**דו"ח עבודה מספר 2 C Pokemons**

**Pokemon.c-** במודל הזה מתבצע כל תהליכי הטעינה

1. קריאה מקובץ
2. טעינה של פרטי הקובץ ל-structs המצויניים למטה
3. קיימים מימושים עבור הדפסות הוספת אפקטים במודל הזה

**Pokemon.h-** תחום אחריות של קובץ זה לקשר בין MainFirst.c ל-Pokemon.c

בקובץ ה-MainFirst אנו מקבלים את כל האינפוטים מהמשתמש

מנתיב קובץ הקונפיגירציה, מספר פוקימונים, מספר סוגים

עד קבלת אינפוטים בתפריט המשתמש.

כל האינפוטים הללו נשלחים לקובץ Pokemon.c המבצע את הפקודה

על מבנה ה-Pokedex ומחזיר\מבצע שינוי\מוחק\מוסיף לפי הפקודה שהוכנסה.

**MainFirst.c-** ממשק המשתמש, מתקבל בו תחילה הנתיב לקובץ,מספר הפוקימונים

מספר הסוגים. לאחר טעינה מוצלחת של Pokedex ינתן למשתמש

תפריט המקבל קלטים ובהתאם לכך מבצע את המטודות הממושות

ב-Pokemons.c

**Defs.h-** משמש בעיקר להדפסת תקלות חוזרות

e\_status, e\_bool הועברו לקובץ Pokemon.h מטעמי קומפליציה.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

כל ה-structs בקוד נכתבו ב-Pokemon.h מהסיבה שיש שימוש בכל struct

גם ב-Pokemon.c MainFirst.c

**תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי**הסבר: זה המידע על הפוקימון

יכיל רק את גודל המשתנים הפרימטיביים הנתונים.

מהסטרינג ממנו אני מלקט את המידע ישחורר

בסוף התוכנית.

גודלו יהיה:

Integer attack -גודל 4

Float weight-גודל 8

**תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי** Float Height-גודל 8

הסבר: מבנה זה בנוי מפוינטרים בלבד לכן

גודלו יהיה 4\*sizeof(pointer)

כל שדותיו מצביעים על זיכרון מוקצה בצורה

אופטימלית, למשל Pickachu אז name יצביע על

זיכרון מוקצה בגודל 9. ניהול המידע המלא יוסבר בקריאה מהקובץ.

בסה"כ גודלו יהיה בערך 32.

**תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי**

הסבר: name לפי ההסבר למעלה ויוסבר בהמשך

מצביע בגודל 8.

Pokemons- מערך של מצביעים לפוקימונים

גודולו המקסימלי יהיה num\_of\_pokemons\*sizeof(pokemon\*)

Effective\_me\other מנוהלים בצורה זהה לפוקימונים. גודלם המקסימלי יהיה

Num\_of\_types\*sizeof(Type\*).

**תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי**

הסבר: Pokedex מכיל את הגישה לכל האובייקטים הקיימים

בקוד, רוב המטודות משתמשות בו על מנת לגשת למקום

הרצוי וביצוע פעולו בהתאם.

גודלו יהיה בגודל קבוע של

Num\_of\_pokemons\*sizeof(pokemon\*)

Num\_of\_types\*size(types\*)

2 integers

Num\_of\_pokemons, num\_of\_types

משמש לדעת כמה פוקימונים ישלנו במהלך הטעינה

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטימשמש בעיקר אם נתקלתי בתקלה של טעינת הפוקימונים שנוכל לצאת מהתוכנית בצורה מסודרת ככל האפשר, שחרור כמות האובייקטים במערך זה לעת זו.

אם במידה באיטרציה I תקלה

בהקצאת זיכרון,

Pokedex->num\_of\_types=i-1

לכן unload\_pokedex(pokedex)

ישחרר בצורה נכונה את הטייפס הנמצאים.

החסכוניות שמימוש הזה נובעת מכך שכל המבנים מלבד Bio בנויים על ints

ומצביעים מסוג מסוים. כל המצביעים הקיימים מצביעים על אותו אובייקט בקוד

כך שלא נוצרים עותקים של אובייקטים כלל בקוד. בדרך זו כל אובייקט שנוצר

למשל pickachu בכל מקום אליו מתאים יצביעו לאובייקט יחיד מסוגו

לדוגמא :

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

בקטע קוד זה למשל

Pokemon->name = Pikachu;

Type->name = Electric;

אז ניצור מצביע חדש במערך הפוקימונים של Electric אשר יצביע לאובייקט Pikachu

הקיים במערכת, כמו כן

Pikachu->type = (Type) Electric

ביצעתי הצבעה הדדית בין האובייקטים כך של קיים עותק מלבד המצביעים.

מכאן ניתן לראות שכל המבנים המבוססים על מצביעים בקוד ישתמשו באובייקטים הקיימים שנוצרו פעם אחת במשך כל התוכנית.

יצירת הפוקימון סטנדרטית, בהינתן המידע נבצע השמה של השדות הדרושים,

שדה type ישאר לא מאותחל, יאותחל מבחוץ בעזרת פוקנציית assigen

המצורפת הנ"ל.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

הריסת פוקימון:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

אני משחרר עבור כל פוקימון במערך pokedex->pokemons

עבור כל פוקימון משחרר את אובייקט Bio\_Info אשר לשדות שלו לא הוקצה זיכרון לכן שחרור האובייקט לבד פעולה מספקת.

שחרור כל אובייקט של פוקימון.

שחרור כל pokemon->type יתבצע בעזרת unload\_types

יתר השדות ישוחרר ב- Unload\_pokedex..

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

מקבלים את השורה הרלוונטית str (שורה 2 בקובץ) אשר אליה הוקצה זיכרון בקריאת הקובץ ויוסבר בהמשך. מבצעים הקצאות זיכרון רלוונטיות למערך המצביעים שיהיה

שייך ל-pokedex->types . עבור כל תא נקצה זיכרון למצביע מסוג (Type\*)

וכל מצביע אל אובייקט (Type). השמה של השדות הרלוונטים.

כרגע 0 פוקימונים השייכים לאובייקט, בהמשך בעזרת פוקנציית assigen\_pokemon

נעדכן בהתאם. מערכי האפקטים יהיו מצביע המצביע ל-NULL מטעמי מימוש.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

הריסת הסוג: הריסת מערך הפוקמונים הריק, כל הפוקימונים שוחררו בטרם עת בעזרת פונקציית unload\_Pokemons שהוזכרה קודם.

שחרור מערכי האפקטים, אין צורך לשחרר את התאים מפני ששורה לאחר מכן

ישחורר הסוג, בגלל מצביעים הדדים בכל התוכנית. כל הסוגים ישוחררו אז

כל רשימות האפקטים בתוכנית ישוחררו בצורה מסודרת. לכן רק שחרור מערכי המצביעים של האפקטים מספקת.

לבסוף שחרור types השייך ל- pokedex->types.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

יצירת האובייקט Pokedex\* , השמה של כל השדות לערכי ברירת מחדל.

לאחר מכן בעזרת הפונקציות שהוצגו קודם נתאחל את כל השדות.

פונקציית load\_effect תטען את כל האפקטים לשדות מערכי האפקטים בסוגים.

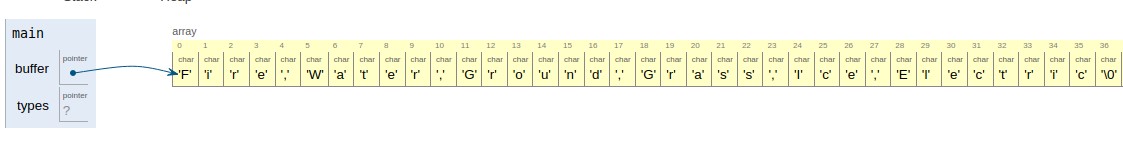
קודם אסביר על מטודה שלדעתי היא הלב של כל התוכנית, **slice\_str**

בקבלת char\* שהיא גם שורה בקובץ הקונפגירציה, בהתאם לצורך

נפריד את כל המילים בשורה הזו בעזרת strtok והשמה של מצביע

עבור כל תחילת מילה הבאה אחרי ה-delimeter ציור c\_tutor.

תמונה שמכילה טקסט, מכשיר, צילום מסך

התיאור נוצר באופן אוטומטי

במקרה המתואר כל מצביע ישמש את לבנות את כל המצביעים ב- pokedex->types[i]

כל המצביעים נבנים על זיכרון שכבר הוקצה בתחילת התוכנית.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטילכן שחרור buffer. הנעשה ב-unload\_pokedex ישחרר את כל הערכים שהתאים מצביעים אליהם. לכן שחרור רק המצביע יהיה מספיק ב-unload\_types.

Load\_effect

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיכאן ניתן לראות שימוש מלא של slice\_str בטעינת האפקטים מקובץ הקונפיגרציה

אינפוט לדוגמא:

Fire effective-against-other:Grass,Ice

צריך לפרק את האינפוט הנ"ל לפי 3 סוגים של delimeters

ופירוק של הסוגים לפי delimeter =’,’

במימוש שהשימוש בסטרינגים הנתונים הינו לחיפוש ומציאה של אובייקטים קיימים במערכת כמו (Type) Grass לכן אשחרר את כל המצביעים שהוחזרו מהתוכנית.

קיים הסבר לפונקציית typeA\_effect\_typeB בקובץ Pokemon.h.

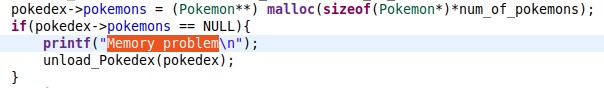
דוגמת קוד להרחבהתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

דוגמאות קוד שהקצאה דינמית לא הצליחה:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט

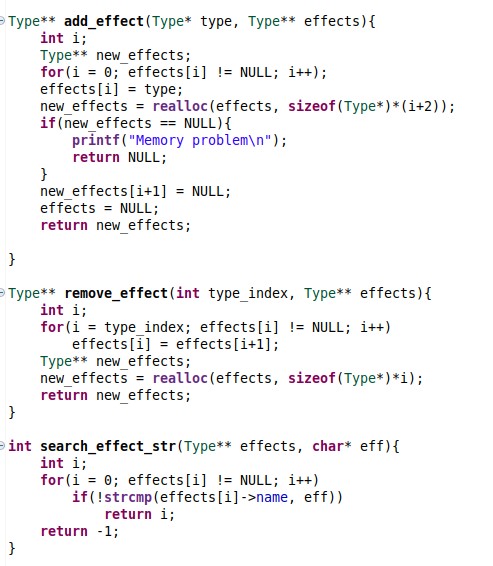
התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

מחיקה הוספה של פוקימונים מרשימה :

מערך האפקטים שלי בנוי כך שבסוף כל מערך קיים ערך null

כך ניתן לעבור על המערך ולספור את כמו האובייקטים הקיימים בו,

בעזרת realloc והגודל הנתון נבנה מערך הגדול ב-1 או קטן ב-1 מהקודם.

וכמובן השמה של ערך NULL. בסוף המערך החדש לפעולות עתידיות נוספות.

כאשר הרשימה ריקה, יהיה מצביע ל-NULL שישמש בהדפסות עתידיות.

בקבלת רשימת האפקטים בהוספה מקבלים את המצביע לסוג ומצביע לרשימת האפקטים, כולם שייכים ל-pokedex

Pokedex->types = effects

Pokedex->types[i](toAdd) = type;

כך בהגדלת המערך המצביע האחרון יצביע לטייפ הקיים, כך נמנע שוב מעותקים ונשתמש באובייקטים קיימים והעברתם בעזרת מצביעים.

המחיקה משחררת את המצביע הדרוש, לא נשחרר את המקום אליו מצביע כי זהו סוג הקיים במבנה נתונים אשר ישוחרר בתום התוכנית.

קריאת הקובץ

אצלי בתכנית הקריאה מהקובץ התבצע בתוך הקובץ Pokemon.c

מהסיבה העיקרית ששמירת כל השורות במערך מצביעים char\* המחזיק

*מצביעים לכל הזיכרונות שהוקצו עבור כל תוכנית במערכת.*

*כל שורה במערכת תשמש עבור שדות האובייקטים. כך בקריאה אחת מהקובץ נקצה את כל הזיכרון הנדרש עבור כל char\* שקיים במבנה הנתונים, דבר שיאפשר גם שחרור קל של רוב הזיכרון שהוקצה במערכת.*

*באמצעות חישוב חמדן חישבתי את כמות השורות המקסימלית שיכולה להיות,*

*כנראה שיכולתי לחסוך על כמות השורות בריצה ראשוניית על הקובץ וספירת כל השורות ולאחר מכן להקצות את כמות השורות הדרושה.*

*כל הפעולות יבוצעו על ידי slice\_str, וכמו שהוסבר קודם לא יקוצה זיכרון נוסף מעבר לנדרש.*

*(מבדיקה שעשיתי יצא כי דרך זו יצאה יקרה יותר, לפי החישוב כנראה שיש סוג רב של טייפים השיטה המוצעת היא המועדפת, אך לא למקרה של fullrun\_data).*



תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

במידה וישנו את קובץ ה-main

לא יובצע שינוי בתוכנית כלל. הדבר הכרחי לכך שהתוכנית תעבוד כראוי זה

נתינת הארגומנטים התקינים (נתיב קובץ קונפגירציה תקין, מספר פוקימונים, מספר סוגים). כך התוכנית תטען את המבנה נתונים בצורה תקינה תמיד.

שינוי קובץ ה-Main ישנה את הפקודות שניתנות לקובץ הפוקימונים, דרישותיו יענו בקובץ ה Pokemon.h ובמידת הצורך ימוששו מטודות על מבנה הנתונים הקיים של הפוקימונים בקובץ Pokemon.c.

בהרצת התוכנית עם valgrind ישנה תקלה שלא הצלחתי לתקן של uncondtional jump, מבדיקות שעשיתי אינה פוגעת ברצף התוכנית או בדליפות הזיכרון.