תכנות מתקדם 2

מועד א' תשע"ד

15.7.14

המחלקה למדעי המחשב

89-211

1	30 נק'	
2	15 נק'	
3	25 נק'	
4	10 נק'	
5	20 נק'	
סה"כ:	100 נק'	

מרצה: יריב טל.

מתרגל: איגור רוכלין.

משך הבחינה: שלוש שעות. אין הארכה.

חומר עזר: אסור להכניס כל חומר עזר.

הנחיות כלליות: רצוי לענות בגוף הבחינה. אחרי כל שאלה יש מקום לתשובות.
אם עניתם על שאלה במחברת – ציינו זאת בגוף הבחינה!
<u>חובה להגיש את המחברת ביחד עם טופס הבחינה בסיום הבחינה.</u>
חובה לענות על השאלות בעברית, אלא אם יש אישור מהדיקן.

הנחיות טכניות: במידה ונדרשתם לתת נימוק, אזי הוא <u>חובה</u>. כלומר, <u>תשובה לא מנומקת לא תקבל נקודות כלל.</u>
<u>נימוק לא נכון יגרור פסילת השאלה.</u>
בכל השאלות – מספר השורות שניתנו לפתרון אינו מרמז על אורך התשובה.

בהצלחה ☺

שאלה 1: Code Improvements (30 נקודות)

נתון הקוד הבא:

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

class Node{
public:
   int m_data;
   Node *m_next;
   Node(int x, Node *n = NULL) : m_data(x), m_next(n) { }
};

1.  class LinkedList {
2.     Node *m_head;
3.
4.  public:
5.     LinkedList() {
6.        m_head = NULL;
7.     }
8.
9.     ~LinkedList() {
10.       while (m_head)
11.          erase(m_head);
12.    }
13.
14.    void addNode(int value) {
15.          m_head = new Node(value, m_head);
16.       }
17.
18.    void erase(const Node *p) {
19.       Node *c;
20.       if (m_head == p) {
21.          m_head = p->m_next;
22.          delete p;
23.       }
24.
25.       c = m_head;
26.       while(c != NULL && c->m_next != p)
27.          c = c->m_next;
28.       c->m_next = p->m_next;
29.       delete p;
30.    }
31. };
```

א. (6 נקודות) בקוד הנתון יש באג שגורם לתוצאות לא נכונות ואף לביצוע כפול של delete של p בפונקצייה erase.

הציעו שיטה אשר הייתה מובילה לזיהוי המצב הבעייתי (בהנחה שקימפלנו ב-debug). הסבירו מדוע השיטה שאתם מציעים הייתה מובילה לזיהוי בעייה.

השיטה: אחרי כל delete נאפס את המצביע ל-NULL, ולפני כל delete נבדוק עם assert שהמצביע לא NULL.

שימו לב: delete על NULL זה חוקי ופשוט לא עושה כלום.

כמו כן – delete לא משנה את המצביע בשום צורה ולכן לא ניתן דרך המצביע לדעת שכבר בוצע delete.

בנוסף – כאשר p הוא NULL, הגישה p->m_next היא לכתובת 4 ולא לכתובת 0 ולכן לא בהכרח תגרום לקריסה (כפי שיכול להעיד כל מי שנתקל בתוכנית שעובדת למרות שמצביע היה NULL).

ב. (18 נקודות) איכות הקוד הנתון שנויה במחלוקת...
הציעו 6 שיפורים, לא כולל הוספת assert׳ים והשיפור/שיטה שהצעתם בסעיף א׳, אשר עשויים להפוך את *הקוד הממוספר במספרי שורות* ליותר קריא או להוריד את הפוטנציאל שלו לבאגים.
נמקו את הצורך בכל שיפור! רק 6 הצעות השיפורים *הראשונות* תיבדקנה.
אין לחזור על אותו *סוג* שיפור פעמיים. אם סוג בעייה חוזר על עצמו רשמו רק אחת מהפעמים.
שימו לב – הצעה של שינוי שאינו משפר את הקוד *עלולה* לגרור *הורדת* ניקוד!

להלן רשימה לא ממצה של שיפורים אפשריים:

	שורה	הבעיה והשיפור
1	10	יש לפתוח בלוק גם לשורה אחת. נוסיף { }
2	9	להפוך destructor לווירטואלי
3	15,16	הזחה לא נכונה (יותר מדי פנימה)
4	18,19	c, p שמות לא משמעותיים
5	19	משתנה c לא מאותחל
6	19	משתנה c מוגדר לפני הזמן (צריך להיות בשורה 25)
7	26	קבוע צריך להיות מצד שמאל – NULL != c (שימו לב שזה לא נכון ל-p == m_head כי ל-p יש const על הערך אליו הוא מצביע ולא על המצביע)
8	26	אסור להסתמך על קדימויות של אופרטורים – יש להשתמש בסוגריים
9	18	const Node *p – ה-const מטעה כי אנחנו הולכים לבצע בו שינוי דרסטי: למחוק אותו.

ג. (6 נקודות) אילו הנחות מניח הקוד של erase? הוסיפו עד 3 assert׳ים המתעדים ובודקים את ההנחות בקוד. *שימו לב:* הוספת assert׳ים לא מתאימים או מיותרים עלולה לגרור הורדת נקודות.

	שורה	ההנחה וה-assert שיש לשים לפני השורה
1	19	מניחים ש-P מצביע לאיבר, לכן: assert(NULL != p)
2	28	מניחים ש-p אכן נמצא ברשימה ולכן אחרי הלולאה מתקיים assert(NULL != c)

שימו לב: אלה 2 ההנחות המרכזיות בקוד שציפיתי שתגלו. קיבלתי גם מקרים שפיזרו את ה-assert על p בשני מקומות: שורות 12 ו-28 ואז יש 3 assert׳ים ולא 2 (זו אחת הסיבות למה היה מקום ל-3 assert׳ים).

שאלה 2: Templates Meta-Programming (15 נקודות)

בכיתה השתמשנו בטמפלטים בכדי להגדיר מבנה המייצג רשימה של טיפוסים:

```
struct Nil
{
      typedef Nil Head;
      typedef Nil Tail;
};

// Usage example: Cons<int, Cons<long, Cons<Foo> > >
template <class HEAD, class TAIL = Nil>
struct Cons
{
      typedef HEAD Head;
      typedef TAIL Tail;
};
```

נגדיר את פעולת המיסתורין הבאה על רשימות של טיפוסים:

```
(1)     template <class TLIST, class T> struct Mystery;

(2)     template <class T>
        struct Mystery<Nil, T>
        {
              enum { value = -1 };
        };
(3)     template <class TAIL, class T>
        struct Mystery<Cons<T, TAIL>, T>
        {
              enum { value = 0 };
        };
(4)     template <class HEAD, class TAIL, class T>
        struct Mystery<Cons<HEAD, TAIL>, T>
        {
        private:
              enum { temp = Mystery<TAIL, T>::value } ;
        public:
              enum { value = (-1 == temp) ? -1 : 1 + temp };
        };
```

ואת ההגדרות הבאות של מחלקות:

```
class Blabla { };
class Something { };
class Bigger : public Something { };
```

א. (5 נקודות) כיתבו אילו אינסטנסיאציות יבצע הקומפיילר ומה יודפס כתוצאה מביצוע שורת הקוד הבאה (נמקו!):

```
std::cout <<
   Mystery<Cons<Blabla, Cons<Something> >, Blabla>::value;
```

נבין קודם מה הפרמטרים ל-Mystery (נזכור גם של-Cons יש פרמטר ברירת מחדל ונרשום אותו

מפורשות): Mystery< **Cons<BlaBla,Cons<Something,Nil>** > , *Blabla* >

ספשיאליזציה 2 לא מתאימה כי הפרמטר הראשון שלה צריך להיות Nil ופה יש Cons מורכב.

ספשיאליזציה 3 מתאימה: אם נציב Tail = Cons<Something, Nil>, T=Blabla ואז נציב אותם בתנאי

של הספשיאליזציה: Mystery<Const<T,TAIL>, T> נראה שנקבל חזרה את ההגדרה המקורית.

ספשילאיזציה 4 גם מתאימה – אבל ספשיאליזציה 3 יותר ספציפית ולכן הקומפיילר יעדיף אותה.

לכן, יודפס 0.

שימו לב: מעט מאוד אנשים נימקו למה ולכן הוענקה להם 1+ נקודת בונוס, גם על נימוק חלקי.

נקודת בונוס נוספת הוענקה למי שהסביר שגם 4 מתאימה ולמה 3 מועדפת על פני 4.

ב. (5 נקודות) כיתבו אילו אינסטנסיאציות יבצע הקומפיילר ומה יודפס כתוצאה מביצוע שורת הקוד הבאה (נמקו!):

```
std::cout <<
   Mystery<Cons<Blabla, Cons<Something> >, Bigger>::value;
```

רק ספשיאליזציה 4 מתאימה: HEAD=BlaBla, TAIL=Cons<Something, Nil>, T=Bigger.

כדי לחשב את value יש לחשב את temp: Mystery<Cons<Something, Nil>, Bigger>::value

לכן תופעל שוב ספשיאליזציה 4: HEAD=Something, TAIL=Nil, T=Bigger

כדי לחשב את value יש לחשב את temp: Mystery< Nil, Bigger>::value וזה ספשיאליזציה 2.

לכן, יודפס 1-.

שימו לב: טמפלטים זה copy-paste לא OO– לכן הירושה של Bigger חסרת חשיבות.

1+ נקודות בונוס הוענקו למי שרשם את כל הפירוט נכון, 4 נקודות (מתוך 5) ניתנו למי שטעה ולא רשם

את הביצוע השני של ספשיאליזציה 4, 3 נקודות הוענקו למי שביצע תחת ההנחה שיש משמעות לירושה.

ג. (5 נקודות) מה עושה הטמפלט? תארו את התוצאה עבור כל המקרים האפשריים. ניתן להעזר בקוד הנתון למטה בכדי לקבל הבנה טובה יותר:

```
std::cout <<
   Mystery<Cons<Blabla, Cons<Something> >, Something>::value;
```

יודפס 1 (תנסו לפתור לבד).

הטמפלט מחזיר את האינדקס ברשימה שבו מופיע האיבר מטיפוס T לראשונה. אם הרשימה לא

מכילה איבר מטיפוס T אזי נקבל 1-.

שאלה 3: Unit Testing, Event Programming & Exception Safety (25 נקודות)

נתון הקוד הבא:

```
class BankAccess {
   static BankAccess s_instance;
public:
   static BankAccess &instance() { return s_instance; }

   void transferMoney(unsigned int fromAccountNum, unsigned int toAccountNum,
              unsigned char dollarAmount); // *** throws std::exception on failure

   size_t getCurrentAmount(unsigned int accountNum);
};

class TrasferMoneyButton : public Button
{
public:
   TrasferMoneyButton(unsigned int from, unsigned int to)
      : m_from(from), m_to(to) { }

   //..

   void onClick() {
      BankAccess::instance().transferMoney(m_from, m_to, 250);
      BankAccess::instance().transferMoney(m_from, m_to, 250);
   }

   //...

   unsigned int m_from;
   unsigned int m_to;
};
```

המחלקה TransferMoneyButton מגדירה כפתור UI המאפשר העברת כסף בין שתי חשבונות בנק.
בעת לחיצה על הכפתור נקראת הפונקציה onClick.
ברצוננו לכתוב בדיקות unit testing עבור הפונקצייה onClick.
באופן טבעי, לא נרצה להעביר כספים בין חשבונות אמיתיים בזמן הבדיקות.

א. (5 נקודות) הניחו שהבנק מספק לכם שני מספרי חשבונות לא אמיתיים למטרת בדיקות: 101 ו-102. כיתבו מטודה testOnClickSuccess הבודקת את הצלחת הפונקצייה onClick בהעברת 500 דולר מחשבון 101 לחשבון 102. השתמשו ב-assert'ים בכדי לוודא את תוצאות הפעולה בדרכים רבות ככל האפשר.

```
void testOnClickSuccess() {

   int sumBeforeA = BankAccess::instance().getCurrentAmount(101);

   int sumBeforeA = BankAccess::instance().getCurrentAmount(102);

   TransferMoneyButton b(101, 102);

   b.onClick();

   assert(sumBeforeA – 500 == BankAccess::instance().getCurrentAmount(101));
```

```
    assert(sumBeforeB + 500 == BankAccess::instance().getCurrentAmount(102));
}
```

ב. (5 נקודות) הניחו שהבנק מספק לכם גם מספר חשבון לא קיים: 99. כיתבו מטודה testOnClickFailure הבודקת שבעת כישלון בהעברת 500 דולר מחשבון 101 לחשבון 99 (transferMoney אמורה לזרוק std::exception) הפונקצייה onClick אכן זורקת std::exception. השתמשו ב-assert׳ים בכדי לוודא את תוצאות הפעולה בדרכים רבות ככל האפשר.

```
void testOnClickFailure() {
    TransferMoneyButton b(101, 102);
    try{
        b.onClick();
        assert(!"exception was thrown!");
    }
    catch(std::exception &) { }
}
```

2 נקודות בונוס ניתנו למי שגם בדקה שהסכום בחשבון לא השתנה.

ג. (5 נקודות) כעבור זמן הבנק הודיע לכם שעקב שינוי פנימי אצלם לא ניתן יותר להשתמש בחשבונות לא אמיתיים ועליכם לוודא שבמהלך הבדיקות שלכם אינכם קוראים למטודות ב-BankAccess מכיוון שהקריאות משפיעות על חשבונות וכספים אמיתיים. אילו שינויים יש לבצע בקוד כך שניתן יהיה לבצע עבורו Unit Testing למרות הדרישות החדשות? הסבירו במילים ונמקו את הצורך בכל שינוי.

עיקר הבעייה בכך ש-onClick ״נעולה״ על שימוש ב-BankAccess ואין לנו שליטה על כך.

הפתרון: להחזיר את השליטה לידיים שלנו ולשלוט על איזה אובייקט ייספק ל-onClick שרותים של ״העברת כסף״.

שיטה אחת אפשרית היא Dependency Injection. ל-onClick יש *תלות* פנימית ב-BankAccess,

במקום זה *נזריק* לה מבחוץ במי להשתמש ע"י פרמטר ל-onClick.

בצורה זו נוכל להעביר מחלקה שעובדת מול הבנק (בעזרת שימוש ב-BankAccess) בסביבת אמת ומחלקה שעובדת מול סביביה מדומה בסביבת בדיקות.

ד. (4 נקודות) בזמן בדיקות ע"י אחד הבודקים התגלתה בעיה – כאשר לוחצים על הכפתור של TransferMoneyButton מנשק המשתמש (ה-UI) נתקע למספר שניות. לאחר בדיקות נוספות הגיעו הבודקים למסקנה שהבעיה נובעת מכך שהפונקצייה transferMoney לוקחת מספר שניות.
הסבירו מדוע אורך זמן הביצוע של transferMoney גורם למנשק המשתמש להיתקע (רמז: היזכרו איך עובד המנגנון של events)

ה-UI מציב event"ים בתור. ההוצאה מהתור מתבצעת ע"י ה-message pump אשר מוציא איבר איבר וקורא לפונקציית טיפול (handler) שתטפל באירוע. הטיפול באירוע הוא **על אותו thread** של ה-Message pump ולכן הלולאה לטיפול באירועים לא מתבצעת ואף אירוע נוסף לא יכול להיות מבוצע כל עוד פונקציית הטיפול הנוכחית לא סיימה את פעולתה.

ה. (6 נקודות) בהנחה ש-transferMoney מקיימת strong exception safety guarantee, מה ה-exception safety guarantee של onClick? נמקו!

נשים לב ש-()instance היא nothrow (בסה"כ מחזירה רפרנס למשתנה).

()onClick מורכבת מ-2 קריאות ל-transferMoney אשר לפי ההגדרות מקיימת strong. מכיוון שהיא עצמה לא מקצה משאבים אלא הם מוקצים (אם בכלל) ע"י transferMoney הרי שבוודאות אין בעייה של leak. כמו כן – כל קריאה ל-transferMoney בהכרח תשאיר את המערכת במצב תקין ולכן יש כאן לפחות Basic. נשאר לראות האם מתקיים "הכל או כלום" בשביל שיהיה Strong.

לשם כך צריך להראות שאם נזרק exception **איפשהו** במהלך ביצוע הקוד אז נוחזר לערכים שלפני תחילת הביצוע.

אם ה-exception ייזרק בשורה הראשונה אז אכן זה מתקיים מכיוון ש-transferMoney תחזיר את הערכים למה שהיו לפני הביצוע. לעומת זאת, אם ה-exception נזרק בקריאה השנייה ל-transferMoney אז נמצא עצמנו במצב שבו כבר הועברו 250 דולר. לכן לא מתקיים "הכל או כלום" והפונקצייה מקיימת Basic.

שאלה 4: Java (10 נקודות)

א. (5 נקודות) הסבירו בקצרה מה זה Thread, מה זה Runnable ומה ההבדלים ביניהם.

Thread – מחלקה המממשת הרצת קוד ב-thread נפרד.

Runnable – אינטרפייס המשמש להעברת משימות לביצוע ע"י thread נפרד (ע"י מימוש פונק' run

והעברת המחלקה המומשת ל-thread לביצוע).

ההבדל העיקרי: מ-Thread צריך לרשת (מה שלא תמיד אפשר כי ניתן לרשת רק ממחלקה אחת),

ב-Runnable ניתן להשתמש כאשר לא ניתן לרשת. (ניתן בונוס של נקודה למי שציין שהצורך ל-

Runnable נובע מכך שלא תמיד ניתן לרשת מ-Thread עקב מגבלת הירושה ממחלקה אחת).

ב. (5 נקודות) הסבירו מה הם ההבדלים בין JDBC ל Hibernate.

JDBC – גישה לבסיסי נתונים בצורה ישירה, ע"י שליחת פקודות SQL.

Hibernate – גישה לבסיסי נתונים דרך שכבה מתווכת המתאמת בין טבלאות בסיס הנתונים לאובייקטים

כך שהעבודה מול בסיס הנתונים היא OO.

שאלה 5: Java (20 נקודות)

א. (10 נקודות) מה הפלט של קטע קוד הבא:

```
public class ThreadTest {
	public static void main(String[] args) throws InterruptedException{
		new Thread(new Runnable(){
			public void run() {
				while(true){
					System.out.println("Home");
					try {
						Thread.sleep(1000);
					} catch (InterruptedException e) {}
				}
			}
		}).start();
		Thread.sleep(500);
		new Thread(new Runnable(){
			public void run() {
				while(true){
```

```
                                System.out.println("Sweet");
                                try {
                                        Thread.sleep(1000);
                                } catch (InterruptedException e) {}
                        }
                }
        }).start();
    }
}
```

Home

Sweet

Home

Sweet

...

(ניתן בונוס של נקודה על הסברים מפורטים למה)

ב. (10 נקודות) נתונים a.jsp הבא:

```
<%@ page import="java.lang.*"%>
<html>
<body>
        Result for <%=request.getParameter("a1")%>
        <%
        int i=Integer.parseInt(request.getParameter("t1"));
        int j=Integer.parseInt(request.getParameter("t2"));
        Integer k;
        String str=request.getParameter("a1");
        if(str.equals("Addition"))
                k=i+j;
        if(str.equals("Multiplication"))
                k=i*j;
        if(str.equals("Division"))
                k=i/j;
        %>
        Result is
        <%
        if(k == null)
                System.out.println("Error");
        else
                out.println(" "+k);
        %>
```

```
</body>
</html>
```

ו-main.html הבא:

```
<html><title>Mathematics Example</title><body>
	<form method="post" action="a.jsp">
		Enter mathematics action: <input type="text" name="a1" value=""><br>
		Enter first Value: <input type="text" name="t1" value=""><br>
		Enter second Value: <input type="text" name="t2" value=""><br>
		<input type="submit" name="result">
	</form>
</body></html>
```

הסבירו מה מבצע הקוד וציינו את הפלטים האפשריים של 2 הקבצים האלה (רמז: שימו לב לאפשרות של קלטים "בעיתיים").

עבור קלט תקין יוחזר למשתמש Result for <value of a1> Result is <value of k> כאשר k

הוא תוצאת הפעולה המתמטית.

עבור a1 שאינו נתמך k יהיה null ולכן יוחזר למשתמש Result for <value of a1> Result is

ויודפס Error בצד השרת (ב-console).

עבור t1, t2 שאינם מספריים ייזרק exception. כנ"ל עבור a1=Division ו-t2=0.