית מתקמפלת ורצה בלי שגיאות.

א [2 נק']. באיזה איזור בזיכרון נמצא המשתנה ixx (**מחסנית** / ערימה / אחר)?

ב [2 נק']. באיזה איזור בזיכרון נמצא המשתנה iyy (**מחסנית** / ערימה / אחר)?

ג [2 נק']. באיזה איזור בזיכרון נמצא המשתנה izz (**מחסנית** / ערימה / אחר)?

ד [2 נק']. התוכנית תורגמה לשפת Java. התוכנית הראשית נשארה ללא שינוי:

```
public static void main(String[] args) {
      ZZ ozz;
}
```

התוכנית מתקמפלת ורצה בלי שגיאות.

באיזה איזור בזיכרון נמצא המשתנה ixx (מחסנית / ערימה / **אחר**)?

[לא נמצא בשום מקום כי העצם לא נוצר - זה רק פוינטר].

ה [2 נק']. התוכנית תורגמה לשפת Java. התוכנית הראשית שונתה ל:

```
public static void main(String[] args) {
      ZZ ozz = new ZZ();
}
```

התוכנית מתקמפלת ורצה בלי שגיאות.

באיזה איזור בזיכרון נמצא המשתנה ixx (מחסנית / ערימה / **אחר**)?

[לא נמצא בשום מקום - העצם מסוג ZZ אמנם נוצר, אבל השדה מסוג YY לא נוצר - הוא רק פוינטר].

שאלה 4 [10 נק']

נתונה תוכנית המורכבת משלושה קבצים.

AbstractShape.h:

```
class AbstractShape {
public:
    virtual void plot();
};
```

Circle.h:

```
#include "AbstractShape.h"
#include <iostream>
class Circle: public AbstractShape {
public:
    void plot() { /* code for plotting a circle. not shown. */ }
};
```

main.cpp:

```
#include "Circle.h"
int main() {
    AbstractShape* s = new Circle();
  s->plot();
}
```

כשמריצים make מתקבלת הודעת השגיאה הבאה:

```
/tmp/main-3bcff2.o: In function `AbstractShape::AbstractShape()':
main.cpp:(.text._ZN13AbstractShapeC2Ev[_ZN13AbstractShapeC2Ev]+0x6): undefined
reference to `vtable for AbstractShape'
```

א [5 נק']. מה סוג השגיאה (קומפילציה / קישור / זמן-ריצה / לוגית)? מה בדיוק גורם לשגיאה?

שגיאת קישור [1 נק'].

בקובץ AbstractShape.h מגדירים פונקציה וירטואלית בשם plot, אבל לא מממשים אותה בשום מקום [1 נק'].

כתוצאה מכך לא נוצרת טבלת פונקציות וירטואליות עבור AbstractShape [1 נק'].

הבנאי של AbstractShape צריך לאתחל את ה-vptr של המחלקה כך שיצביע לטבלה הוירטואלית, אבל הטבלה לא קיימת [1 נק'].

המקשר מגלה את זה ומתלונן על הפניה לא מוגדרת ל-vtable for AbstractShape [1 נק'].

ב [5 נק']. יש לתקן את הבאג ע"י שינוי/הוספה של שורה אחת. איזה שינוי יש לבצע כך שהתוכנית תרוץ בלי שגיאות ותצייר עיגול?

כיוון שהמחלקה AbstractShape מייצגת צורה מופשטת, לא אמור בכלל להיות לה בנאי. הדרך הטובה ביותר להשיג מטרה זו היא להפוך את הפונקציה plot לוירטואלית-טהורה ע"י הוספת "=0" בשורה 4:

```
04      virtual void plot() = 0;
```

פתרון פחות טוב [4 נק'] הוא להוסיף מימוש ריק לפונקציה, למשל:

```
04      virtual void plot() {}
```

שאלה 5 [10 נק']

נתונה התוכנית:

```
01  #include <set>
02  #include <iostream>
03  using namespace std;
04
05  struct XX {
06      int x;
07      XX(int x): x(x) {}
08  };
09
10  int main() {  // the program should print "100"
11      multiset<XX> myset;
12      myset.emplace(1);
13      myset.emplace(0);
14      myset.emplace(0);
15      for (auto y: myset)
16          cout << y.x;
17  }
```

כשמריצים make מתקבל הפלט הבא:

```
/usr/bin/../lib/gcc/x86_64-linux-
gnu/7.1.0/../../../../include/c++/7.1.0/bits/stl_function.h:386:20: error: invalid operands to
binary expression ('const XX' and 'const XX')
{ return __x < __y; }
~~~ ^ ~~~
/usr/bin/../lib/gcc/x86_64-linux-gnu/7.1.0/../../../../include/c++/7.1.0/bits/stl_tree.h:2069:10:
note: in instantiation of member function 'std::less<XX>::operator()' requested here
__x = _M_impl._M_key_compare(__k, _S_key(__x)) ?
^
/usr/bin/../lib/gcc/x86_64-linux-gnu/7.1.0/../../../../include/c++/7.1.0/bits/stl_tree.h:2381:19:
note: in instantiation of member function 'std::_Rb_tree<XX, XX, std::_Identity<XX>,
std::less<XX>, std::allocator<XX> >::_M_get_insert_equal_pos' requested here
auto __res = _M_get_insert_equal_pos(_S_key(__z));
^
/usr/bin/../lib/gcc/x86_64-linux-
gnu/7.1.0/../../../../include/c++/7.1.0/bits/stl_multiset.h:448:16: note: in instantiation of
function template specialization 'std::_Rb_tree<XX, XX, std::_Identity<XX>, std::less<XX>,
std::allocator<XX> >::_M_emplace_equal<int>' requested here
{ return _M_t._M_emplace_equal(std::forward<_Args>(__args)...); }
^
main:12:8: note: in instantiation of function template specialization 'std::multiset<XX,
std::less<XX>, std::allocator<XX> >::emplace<int>' requested here
myset.emplace(1);
```

א [5 נק']. מה סוג השגיאה (קומפילציה / קישור / זמן-ריצה / לוגית)? מה בדיוק גורם לשגיאה?

שגיאת קומפילציה [1 נק'].

בשורה11 אנחנו יוצרים עצם מסוג multiset של XX.

הסוג multiset הוא בעצם תבנית (template) שבונה קבוצה מסודרת, ומשתמשת באופרטור "קטן מ" כדי לסדר [2 נק'].

אבל, לסוג XX אין אופרטור "קטן מ". השגיאה מתגלה כשהקומפיילר מקמפל את הקוד של התבנית - לכן השגיאה מופיעה בקוד של הספריה התקנית (stl). [2 נק'].

ב [5 נק']. איך אפשר לתקן את התוכנית כך שתרוץ בלי שגיאות ותדפיס 100?

אפשרות אחת היא להוסיף אופרטור "קטן מ" למחלקה XX. אבל כדי שהתוכנית תדפיס 100, האופרטור צריך לשים את 1 לפני 0. למשל, אפשר להוסיף בשורה 9:

```
bool operator< (const XX& a, const XX& b) { return a.x > b.x; }
```

פתרון נוסף: להוסיף אופרטור בתוך המחלקה - בין שורות 7,8:

```
bool operator< (const XX& other) const {return x > other.x;}
```

פתרון נוסף שהתקבל (למרות שלא לזה התכוונו) היה לשנות את מבנה-הנתונים מ-multiset ל-vector - וקטור לא צריך אופרטור השוואה.

שגיאות אפשריות:

- *חסר מימוש האופרטור עצמו (3 נק').*
- *האופרטור בכיוון הפוך כך שהתוכנית מדפיסה 001 (2 נק').*
- *שכחת const (נק' אחת לכל אחד).*

שאלה 6 [40 נק']

נתונה התוכנית:

```
01 #include <cmath>
02 #include <iostream>
03 #include <vector>
04 using namespace std;
05
06 class PositiveMatrix {
07     double* vals;
08     int rows,cols;
09 public:
10     PositiveMatrix(int rows, int cols):
11     rows(rows), cols(cols), vals(new double[rows*cols]) {
12         for (int i=0; i<rows*cols; ++i)
13             vals[i]=0.0;
14     }
15
16     double operator() (int x, int y) const {
17         return vals[x*cols+y];
18     }
19
20     double& operator() (int x, int y) {
21         return vals[x*cols+y];
22     }
23 };
24
25 int main() {
26     PositiveMatrix m1(10,4);
27     cout << m1(1,2) << endl; // prints 0 (OK)
28     m1(1,2) = 3.0;
29     cout << m1(1,2) << endl; // prints 3 (OK)
30
31     PositiveMatrix m2{m1};
32     m2(1,2) = 5;
33     cout << m1(1,2) << endl; // prints 5 (should print 3)
34
35   m1 = m2;
36   m1(1,2) = 7;
37     cout << m2(1,2) << endl; // prints 7 (should print 5)
38
39     m1(1,2) = -5.0; // should throw an exception "illegal value"
40 }
```

התוכנית מתקמפלת ורצה אולם המחלקה PositiveMatrix לא גמורה - יש להשלים בה מספר חלקים לפי הפירוט בסעיפים הבאים. **אין לשנות את התוכנית הראשית** - עליכם לוודא שהתוכנית הראשית כפי שהיא כתובה עכשיו רצה בלי שגיאות. אם השינוי שלכם דורש להחליף קוד שכבר קיים במחלקה PositiveMatrix - ציינו בפירוש את מספרי השורות שבהן יש להחליף את הקוד ומה יש לכתוב במקומו.

א [10 נקודות]. יש להוסיף קוד למחלקה PositivcMatrix כך שלא תהיה דליפת זיכרון בתוכנית.

להוסיף מפרק:

```
~PositiveMatrix()
{
   delete[] vals;
}
```

שגיאות אפשריות:

- *מחיקה בעזרת לולאה (3 נק').*
- *לשכוח לכתוב סוגריים (2 נק').*
- *מחיקה של rows, cols או מחיקה כפולה (5 נק').*
- *שגיאה אחרת בתחביר של פעולת המחיקה (1 נק').*

ב [10 נקודות]. יש להוסיף קוד למחלקה PositivcMatrix כך שבשורה 33 יודפס הערך הנכון - 3.

להוסיף בנאי מעתיק:

```
PositiveMatrix(const PositiveMatrix& other):
rows(other.rows), cols(other.cols), vals(new double[rows*cols])
{
   for (int i=0; i<rows*cols; ++i)
      vals[i]=other.vals[i];
}
```

שגיאות אפשריות:

- *חסר איתחול של אחד הפרמטרים (3 נק').*
- *לשכוח רפרנס בפרמטר (2 נק').*
- *לשכוח const בפרמטר (1 נק').*
- *מימוש נכון אבל של אופרטור {} (5 נק').*
- *מימוש נכון אבל החזרת עצם חדש במקום שינוי העצם הנוכחי (5 נק').*
- *לא יורדו נקודות על איתחול בתוך גוף הבנאי.*

ג [10 נקודות]. יש להוסיף קוד למחלקה PositivcMatrix כך שבשורה 37 יודפס הערך הנכון - 5.

להוסיף אופרטור השמה:

```
PositiveMatrix& operator=(const PositiveMatrix& other)
{
   if (this != &other)
   {
      delete[] vals;
      rows = other.rows;
      cols = other.cols;
```

```
        vals = new double[rows*cols];
        for (int i=0; i<rows*cols; ++i)
            vals[i]=other.vals[i];
    }
    return *this;
}
```

שגיאות אפשריות:

- להעתיק רק את התוכן בלי הגודל (5 נק').
- החזרת עצם חדש במקום שינוי העצם הנוכחי (5 נק').
- להעתיק את התוכן והגודל אבל לא להקצות מקום חדש כשהגודל שונה (3 נק').
- לזרוק חריגה כשהגודל שונה (2 נק').
- לשכוח לבדוק האם זה אותו אובייקט או לפחות אותו גודל (1 נק').
- לאתחל שדות ב"רשימת האיתחול" (2 נק').
- לשכוח רפרנס בפרמטר (2 נק').
- לשכוח const בפרמטר (1 נק').
- ערך מוחזר לא נכון (1 נק').
- חסרה מחיקה של התוכן הקודם (3 נק').
- חסרים סוגריים במחיקה (1 נק').

ד [10 נקודות]. יש להוסיף ולשנות קוד כך שכל ניסיון לשנות ערך במטריצה למספר שלילי (כמו בשורה 39) יגרום לזריקת חריגה.

כמו שעשיתם במטלה 6, יש להוסיף מחלקת פרוקסי ולהחזיר אותה מתוך אופרטור הסוגריים. למשל, אפשר להוסיף את הקוד הבא בשורה 5:

```
class negative_value { };

class Proxy {
    double& _d;
public:
    Proxy(double& d): _d(d) { }
    Proxy& operator= (double d) {
        if (d<0)
            throw negative_value();
        _d = d;
        return *this;
    }
    operator double() const { return _d; }
};
```

ולשנות את שורה 20 ל:

```
20   Proxy operator() (int x, int y) {
21          return Proxy(vals[x*cols+y]);
22   }
```

***הערה**: במקום לזרוק מחלקת-חריגה, אפשר לזרוק מחרוזת "negative value" (לא מורידים נקודות).*
***מפתח ניקוד**:*

- *על זריקת החריגה מקבלים 2 נק'.*
- *על מחלקת פרוקסי מקבלים 6 נק'.*
- *על התאמת הפרוקסי כך שאופרטור הפלט ימשיך לעבוד (למשל כתיבת אופרטור פלט מעודכן או הוספת אופרטור המרה) - 2 נק'.*

הערה: הרבה סטודנטים כתבו שצריך לשנות את אופרטור ההשמה בין double ל-double, למשל:

```
double& operator=(double& a, double b) {
        if (b<0)
                throw negative_value{};
        return (a = b);
}
```

אז קודם-כל זה לא יתקמפל כי אי אפשר להעמיס אופרטורים על טיפוסים פרימיטיביים.
אבל הבעיה הגדולה יותר היא, שאילו זה היה מתקמפל, היתה כאן שגיאה לוגית חמורה - זה למעשה משנה את ההשמה של מספרים ממשיים **בכל התוכנית** ולא רק במחלקה PositiveMatrix! אם במקום אחר בתוכנית, בלי שום קשר ל-PositiveMatrix, מישהו ינסה למשל לכתוב:

```
double a = -1;
```

הוא יקבל חריגה.
הפתרון הוא לכתוב מחלקת פרוקסי כמו שעשיתם במטלה 6.

שאלה 7 [10 נק']

נתון קוד חלקי של המחלקה Fraction. עליכם לכתוב קוד חדש, כך שכל עצם מהמחלקה:

```
Fraction<int>
```

יתנהג כמו Fraction הנתון, אבל יהיה ניתן גם לייצר עצמים מהטיפוס:

```
Fraction<short>
Fraction<long>
```

בלי לכתוב קוד נוסף. אין צורך לממש אופרטורים או פונקציות אחרות שלא מופיעות בקוד להלן. עליכם לכתוב קוד מלא גם אם חלקו העתקה מהקוד להלן.

```
class Fraction {
   private:
      int nom, den;
   public:
      Fraction(int nn, int dn=1): nom(nn), den(dn) { }

      int Nom() const {
         return nom;
      }

      int Den() const {
```

```
        return den;
    }

    Fraction operator+ (const Fraction& other) {
                int nn = nom * other.den + den * other.nom;
                int dd = den * other.den;
                return Fraction(nn, dd);
    }
};
```

פתרון:

```
template <typename T>

class Fraction
{
   private:
      T nom, den;
   public:
      Fraction(T nn, T dn=1): nom(nn), den(dn) { }

      T Nom() const {
         return nom;
      }

      T Den() const {
         return den;
      }

      Fraction operator+ (const Fraction& other) {
            T nn = nom * other.den +
               den * other.nom;
            T dd = den * other.den;
            return Fraction(nn, dd);
         }
};
```

שגיאות אפשריות:

- *לשכוח בשורה ראשונה template (3 נק');*
- *לשכוח להמיר ל-T (1 נק' לכל אחד);*
- *שני סוגים או יותר ב-template (שלוש נק');*
- *שגיאת תחביר משמעותית ב-template (עד 5 נק');*