# פרויקט מסכם תכנות מתקדם – תשפ"א סמסטר ב'

בלוח שחמט יש שמונה שורות ושמונה עמודות של משבצות. מיקום משבצת מתואר ע"י מערך בגודל שני תווים – אות לשורה (A עד H) ומספר לעמודה (1 עד 8). לדוגמא: המיקום של המשבצת שמכילה X בלוח הבא מסומן ע"י B4 ומיקום המשבצת שמכילה Y הוא F1.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | Z | | | | | | | |
| B | | | 2 | X | | | | |
| C | | 2 | | | 1 | | 1 | |
| D | | | | 1 | | | | 1 |
| E | | | | | | Q | | |
| F | Y | | | 1 | | | | 1 |
| G | | | | | 1 | | 1 | |
| H | | | | | | | | |

על-מנת לייצג משבצת בלוח שח-מט, עושים שימוש בהגדרה הבאה:

```
typedef char chessPos[2];
```

שימו-לב כי זה מערך בגודל ידוע והוא בן שני בתים. זו אינה מחרוזת ולכן אין 0\ בסופה.

על-מנת לייצג מערך של משבצות עושים שימוש בהגדרה הבאה:

```
typedef struct _chessPosArray {
     unsigned int   size;
     chessPos  *positions;
} chessPosArray;
```

פרש (knight) הוא אחד הכלים במשחק השחמט. פרש יכול לנוע ממיקומו הנוכחי אך ורק לאחד משמונת המיקומים הבאים:

- 2 שורות מעלה או מטה ועמודה אחת ימינה או שמאלה
- 2 עמודות ימינה או שמאלה ושורה אחת מעלה או מטה

לפרש אסור לצאת מגבולות הלוח.

**לדוגמא:**

בלוח המופיע לעיל, לפרש אשר ממוקם במשבצת אשר מכילה Q מותר לעבור אך ורק לאחת מהמשבצות המכילות 1. לפרש אשר ממוקם במשבצת אשר מכילה Z מותר לנוע רק לאחת מהמשבצות שמכילות 2.

## סעיף 1

בסעיף זה יש לכתוב את הפונקציה:

```
chessPosArray *** validKnightMoves( )
```

הפונקציה מחזירה מערך דו-מימדי של מצביעים למערכי מיקומים אפשריים לתנועת פרש. כל תא בשורה r ובעמודה c יכיל מערך של מיקומים אליהם מותר לפרש לנוע מתא זה.

**סעיף 2**

על-מנת לייצג רשימה של משבצות, עושים שימוש ברשימה מקושרת חד-כיוונית שמוגדרת כדלקמן:

```
typedef struct _chessPosCell {
    chessPos   position;
    struct _chessPosCell *next;
} chessPosCell;

typedef struct _chessPosList {
    chessPosCell *head;
    chessPosCell *tail;
} chessPosList;
```

כתבו את הפונקציה:

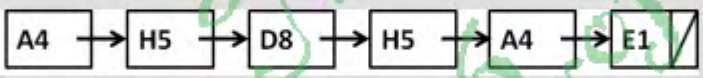
**void display( chessPosList *lst)**

המקבלת רשימה של משבצות.

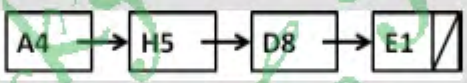
ראשית, הפונקציה מסירה מהרשימה משבצות אשר חוזרות על עצמן. במקרה שמיקום חוזר על עצמו, המופע הראשון שלו מתחילת הרשימה הוא זה שיושאר וכל המופעים אשר נמצאים אחריו ימחקו.

**לדוגמא**:

אם הקלט לפונקציה יהיה




לאחר הסרת המשבצות אשר חוזרות על עצמן, הרשימה תהפוך ל:




לאחר הסרת המשבצות אשר חוזרות על עצמן, הפונקציה תציג את המשבצות שנותרו ברשימה על המסך באופן הבא:

נניח כי מספר המיקומים ברשימה הוא N. הפונקציה תצייר לוח שח-מט על המסך (יש לכם את החופש לקבוע כיצד יראה הלוח) כאשר בכל אחת מהמשבצות אשר מופיעות ברשימה 1st יופיע מספר בין 1 ל- N ששווה למיקומה ברשימה. בדוגמא לעיל N=4 - בתא A4 יופיע המספר 1, בתא H5 יופיע המספר 2 וכן הלאה.

בעמוד הבא תמצאו פלט אפשרי לדוגמא לעיל.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | | | | 1 | | | | |
| B | | | | | | | | |
| C | | | | | | | | |
| D | | | | | | | | 3 |
| E | 4 | | | | | | | |
| F | | | | | | | | |
| G | | | | | | | | |
| H | | | | | 2 | | | |

**סעיף 3**

נתונות ההגדרות הבאות עבור עץ אשר מתאר את כל מסלולי התנועה האפשריים של פרש החל ממיקום התחלתי שנתון בשורש (עליכם למצוא בעצמכם מה יש לרשום במקום שבו רשומות 3 נקודות על-מנת שההגדרות יעברו קומפילציה. נושא זה קרוי forward declaration):

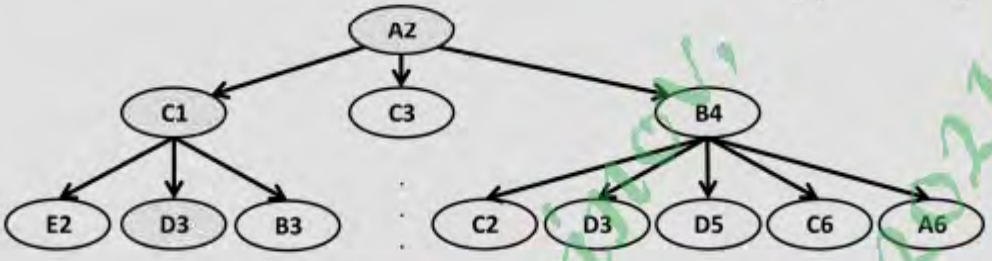
```
...

typedef struct _treeNode{
    chessPos   position;
    treeNodeListCell *next_possible_positions;  // רשימת מיקומים
} treeNode;

typedef  struct  _treeNodeListCell {
    treeNode   *node;
    struct _treeNodeListCell  *next;
} treeNodeListCell;

typedef  struct  _pathTree {
    treeNode   *root;
} pathTree;
```

צומת כלשהו X בעץ מייצג משבצת בלוח שחמט. ילדיו של הצומת X הם כל המיקומים האפשריים אליהם יכול להמשיך הפרש בצעד אחד מ-X. הילדים נתונים ברשימה המקושרת שראשה נתון ב-next_possible_positions.

**כדי להימנע ממצב שבו פרש הולך במעגלים, אסור במסלול מסוים לחזור אל משבצת שהופיעה כבר במסלול, כלומר, אסור שרשימת הילדים של X תכיל מיקומים אשר מופיעים כבר במסלול העובר בין השורש ל-X** (לדוגמא, בכל צומת שאינו השורש, תמיד אחד מהמיקומים האפשריים שאליו יכול לזוז הפרש הוא צומת האב (לחזור על עקבותיו). לכן אסור, בין השאר, שצומת זה יופיע ברשימת מיקומי ההמשך שמהווים את ילדיו של הצומת).

**לדוגמא:**

אם המיקום ההתחלתי של הפרש הוא A2 אזי תיאור חלקי של מסלולים באורך 2 (3 הרמות העליונות של העץ) ייראה כדלקמן:




לנוחיותכם, המסלולים מתוארים גם בלוח הבא כאשר מיקומי המסלול המודגש בעץ מודגשים גם בלוח. תוכן המשבצת מורכב ממספר שהוא הרמה בעץ ואות שמתארת את מספר האפשרות (האפשרויות ממוספרות משמאל לימין).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| A | | 1 | | | | 2c3e | | |
| B | | | 2a3c | 2c | | | | |
| C | 2a | 2c3a | 2b | | | 2c3d | | |
| D | | | 2a3b/ 2c3b | | 2c3c | | | |
| E | | 2a3a | | | | | | |
| F | | | | | | | | |
| G | | | | | | | | |
| H | | | | | | | | |

שימו לב שמגבלת החזרה שמתוארת לעיל מונעת מעבר של פרש העומד ב-D5 אל B4 היות וביקרנו בו לפני שהגענו ל-D5. יחד עם זאת, ייתכן שמשבצת השתתף ביותר ממסלול אחד – לדוגמא D3.

יש לכתוב את הפונקציה:

```
pathTree  findAllPossibleKnightPaths( chessPos *startingPosition)
```

הפונקציה בונה עץ עם כל מסלולי הפרש האפשריים אשר מתחילים במשבצת שמתקבלת כפרמטר. בכל צומת X הפונקציה מחפשת את כל הצמתים האפשריים שאליהם יכול הפרש להמשיך תוך הימנעות מהליכה במעגלים. את כל צמתי ההמשך האפשריים יש לשמור ברשימה ולאחסנם בצומת X. יש למצוא את המשכי המסלולים מכל אחד מילדים. אם לא ניתן להמשיך לאף צומת מ- X אזי X נמצא בסוף אחד מהמסלולים אשר החלו בשורש והוא יהיה עלה בעץ המוחזר.

**<u>שימו לב:</u>**

**א. יש לעשות שימוש בפונקציה אשר יצרתם בסעיף 1.**

**ב. עקב מספר המסלולים הרב, בדקו סעיף זה עבור לוח בגודל 4X4 ו-5X5. לוחות גדולים יותר יצריכו זמן ריצה רב יותר ויתכן שיגרמו לנפילת התכנית עקב מחסור בזיכרון.**

## סעיף 4

בסעיף זה עליכם לכתוב את הפונקציה:

```
chessPosList *findKnightPathCoveringAllBoard( pathTree *path_tree)
```

הפונקציה מוצאת מסלול שמכסה את **כל** משבצות הלוח החל מהשורש עץ המסלולים שהתקבל כפרמטר. המסלול ייוצג כרשימה מקושרת אשר התא הראשון שבה יכיל את המיקום שבשורש עץ המסלולים ויתר המיקומים ברשימה יהיו התאים הבאים במסלול לפי סדרם. במידה ואין מסלול אשר עובר דרך כל משבצות הלוח, הפונקציה תחזיר NULL.

## סעיף 5

הוחלט לשמור באופן חסכוני רשימת מיקומים בקובץ בינארי באופן הבא: כל מיקום ייוצג ע"י 6 ביטים. 3 הביטים השמאליים ייצגו את מספר השורה פחות 1 ו-3 הביטים הימניים ייצגו את מספר העמודה פחות 1. המיקומים יישמרו ברצף כדי לחסוך במקום. בתחילת הקובץ, יישמר מספר מטיפוס short שיכיל את מספר המיקומים ברשימה. במידת הצורך יש לרפד בסיביות אפס בסוף הקובץ. להלן דוגמא לאופן שמירת מיקומים (הרווחים הם למטרת נוחות התצוגה בלבד ואינם חלק מהקובץ):

$$\underbrace{00000000\ 00000101}_{5\ positions}\ \underbrace{010}_{C}\ \underbrace{100}_{5}\ \underbrace{000}_{A}\ \underbrace{011}_{4}\ \underbrace{001}_{B}\ \underbrace{010}_{3}\ \underbrace{011}_{D}\ \underbrace{000}_{1}\ \underbrace{100}_{E}\ \underbrace{010}_{3}\ 00$$

המיקום הראשון ישמר ב-most significant bits ויתר המיקומים ישמרו אחריו. ביטים שנותרים לא מנוצלים יכילו אפסים ב-least significant bits של הבית האחרון. בדוגמא לעיל הקובץ יכיל 6 בתים. שימו לב שיתכן שמיקום יכול להתחיל בבית מסוים ולהסתיים בבית הבא: לדוגמא האות A בדוגמא לעיל. אין לשמור מיקום (שורה ועמודה) בבית שלם ולהשאיר שני ביטים ללא שימוש שכן זה לא חסכוני ולכן לא עונה על דרישת הסעיף.

יש לכתוב את הפונקציה:

```
void saveListToBinFile( char *file_name, chessPosList *pos_list)
```

אשר מקבלת שם של קובץ בינארי ורשימת מיקומים ושומרת אותם באופן המתואר לעיל.

## סעיף 6

בסעיף זה יש לכתוב את הפונקציה:

```
int  checkAndDisplayPathFromFile ( char *file_name)
```

הפונקציה מקבלת שם של קובץ בינארי אשר מכיל רשימה של מיקומים בפורמט שצוין בסעיף 5. הפונקציה קוראת את הרשימה ומבצעת את הפעולות הבאות:

אם הרשימה אינה מכילה מסלול חוקי של פרש, אזי הפונקציה מסיימת ומחזירה את הערך 1.

אחרת, הפונקציה מסירה מהמסלול מיקומים אשר הפרש מבקר בהם יותר מפעם אחת (במידה ויש כאלה) ומדפיסה לוח שחמט עם המסלול בעזרת הפונקציה מסעיף 2.

אם המסלול מכסה את כל משבצות הלוח, הפונקציה מחזירה 2, אחרת הפונקציה מחזירה 3.

אם הקובץ לא קיים, הפונקציה מחזירה 1-.

## סעיף 7

בסעיף זה עליכם ליצור תפריט אשר יאפשר את הפעלת הסעיפים שכתבתם. הפונקציה תציג את התפריט הבא:

1. Enter a knight's starting position
2. Create all possible knight paths
3. Find a knight path covering all board
4. Save knight path covering all board to file
5. Load and display path from file
6. Exit

באופציה מספר 1 המשתמש יקליד מיקום על הלוח. יש לוודא שהוקלד מיקום חוקי.
באופציה 2, ייבנה עץ של כל המסלולים האפשריים שמתחילים מהמיקום שהוזן באופציה 1.
באופציה 3 יש למצוא מסלול פרש אשר מכסה את כל הלוח. אם טרם נבנה עץ מסלולים ע"י הפעלת אופציה 2, יש ליצור אותו ללא בקשה מהמשתמש.
באופציה 4, יש לקלוט שם קובץ מהמשתמש ולשמור בו מסלול פרש אשר מכסה את הלוח. אם טרם הופעלה אופציה 3, יש להפעילה ללא בקשה מהמשתמש.
באופציה 5, יש לקלוט שם קובץ מהמשתמש ולהפעיל את הפונקציה מסעיף 6. יש להוציא הודעה למשתמש לגבי כל אחד מהמקרים שהפונקציה מכסה בהחזרת הערכים (-1,1,2,3).

**שימו לב:**

- באופציות 2-4, אם טרם הוזן מיקום התחלתי יש להודיע למשתמש.
- באופציות 3-4, אם אין מסלול כדרוש, יש להודיע על כך למשתמש.

**הנחיות כלליות:**

יש להקפיד על יעילות וחסכון בזמן ריצה וזיכרון.
יש להקפיד שגודל פונקציה לא יחרוג ממסך אחד.
יש לפנות זיכרון שהוקצה דינמית ואשר אין בו צורך יותר.
יש להשתמש ב#define היכן שנחוץ.