גיליון תשובות

מספר נבחן:_____

Q1	Q2
Q3	Q4
Q5	Q6
Total	

שאלה 1 (30 נקודות)

סעיף א (10 נקודות)

על אף שבגירסה הראשונה מבנה התור ClicksList תמיד יהיה (L,R,L) בגלל שהקריאה מ ClicksList והפעלת Firemen הן בשתי פעולות שונות שאינן תחת סנכרון במתודה run() של Left ו Right נקבל עבור שתי הגרסאות את כל האפשרויות:

4,3,4 [עבור LLR]
4,5,4 [עבור LRL]
6,5,4 [עבור RLL]

סעיף ב (10 נקודות)

המתודה onStop() תוסיף לתור ClicksList תו "S" המציין בקשה לסיום פעולת הת'רדים מכאן ואילך. כאשר קוראים הת'רדים תו זה מסתיימת מתודת run()

```
class ClicksList
{

   // remove the first string in the list if equals to given parameter s
   // returns false if stop control ("S") was identified
   public synchronized boolean get(String s) throws InterruptedException
   {
      while (list_.size() == 0 ||
             ! (list_.getFirst().equals(s) ||
             ! list_.getFirst().equals("S")
         wait();
      if (list_.getFirst().equals("S"))
         return false;
      list_.removeFirst(); // Remove first element of list
      notifyAll();
      return true;
   }
   ...
}

class UserInterface extends Thread
{
```

```
      ...

      public void onStop() {
         try {
            clicksList_.add("S");
         } catch (InterruptedException e) {
         }
      }
      ...
   }

   class Left extends Thread
   {
      ...

      public void run() {
         while (true) {
            try{
               if (clicksList_.get("L"))
                  firemen_.moveLeft();
               else
                  return;
            } catch (InterruptedException e) {
            }
         }
      }
      ...
   }


   class Right extends Thread
   {
      ...

      public void run() {
         while (true) {
            try{
               if (clicksList_.get("R"))
                  firemen_.moveLeft();
               else
                  return;
            } catch (InterruptedException e) {
            }
         }
      }
      ...
   }
```

סעיף ג (5 נקודות)

אין צורך לסנכרן את _v כי _x הוא final ולא משנה את ערכו.
אין צורך לסנכרן את _f כי getX() מסונכרנת.
מי שדרש/ה סינכרון על _f כדי למנוע תזוזה של האלונקה תוך פיספוס זיהוי תפיסת הניצול, קיבל את מלוא הנקודות, אף שזה לא יעזור במקרה זה.

סעיף ד (5 נקודות)

```
class Firemen
{
   ...

   public synchronized void moveLeft()
   {
      if (x_ > MIN_X) {
         synchronized (getThrower().getVictim()) {
             if (!(getThrower().getVictim().getX() = x_ &&
                   getThrower().getVictim().getY() = 1))
             x_--;
         }
      repaint();
   }

   public synchronized void moveRight()
   {
      if (x_ < MAX_X)
         synchronized (getThrower().getVictim()) {
             if (!(getThrower().getVictim().getX() = x_ &&
                   getThrower().getVictim().getY() = 1))
             x_++;
      repaint();
   }

   ...
}
```

<u>הערה</u>: יש לסנכרן גם את הקריאה v_.fall() במתודה run() של Thrower, אך מאחר והשינויים הנדרשים צומצו ל Firemen, לא ירדו על כך נקודות – מי שהזכיר צורך זה קיבל בונוס של נקודה.

שאלה 2 (10 נקודות)

סעיף א (5 נקודות)

במחלקה <iostream> אין שימוש.
המחלקה B נידרשת להעברת פרמטרים והחזרת ערכים ממתודות, ועל כן ניתן רק להצהיר עליה (ה includes יכתבו בקובץ המימוש x.cpp)
C,string, list, D, הינם טיפוסים של שדות במחלקה X ועל כן נידרש עבורם include.

```
// Contents of x.h
#include <list>
#include <string>
#include "d.h"  // class D
#include "c.h"  // class C

class B;

class X {
```

```
public:
    X( const C& );
    D Function1( int, char* );
    D Function1( int, C );
    B& Function2( B );

private:
   std::string  name_;
   std::list<C> clist_;
   D d_;
};
```

סעיף ב (5 נקודות)

מגדירים מחלקה Ximpl בקובץ אחר, המחזיקה את השדות של X , ומבצעת את ה includes הנחוצים. המצב הפנימי של X מורכב ממצביע ל Ximpl, כך שכל פניה לשדות עוברת דרכו. אין צורך לבצע include כי הוא מצביע וניתן להסתפק בהצהרה על קיומו (class XImpl).

```
// Contents of x.h
class XImpl;
class B;
class C;

class X {

public:
    X( const C& );
    D Function1( int, char* );
    D Function1( int, C );
    B& Function2( B );

private:
   Ximpl* pXImpl_;
};

// Contents of ximpl.h
#include <list>
#include <string>
#include "d.h"  // class D
#include "c.h"  // class C

class XImpl {
public:
   std::string  name_;
   std::list<C> clist_;
   D d_;
};
```

שאלה 3 (20 נקודות)

סעיף א.1 (5 נקודות)

ה destructor של Person לא ווירטואלי, מה שהופך את הפעולה

```
delete p;
```

ללא מוגדרת (גורם להתרסקות).

תיקון המחלקה Person:

```
virtual ~Person() {}
```

סעיף א.2 (7 נקודות)

ב destructor של Student מנוקה זיכרון שאינו באחריותו (התוכן של _courses לא מוקצה על ידו), דבר הגורם במקרה שלנו להתרסקות התוכנית עקב פניה זיכרון משוחרר – ה destructor של s, שיחרר את הזיכרון של courses.

תיקון המחלקה Student:

```
virtual ~Student() {}
```

סעיף ב (3 נקודות)

קטע הקוד לא מתקמפל.
ה constructor של Student מצפה לקבל reference ל string ותחת זאת נשלחת לו המחרוזת "Rina" אשר אינה מוגדרת בזיכרון וממילא לא ניתן להגדיר עבורה reference (למעשה זו העברה by value על ידי העתקה למחסנית של המחרוזת)

תיקון קטע הקוד:

```
{
   std::string name("Rina");
   Student s(name,37);
   std::vector<int*>::const_iterator it;
   for (it = s.getCourses().begin(); it != s.getCourses().end(); ++it)
      std::cout << (*(s. getCourses().at(0)));
}
```

סעיף ג (5 נקודות)

המתודה getCourses() ב Student מחזירה const reference לווקטור של הקורסים _courses ועל כן לא ניתן להפעיל עליו מתודות שאינן const דוגמת push_back.
כדי למלא אחר דרישת השאלה בדבר הוספת קורס, יש לבצע const cast:

```
{
   const_cast<std::vector<int*>&>(student.getCourses()).push_back
                                                        (new int(1200));
}
```

שאלה 4 (18 נקודות)

א	X
ב	X
ג	V
ד	X
ה	X
ו	V
ז	X
ח	V
ט	X

שאלה 5 (12 נקודות)

הצעה ראשונה

חסרונות

1. מספר פעולות מבוצעות על ידי מספר רב של ת'רדים מעל משאב מסונכרן אחד:
- קריאה מתור העבודות של ה thread pool
- פניה של מספר רב של לקוחות ל port אחד.

2. הת'רד של ה selector מבצע עבודה רבה 999: זיהוי ארועים על ה channels, טיפול בבקשות התחברות, ופיענוח הקלט במספר רב של ערוצים. שאר 999 הת'רדים ממתינים לפיענוח ההודעות והגדרתם כמשימות בתור של ה thread pool.

3. קיימת מגבלה על מספר ה channels במחלקה Selector.

יתרון

חלוקה שווה של עבודה בין הת'רדים ב thread pool

הצעה שנייה

חסרונות

1. חלוקה לא שווה של ביצוע עבודות על ידי הת'רדים ב thread pools השונים, במידה והתפלגות הפניה לעשרת ה ports בהתחברות הלקוחות לא הייתה אחידה.

2. מספר רב של ת'רדים בתהליך אחד.

<u>יתרונות</u>

כל החסרונות שצויינו בהצעה הראשונה נמנעות.

הצעה שלישית

<u>חסרונות</u>

1. מעבר בין תהליך לתהליך כרוך ב context switch כבד.

2. תקשורת בין reactor אחד למשנהו (אם נדרשת) מסובכת יותר מתקשורת בין שני ת'רדים באותו תהליך.

<u>יתרונות</u>

1. מספר קטן של ת'רדים בתהליך אחד.
2. ניתן להריץ את ה reactors על מחשבים שונות (השינוי היחיד: הלקוח מקבל רשימת זוגות של <host,port> תחת רשימה של ports)

הבחירה

באפליקציה הנתונה, נראה לוועדה, כי יש להעדיף את ההצעה השנייה עקב ריבוי (כמותית ואיכותית) יתרונותיה ומיעוט חסרונותיה (כמותית ואיכותית).

שאלה 6 (10 נקודות)

סעיף א (3 נקודות)

יש להוסיף את השדה Prob לטבלה WordsAnalyses.

סעיף ב (7 נקודות)

```
SELECT      Analyses.POS, Analyses.Gender, Analyses.Number

FROM        Analyses, Words, WordsAnalyses

WHERE       Words.Str = WordsAnalyses.WordStr       AND
            Analyses.ID = WordsAnalyses.AnalysisID  AND
            Words.Str = 'מספרים'

ORDER BY    WordsAnalyses.Prob Asc
```