מטלה מספר 6 – שפות תכנות והידור (ML)

הנחיות כלליות

- הגשה דרך מערכת ההגשות
- התרגיל צריך לרוץ על שרתי האוניברסיטה u2.
- יש לצרף בהגשה את הקובץ Ex6.sml.
- בסוף הקובץ יש להוסיף את השורה **(use "test.sml")**
- בראש הקובץ יש להוסיף הערה בה רשום:
  - שם הסטודנט
  - ת.ז.
  - מספר קבוצה
  - שם משתמש
- הערה ב- ML היא ע"י (* Comment *)

במטלה אנו נתרגל את השפה ML.

- מותר להשתמש בפונקציות שנלמדו בשיעור **בלבד**
- אסור להשתמש ב- ref (או כל רעיון אימפרטיבי במטלה), הגשה עם חלק אימפרטיבי לא תיבדק!

בכל החלקים יש להשתמש ב- Exception הבא:

```
exception IllegalArgumentException;
```

במקרה ובפונקציה מתקבל קלט לא תקין, משמע ההנחות של הפונקציה לא מתקיימות, אנו נזרוק את Exception מהטיפוס: IllegalArgumentException.
(משמע לא נחזיר ערך, אלא רק נזרוק exception).

חלק 1 (ניתן לפתור רק לאחר שיעור 9)

- **הפונקציה rotate**

```
val rotate = fn : 'a list * int -> 'a list
```

קלט: ls – רשימה מטיפוס כלשהוא
n – מספר שלם חיובי
פלט: מחזיר את הרשימה ls מסובבת n פעמים

```
- rotate ([1,2,3,4,5], 1);
val it = [2,3,4,5,1] : int list
- rotate ([1,2,3,4,5], 3);
val it = [4,5,1,2,3] : int list
- rotate ([1,2,3,4,5], 6);
```

שים/י ♥: במקרה ומתקבל n<0 יש לזרוק Exception מטיפוס **IllegalArgumentException**

- **הפונקציה split**

```
val split = fn : 'a list -> 'a list * 'a list
```

קלט: ls – רשימה מטיפוס כלשהוא
פלט: מחזיר 2 רשימות, אשר מהוות רשימות "סריג" (ראה דוגמא)

```
- split [1,2,3,4,5,6,7,8,9];
val it = ([1,3,5,7,9],[2,4,6,8]) : int list * int list
- split [1];
val it = ([1],[]) : int list * int list
```

- **הפונקציה merge**

```
val merge = fn : int list * int list -> int list
```

קלט: ls1 (רשימה **ממוינת** מטיפוס int)
ls2 (רשימה **ממוינת** מטיפוס int)
פלט: מחזיר רשימה חדשה **ממוינת** המורכבת ממיזוג שתי הרשימות
(אם רשימה אחת קצרה מהשנייה, נשרשר את הרשימה הארוכה יותר לסוף הרשימה החדשה)

```
- merge([1,4,7,9] , [2,3,5,6,8,10]);
val it = [1,2,3,4,5,6,7,8,9,10] : int list
```

- **הפונקציה sort**

```
val sort = fn : int list -> int list
```

קלט: ls – רשימה (מכל טיפוס)
פלט: הפונקציה מחזירה רשימה ממויינת (בשיטת Merge-Sort)

```
- sort [1, ~4, 3, 8, ~1, 5, 2];
val it = [~4,~1,1,2,3,5,8] : int list
```

חלק 2 (ניתן לפתור רק לאחר שיעור 10)

- **הפונקציה choose**

```
val choose = fn : int * 'a list -> 'a list list
```

קלט: k - מספר שלם חיובי
ls – רשימה (מכל טיפוס)
פלט: הפונקציה מחזירה את כל הסידורים האפשריים של לבחור k איברים מתוך הרשימה ls (ללא חשיבות לסדר).

```
- choose(3, [1,2]);
val it = [] : int list list
- choose(3, [1,2,3]);
val it = [[1,2,3]] : int list list
- choose(3, [1,2,3,4,5]);
val it = [[1,2,3],[1,2,4],[1,2,5],[1,3,4],[1,3,5],[1,4,5],[2,3,4],[2,3,5],[2,4,5],[3,4,5]] :
```

שים/י ♥: במקרה ומתקבל n<0 יש לזרוק Exception מטיפוס **IllegalArgumentException**

- **פונקציית isPolindrom**

```
val isPolindrom = fn : string -> bool
```

קלט: str – מחרוזת

פלט: מחזירה true אם המחרוזת מהווה פולינדרום (משמע אם נקרא את המחרוזת מתחילתה ועד הסופה ולהפך נקבל את אותה המחרוזת), אחרת false.

```
- isPolindrom("abcdedcba");
val it = true : bool
- isPolindrom("abdda");
val it = false : bool
```

חלק 3 (ניתן לפתור רק לאחר שיעור 10)

להלן הגדרות של טיפוסי נתונים חדשים:

```
datatype Arguments
  = IntPair of int * int
  | RealTriple of real * real * real
  | StringSingle of string

datatype OutputArgs = IntNum of int | RealNum of real | Str of string
```

יש להשתמש בהגדרות הללו לצורך בניית הפונקציה הבאה:

- **פונקציית multiFunc**

```
val multiFunc = fn : Arguments -> OutputArgs
```

קלט:

- IntPair(x,y) – זוג של מספרים שלמים.
- RealTriple(x,y,z) – שלישיה של מספרים ממשיים.
- SingleString(str) – מחרוזת string.

פלט:

- אם התקבל IntPair(x,y) יוחזר מכפלתם.
- אם התקבל RealTriple(x,y,z) אז יוחזר הממוצע של האיברים.
- אם התקבל SingleString(str) אז תוחזר מחרוזת בסדר הפוך למחרוזת str.

```
- multiFunc(IntPair(5,10));
val it = IntNum 50 : OutputArgs
- multiFunc(RealTriple(1.0, 4.0, 3.0));
val it = RealNum 2.66666666667 : OutputArgs
- multiFunc(StringSingle("abcdefg"));
val it = Str "gfedcba" : OutputArgs
```

חלק 4 (ניתן לפתור רק לאחר שיעור 10)

בחלק זה תדרשו לרשום סימולטור למשחק המפורסם "משחק החיים"
(ראו הסבר מפורט בקישור : משחק החיים - ויקיפדיה)

במשחק החיים שלנו, יש לנו מטריצה בגודל NxN , כאשר תוכן כל משבצת יכול להיות:
Alive(age) – יצור חי בגיל age.
Dead(n) – יצור מת שכבר לא חי n דורות

בתחילת המשחק, כל התאים שהם חיים יהיו בגיל 0 וכל התאים המתים, מתים 0 דורות.

כל משבצת מוקפת ב-8 משבצות, לכן לכל תא יכולים להיות עד 8 שכנים. המשחק מתנהל לפי הכללים הבאים, לכל תא מסתכלים על השכנים שלו בדור הנוכחי:

- אם הוא חי, ויש לו שכן אחד או שאין לו שכנים כלל, הוא מת מבדידות (משמע, הופך למת עם 0 דורות)
- אם הוא חי, אם יש לו יותר משלושה שכנים, הוא מת מצפיפות (משמע, הופך למת עם 0 דורות)
- אם הוא מת, ויש לו בדיוק שלושה שכנים, הוא הופך להיות חי ("נולד" – וגילו הופך להיות 0).
- אחרת (שני שכנים, או שלושה שכנים והוא חי) הוא נשאר חי **וגדל בשנה**.

טיפוס הנתונים המייצג משבצת הינו

```
datatype Square = Alive of int | Dead of int;
```

סטאטוס האוכלוסייה במשחק החיים יכול להיות On-Going (עדיין חיים בלוח) או Extinct (האוכלוסייה נכחדה)

טיפוס הנתונים המייצג את סטאטוס האוכלוסייה הינו:

```
datatype GameStatus = Extinct | OnGoing
```

מצב המשחק הינו זוג המורכב מלוח המשחק וסטאטוס האוכלוסייה.
לפיכך נגדיר את טיפוס אשר ייצג את סטאטוס המשחק

```
type LifeState = Square list list * GameStatus;
```

שים ♥
המיקום (0,0) מייצג את הנקודה השמאלית העליונה של לוח המשחק

מיקום (i,j) מתייחס למשבצת בשורה i ובעמודה j.

עלייך לממש את הפונקציות הבאות:

- **פונקציית createLifeGrid**

```
val createLifeGrid = fn : int * (int * int) list -> Square list list
```

קלט: n – מספר שלם חיובי

lives – רשימה של מיקומים (pairs) אשר קובעים היכן נמצאים תאים חיים

פלט: מחזירה מטריצה מאותחלת עם המשבצות המתאימות (כאשר כל החיים וכל המתים בגיל 0)

```
createLifeGrid(5, [(0,4), (1,1), (3,0), (3,2), (3,4), (4,3)]);
val it =
  [[Dead 0,Dead 0,Dead 0,Dead 0,Alive 0],
   [Dead 0,Alive 0,Dead 0,Dead 0,Dead 0],[Dead 0,Dead 0,Dead 0,Dead 0,Dead 0],
   [Alive 0,Dead 0,Alive 0,Dead 0,Alive 0],
   [Dead 0,Dead 0,Dead 0,Alive 0,Dead 0]] : Square list list
```

שים/י ♥: במקרה ומתקבל n<0 יש לזרוק Exception מטיפוס **IllegalArgumentException**

- **פונקציית determineStatusOf**

```
val determineStatusOf = fn : Square list list -> GameStatus
```

קלט: grid – לוח המשחק

פלט: מחזירה את סטאטוס האוכלוסיה. (אם כולם מתים יוחזר Extinct, אחרת OnGoing)

```
- val mat = createLifeGrid(5, [(0,4), (1,1), (3,0), (3,2), (3,4), (4,3)]);
val mat =
  [[Dead 0,Dead 0,Dead 0,Dead 0,Alive 0],
   [Dead 0,Alive 0,Dead 0,Dead 0,Dead 0],[Dead 0,Dead 0,Dead 0,Dead 0,Dead 0],
   [Alive 0,Dead 0,Alive 0,Dead 0,Alive 0],
   [Dead 0,Dead 0,Dead 0,Alive 0,Dead 0]] : Square list list
- determineStatusOf mat;
val it = OnGoing : GameStatus
```

- **פונקציית nextGeneration**

```
val nextGeneration = fn : Square list list -> Square list list
```

קלט: grid – לוח המשחק
פלט: מחזירה את לוח המשחק לאחר דור אחד.

```
val mat = createLifeGrid(5, [(0,4), (1,1), (3,0), (3,2), (3,4), (4,3)]);
val mat =
  [[Dead 0,Dead 0,Dead 0,Dead 0,Alive 0],
   [Dead 0,Alive 0,Dead 0,Dead 0,Dead 0],[Dead 0,Dead 0,Dead 0,Dead 0,Dead 0],
   [Alive 0,Dead 0,Alive 0,Dead 0,Alive 0],
   [Dead 0,Dead 0,Dead 0,Alive 0,Dead 0]] : Square list list
- nextGeneration mat;
val it =
  [[Dead 1,Dead 1,Dead 1,Dead 1,Dead 0],[Dead 1,Dead 0,Dead 1,Dead 1,Dead 1],
   [Dead 1,Alive 0,Dead 1,Dead 1,Dead 1],
   [Dead 0,Dead 1,Dead 0,Alive 0,Dead 0],
   [Dead 1,Dead 1,Dead 1,Alive 1,Dead 1]] : Square list list
```

- **פונקציית determineNState**

```
val determineNState = fn : LifeState * int -> LifeState
```

קלט: state – מצב המשחק (משמע זוג המכיל את לוח המשחק וסטאטוס האוכלוסיה). **יש לקבוע כי הטיפוס של הזוג הוא LifeState**
n – מספר הדורות שהאוכלוסייה עוברת.
פלט: מחזירה את מצב המשחק לאחר n דורות.
שים ♥ : אם אוכלוסייה נכחדה בדור k $(k < n)$ אז לא נעדכן את מצב האוכלוסייה עבור k+1 והלאה
(שים לב, שכל שהדורות עוברים, כך גיל המתים עולה)

```
- val mat = createLifeGrid(5, [(0,4), (1,1), (3,0), (3,2), (3,4), (4,3)]);
val mat =
  [[Dead 0,Dead 0,Dead 0,Dead 0,Alive 0],
   [Dead 0,Alive 0,Dead 0,Dead 0,Dead 0],[Dead 0,Dead 0,Dead 0,Dead 0,Dead 0],
   [Alive 0,Dead 0,Alive 0,Dead 0,Alive 0],
   [Dead 0,Dead 0,Dead 0,Alive 0,Dead 0]] : Square list list
- val newMat = determineNState((mat, OnGoing), 1);
val newMat =
  ([[Dead 1,Dead 1,Dead 1,Dead 1,Dead 0],[Dead 1,Dead 0,Dead 1,Dead 1,Dead 1],
    [Dead 1,Alive 0,Dead 1,Dead 1,Dead 1],
    [Dead 0,Dead 1,Dead 0,Alive 0,Dead 0],
    [Dead 1,Dead 1,Dead 1,Alive 1,Dead 1]],OnGoing)
  : Square list list * GameStatus
- val newMat = determineNState((mat, OnGoing), 3);
val newMat =
  ([[Dead 3,Dead 3,Dead 3,Dead 3,Dead 2],[Dead 3,Dead 2,Dead 3,Dead 3,Dead 3],
    [Dead 3,Dead 1,Dead 3,Dead 3,Dead 3],[Dead 2,Dead 3,Dead 0,Dead 1,Dead 2],
    [Dead 3,Dead 3,Dead 3,Dead 1,Dead 3]],Extinct)
  : Square list list * GameStatus
- val newMat = determineNState((mat, OnGoing), 4);
val newMat =
  ([[Dead 3,Dead 3,Dead 3,Dead 3,Dead 2],[Dead 3,Dead 2,Dead 3,Dead 3,Dead 3],
    [Dead 3,Dead 1,Dead 3,Dead 3,Dead 3],[Dead 2,Dead 3,Dead 0,Dead 1,Dead 2],
    [Dead 3,Dead 3,Dead 3,Dead 1,Dead 3]],Extinct)
  : Square list list * GameStatus
- val newMat = determineNState((mat, OnGoing), 5);
val newMat =
  ([[Dead 3,Dead 3,Dead 3,Dead 3,Dead 2],[Dead 3,Dead 2,Dead 3,Dead 3,Dead 3],
    [Dead 3,Dead 1,Dead 3,Dead 3,Dead 3],[Dead 2,Dead 3,Dead 0,Dead 1,Dead 2],
    [Dead 3,Dead 3,Dead 3,Dead 1,Dead 3]],Extinct)
  : Square list list * GameStatus
```

שים/י ♥: במקרה ומתקבל n<0 יש לזרוק Exception מטיפוס **IllegalArgumentException**

חלק 5 (ניתן לפתור רק לאחר שיעור 11)

עפ"י הפונקציות שלמדנו בשיעור על רצף אינסופי:

```
datatype 'a Seq = Cons of 'a * (unit -> 'a Seq) | Nil
val head = fn : 'a Seq -> 'a
val tail = fn : 'a Seq -> 'a Seq
val take = fn : int * 'a Seq -> 'a list
```

פונקציית upF:

```
val upF = fn : 'a * ('a -> 'a) -> 'a Seq
```

קלט: u – האיבר הראשון ברצף

f – הפונקציה מקבלת איבר ברצף ומחזירה את האיבר הבא ברצף

פלט: מחזירה את הרצף האינסופי:

```
u, f(u), f(f(u)), f(f(f(u))),....
```

דוגמאות:

```
-val x = upF(2, fn (n) => n*n);
val x = Cons (2,fn) : int Seq
- tail(x);
val it = Cons (4,fn) : int Seq
- tail(tail(x));
val it = Cons (16,fn) : int Seq
- take(4, x);
val it = [2,4,16,256] : int list

-val x = upF(2, fn (n) => 2*n);
val x = Cons (2,fn) : int Seq
- take(10, x);
val it = [2,4,8,16,32,64,128,256,512,1024] : int list
```