פרויקט מסכם תכנות מתקדם – סמסטר ב' תשע"ט

הנחיות כלליות

יש לתעד את התכניות.
יש להקפיד על יעילות והסכון בזמן ריצה וזיכרון.
יש להקפיד שגודל פונקציה לא יחרוג ממסך אחד.
יש לפנות זיכרון שהוקצה דינמית ואשר אין בו צורך יותר.
יש לבדוק אם הקצאות הזיכרון הצליחו.
יש לבדוק שפתיחת קובץ הצליחה.
יש לסגור קובץ לאחר סיום השימוש בו.
יש להשתמש ב#define היכן שנחוץ.
אין לחרוג מה-prototype-ים שהוגדרו אך ניתן ליצור פונקציות עזר לפי הצורך.

הגדרת הפרויקט

מטרת הפרויקט היא לתכנת גרסה פשוטה של משחק הדמקה.

בלוח דמקה יש שמונה שורות ושמונה עמודות של משבצות. מיקום משבצת מתואר ע"י מערך בן שני תווים – אות לשורה (A עד H) ומספר לעמודה (1 עד 8). לדוגמא: המיקום של המשבצת שמכילה X בלוח הבא מסומן ע"י B3 ומיקום המשבצת שמכילה Y הוא F1.

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

חוקי המשחק

למשחק שני שחקנים שנקראים T ו-B. המשחק מתנהל רק על המשבצות האפורות כאשר T נע מלמעלה כלפי מטה ו-B נע מלמטה כלפי מעלה. לכל שחקן בתחילת המשחק יש 12 כלי משחק אשר מסודרים כדלקמן:

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

כל שחקן מתקדם בתורו.

בכל תור השחקן מזיז את אחד מכלי המשחק שלו. כלי משחק יכול לנוע לאחת משתי המשבצות אשר בכיוון תנועתו במידה והיא פנויה. במקרה שהמשבצת אינה פנויה ומכילה כלי משחק של היריב אבל המשבצת שאחריה (בכיוון תנועת השחקן) פנויה, יכול כלי המשחק לדלג מעל כלי המשחק של היריב ובכך להוציא את כלי משחק זה מהמשחק. מהלך זה קרוי capture או אכילה. במידה ומצב זה מתקיים גם במשבצת שהגיע אליה, יכול כלי המשחק להמשיך ולדלג ולהוציא עוד כלי משחק של יריבו מהמשחק.

לדוגמא, במקרה שתורו של השחקן T ומצב הלוח הוא:

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

יכול כלי המשחק שבמשבצת A4 לנוע אל אחת מהמשבצות שמסומנות ב* כאשר כלי משחק שהוא מדלג מעליהם יוצאים מהמשחק. במקרה ויש מספר אפשרויות תנועה, **חייב** השחקן לבחור בזו עם מספר הדילוגים הגדול ביותר. אם יש מספר אפשרויות עם מספר הדילוגים הגדול ביותר, השחקן יכול לבחור באחת מהן לפי רצונו. בדוגמא לעיל, חייב השחקן T לנוע אל המשבצת G6 ומצב הלוח לאחר המהלך יהיה:

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

שימו לב שאפשרויות התנועה של שחקן יוצרות עץ בינארי.

המשחק מסתיים באחד מהמקרים הבאים:

א. כאשר אחד השחקנים נותר ללא כלי משחק
ב. כאשר אחד מכלי המשחק של T מגיע אל שורה H
ג. כאשר אחד מכלי המשחק של B מגיע אל שורה A.

<u>מימוש הפרויקט</u>

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

```
typedef struct _checkersPos
{
```

```
        char row, col;
} checkersPos;
```

על-מנת לייצג לוח עושים שימוש בהגדרה הבאה:

```
typedef unsigned char Board[BOARD_SIZE][BOARD_SIZE];
```

על-מנת לייצג שחקן הוגדר הטיפוס

```
typedef unsigned char Player;
```

סעיף 1

כדי לתאר את העץ הבינארי של אפשרויות התנועה של כלי משחק **אחד** עושים שימוש בהגדרות הבאות:

```
typedef struct _SingleSourceMovesTreeNode{
        Board                   board;
        checkersPos             *pos;
        unsigned short          total_captures_so_far; // מספר הדילוגים עד כה
         struct  _SingleSourceMovesTreeNode      *next_move[2]; //  יעדי התנועה
} SingleSourceMovesTreeNode;

typedef  struct _SingleSourceMovesTree {
        SingleSourceMovesTreeNode       *source;
} SingleSourceMovesTree;
```

יש לכתוב את הפונקציה:

SingleSourceMovesTree *FindSingleSourceMoves(Board board, checkersPos *src)

הפונקציה מקבלת לוח משחק עם מצב נתון ומשבצת בלוח.
אם קיים כלי משחק במשבצת, הפונקציה מחזירה עץ בינארי של כל אפשרויות התנועה שלו. אחרת, מחזירה הפונקציה NULL.

סעיף 2

כדי לתאר את התנועה שנבחרה מבין כל אפשרויות התנועה של כלי משחק אחד עושים שימוש בהגדרות הבאות:

```
typedef struct _SingleSourceMovesListCell {
        checkersPos                             *position;
        unsigned short                          captures;
        struct  _SingleSourceMovesListCell      *next;
} SingleSourceMovesListCell;

typedef struct _SingleSourceMovesList {
        SingleSourceMovesListCell  *head;
        SingleSourceMovesListCell  *tail;
} SingleSourceMovesList;
```

יש לכתוב את הפונקציה:

SingleSourceMovesList *FindSingleSourceOptimalMove(SingleSourceMovesTree *moves_tree)

הפונקציה מקבלת עץ של כלי משחק מסוים ומחזירה את מסלול התנועה עם מספר הדילוגים (captures) הרב יותר מבין כל מסלולי התנועה האפשריים שמתוארים בעץ. במידה ויש מספר כאלה, על הפונקציה לבחור אחד מהם ולהחזירו.

סעיף 3

כדי לתאר את כל אפשרויות התנועה של שחקן, עושים שימוש ברשימה מקושרת אשר מכילה עבור כל אחד מכלי המשחק של אותו שחקן את מסלול התנועה הטוב ביותר מהמשבצת שכלי המשחק נמצא בה:

```
typedef struct _multipleSourceMovesListCell {
  SingleSourceMovesList                  *single_source_moves_list;
  struct _multipleSourceMovesListCell    *next;
} MultipleSourceMovesListCell;

typedef struct _multipleSourceMovesList {
  MultipleSourceMovesListCell     *head;
  MultipleSourceMovesListCell     *tail;
} MultipleSourceMovesList;
```

כתבו את הפונקציה:

MultipleSourceMovesList *FindAllPossiblePlayerMoves(Board board, Player player)

הפונקציה מקבלת לוח משחק ושחקן ומחזירה את רשימת המהלכים הטובים ביותר מכל משבצת בלוח אשר מכילה כלי משחק של שחקן זה ואשר ניתן לנוע ממנה.

סעיף 4

ממשו את הפונקציה:

void Turn(Board board, Player player)

הפונקציה מקבלת לוח משחק ושחקן ומבצעת את המהלך הטוב ביותר מבין המהלכים האפשריים בלוח הנתון עבור שחקן זה. במידה והמהלך הנבחר מכיל דילוגים, יש להסיר מלוח המשחק את כלי המשחק של היריב שמדלגים מעליהם.

סעיף 5

ממשו את הפונקציה :

void StoreBoard(Board board, char *filename)

הפונקציה מקבלת לוח משחק ושם קובץ בינארי ושומרת את לוח המשחק לקובץ באופן הבא. כל משבצת תישמר בשתי סיביות על-פי הערכים הבאים: 00 מייצג משבצת ריקה, 01 מייצג משבצת שמכילה כלי משחק של השחקן T, 10 מייצג משבצת שמכילה כלי משחק של השחקן B. הלוח יישמר שורה אחר שורה, כאשר כל שורה תישמר משמאל לימין.

לדוגמא, שלוש השורות של הלוח שבחלקו העליון של עמוד 2 תישמרנה כך (הרווחים הם לנוחיות הקריאה בהפרדה בין שורות הלוח ואינם חלק מהקובץ).

0000000100000000 0000100010000000 0000000000000000

חמש השורות הנותרות תישמרנה בהמשך הקובץ באופן דומה.

סעיף 6

ממשו את הפונקציה

void LoadBoard(char *filename , Board board)

הפונקציה מקבלת שם של קובץ שנוצר בסעיף 5 ולוח. הפונקציה טוענת אל המשתנה board את נתוני הלוח מהקובץ.

סעיף 7

ממשו את הפונקציה

void PlayGame(Board board, Player starting_player)

הפונקציה מקבלת לוח ושחקן שתורו לשחק.
הפונקציה משחקת משחק שבו המחשב מחליט על מהלכי שני הצדדים (המחשב משחק מול עצמו) כאשר הוא מתחיל מהשחקן שהתקבל בפרמטר starting_player. הפונקציה תדפיס את הלוח שהתקבל ובכל תור הפונקציה תדפיס את השחקן שזה תורו ואת המהלך שהתבצע עבורו. על הפלט להיות בפורמט הבא (להלן דוגמא לפלט):

```
+-+-+-+-+-+-+-+-+
+ |1|2|3|4|5|6|7|8|
+-+-+-+-+-+-+-+-+
|A| |T| |T| |T| |T|
+-+-+-+-+-+-+-+-+
|B|T| |T| |T| |T| |
+-+-+-+-+-+-+-+-+
|C| |T| |T| |T| |T|
+-+-+-+-+-+-+-+-+
|D| | | | | | | | |
+-+-+-+-+-+-+-+-+
|E| | | | | | | | |
+-+-+-+-+-+-+-+-+
|F|B| |B| |B| |B| |
+-+-+-+-+-+-+-+-+
|G| |B| |B| |B| |B|
+-+-+-+-+-+-+-+-+
|H|B| |B| |B| |B| |
+-+-+-+-+-+-+-+-+
player TOP_DOWN's turn
C8->D7
player BOTTOM_UP's turn
F7->E8
player TOP_DOWN's turn
D7->E6
player BOTTOM_UP's turn
F5->D7
player TOP_DOWN's turn
C6->D5
player BOTTOM_UP's turn
G8->F7
player TOP_DOWN's turn
D5->E6
player BOTTOM_UP's turn
F7->D5
player TOP_DOWN's turn
C4->E6
player BOTTOM_UP's turn
H7->G8
player TOP_DOWN's turn
E6->F7
player BOTTOM_UP's turn
```

```
G8->E6
player TOP_DOWN's turn
C2->D3
player BOTTOM_UP's turn
G6->F7
player TOP_DOWN's turn
D3->E4
player BOTTOM_UP's turn
F3->D5
player TOP_DOWN's turn
B7->C8
player BOTTOM_UP's turn
H5->G6
player TOP_DOWN's turn
B5->C6
player BOTTOM_UP's turn
D7->B5
player TOP_DOWN's turn
A4->C6->E4
player BOTTOM_UP's turn
G6->F5
player TOP_DOWN's turn
E4->G6
player BOTTOM_UP's turn
G4->F5
player TOP_DOWN's turn
G6->H7
player BOTTOM_UP's turn
```