הצעת פרויקט – י"ג הנדסת תוכנה

סמל מוסד: 471029

שם מכללה: מכללת אורט הרמלין נתניה

שם הסטודנט: עידו הירש

ת.ז הסטודנט: 214290249

שם הפרויקט: בניית **Compiler**

רקע תיאורטי

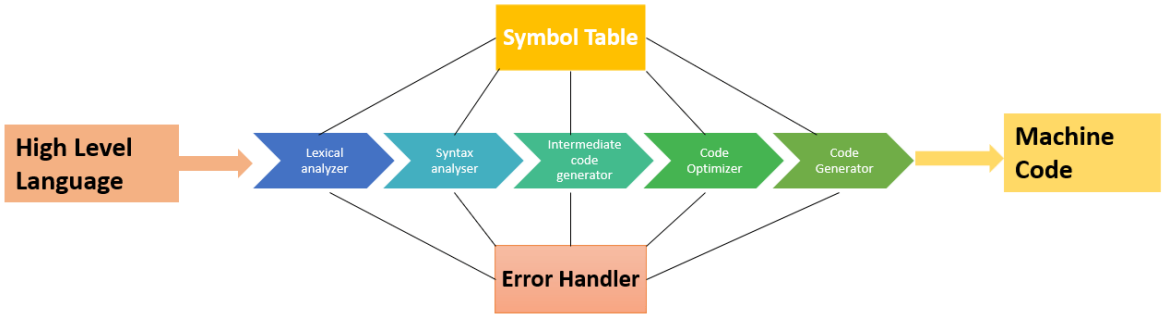
אלגוריתם

אלגוריתם הוא דרך שיטתית וחד-משמעית לביצוע של משימה מסוימת, במספר סופי של צעדים. אלגוריתם נועד לפתור בעיה או לבצע משימה על ידי מעקב אחר הוראות, אלגוריתם משמש לדוגמה עבור חישובים, קבלת החלטות, עיבוד מידע ועוד.

אפליקציה

היא סוג של תוכנת מחשב אשר מנצלת את יכולות המחשב ישירות לביצוע משימות אותן המשתמש מבקש לבצע.

שלבי ה - **Compiler**



שלב 1: **Lexical analysis**

Lexical analysis, או ניתוח מילוני בעברית, הוא השלב הראשון בתהליך הקומפילציה.

בשלב זה, ה – Compiler מנתח את הקלט (הטקסט) אות אחר אות ומקבץ את האותיות למילים ורצפים בעלי משמעות על ידי זיהוי Tokens.

מעביר את ה – Tokens שיצר לשלב הבא, Syntax analysis או ה – Parser.

תפקידים עיקריים של שלב זה

- זיהוי יחידות מילוניות בקוד המקור.
- סיווג יחידות מילוניות אלו לסוגים השונים. כמו קבועים, מילים שמורות, ושמירה שלהם בטבלאות שונות.
- התעלמות מהערות בקוד המקור.
- השמה של Identifiers (מזהים) ב – Symbol table (טבלת הסמלים).

שלב 2: **Syntax analysis - Parser**

Syntax analysis, או ניתוח תחבירי בעברית, הוא שלב העוסק בזיהוי מבנים בקוד.

מטרתו העיקרית של שלב זה היא לקבוע אם הטקסט עומד בפורמט הרצוי של השפה שלנו.

קובע את המבנה והתחביר של שפת המקור.

תפקידים עיקריים של שלב זה

- קבלת Tokens מהשלב הראשון, Lexical analysis.
- בדיקה אם הביטוי הנוכחי נכון תחבירית או לא.
- דיווח על כל השגיאות התחביריות.

- בניית מבנה היררכי הידוע גם כ – Parse tree או Syntax tree.

שלב 3: Semantic analysis

Semantic analysis, או ניתוח סמנטי (חקר המשמעות), הוא השלב השלישי בתהליך הקומפילציה.

שלב זה בודק את העקביות הסמנטית של הקוד.

בעזרת ה – Syntax tree וה – Symbol table, מוודא שהקוד הנתון הוא נכון מבחינה סמנטית.

לדוגמא, בשלב זה נבדוק חוסר התאמת טיפוסי משתנים, קריאה לפעולה בלי כל הארגומנטים שלה, שימוש במשתנים לא מוצהרים, ועוד.

שלב 4: Intermediate code generator

ברגע שהשלב השלישי, הניתוח הסמנטי, מסתיים, הקומפיילר מייצר קוד ביניים בשביל המכונה הרצויה.

קוד הביניים נמצא בין שפה עילית לבין שפת מכונה, והוא צריך להיות מיוצר כך שיהיה קל לאחר מכן לתרגם אותו לשפת מכונה.

שלב 5: Code optimizer

מטרתו של שלב זה הוא לשפר את קוד הביניים שנוצר בשלב הקודם, שלב 4. סידור מחדש של שורות קוד, והסרת שורות קוד לא נחוצות, על מנת ליעל את ביצוע התוכנית, וכדי לבזבז כמה שפחות משאבים (זמן, מקום).

שלב 6: Code generator

השלב הסופי והאחרון בתהליך הקומפילציה.

מקבל את הקוד המיועל מהשלב הקודם, Code optimizer, ומייצר ממנו את ה – Object code.

Symbol table

ה – Symbol table או טבלת הסמלים, מכילה רשומה עבור כל Identifier (מזהה) עם שדות עבור התכונות של אותו המזהה.

טבלה זו עוזרת לקומפיילר למצוא רשומה של מזהה כלשהו בתוכנית ולקבל את הפרטים עליו באופן מהיר יחסית.

ה – Symbol table עוזרת גם ב – Scope management.

טבלה זו לוקחת חלק בכל אחד מהשלבים שצוינו לעיל, ומתעדכנת בהתאם.

Error handling

בכל שלב ושלב בתהליך הקומפילציה יכולות להיווצר שגיאות. שגיאות אלו צריכות להיות מדווחות חזרה למתכנת, ולרוב מוצגות בתצורה של הודעה.

תיאור הפרויקט

Compiler, או מהדר בעברית, הוא תוכנית מחשב המתרגמת שפת מחשב אחת לשפת מחשב אחרת. המהדר הקלאסי, שהוא מה שאני אצור, מקבל כקלט תוכנית הכתובה בשפה עילית, במקרה שלי do, ומתרגם אותה לתוכנית בשפת מכונה.

תיאור השפה do

מקור שמה של שפת do מגיע מקיצור שמי, Ido Hirsh, ומהמילה "תעשה!" באנגלית, מילה המעוררת מוטיבציה לעבודה ועשייה.

שפת do דומה בסינטקס שלה לשפות התכנות C ו – ++C.

השפה תכיל:

- ליטרלים
- משתנים
- תנאים
- לולאות
- פעולת השמה
- אופרטורים חשבוניים
- אופרטורים לוגיים
- הערות

תיאור הבעיה האלגוריתמית

מטרתו של Compiler הוא לתרגם משפה עילית לשפת מכונה, ולעשות זאת בצורה היעילה ביותר הן מבחינת זמן והן מבחינת מקום.

תת-בעיות אלגוריתמיות בפרויקט:

- אלגוריתם ניתוח טקסט מתוכנית המשתמש
- אלגוריתם לקביעה האם המשתמש כתב נכון בשפה שלנו
- אלגוריתם ליצירת קוד ביניים
- אלגוריתם לייעול קוד הביניים
- אלגוריתם להמרת קוד הביניים לקוד היעד

תהליכים עיקריים בפרויקט

עיצוב שפת התכנות שלי, do

השלב הראשון ביצירת קומפיילר הוא ההחלטה על איזו שפה הקומפיילר הולך לתרגם.

אני בחרתי ליצור שפה חדשה משלי, עם חוקים משלה, ו - grammar משלה, אותה אני אעצב, כלומר אקבע את החוקים והכללים שיצרו את השפה.

שמה יהיה do. גם קיצור של השם שלי באנגלית, Ido, וגם "תעשה!" בעברית, מילה שנותנת מוטיבציה.

שלבי ה – Compiler

שלב 1: **Lexical analysis**

שלב 2: **Syntax analysis - Parser**

שלב 3: **Semantic analysis**

שלב 4: **Intermediate code generator**

שלב 5: **Code optimizer**

שלב 6: **Code generator**

GUI

בשלב זה אצור סוג של IDE (integrated development environment), לשפה שלי, do, שישתמש בקומפיילר שאצור.

זה יהיה השלב האחרון בפרויקט שלי.

תיאור הטכנולוגיה

תכנות מונחה עצמים – **OOP (Object Oriented Programming)**

תכנות מונחה עצמים היא תבנית תכנות, המשתמשת ב"עצמים" בעלי תכונות לפתרון בעיות שונות. התכונות של העצמים יכולים להיות מטיפוסים פשוטים הקיימים בשפה (int, char וכו'), ויכולים גם להיות מ - "עצמים" אחרים נוספים שכבר שהוגדרו בתוכנית.

תכונות שונות של תכנות מונחה עצמים הן:

- פולימורפיזם
- ירושה
- היררכיה
- הפשטת נתונים

דרך נפוצה ליצירת עצמים היא באמצעות מחלקות. בין מחלקות יכולים להיות קשרים שונים. דוגמא לקשר היא ירושה.

אפליקציה

ישנם כמה סוגים של אפליקציות:

- Desktop application – אפליקציה הרצה על מערכת ההפעלה של המחשב.
- Mobile application – אפליקציה הרצה על מערכת ההפעלה של סמארטפון / מכשיר נייד. לדוגמא, iOS ו – Android.
- Web application – אפליקציה הרצה בדפדפן.

ישנם Compilers היכולים לרוץ כאפליקציות על כל אחת מהפלטפורמות הנ"ל.

ה – Compiler שלי יהיה מסוג Desktop application.

מכונת מצבים

מכונת מצבים היא מודל מתמטי חישובי. זוהי מכונה מופשטת בעלת מספר סופי של מצבים. כך שבהתבסוסת על המצב הנוכחי, והקלט הניתן, מחזירה פלט או עוברת למצב אחר.

שפות תכנות

++C - השפה בה אכתוב את ה – Compiler לשפה do.

#C / Java - באחת מהשפות האלה אכתוב את ה – IDE.

סביבת עבודה

OS

- Windows 10

Code editor

- Visual Studio Code
  - version 1.61.2 (user setup)

C++ compiler

- g++
  - (x86_64-win32-seh-rev0, Built by MinGW-W64 project) 8.1.0

לוחות זמנים

	משימה	מועד אחרון להגשה/ביצוע
1	עיצוב השפה do	12/2021
2	בניית קומפיילר – שלב ראשון: Lexical analysis	01/2022
3	בניית קומפיילר – שלב שני: Syntax analysis	02/2022
4	בניית קומפיילר – שלב שלישי: Semantic analysis	02/2022
5	בניית קומפיילר – שלב רביעי: Intermediate code generator	02/2022
6	בניית קומפיילר – שלב חמישי: Code optimizer	03/2022
7	בניית קומפיילר – שלב שישי: Code generator	03/2022
8	יצירת IDE (GUI)	04/2022

חתימות

סטודנט

רכז המגמה

אישור משרד החינוך