מגיש : ירין בנימין

מבנה הדו"ח בנוי סעיפים אחד אחר השני כמו בהסבר הכללי. ניסיתי לפרט ולדייק כמה שיותר בניסוח כדי שלא תצטרכ/י 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.
- קובץ Pokemon.h הוא קובץ החתימות של הפונקציות אפשריות שהמשתמש יכול לבצע. בנוסף מכיל את הסטרקטים שאפשר ליצור (פוקימון, ביו של הפוקימון וסוג פוקימון).
- קובץ Pokemon.c הוא קובץ הממש את כול הפונקציות שנמצאת בקובץ הH.
מטרת קובץ זה הוא "להסתיר" את מימוש הפונקציות מהמשתמש.
- קובץ MainFirst.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 הפוקימון לשם השלמתן.

2. יצרתי 3 סטרקטים - פוקימון, ביו של הפוקימון וסוג פוקימון. שלושתם נמצאים בקובץ ה-H וזאת כדי "להראות" למשתמש במערכת את המשתנים השונים שיש לכול אחד מהם.
יש לשים לב שפונקציות ההריסה אינן יכולות להחזיר הצלחה או כישלון כי במקרה של כישלון הפונקציה קיבלה NULL דבר שבפונקציה יגרום לחזרה ולא המשכה.

שתי הדפים הבאים יתמקדו על הסטרקטים השונים בעבודה. אנסה להסביר בצורה הטובה יותר את הבחירות שעשיתי והסיבות להן.

א. פוקימון :

a.

```c
typedef struct pokemon_st {
    char*  name;
    char*  species;
    Type* type;
    Bio_Info* bio;
}Pokemon;
```

b. הסטרקט בנוי מ-4 מצביעים כאשר כול מצביע מגודל 4 bytes בסך הכול 16 bytes.

c. הקוד נרשם בצור החסכונית ביותר שאפשר, כול פוקימון צריך להחזיק את השם, זן סוג וביו שלו. (הסבר מפורט יותר בסעיף הבא)

```c
type->name = (char*)malloc((strlen(name)+1) * sizeof(char));
```

לדוגמא בקבלת השם מהמשתמש (או מהקובץ) לא נשתמש בגודל יותר גדול ממה שצריך.

d. פונקציית היצירה היא:

```c
Pokemon* create_pokemon(char* name, char* species, Type* type, double h, double w, int att);
```

עם המשתנים שהפונקציה מקבלת היא עושה העתקה עמוקה לשם ולזן לפי הגודל האמיתי שלו. מעתיקה את המקום בזיכרון של הסוג. עם הגובה, רוחב והתקפה היא יוצרת סטרקט מסוג ביו של פוקימון ושומרת את המצביע שלו.
בסוף הפונקציה מחזירה מצביע לפוקימון שנוצר.

הפונקציה ההורסת שלו היא:

```c
void kill_pokemon(Pokemon* poke);
```

פונקציה זו מקבלת מצביע לפוקימון ומשחררת את 3 המצביעים השייכים לפוקימון.
כדי לשחרר את הביו של הפוקימון נקרא לפונקציה המתאימה.

ב. ביו של פוקימון :

a.

```c
typedef struct bio_info_st {
    double  height;
    double weight;
    int attack;
}Bio_Info;
```

b. הסטרקט בנוי מ-3 משתנים 2 דאבל כול אחד בגודל 8 bytes ואינט אחד בגודל 4 int בסך הכול 24 bytes.

c. הקוד יכל לחסוך במקום נוסף על ידי שימש ב float במקום double ולחסוך ב8 bytes אך העדפתי לשמור על דיוק גבוה יותר.

d. פונקציית היצירה היא:

```c
Bio_Info* create_bio(double h, double w, int att);
```

את המשתנים שהפונקציה מקבלת היא מזינה במשתנים הלוקלים שלה.
בסוף הפונקציה מחזירה מצביע לביו של הפוקימון שנוצר.

הפונקציה ההורסת שלו היא:

```
void del_bio(Bio_Info* bio);
```

פונקציה זו מקבלת מצביע לביו ומשחררת אותו. רציתי לפרק את האחריות של הפוקימון והביו שלו ולכן על אף שלביו אין משתנים לוקלים שצריך לשחרר היא משחררת את עצמה מאשר שהפוקימון עצמו ישחרר אותה .

ג. סוג פוקימון :

a.

```
typedef struct type_st {
    char*  name;
    int counter;
    int stronger_vs_me;
    struct type_st **effective_against_me;
    int weeker_vs_me;
    struct type_st **effective_against_other;
}Type;
```

b. הסטרקט בנוי מ-3 מצביעים ו-3 אינטים כאשר כול אחד מגודל 4 bytes בסך הכול 24 bytes.

c. הקוד נרשם בצורה החסכונית ביותר שאפשר לתנאי הבקשה בתרגיל, לכול מערך דינאמי חייב לשמור את הגודל שלו מכיוון שאסור לשמור זיכרון גדול יותר ממה שצריך (כאשר ללא תנאי זה יכלנו לחסוך 8 bytes ובסוף כול מערך לשמור NULL שנדע מתי נגמר המערך). כמו בסטרקט של הפוקימון הפוקימון לא נשמור יותר מגודל השם שצריך. בכול הוספה או הסרה מרשימת האפקטיב נמחק/נוסיף זיכרון לכמות הנחוצה.

d. פונקציית היצירה היא:

```
Type* create_type(char* name);
```

הפונקציה מקבלת שם, עושה לו העתקה עמוקה ו"מאפסת" את שאר המשתנים. בסוף הפונקציה מחזירה מצביע לסוג פוקימון.

הפונקציה ההורסת שלו היא:

```
void extinct_type(Type* type);
```

פונקציה זו מקבלת מצביע לסוג פוקימון ומשחררת את 3 המצביעים השייכים לה (ללא שחרור הסוגים עצמם (כי אינם מוקצים בזיכרון בצורה דינמית) אלה רק המצביע שמחזיק את הרשימה הזאת).

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:

```
if(poke->species == NULL){
    free(poke->name);
    free(poke->bio);
    free(poke);
    return NULL;
}
```

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 ב-MainFirst לבצע שחרור זיכרון והדפסת שגיאה.

```
if(done == err){
    armageddon(pokes, num_pokes, types, num_types);  // free memory
    printf("Memory Problem\n");
    return -1;
}
```

4.

a. קובץ ה H יצורף בדף הבא.

b. כמו שכבר כתבתי בתחילת העבודה קובץ ה-H הוא קובץ החתימות של הפונקציות אפשריות שהמשתמש יכול לבצע. כלל המימושים של הפונקציות השונות מבוצע בקובץ ה-C ובכך מסתיר מהמשתמש הסופי את המימוש. בנוסף קובץ ה-H מכיל את הסטרקטים שאפשר ליצור (פוקימון, ביו של הפוקימון וסוג פוקימון) וזאת בשביל לאפשר למשתמש ליצור פונקציות נוספות שנוגעות לסטקרטים אלו.

c. בכלל הפונקציות שמשתמשות בסטרקטים וסטרינגים השתמשתי בהעברה by pointer ובכלל המשתנים הפרימיטיבים העברה by value.
הסברים על הפונקציות בהערה מעל כוך פונקציה. כאן הוסיף ואומר שכול יצירת סטרקט תעשה על ידי יצירת מצביע והחזרתו. סטטוסים אפשריים הם success כאשר הפעולה נעשתה בהצלחה, failure כאשר הפעולה לא נעשתה בהצלחה (כאשר הפונקציה מקבלת משתנה כ NULL היא מתייחסת לזה כלא נעשה בהצלחה) ו סטטוס אחרון err במקרה שהקצאה דינאמית של זיכרון לא הצליחה.

```c
/*
 * Pokemon.h
 *
 *  Created on: Nov 19, 2021
 *      Author: yarin bnyamin 208896548
 */

#ifndef POKEMON_H_
#define POKEMON_H_
#include "Defs.h"


typedef struct type_st {
    char*  name;
    int counter;
    int stronger_vs_me;
    struct type_st **effective_against_me;
    int weeker_vs_me;
    struct type_st **effective_against_other;
}Type;

Type* create_type(char* name);
// add type other to the effective_against_me list of me
status add_stronger_vs_me(Type* me, Type* other);
// add type other to the effective_against_other list of me
status add_weeker_vs_me(Type* me, Type* other);
// delete type other to the effective_against_me list of me
status del_stronger_vs_me(Type* me, Type* other);
// delete type other to the effective_against_other list of me
status del_weeker_vs_me(Type* me, Type* other);
// print type
status print_type(Type* type);
// free memory of type
void extinct_type(Type* type);


typedef struct bio_info_st {
    double  height;
    double weight;
    int attack;
}Bio_Info;

Bio_Info* create_bio(double h, double w, int att);
// print bio info
status print_bio(Bio_Info* bio);
// free memory of bio info
void del_bio(Bio_Info* bio);


typedef struct pokemon_st {
    char*  name;
    char*  species;
    Type* type;
    Bio_Info* bio;
}Pokemon;

Pokemon* create_pokemon(char* name, char* species, Type* type, double h, double w, int att);
// print pokemon
status print_pokemon(Pokemon* poke);
// free memory of pokemon
void kill_pokemon(Pokemon* poke);


#endif /* POKEMON_H_ */
```

d. פונקציית הוספת סוג לרשימה (בדוגמא הנ"ל הוספה לרשימת האפקטיביים נגדי) :

```
status add_stronger_vs_me(Type* me, Type* other){
    // add type other to the effective_against_me list of me

    if(me == NULL || other == NULL)
        return failure;

    // check if already exist in the list
    int i;
    for(i=0; i < me->stronger_vs_me ; i++)
        if(me->effective_against_me[i] == other)
            return failure;

    me->stronger_vs_me++;
    if(me->effective_against_me != NULL)
        me->effective_against_me = (Type**)realloc(me->effective_against_me, me->stronger_vs_me*sizeof(Type*));
    else
        me->effective_against_me = (Type**)malloc(sizeof(Type*));


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

    me->effective_against_me[me->stronger_vs_me-1] = other;
    return success;
}
```

במקרה שהפוקימון לא נמצא כבר ברשימה נתחלק לשתי מקרים :
- אין כלל רשימה ניצור רשימה בגודל אחד ונוסיף את הסוג.
- במקרה בו ישנה רשימה כבר נגדיל את הרשימה ב-1 ונצרף את הסוג בסוף הרשימה.
כלומר אף פעם לא נגדיל את הרשימה הנוכחתי יותר מהכמות הנצרכת בפועל.
הוספה בהצלחה תחזיר סטטוס success , במקרה ולא נצליח להקצות זיכרון נחזיר סטטוס err ובמקרה בו הפוקימון כבר נמצא ברשימה או שהפונקציה קיבלה NULL הסטטוס שיוחזר יהיה failure.

פונקציית הסרת סוג מרשימה (בדוגמא הנ"ל הסרה מרשימת האפקטיביים נגדי) :

```
status del_stronger_vs_me(Type* me, Type* other){
    // delete type other to the effective_against_me list of me

    if(me == NULL || other == NULL)
        return failure;

    int i;
    for(i=0; i < me->stronger_vs_me ; i++){
        if(me->effective_against_me[i] == other){

            me->stronger_vs_me--;

            for(;i < me->stronger_vs_me ; i++)
                me->effective_against_me[i] = me->effective_against_me[i+1];

            me->effective_against_me = (Type**)realloc(me->effective_against_me, me->stronger_vs_me*sizeof(Type*));

            if(me->stronger_vs_me != 0 && me->effective_against_me == NULL)
                return err;

            return success;
        }
    }

    return failure;
}
```

במקרה שהפוקימון נמצא כבר ברשימה נדרוס את המקום שלו ע"י הסוג הבא בתור (כך עד סוף הרשימה בלולאה) ולבסוף נקטין את הרשימה ב-1.
כלומר אף פעם לא נשמר המקום של הפוקימון שנמחק ובכך הרשימה הנוכחתי לא תכיל יותר מקומם מהכמות הנצרכת בפועל.
מחיקה בהצלחה תחזיר סטטוס success , במקרה ולא נצליח להקצות זיכרון נחזיר סטטוס err ובמקרה בו הפוקימון לא נמצא ברשימה או שהפונקציה קיבלה NULL הסטטוס שיוחזר יהיה failure.

5. a. 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 – 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.

b. 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 3 במניו – הוספת סוג לרשימת החולשות של סוג אחר :

```
        case '3':  // Add type to effective against me list
            done = add_stronger_against_me(types, num_types);
            break;
```

תקרא לפונקציה הבאה :

```
status add_stronger_against_me(Type** types, int num_types){
    /* this function add type to effective against me list */
```

כאשר במקרה של שגיאת זיכרון :

```
if(done == err){
    armageddon(pokes, num_pokes, types, num_types);  // free memory
    printf("Memory Problem\n");
    return -1;
}
```

6. 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 כללית של קובץ ה H ולאחריו במיין על קריאת הקובץ. משם עברתי למימוש של כלל הפונקציות בקובץ ה C ולאחריו 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.