דו"ח על תרגיל בית מספר 3

מגיש : ירין בנימין

מבנה הדו"ח בנוי סעיפים אחד אחר השני כמו בהסבר הכללי. ניסיתי לפרט ולדייק כמה שיותר בניסוח כדי שלא תצטרך לעבור כול פעם לשאלות שנשאלו ותוכל/י לקרוא את הקובץ בשלמותו. בנוסף כלל קטעי הקוד המצורפים בקובץ הם כתמונה ולא כטקסט כדי לשמור על המבנה כמה שיותר קריא.

1. העבודה עצמה מחולקת ל-4 קבצים :

- קובץ Defs.h הוא קובץ הגדרות המכיל סטטוסים אפשריים לסוגי החזרות שונים.
- קבצי H לכול אחת מהמחלקות אשר מכילה את הפונקציות אפשריות שהמשתמש יכול לבצע.
- קובץ Pokemon.c הוא הגדרות הפוקימון שלנו מפרוייקט קודם.
- קובץ LinkedList.c אשר מכיל את הסטרקטים LinkedList ו- Node , כאשר Node הוא חולייה ברשימה המקושרת.
- קובץ MaxHeap.c אשר מכיל את הסטרקט MaxHeap, מימוש ערמת עדיפויות.
- קובץ BattleByCategory.c אשר ממומש על ידי רשימה מקושרת של ערמות מקסימום. כול ערימה תחזיק את האובייקטים בעלי קטגוריה מהסוג שלה.
- קובץ MainSecond.c הוא קובץ המניו שלנו , הוא אחראי על קריאת הקובץ מהמשתמש והמרתו למשתנים הנדרשים. בנוסף הוא אחראי להצגת התפריט בו בכול בחירת משתמש נכנס לפונקציה ייעודית שתשתמש בפונקציות של BattleByCategory לשם השלמתן.

makefile
~/eclipse-workspace/Pokemon2/src

```makefile
PokemonsBattles: Pokemon.o LinkedList.o MaxHeap.o BattleByCategory.o MainSecond.o
	gcc Pokemon.o LinkedList.o MaxHeap.o BattleByCategory.o MainSecond.o -o PokemonsBattles

Pokemon.o: Pokemon.c Pokemon.h Defs.h
	gcc -c Pokemon.c

LinkedList.o: LinkedList.c LinkedList.h Defs.h
	gcc -c LinkedList.c

MaxHeap.o: MaxHeap.c MaxHeap.h Defs.h
	gcc -c MaxHeap.c

BattleByCategory.o: BattleByCategory.c BattleByCategory.h LinkedList.h MaxHeap.h Defs.h
	gcc -c BattleByCategory.c

MainSecond.o: MainSecond.c Pokemon.h BattleByCategory.h Defs.h
	gcc -c MainSecond.c

clean:
	rm -f *.o PokemonsBattles
```

.2

.3 כאשר הקצאת זיכרון לא הצליחה , מתבצע שחרור זיכרון של כלל הזיכרון שהוקצה כבר בפונקציה והחזרת סטטוס err או NULL (במקרה שהפונקציה היא פונקציה יוצרת). החזרת הסטטוס err היא במשמעות של כישלון להקצאת זיכרון דינמי (בשונה מ failure - כישלון עקב פרמטרים לא נכונים). מתבצע החזרת סטטוס כדי לתת למשתמש המערכת (במקרה שלנו יוצר המיין) אפשרות להחליט מה לעשות במקרה של כישלון מערכת (ולא על דעת עצמי להחליט בפונקציה עצמה להקריס/לסגור את המערכת).

לדוגמא:

```
done = print_type(types[i]);
if(done == err)
    return err;
```

או

```
if(me->effective_against_me == NULL)
    return err;
```

המשתמש במערכת יקבל חזרה את השגיאה ויחליט מה לעשות.

במקרה שלנו אני הוא משתמש המערכת והחלטתי לבצע שחרור זיכרון והדפסת שגיאה.

```
if(done == err){
    freeSys(pokes, types, num_types);  // free memory
    printf("No memory available.\n");
    return -1;
}
```

.4

a. קובצי ה H יצורפו לפני הסבר מלא על ה ADT הספציפי.

b. כמו שכבר כתבתי בתחילת העבודה קובץ ה-H הוא קובץ החתימות של הפונקציות אפשריות שהמשתמש יכול לבצע. כלל המימושים של הפונקציות השונות בנוסף למימוש הסטרקט עמנו מבוצעים בקובץ ה-C ובכך מוסתרים מהמשתמש הסופי את המימוש שלהם.

:LinkedList

```
#ifndef LINKEDLIST_H_
#define LINKEDLIST_H_
#include "Defs.h"


// Node
typedef struct a_Node *Node;
Node createNode(element);
element getVal(Node);
Node getNext(Node);

// LinkedList
typedef struct a_LinkedList *LinkedList;
LinkedList createLinkedList(freeFunction, equalFunction cmpByVal, printFunction, copyFunction, equalFunction cmpByKey);
status destroyList(element);
status appendNode(LinkedList, element);
status deleteNode(LinkedList, element);
status displayList(element);
element searchByKeyInList(LinkedList, element);
Node getHead(LinkedList);


#endif /* LINKEDLIST_H_ */
```

c. כמו שכבר אמרתי הסטרקט ממומש בקובץ הC כדי להיות מוסתר מהמשתמש:

```
struct a_LinkedList
{
    Node head;
    Node tail;
    int size;
    freeFunction freeFunc;
    equalFunction cmpByVal;
    printFunction print;
    copyFunction copy;
    equalFunction cmpByKey;

};
```

d. קיימים 7 מצביעים מסוגים שונים + משתנה מסוג int כלומר 8 משתנים בגודל 4 bytes כול אחד - בסך הכול גודל של 32 bytes.

e. הקוד נרשם בצורה החסכונית ביותר שאפשר כדי לחסוך בזמן ריצה מיותר. כלומר המשתנה היחיד שלא היינו חייבים פה הוא ה tail אבל מחיקתו תגרור בכול הוספת איבר חדש ריצה על כול הרשימה עד הגעה לסופו.

f. פונקציית היצירה מצורפת בדף הבא. הפונקציה מקבלת את הפונקציות הבאות – פונקציית שחרור, השוואת תוכן, הדפסה, העתקה והשוואה ע"י key ספציפי שהמשתמש נתן לנו. הפונקציה מאתחלת את ה ADT ושומרת את כול הפונקציות הנדרשות.

```c
LinkedList createLinkedList(freeFunction freeFunc, equalFunction cmpByVal, printFunction print, copyFunction copy, equalFunction cmpByKey){

    if(freeFunc == NULL || cmpByVal == NULL || print == NULL || copy == NULL || cmpByKey == NULL)
        return NULL;

    LinkedList list = (LinkedList)malloc(sizeof(struct a_LinkedList));
    if(list == NULL)
        return NULL;

    list->head = NULL;
    list->tail = NULL;
    list->size = 0;
    list->freeFunc = freeFunc;
    list->cmpByVal = cmpByVal;
    list->print = print;
    list->copy = copy;
    list->cmpByKey = cmpByKey;

    return list;
}
```

פונקציית הריסה :

```c
status destroyList(element l){
    // free all memory
    if(l == NULL)
        return failure;

    LinkedList list = (LinkedList)l;

    Node cur = list->head;
    Node next;

    while(cur != NULL){
        next = cur->next;
        list->freeFunc(cur->val);
        free(cur);
        cur = next;
    }

    free(list);
    return success;

}
```

הפונקציה משחררת את הזיכרון של כול איבר ברשימה ובנוסף את הרשימה.

:MaxHeap

```c
#ifndef MAXHEAP_H_
#define MAXHEAP_H_
#include "Defs.h"

typedef struct a_MaxHeap *MaxHeap;
MaxHeap createHeap(char*, int, copyFunction, freeFunction, equalFunction, printFunction);
status destroyHeap(element);
status insertToHeap(MaxHeap, element);
status printHeap(element);
element PopMaxHeap(MaxHeap);
element TopMaxHeap(MaxHeap);
char* getHeapId(MaxHeap);
int getHeapCurrentSize(MaxHeap);
int equalHeap(element, element);  // compare by name
element shallowCopyHeap(element);
void maxHeapify(MaxHeap, int);  // sort the heap

#endif /* MAXHEAP_H_ */
```

c. כמו שכבר אמרתי הסטרקט ממומש בקובץ הC כדי להיות מוסתר מהמשתמש:

```
struct a_MaxHeap
{
    char* id;
    int max_size;
    int cur_size;
    element* list;

    copyFunction copy;
    freeFunction freeFunc;
    equalFunction cmpByVal;
    printFunction print;

};
```

d. קיימים 6 מצביעים מסוגים שונים + 2 משתנים מסוג int כלומר 8 משתנים בגודל 4 bytes כול אחד - בסך הכול גודל של 32 bytes.

e. הקוד נרשם בצורה החסכונית ביותר שאפשר. חייבים לשמור את כלל הפונקציות כי אנחנו משתמשים בהם. בנוסף שמירה על גודל מינימאלי -

```
heap->id = (char*)malloc((strlen(id)+1) * sizeof(char));
```

לדוגמא בקבלת השם לא נשתמש בגודל יותר גדול ממה שצריך.

f. פונקציית היצירה מקבלת את הפונקציות הבאות – שם, גודל מקסימלי, פונקציית העתקה, פונקציית שיחרור, פונקציית השוואה ופונקציית הדפסה. הפונקציה מאתחלת את ה ADT ושומרת את כול הפונקציות הנדרשות.

```
MaxHeap createHeap(char* id, int max_size, copyFunction copy, freeFunction freeFunc, equalFunction cmpByVal, printFunction print){

    if(id == NULL || copy == NULL || freeFunc == NULL || cmpByVal == NULL || print == NULL)
        return NULL;

    MaxHeap heap = (MaxHeap)malloc(sizeof(struct a_MaxHeap));
    if(heap == NULL)
        return NULL;

    heap->id = (char*)malloc((strlen(id)+1) * sizeof(char));
    if(heap->id == NULL){
        free(heap);
        return NULL;
    }
    strcpy(heap->id, id);

    heap->list = (element*)malloc(sizeof(element)*max_size);
    if(heap->list == NULL){
        free(heap->id);
        free(heap);
        return NULL;
    }

    heap->max_size = max_size;
    heap->cur_size = 0;
    heap->copy = copy;
    heap->freeFunc = freeFunc;
    heap->cmpByVal = cmpByVal;
    heap->print = print;

    return heap;
}
```

פונקציית הריסה :

```
status destroyHeap(element h){

    if(h == NULL)
        return failure;

    MaxHeap heap = (MaxHeap)h;

    int i;
    for(i=0;i<heap->cur_size;i++)
        heap->freeFunc(heap->list[i]);

    free(heap->list);
    free(heap->id);
    free(heap);
    return success;

}
```

הפונקציה משחררת את הזיכרון של כל איבר ברשימה ובנוסף את הרשימה והשם שלה.

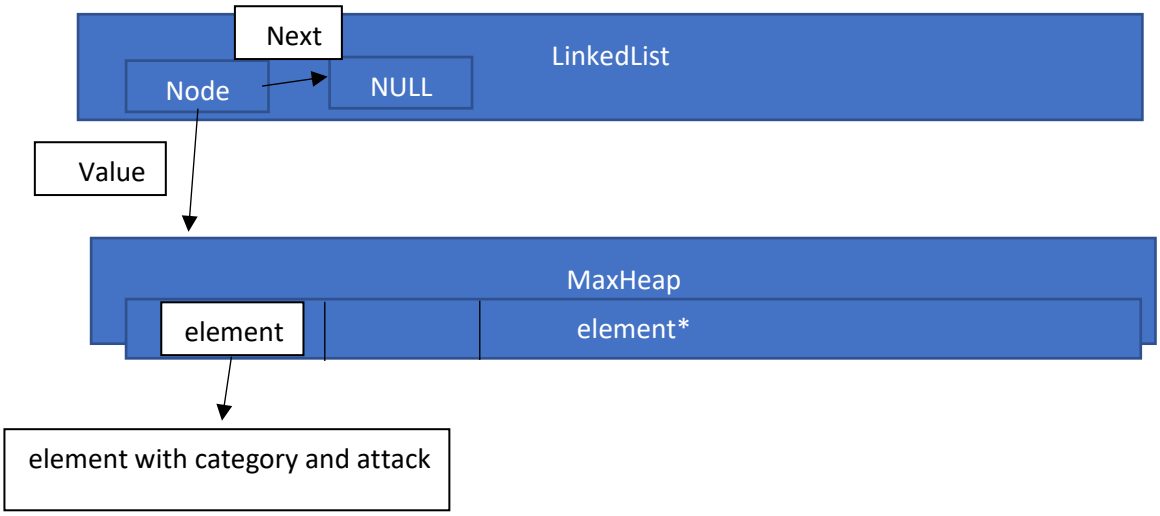
5. BattleByCategory:

a. כמו שכבר אמרתי הסטרקט ממומש בקובץ הC כדי להיות מוסתר מהמשתמש:

```
struct battle_s
{

    LinkedList list;  // linked list of types , each type is heap of elements in this type

    int numberOfCategories;
    getCategoryFunction getCategory;
    getAttackFunction getAttack;
    printFunction printElement;

};
```

b. קיימים 4 מצביעים מסוגים שונים + משתנה מסוג int כלומר 5 משתנים בגודל 4 bytes כול אחד - בסך הכול גודל של 20 bytes.

c. שמירת האיברים בADT נעשה על ידי רשימה מקושרת של ערמות מקסימום. כול ערימה תחזיק את האובייקטים בעלי קטגוריה מהסוג שלה.



.d

```
Battle createBattleByCategory(int capacity,int numberOfCategories,char*
categories,equalFunction equalElement,copyFunction copyElement,freeFunction
freeElement,getCategoryFunction getCategory,getAttackFunction getAttack,printFunction
printElement){

	if(categories == NULL || equalElement == NULL || copyElement == NULL ||
freeElement == NULL || getCategory == NULL || getAttack == NULL || printElement == NULL)
		return NULL;

	Battle battle = (Battle)malloc(sizeof(struct battle_s));
	if(battle == NULL)
		return NULL;

	battle->list = createLinkedList(destroyHeap, equalHeap, printHeap,
shallowCopyHeap, sameCategory);
	if(battle->list == NULL){
		free(battle);
		return NULL;
	}

	// insert the linked list a max heap for each category
	char* word= NULL;
	word = strtok(categories, ",");
	int i;
	status st;
	for(i=0; i< numberOfCategories; i++){
		st = appendNode(battle->list, createHeap(word, capacity, copyElement,
freeElement, equalElement, printElement));
		if (st != success){ // failed to create heap or insert to list it's fatal error
			destroyList(battle->list);
			free(battle);
			return NULL;
		}

		word = strtok(NULL, ",");
	}


	battle->numberOfCategories = numberOfCategories;
	battle->getCategory = getCategory;
	battle->getAttack = getAttack;
	battle->printElement = printElement;

	return battle;

	}
```

יצירת הADT נעשת בצורה החסכונית ביותר שאפשר. נשמרים רק הפונקציות שהאובייקט משתמש בהם בהמשך, כשאר הפונקציות האחרות כבר שמורות בתוך ערמות המקסימום ולכן לא נשמרים פעמיים. ניצור רשימה מקושרת ונשלח אליה את כול הפונקציות הקשורות להעתקת, הריסת והדפסת Heap (אשר שייכות למחלקת MaxHeap) בנוסף לחיפוש על פי קטגוריה. לכול קטגוריה ניצור Heap חדש (שיתווסף לרשימה המקושרת) שישמור את הגודל המקסימלי, שם הקטגוריה וכלל פונקציות הנחוצות לה.

e. ההתאמה היחידה שנדרשה היא כתיבת הפונקציות השל Heap בצורה גנרית. בנוסף בניית הפונקציה sameCategory אשר תקל על חיפוש קטגוריה ברשימה המקושרת.

f. הפונקציה fight ממומשת ממש לפי ההסבר של מה שהיא צריכה לעשות. נעבור ברשימה המקושרת שלנו על כול ערמות המקסימום שיש לנו, כאשר אין איברים בערמה נעבור לאחת הבאה, כאשר בערמה יש איברים נקבל את הפוקימון הכי חזק ממנה ונשמור כמה הוא חזר לאומת הפוקימון שקיבלנו (ע"י חישוב getAttack). נעבור על כלל הפוקימונים הכי חזקים שלנו מכול קטגוריה ונבחר בסוף את הפוקימון שהתוצאה שלו לאומת הפוקימון הנתון היא הגדולה ביותר. לסיום נעשה הדפסות ונחזיר את הפוקימון המנצח. (כאשר לא נמצא פוקימון כלל נחזור).

g. כמו שנאמר בסעיף e הפונקציה שיצרתי להקל על חיפוש קטגוריה ברשימה המקושרת היא:

```
int sameCategory (element heap, element category){
    return strcmp(getHeapId((MaxHeap)heap), (char*)category);
}
```

6. a. עבור המודל של הפוקימון לא היו פונקציונאליות שנאלצתי למחוק עקב הפרדה טובה של הקוד בעבודה קודמת.

b. הפונקציות הדפסה ושחרור הפכתי לגנריות יותר והוספתי את שאר הפונקציות כדי לתמוך בהכנסת פוקימון לזירת הקרבות.

```
// print pokemon
status print_pokemon(element);
// compare pokemon attack - 1 first stronger, 0 same, -1 second stronger
int cmp_pokemon_attack(element, element);
// free memory of pokemon
status free_pokemon(element);
// shallow copy pokemon
element shallow_copy_pokemon (element);
// return type of pokemon
char* pokemon_type(element);
// pokemon fight
int pokemon_fight(element firstElem, element secondElem, int* attackFirst, int* attackSecond);
```

כאשר shallow_copy תחזיר את האיבר עצמו וfight תחשב את ההתקפה בהתאם לקטגוריה של הפוקימון עצמו.

7. a. בקובץ הmain השתמשתי במערך של מצביעים לשמירת ה types כפי שהיה בעבודה הקודמת ואת הפוקימונים שמרתי בBattle בלבד.

b. השתמשתי בפונקציה fgets כדי לקרוא שורה שורה מהקובץ. חילקתי את הקריאה ל-2 חלקים עיקריים : קריאת סוגי הפוקימונים וקריאת הפוקימונים עצמם.
סוגי הפוקימונים יעשה באופן הבא - כאשר השורה הראשונה היא Types נתחיל בקריאת הסוגים כאשר לאחר שורה זו נקבל את כלל הסוגים הקיימים לתור רשימת סוגים. לאחר מכאן נמשיך בקריאת השורות עד שנגיע לשורה Pokemons. בכול שורה נוספת כזו שנקרא ננקה אותה (נמחק רווחים ותווים מיוחדים) ונוסיף את החולשה או החזקה הזאת לאותו סוג.
קריאת הפוקימונים עצמם יעשה באופן הבא בלולאה עד סיום קריאת הקובץ – לכול פוקימון שורה אחת בלבד המתארת אותו , ננקה את השורה (נמחק רווחים ותווים מיוחדים) , ניצור את הפוקימון ונכניס לרשימת הפוקימונים.
קריאת השורות נעשה עד גודל של 300 אבל בשליחת הארגומנטים לפונקציות היצירה עצמם נשלח את הגודל האמיתי של המילה עצמה ובכך נמנע שימוש בזיכרון גדול יותר מהנחוץ.

c. בתחילת המיין קראתי את כלל הארגומנטים שקיבלתי מהמשתמש, קראתי את קובץ הדאטה לתוך המשתנים של הפונקציה ולאחר מכן הצגתי את המניו. לכול אופציה במניו עשיתי תת פונקציה ייעודית משלו שתבצע את הפעולה הנדרשת. בכך אני יכול לפנות לפונקציה עצמה ללא התעסקות עם הקוד של הצגת המניו עצמו , לקחת את הפונקציה ולשים אותה כאופציה אחרת או כדי להשתמש בה במקום אחר בקוד.
בכול מקרה של שגיאת מערכת (אי הצלחה להקצאת זיכרון דינמי) בכול אחת מהאופציות במניו יקרא פונקציית שחרור הזיכרון ולאחר מכן הצגת השגיאה ויציאה מהתוכנית.

לדוגמא שימוש בפונקציה 1 במניו – הדפסת כלל הפוקימונים לפי סוג:

```
case '1':  // Print all Pokemons by types
    done = print_all_pokes(pokes);
    break;
```

תקרא לפונקציה הבאה :

```
status print_all_pokes(Battle pokes){
    /* this function print all the Pokemons in the system */
```

כאשר במקרה של שגיאת זיכרון :

```
if(done == err){
    freeSys(pokes, types, num_types);  // free memory
    printf("No memory available.\n");
    return -1;
}
```

8. a. ניסיתי להסביר כול דבר בקוד והסיבה למה עשיתי אותה ולכן אני חושב שעשיתי את ההפרדה הלוגית הטובה והמעילה ביותר שיכולתי לעשות בקוד.
b. כמו שכבר רשמתי בקובץ ניסיתי לאפשר כמה שיותר שליטה למשתמש כלומר יוצר המיין ולכן אם נחליף את המיין לקובץ אחר לא נצטרך לשנות שום דבר בקוד, המיין המחליף יהיה אחראי על מה יקרה במקרה של שגיאה ואם הוא ירצה לטפל בזה בצורה אחרת הוא יכול – כמו במקרה שלנו שאני בחרתי לשחרר את הזיכרון ולהתריע על שגיאה.

c. ההתמקדות שלי בכתיבת הקוד הייתה לתת את האפשרות למשתמש החליט מה לעשות בכול מקרה בו בפונקציות לא מתפקדות כמו שצריך וזאת בלי לחשוף את אופן המימוש המלא של הפונקציות . התחלתי מכתיבה כלל ה ADT ולאחריהם שימוש בהם בתתי הפונקציות של המיין. השותי את כלל הפלטים עם הקבצים שקיבלנו וביצעתי בדיקות נוספות משלי. יצרתי קובץ make ובדקתי את דליפות הזיכרון.

בנוסף הייתה פונקציונליות שחזרה על עצמה בקובץ המיין והיא חיפוש אינדקס של סוג ברשימת הסוגים ולכן יצרתי פונקציה יעודת לכך :

```
int index_in_types(Type** types, int size, char* search){
    /* this function return the index of the string search in the list types
        if not found return -1 */
```

תודה.