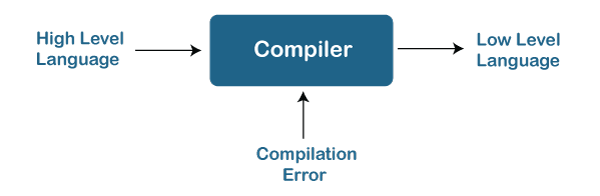
תכנון פרוייקט

פיתוח קומפיילר



שם סטודנט: ליחי סויסה

ת.ז: 214907651

שם פרויקט: ACE compiler

שם מנחה: מיכאל צ'רנובילסקי

מבנה וארכיטקטורת הפרויקט

**תהליך ההידור הוא תהליך המורכב מחמישה חלקים מרכזיים, בפרק זה אתמקד בחלקים אלו ואפרט עליהם**

**חלק ראשון - ניתוח מילולי**

על חלק זה אחראי הלקסר והמתכנת, בחלק זה בונה השפה מגדיר את שפת התכנות בדרך כלל בעזרת [קונבציית BNF](https://condor.depaul.edu/ichu/csc447/notes/wk3/BNF.pdf) שהיא הקונבנציה הנפוצה ביותר לתיאור שפות תכנות

לאחר קבלת מחרוזת הקלט (בצורת קובץ או פשוט רצף תווים) הלקסר ימיר אותה לאסימונים מוגדרים לפי חוקי הדקדוק שהגדיר בונה השפה בעזרת אלגוריתם כלשהו, בדרך כלל מממשים חלק זה על ידי [אוטומט סופי דטרמיניסטי.](https://he.wikipedia.org/wiki/%D7%90%D7%95%D7%98%D7%95%D7%9E%D7%98_%D7%A1%D7%95%D7%A4%D7%99_%D7%93%D7%98%D7%A8%D7%9E%D7%99%D7%A0%D7%99%D7%A1%D7%98%D7%99)

**חלק שני - ניתוח תחבירי**

בחלק זה נלקחים האסימונים מהשלב הקודם ונבדקת תקינותם על פי המבנה התחבירי של השפה, כלומר נענה על השאלה האם רצף מחרוזות\תווים מסויים עומד בתקן השפה שהוגדר מראש? תהליך זה יוצר, במידה ואין שגיאות, מבנה הנקרא "עץ בינארי אבסטרקטי" שהוא חלק חשוב ומרכזי בתרגום השפה.

**חלק שלישי - ניתוח סמנטי**

בשלב זה נשתמש בעץ בינארי אבסטרקטי שהתקבל מהשלב הקודם ובטבלת סמלים על מנת לוודא שהקוד הכתוב בשפה הוא נכון מבחינה סמנטית, כלומר, נבדוק עבור כל אופרטור האם האופרנדים הם מהטיפוס הנכון (למשל בתרגילים חישוביים יש לוודא כי האופרנדים משני צידי האורטור תואמים למה שהוגדר בשפה כתקין) דוגמא נוספת היא וידוא כי כל מזהה שנעשה בו שימוש בתוכנית הוצהר קודם לכן (תלוי שפה).

**חלק רביעי - תרגום לשפת ביניים ואופטימיזציית קוד**

שלב זה הוא השלב האהוב עליי מכיוון שבו נתרגם את קוד המקור למעין "שפת אמצע" שממנה יהיה קל יותר לעשות את המעבר לשפת מכונה, בנוסף שלב זה מאפשר הוספת אופטימזציות לקוד המקור ושימוש יעיל יותר בשפה גם אם המשתמש כתב קוד בצורה לא יעילה, כאן המקום של המהדר להראות את הייחודיות ואת התפקיד שלו.

**חלק חמישי - הפקת קוד היעד**

חלק זה הוא החלק הסופי כאמור, של תהליך ההידור, בחלק זה ניקח את הקוד המיוצג בשפה אמצעית הקרובה לשפת מכונה (שפת אסמבלי לדוגמא) ונתרגם אותו לשפת מכונה טהורה, תהליך זה ניתן להשיג בעזרת קומפיילרים קיימים אשר תפקידם להמיר בין שפה נמוכה לשפת מכונה (כגון MASM) .

בנוסף, במהלך תהליך ההידור, תפקיד המהדר הוא גם להתריע על שגיאות משתמש ולהחזיר פלט בהתאם,, שגיאות אלה נקראות "שגיאות קומפילציה" והן מתחלקות לשלושה סוגים עיקריים

**סוג ראשון - שגיאות לקסיקליות**

איות שגוי של אסימונים שמורים, איות שונה של מזהה מוצהר וכדומה

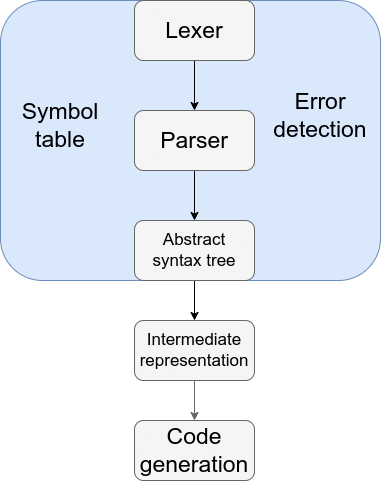
**סוג שני - שגיאות תחביריות**

שימוש שגוי בשפה כגון פתיחת סוגריים ללא סגירה, הצהרה לא חוקית של משתנה מסוג מסוים וכו'

**סוג שלישי - שגיאות סמנטיות**

למשל חיבור בין משתנה מסוג תו ומשתנה מסוג מספר טבעי

תרשים עיצוב מחלקות



פירוט מחלקות:

* Lexer
  + **תפקיד:** עיבוד הקלט והפרדה למשפחות אסימונים, כאן גם נבצע זיהוי שגיאות לקסיקליות בתוכנית ונחזיר קוד שגיאה בהתאם
  + **קלט:** קובץ שבו נכתב קוד המקור
  + **פלט:** רשימת אסימונים
* Parser
  + **תפקיד:** מחלק TOKENS לקבוצות ומשפחות על מנת להפוך אותם לבסיס עבור השלב הבא שהוא עץ בינארי אבסטרקטי. הפרסר מזהה מילות מפתח אותן הוא ממיין לקטגוריות שונות שנקבעות על פי סוג מילת המפתח והקשרה.
  + **קלט:** רשימת האסימונים שהתקבלה בשלב הקודם מהלקסר
  + **פלט:** בסיס לעץ בינארי אבסטרקטי (יפורט בהמשך)
* (AST (Abstract Syntax Tree
  + **תפקיד:** התפקיד של עץ בינארי אבסטרקטי הוא לחלק את הקוד המפורסר לתתי משפחות ומחלקות על פי הגדרות שניתנו מראש העץ הבינארי מזהה משפחות שונות של מילות מפתח ומחלק אותן לקבוצות קטנות
  + **קלט:** קוד ששעבר תהליך פרסור על ידי הפרסר (הבסיס לעץ בינארי אבסטרקטי)
  + **פלט:** קוד מחולק לתתי משפחות ומוכן לעבור לשלב התרגום לקוד המיוצג על ידי שפת אמצע
* intermediate representation
  + **תפקיד:** מייעל ומסדר את הקוד וממיר את שפת הקוד לשפה אמצעית אשר נראית דומה לאסמבלי בדרך כלל וקל יותר להמיר אותה לשפת מכונה ולהריץ.
  + **קלט:** הוא קוד ממויין ומסודר לקטגוריות בעץ בינארי מופשט.
  + **פלט:** הוא קוד בשפה אמצעית שדומה לאסמבלי אשר מיועל ומוכן להפוך

לקוד בשפת מכונה.

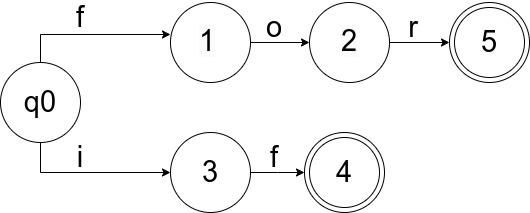
* Code generation
  + **תפקיד:** קורא את ייצוג הביניים וממיר אותו לשפת מכונה ויוצר ממנו object.file/executable file הוא ממיר את ייצוג הביניים הקרוב לאסמבלי לפקודות אסמבלי אשר מיוצגות בייצוג הביניים ומשם הקוד מומר ישירות לשפת מכונה ומתווסף לקובץ ההרצה הסופי
  + **קלט:** ייצוג ביניים של הקוד שקרוב יותר לאסמבלי אשר מיועל ומוכן להפוך לשפת מכונה
  + **פלט:** קוד הרצה אשר מכיל את הקוד המומר לשפת מכונה ויכול לרוץ על המחשב.

מבני נתונים

כאן אפרט על מבני הנתונים שהפרויקט ידרוש

בפרק זה לא ארחיב על המימוש יתר על המידה אלא אשאר בתחום התיאורטי

על מנת לבצע את תהליך התרגום עלינו להשתמש באוטומט סופי דטרמיניסטי כפי שהוזכר [קודם לכן](#ljvpho7n00r6) הנה שרטוט מופשט של האוטומט שבו אשתמש:



את האוטומט אממש באמצעות מטריצה, שורות המטריצה ייצגו את המצבים אליהם ניתן להגיע ועמודות המטריצה הם תווי הקלט, כלומר, בתא [I,J] במטריצה נמצא אינדקס המצב שיש לעבור אליו

כלומר עבור האוטומט המוצג לעיל המטריצה תיראה כך:

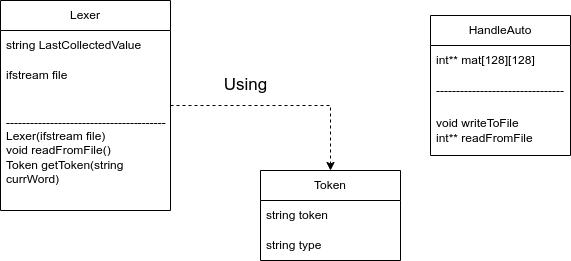
| f | i | r | o | f |  |
| --- | --- | --- | --- | --- | --- |
| 1- | 3 | 1- | 1- | 1 | 0 |
| 1- | 1- | 1- | 2 | 1- | 1 |
| 1- | 1- | 5 | 1- | 1- | 2 |
| 4 | 1- | 1- | 1- | 1- | 3 |
| 1- | 1- | 1- | 1- | 1- | 4 |
| 1- | 1- | 1- | 1- | 1- | 5 |

כאשר 1- מייצג מצב שאין ממנו מעבר למצב אחר עבור אותו מצב ותו

מבנה זה מאפשר גישה לערך(שליפה) בסיבוכיות (1)O

תיאור מערכת

בפרק זה אציג תרשים עבור מחלקת הLexer



תיאור מחלקות

Lexer

| כותרת הפונקציה | הסבר |
| --- | --- |
| Lexer(fstream file) | פעולה המאתחלת מחלקת לקסר חדשה עם קובץ לקריאת מידע |
| Void readFromFile() | פעולה הקוראת מילים מקובץ |
| Token getToken(string currWord) | פעולה המחלצת אסימון ממחרוזת קלט הנקראת מתוך קובץ בעזרת [פעולת קריאה](#y1jjow33etce) |

HandleAuto

| כותרת הפונקציה | הסבר |
| --- | --- |
| Void writeToFile() | פעולה הכותבת מטריצה לתוך קובץ בינארי |
| int\*\* readFromFile() | פעולה הקוראת מטריצה מתוך קובץ בינארי |

תיאור השפה

בפרק זה אעסוק בתיאור שפת התכנות שותה אני ממציא

איעזר בפרוטוקול BNF כאמור פרוטוקול נפוץ לתיאור שפות תכנות

<type> ::= "int" | "float" | "char"

<identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>

<letter> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"

<digit> ::= "0" | "1" | ... | "9"

<assignment> ::= <identifier> ":=" <expression>

<factor> ::= <identifier> | <number> | "(" <expression> ")"

<term> ::= <factor> | <factor> "\*" <term> | <factor> "/" <term>

<expression> ::= <term> | <term> "+" <expression> | <term> "-" <expression>

<expression-statement> ::= {<expression>}? ;

number> ::= <digit> | <digit> <number>

<if-statement> ::= "if" "(" <condition> ")" "{" <statement-list> "}" ["else" "{" <statement-list> "}"]

<condition> ::= <expression> <relational-operator> <expression>

<relational-operator> ::= "==" | "!=" | "<" | ">" | "<=" | ">="

<while-loop> ::= "while" "(" <condition> ")" "{" <statement-list> "}"

<for-loop> ::= "for" "(" <for-initialization> ";" <condition> ";" <for-increment> ")" "{" <statement-list> "}"

<for-initialization> ::= <assignment> | ε

<for-increment> ::= <assignment> | ε

ביבליוגרפיה

בפרק זה אפרט את מקורות המידע שלי

מקור 1: [ספר בנושא קומפיילרים](https://lib.fbtuit.uz/assets/files/Alfred-V.AhoMonicaS.LamRaviSethiJeffreyD.Ullman-Compilers-PrinciplesTechniquesandTools-Pearson_AddisonWesley2006.pdf)

מקור 2: [ספר של האוניברסיטה הפתוחה בנושא אוטומטים](https://books.google.co.il/books?id=G3-kKfR-tNQC&printsec=frontcover&hl=iw&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)

מקור 3: [ספר אינטרנטי על BNF](https://condor.depaul.edu/ichu/csc447/notes/wk3/BNF.pdf)