

ACE programming language compiler



שם תלמיד: ליחי סויסה
תעודת זהות: 214907651
שם מנהה הפרויקט: מיכאל צ'רנובילסקי
שם המנהה למבני נתונים: אורי וולטמן
תאריך: מאי 2024 תשפ"ד

תוכן עניינים

5.....	מבוא.....
6.....	תיאור הפרויקט
6.....	מטרת הפרויקט
7.....	הקדמה.....
7.....	מונחי יסוד
7.....	קצת על שפת ACE
8.....	בני השפה
8.....	קבועים - literals
8.....	משתנים – Identifier
9.....	ביטויים – Expressions & Statements
11.....	תכולת השפה.....
11.....	השמה
11.....	תנאים ולולאות
12.....	הערות – Comments
12.....	פונקציות – Functions
13.....	דיקוק השפה
13.....	תחביר השפה
13.....	רשימת האסימונים בשפה:
14.....	BNF
14.....	דוגמא לקוד בשפת ACE
15.....	רקע תיאורטי
15.....	מהו קומפיילר
15.....	Compiler vs. Interpreter
15.....	למה נדרש קומפיילר
16.....	כיצד עובד קומפיילר
16.....	הסבר קצר על שלבי תהליך הקומפיילציה
16.....	• לkompile
16.....	• פרט
17.....	• עץ ביןארי אבסטרקטי
17..... intermediate representation
17.....	• קומפול לשפת מכונה
18.....	מודולים נוספים
18.....	Register allocation
18.....	Assembly, linking and loading

18.....	Symbol table
18.....	Error handler
19.....	מכונות מצבים
19.....	הגדלה
19.....	רכיבים עיקריים
19.....	פעולה
19.....	סוגים של מכונות מצבים
19.....	מכונות מצבים דטרמיניסטיות (:DFSM)
19.....	מכונות מצבים לא דטרמיניסטיות (:NFSM)
19.....	"ישומים"
20.....	יתרונות
20.....	חרשות
20.....	שפה פורמלית
20.....	מאפיינים עיקריים
21.....	הגדרת מטרות ומשימות
21.....	מטרת הפרויקט:
21.....	אפיון שפת התוכנות:
21.....	מגבליות הפרויקט:
21.....	אתגרים בפרויקט:
21.....	הנדשת הבעה האלגוריתמית
22.....	אתגרים ספציפיים
22.....	פתרונות
22.....	ניתוח הבעיות בכל שלב
22.....	ניתוח לקסיקלי ((Lexical Analysis))
22.....	ניתוח תחבירי ((Parsing))
23.....	ניתוח סמנטי ((Semantic Analysis))
23.....	יצור קוד ((Code Generation))
23.....	סקירת אלגוריתמים בתחום הבעה
23.....	מונחים
24.....	Left Factoring
25.....	אלגוריתמי פרסר
25.....	Top Down Parsing (TDP)
26.....	LL Parser
26.....	Recursive Descent Parsing
26.....	Back tracking
26.....	Bottom Up Parsing (BUP)
26.....	Reduce Shift
27.....	LR Parser
27.....	Operator Precedence Parser
27.....	LALR parser
28.....	דוגמא לטבלאות הניתוח של LALR parser:
29.....	ניתוח שלבי אלגוריתמיקה
29.....	ניתוח מילוני - Analysis Lexical
30.....	דוגמא לתרשים האוטומט הסופי

31.....	ניתוח תחבירי – Analysis Syntax
32.....	תרשים פעולה LALR parser
32.....	Parsing table & Stack
32.....	Action table
32.....	Goto table
33.....	עץ ניתוח
34.....	מימוש LALR parser
34.....	דוגמאות משפט ACE
34.....	טבלאת goto & action
35.....	ניתוח סמנטי
35.....	מהו ניתוח סמנטי?
35.....	ההבדל בין ניתוח תחבירי לניתוח סמנטי:
35.....	מטרות ניתוח סמנטי:
35.....	יתרונות ניתוח סמנטי:
35.....	טכניות לניטוח סמנטי:
36.....	בדיקה ייחודיות שמות המשתנים
36.....	אסטרטגיה לפתרון
36.....	הפקת קוד
36.....	מטרות יצירת קוד
36.....	אסטרטגיות יצירת קוד
36.....	שיקולים ביצירת קוד
37.....	דוגמה:
37.....	טבלת סמלים
37.....	מטרות טבלת הסמלים:
37.....	יצוג טבלת הסמלים:
38.....	יתרונות שימוש במילון:
38.....	ניהול תחומי גישה:
38.....	טיפול בשגיאות
38.....	גישות שונות לטיפול בשגיאות
39.....	Top down level design
39.....	Compiler
39.....	Lexer
40.....	Parser
40.....	Semantic analyzer
40.....	Code generation
41.....	מודולים
41.....	Token
42.....	Lexer
43.....	Parser
45.....	Stack
47.....	semanticAnalysis
49.....	Code generation
52.....	אלגוריתם ראשי
54.....	מבנה נתונים

54.....	מנתח מילוני – Analysis Lexical
54.....	מנתח תחבירי – Analysis Syntax
55.....	Parse Table
56.....	Parse Stack
56.....	דקדוק השפה – Rules Production
56.....	עץ הניתוח – Tree Parse
57.....	מנתח סמנטי – Analysis Semantic
58.....	Hash table
59.....	סביבת העבודה
59.....	עורך קוד
59.....	מערכת הפעלה
59.....	שפת תכנות
59.....	קומפיילר
60.....	מדריך למשתמש
60.....	דרישות
61.....	סיכום אישי
61.....	מסע מרתק בעולם קומפיילרים: סיורו של סטודנט צייר
62.....	במידה והיה לי זמן נוספת בפרויקט הייתי רוצה לבצע מספר דברים
62.....	סיכום:
63.....	ביבליוגרפיה
64.....	קוד הפרויקט
64.....	Dfa
69.....	Lexer
73.....	Parser
79.....	Production
80.....	semanticAnalysis
84.....	Stack.hpp
85.....	Symbol
86.....	Token.hpp
87.....	Types.hpp
87.....	Dfa
91.....	Code generation

מבוא

תיאור הפרויקט

פרויקט זה עוסק בפיתוח מהדר (compiler) אשר תפקידו לתרגם קובץ טקסט כלשהו אשר רשום בשפת תכנות עם מבנים מסוימים לשפת אסמבלי 64 בית אשר ניתן להריץ על מחשב סטנדרטי, לצורך כך בניתי את שפת התכנות ACE.

שפה תכנות הינה סט של חוקים תחביריים וסמנטים המגדירים תהליכי מסוימים שנרצה שיבצעו על ידי המחשב, לצורך בניית המהדר נוצרה לשפת תכנות קודם לכך.

כאשר מגדירים שפת תכנות, מתייחסים כאמור לשלווה מישורים: מילוני, תחבירי ולשוני. המישור המילוני מגדיר אילו מילים שייכות לשפה, ואילו לא.

לדוגמא, המילה if היא מילה המקובלת בשפת C בעוד שהמילה else# איןנה.

המישור התחבירי מגדיר אילו רצפי מילים של השפה הם חוקים, ואילו הם לא.

לדוגמא, הרצף; 3 = int x והוא רצף חוקי בשפת C, בעוד שהרצף if x is 5 איןנו.

המישור הלשוני מתייחס למשמעות רצפי המילים, והוא מגדיר חוקים כלליים שכחיברים להתקיים בכל רצף מילים בשפה.

לדוגמא, חובת הוצאה – לפני שימוש במשתנה, קיימת חובה להציגו עליו.

הקומפイルר עושה שימוש בהנדשת השפה על מנת לנתח את קטע הקוד שנקלט, וכך לייצר את תוכנית היעד.

מטרת הפרויקט

מטרתי בבחירה בפרויקט זה היא להעמק את ידיעותיו וכולותיו ב"אחרי הקלעים" של עולם התכנות, על ידי בניית קומפイルר אני מבין יותר טוב את תהליך בניית שפת תכנות ומתרסה במגוון דברים אשר יעמיקו את הבנייה בעולם התכנות בכלל ובעולם הלוואו לבסוף בפרט

- בניית כל חלק מהדר בצורה יעילה יחסית
- למידה מעמיקה על תהליך ההידור
- יצירת שפה חדשה מאפס
- פיתוח יכולות למידה אישיות

מכיוון שכבר בעבר כבר בנייתי מהדר אני מודע לתהליך ולרמת הקושי, אך בניגוד לשפה הראשונה שכחבתי הפעם לא יכול בטלולוגיות חיצונית (בעבר השתמשתי ב & flex Bison) אלא אבנה הכל בעצמי ללא עזרה חיצונית (למעט ספריות של מבני נתונים ב CPP)

הקדמה

邏輯基礎

לפני שאסביר על תהליך בניית המהדר ארצה להציג בפני מי שקורא את ספר הפרויקט מונחים בסיסיים עליהם אבנה במהלך כתיבת הספר וחוובה על הקורא להבין ולדעת על מנת לא לפגוע ברכף הקריאה והבנת הפרויקט.

שפת תכנות - כאשר אנחנו מדברים על שפת תכנות אנחנו בעצם מדברים על אוסף הנדרות שכאשר נרכיב אותן ביחד ניצור תוכנית בשפת התכנות, מטרת שפת התכנות היא להוות מעין גשר בין האדם המתכונת לבין המכשיר הטכנולוגי שהוא רוצה להפעיל, מבחינת המחשב לדוגמא, אין משמעות לשפת התכנות עצמה אלא למזה שהוא הופכת אליה לאחר תהליך ההידור, אך מבחינת האדם היא קריטית ביותר שכן קשה מאוד לתכנת בשפת מכונה המורכבת מ-2 ספרות שהן 0 ו-1.

הידור - לשם כך נוצרו המהדרים (קומפיילרים) אשר כל מטרתם היא להמיר בין שפת תכנות שהומצאה על ידי אנשים, לשפת מכונה (כאמור 0 ו-1) אשר מחשב יכול להבין ולהפעיל פקודות בהתאם

תהליך ההידור כולל 5 שלבים מרכזיים שאפרט בהרחבה בהמשך הספר, עת נסתפק בשםם לקסר, פארסר, עץ סינטקטיס אבסטרקט, ייצוג אמצעי, והפקת קוד הינם החלקים המרכיבים קומפיילרים מודרניים.

בפרויקט זה אבנה כל חלק בעצמי ואגדיר שפת תכנות חדשה שתיקרא ACE (ראשי תיבות רקורייביים של **Ace Compiling Environment**)

בפרויקט זה ארצתה להעמק את הידע והבנה שלי בתחום ה-level Low ולהבין מהו קורה מהחורי הקלעים של דבר שנדרה לנו כל כך פשוט ומובן מallowן אף בעצם חביבים בו אתגראים רבים שנגמ עליהם אפרט בהמשך

קוד - הינו רצף של פקודות בשפת תכנות מסוימת אשר לאחר תהליך ההידור הופך לרצף תווים המרכיבים 0 ו-1, מעבד המחשב יודע להפוך תווים אלה לפקודות פיזיות בפועל בעדרת חשמל כאשר 0 מסמל "אין חשמל" ו-1 מסמל "יש חשמל"

קצת על שפת ACE

שםה של שפת התכנות מגיע מראשית התיבות

A - Ace

C- Compiling

E- Environment

שם ראשית תיבות רקורייביים, ההשראה לשם נשאבת מ-**GNCGN**

השפה דומה בסינטקטיס לשפת C סטנדרטית

ארגוני השפה

אלו הם ארגוני היסוד הבסיסיים ביותר של השפה אשר לא ניתן לפרק אותם לביטויים פשוטים יותר והם מרכיבים את כל הביטויים המורכבים בשפה

קבועים - literals

קבוע הוא ערך המופיע ישירות בקוד התוכנית
קבוע יכול להיות מטיפוסים שונים – מספר שלם, true.
טיפוסו של הקבוע נקבע על ידי המהדר בהתאם לערכו.

81 = x : דוגמא

בדוגמה שלעיל 81 הוא קבוע, אשר יובן על ידי המהדר כקבוע מטיפוס מספר שלם.

טיפוסי קבועים

קבוע מטיפוס מספר שלם - int.

על מנת להגדיר קבוע מסווג true נכתב את ערכו ישירות בקוד התוכנית:

ס דוגמא:

a / 2

בדוגמה זו 2 הוא קבוע ו - a הוא שם המשתנה קלשו.

קבוע מטיפוס true - char.

הגדרת קבוע מסווג char תהיה בתוך שני גרשים בודדים:

ס דוגמאות להגדרה:

'b' := ch

הערך של התו a יכנס אל תוך המשתנה ששמו ch.

קבוע מטיפוס בוליאני - bool

הגדרת קבוע מסווג בוליאני תאפשר לפיו ערך מילולי false \neq true בצורה הבאה:

Bool flag := true

ערך בוליאני מייצג אפשרות אחת, דלוק או כבוי וחוסך בזיכרון

משתנים – Identifier

משתנה מייצג מקום בזיכרון בו אפשר לשמור ערכים.

מקום זה בזיכרון מיוצג על ידי שם המשתנה, שנគרא גם מזהה

שמות משתנים

1. שם המשתנה הוא רצף של אותיות בשפה האנגלית, ספרות, והטע '_'.
רצף זה חייב להתחיל באות בשפה האנגלית או בתו '_'.
2. אין להשתמש במילים שמורות כמו זהים.
3. קיימת הבחנה בין אותיות גדולות וקטנות.
4. שם המשתנה הוא ייחודי ואין להשתמש באותו שם יותר מפעם אחת

טיפוסי משתנים

לכל משתנה בשפה יש גם טיפוס אשר מציין את סוג הערכים שהוא יכול להכיל.

ישנם שלושה סוגי טיפוסי משתנים:

- int – משתנה מטיפוס מספר שלם.

- מכיל ערכים מסוג מספרים שלמים. 1, -15, 79, 0 וכו'.

- char – משתנה מטיפוס תוו.

- מכיל ערכים מסוג תוו. a, g, 0, 7, f וכו'.

- Bool - משתנה מסוג בוליאני

- מכיל ערכים מסוג true או false

הנדרת משתנים

הנדרת כללית של משתנה:

`<Identifier> <Data-type>;`

דוגמאות:

int x

Bool f

char c

Expressions & Statements

Expression

יחידה תחבירית בשפת תכנות שניית להערכה על מנת לקבוע את ערכה. שילוב של אחד או יותר קבועים, משתנים, פונקציות, אופרטורים, ו – Expression נוספים, שהשפה מפרשת (לפי הכללים של קדימות ושירות), ומחשבת כדי לייצר ערך. תהליך זה, עברו ביטויים מתמטיים, נקרא הערכה בפשטות, הערך המתתקבל הוא בדרך כלל אחד מהסוגים הפרימיטיביים השונים, כמו ערך מספרי, ערך בוליאני וכו'.

דוגמאות ל – Expressions :

• $3 + 15$

• 4

• $(x - 5) / y$

• $7 \leq 22$

• $X \% 2 == 0$

• $(x + 15) < 3 * (y - 4)$

Statement

יחידה תחבירית בשפת תכנות המבטאת פעולה כלשהי שיש לבצע. תכנית הנקתבת בשפה כזו נוצרת על ידי רצף של אחד או יותר Statements. בשונה מ – Expression, Statement – לא מוערכת לכדי ערך. ל – Statement יכולים להיות רכיבים פנימיים

דוגמאות ל – Statements :

- **תנאים – if, else**
- **וליאות – while**
- **czhera על משתנה – x int;**
- **השמת ערך למשתנה – x set = 4;**

אופרטורים – Operators

חשבוניים

- חיבור - +
- חיסור - -
- כפלה - *
- חילוק - /
- שארית - %

לוגיים

- שווה ל - ==
- לא שווה ל - !=
- נדול מ - <
- קטן מ - >
- נדול או שווה ל - <=
- קטן או שווה ל - >=
- not - !

קשרים לוגיים

- or - ||
- and - &&

תכלת השפה

בפרק זה עוסק בתכלת השפה (כלומר היכולות שהיא מסוגלת לבצע) ובכלל הדקדוק והכתיבה הנכונה בשפה.

תכלת השפה מוגדרת היטב גם לפי תרשימים הקיימים שאותו ניתן למצוא בנספחים של הספר בנוסף אוצר קטעים רלוונטיים במידת הצורך

השמה

סימול של השמה מבוצע באמצעות אחת מהמיילים השמורות לטיפוסים וסימן השווה עם נקודותיים =:

- הגדרה כללית להשמה:
`<type> <Identifier> = <Expression>;`

על מנת שהשמה תהיה חוקית, טיפוס המשתנה אליו עושים את ההשמה, של ה - `<Identifier>` צריך למתאם לטיפוס הערך המושם, .

- דוגמאות:

`Int x := 80`
`Char y := 'v'`
`Bool flag := true`

תנאים ולולאות

תנאים ולולאות הם חלקו קוד המתבצעים כתלות באם ביטוי מסוים הוא אמת או שקר.

תנאים – Conditions

لتנאי יכולם להיות שני חלקים:

- if
- else

אם הביטוי נותן תוצאה אמת, הקוד שבחלק של ה – if יבוצע.

ואם נותן תוצאה שקר, הקוד שבחלק של ה – if לא יבוצע.

אם יש חלק של else, הוא יבוצע כאשר הביטוי נותן תוצאה שקרית.

חלקים אלו של ה – if וה – else יבוצעו 0 או 1 פעמים.

בכל מקרה, לאחר ביצוע התנאי התוכנית תמשיך לקוד שנמצא אחריו.

- דוגמא להגדרת תנאי בעדרת שימוש ב – if בלבד:

```

if (<Expression>
    )
Do if <Expression> is True
    {

```

דוגמה להנדרת תנאי בעזרת שימוש ב - if | - else:

```
if (<Expression>):
{
    Do if <Expression> is True
}
Else:
    Do if <Expression> is false
```

לולאות – Loops

לולאה דומה מאוד מבנה שלה לתנאי, if, אך ההבדל היחיד הוא שחלק הקוד שבתוך הלולאה מתבצע כל עוד הביטוי נותן תוצאהאמת, ולאו דווקא 0 או 1 פעמים. כלומר לולאה יכולה להתבצע מספר רב של פעמים.

- while

דוגמה כללית להנדרת לולאה בעזרת שימוש ב - while:

```
while (<Expression>):
    Do while <Expression> is True
```

הערות – Comments

הערות הן קטעי קוד אשר הקומפיילר מתעלם מהם .

לרוב מתכוונים משתנים בהערות על מנת להסביר למה כתבו שורת קוד מסוימת, ומה עשו זאת דווקא בצורה הספציפית הזאת ועל מנת להשאיר תזכורות לעצם ולמתכוונים אחרים שיקראו את הקוד שלהם בעתיד. כל הקוד מתחילה הערה ייחסב הערה, עד אשר נרד שורה. הערות בשפה יתחלו עם הסימן ~

- דוגמא להערה:

~ This is a comment

פונקציות – Functions

פונקציות הן קטעי קוד קומפקטיים אשר נועד לבצע קטעי קוד מסוימים וניתן לקרוא להן ממוקומות שונות בקוד, פונקציה יכולה לקבל משתנים ולהחזיר משתנים ויכולת שלא, על המשתנים שהוא מוחזירה יש לתאים את סוג המשתנה בהצהרת הפונקציה

דוגמה:

```
Give type identifier ( input_var_list )
{
    statement_list
    return_statement
}
```

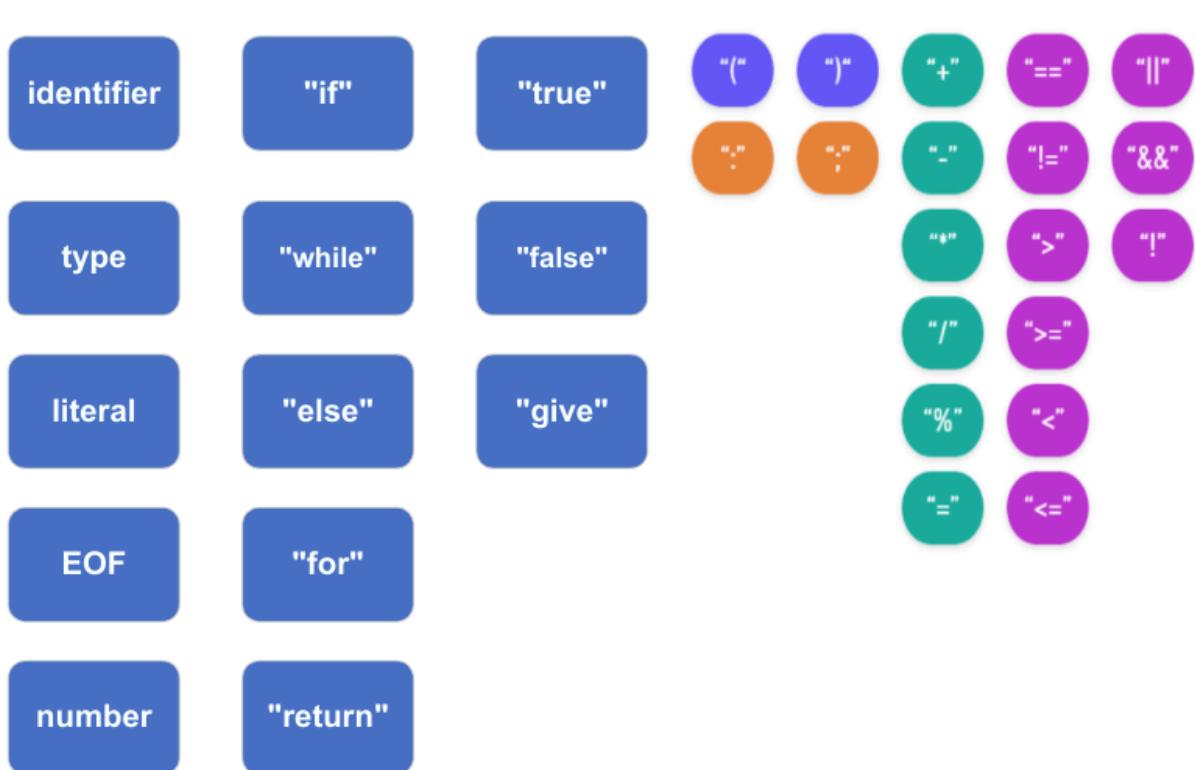
דקדוק השפה

לאחר שהגדרתי את אבני השפה ותכולת השפה,icut אנדר את תחביר / דקדוק השפה. ה – Grammar של השפה.

תחביר השפה

הדקוק מורכב מ – Terminals ו – Non-Terminals. הסימנים Terminals (הם המילים Tokens) שנקלטו כקלט מקטע הקוד, בעודם המשתנים Non-Terminals (הם רצפי סימנים ומשתנים. תחביר השפה מוגדר באמצעות שילוב הסימנים והמשתנים, בכללם שנדרים כללי יצירה . כללי היוצרה בעצם מגדירים את המשתנים, באמצעות הסימנים המוגדרים בשפה ומשתנים אחרים .

רשימת האסימונים בשפה:



BNF

Form Naur-Backus היא צורת כתיבה פורמלית עבור תיאור Grammars של שפות נטולות הקשר. צורת כתיבה זו משמשת לעיתים קרובות לתיאור שפות תכנות עוזר לכתוב בצורה חד-חד משמעותית את כללי ה- Grammar – BNF של שפה מסוימת, באופן יחסית קל וקריא.

תרשים bnf של שפת ACE:

```

1 program ::= statement_list
2
3 statement_list ::= statement | statement_list statement
4
5 statement ::= if_statement | while_statement | assignment | deceleration |
6 | for_statement | expression | return_statement | function
7
8 for_statement ::= for ( assignment ; condition ; expression ) { statement_list }
9
10 if_statement ::= if ( condition ) { statement_list } | if ( condition ) { statement_list } else { statement_list }
11
12 while_statement ::= while ( condition ) { statement_list }
13
14 deceleration ::= type identifier := expression | type identifier
15
16 assignment ::= identifier := expression
17
18 type ::= int | bool | char
19
20 void_type ::= void
21
22 expression ::= factor | expression arop factor | expression bitop factor | identifier unary_expression
23
24 factor ::= number | identifier | literal | ( expression )
25
26 condition ::= expression relational_operator expression
27
28 relational_operator ::= == | != | < | > | <= | >=
29
30 arop ::= - | + | * | /
31
32 unary_expression ::= ++ | --
33
34 bitop ::= & | >> | << | |
35
36 return_statement ::= return factor
37
38 function ::= give type identifier ( input_var_list ) { statement_list return_statement } | |
39 | void_type identifier ( input_var_list ) { statement_list }
40
41 input_var_list ::= input_var | input_var , input_var_list | ''
42
43 input_var ::= type identifier

```

דוגמא לקוד בשפת ACE

```

1 give int main() [
2
3     int firstNumber := 7
4     int seconed_number
5
6     if (sum == 2)
7     {
8         bool flag := 1
9         int avg := (firstNumber + seconed_number) / 3
10    }
11
12    while (flag == 1)
13    {
14        avg := avg + 1
15        if(avg > 10)
16        {
17            flag := 0
18        }
19    }
20
21    return avg
22
23 ]

```

רקע תיאורטי

מהו קומפיילר

תוכנת מחשב אשר מתרגם קוד מקור הכתוב בשפת תכנות אחת לקוד הכתוב בשפת תכנות אחרת, ללא שינוי המשמעות של קוד המקור. לרוב מתרגם משפה גבוהה (C++, Java, C), לשפת מכונה. הקומפיילר גם מייעל ומשפר את קוד המקור כמו שניתן. כמו כן, מתריע על השגיאות / אזהרות שמצא, ומציע הצעות לפתרונות שלדעתו יפתרו שגיאות / אזהרות אלו.

Compiler vs. Interpreter

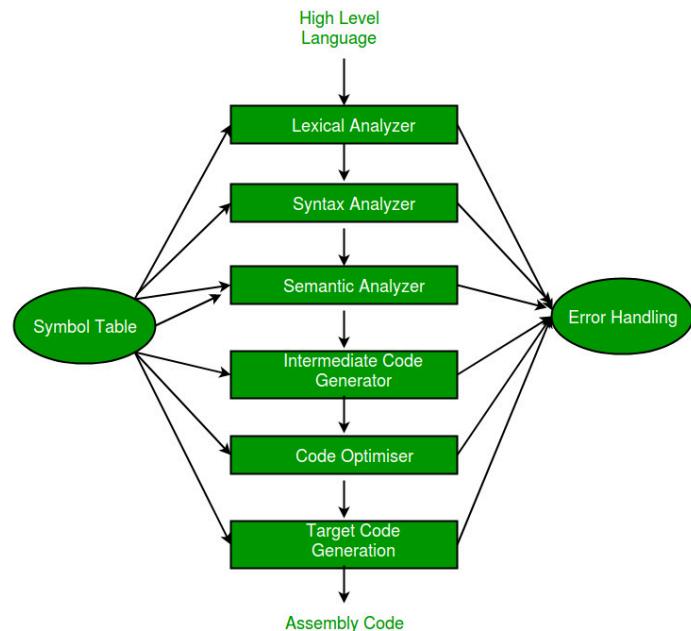
ישנם שני אופני עבודה עיקריים של קומפיילר: תרגום כל קוד המקור לכדי יחידת הריצה אחת או תרגום כל פקודה בנפרד בקוד המקור תוך כדי ריצת התוכנית. כפי שצין לעיל, Compiler, עובר על כל קוד המקור לפני הריצה, בודק את תקיןותו, ומתרגם וממיר אותו ליחידת הריצה אחת בשפת מכונה. שפות שמתורגות על ידי Compiler נקראות שפות מקומפלות . דוגמאות לשפות מקומפלות הן Java, C, C++, Compiler ל – JavaScript, Python ועוד.Interpreter מתרגם וממיר כל פקודה בנפרד בקוד המקור לפקודות בשפת מכונה , תוך כדי ריצת התוכנית, ללא בדיקת תקינות הקוד לפני הריצה. שפות שמתורגות על Interpreter נקראות שפות סקריפטים. דוגמאות לשפות אלו הן JavaScript, Python ועוד.

למה נדרש קומפיילר

מכיוון שלבני האדם קל יותר לכתוב קוד בשפות תכנות עליות אשר יותר קרובות אליהם (יותר קרובות לאנגלית), מאשר לכתחוך קוד בשפת מכונה, אנו משתמשים בשפות עליות אלו לכנתיבת קוד. אך המחשב אינו מבין שפות עליות אלו, הוא מבין רק קוד בשפת מכונה. בשביל לנגרר על הפער בין בני האדם לבין המכונה, צריך קומפיילר. שיתרגם את מה שאנו מתכוונים כאשר אנו כותבים קוד בשפה עליית לשפה שהמחשב יבין, שפת מכונה.

כיצד עובד קומpileר

תהליך הקומPILEציה הוא תהליך מורכב, ולכן מומלץ לחלק אותו לשלבים. הקומPILEר עובד כך שכל שלב מקבל כקלט את התוצאה של השלב הקודם. הוראות של השלבים:



הסבר קצר על שלבי תהליך הקומPILEציה

• לקסר

- הקלט הוא הקוד אשר אנו רוצים לкопל לשפת מכונה, בדר"כ, יקרא תחילת קקובץ ואז נקרא כל מילה בנפרד (התעלמות מרוחחים).
- הפלט הוא **Token**, מספר או תו אשר מזוהה מה סוג של המילה שנקראה(שם/מזהה, מילת מפתח, ערך, פועלה בינארית וכו').
- הוא קורא את המילה/מספר הנוכחי/ת, מזוהה איזה סוג הם ומחזיר את ה-**token** התואם תור שמיירה של המילה/תו לשימוש מאוחר יותר ב-**parser**.
- ע"י קריית התו הראשוני של המילה/מספר. לאחר מכן הוא בודק לאיזה קритריון בסיסי היא תואמת (מזהאים, ערכים, פועלות בינאריות וכו'). לאחר מכן הוא קורא את הערך בהתאם ומזהה איזה **token** שייר לו(למשל למזהאים, האם זה מזהה כללי, או מילת מפתח ספציפית וכו').

• פרסר

- הקלט הוא קוד מחולק לTOKENS.
- הפלט הוא הבסיס לעצבי אבסטרקטוי. (יוסבר בהמשך)

- מחלק TOKENS לקבוצות ומשפחות על מנת להפוך אותם לבסיס עבור השלב הבא שהוא עץ ביןארי אבסטרקטו.
- הפרסר מזהה מילוט מפתח אותן הוא ממין לקטגוריות שונות שנקבעות על פי סוג מילת המפתח והקשרה.

• עץ ביןארי אבסטרקטו

- הקלט הוא קוד ששבוער תהליך פרסור על ידי הפרסר
- הפלט הוא קוד שמחולק לתתי משפחות ומוכן לעבור לשלב התרגום לקוד המיצג על ידי שפת אמצע
- התפקיד של עץ ביןארי אבסטרקטו הוא לחלק את הקוד המפורسر לתתי משפחות ומחלקות על פי הנדרות שניתנו מראש
- העץ הבינארי מזהה משפחות שונות של מילוט מפתח ומחלק אותן לקבוצות קטנות

• intermediate representation

- הקלט הוא קוד ממויין ומסודר לקטגוריות בעץ ביןארי מופשט.
- הפלט הוא קוד בשפה אמצעית דומה לאסמבלי אשר מייעל ומוכן להפוך לקוד בשפת מכונה.
- מייעל ומסדר את הקוד וממיר את שפת הקוד לשפה אמצעית אשר נראה דומה לאסמבלי בדרך כלל וכל יותר להמיר אותה לשפת מכונה ולהריץ.
- לקח את הקוד הממוין ומסדר אותו בדרך היעילה ביותר בעזרת מספר אלגוריטמים של ספריית LLVM, למשל: יצירה מבנה קבוע של ייצוג הביניים, דבר המאפשר לעبور על פונקציות מספר פעמים בשבייל לוודא את הרצאה הcy'יעילה של הפונקציה בקומפיילר, או זיהוי והתאמה בין קבצי קוד ויבוא רק של קטעי הקוד בשימוש, דבר המוביל לייצירה של קבצי הרצאה קטנים יותר ויעילים יותר (optimization Time-Link).
- שחוסכת הcy' הרצה זיכרון וננתנת זמן הרצאה מינימלי על ידי מנגנון מכונה המוני.

• קומפול לשפת מכונה

- יצוג הביניים של הקוד שקרוב יותר לאסמבלי אשר מייעל ומוכן להפוך לשפת מכונה
- קוד הרצאה אשר מכיל את הקוד המומר לשפת מכונה יוכל לרוץ על המחשב.
- קורא את ייצוג הביניים וממיר אותו לשפת מכונה ויוצר ממנו object file/executable file.
- הוא ממיר את ייצוג הביניים הקרוב לאסמבלי לפקודות אסמבלי אשר מייצגות ביצוג הביניים ומשם הקוד מומר ישירות לשפת מכונה ומתווסף לקובץ ההרצה הסופי.

מודולים נוספים

Register allocation

לתוכנית יש מספר ערכיים שהוא צריך לשמר במהלך הריצה שלה. יתרון שארכיטקטורת מכונת היעד לא מאפשר כל הערכיים להישמר בזיכרון המעבד, או ה – registers. השלב של ה – generator code depended machine שומרו ערכיים אלו. ואילו registers שומרו ערכיים אלו.

Assembly, linking and loading

אסמבלי מתרגם שפת אסמבלי לשפת מכונה. הוא יוצר מקובץ asm שמכיל שפת אסמבלי קובץ object. קובץ object מכיל הוראות בשפת מכונה, כמו גם המידע הדרוש על איפה צריך לשים את ההוראות האלה בזיכרון.

לינקר לחבר כמה קבצי object לקובץ exe אחד.

כל הקבצים שהוא לחבר יכולים להיות מוקומפלים על ידי אסמבליים שונים. משימתו העיקרי של הלינקר היא לקבוע את המיקום בזיכרון של כל אחד מהקבצים בעת הטעינה שלהם לזכרון (על ידי ה

– Loader), כך שההוראות מקבציו – jpg השונים יתבצעו בסדר הגיוני בעת הריצה. ה – loader הוא חלק ממכלול הפעלה שאחראי על הטעינה של קבצי הריצה (exe לזכרון, והביצוע שלהם).

הוא מתחשב את גודל התוכנית ומנסה给她 מקום בזיכרון. הוא גם מאתחל מספר רגיסטרים שונים שיתחילו את תהליך הביצוע/הריצה של התוכנית.

Symbol table

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

טבלה זו עוזרת לקומpileר למצאו רשומה של מזהה כלשהו בתוכנית ולקבל את הפרטים עליו באופן מהיר וחסית.

ה – table Symbol עוזרת גם ב – managment Scope .

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

Error handler

כפי שכתבתי בתיאור השלבים של הקומpileר, בכל שלב ושלב בתהליכי הקומpileציה יכולות להיווצר שונות.

בשביל כך יש את ה – handler Error .

השגיאות שמתגלות מדווחות ל – handler Error והוא מדווח, ומצביעו על חזרה למתקנת במקרה של הودעה. אם

קומpileר יש הצעה מסוימת לפתרון הבעיה, גם היא תוצג בהודעה.

מכונות מצבים

הנדסה

מכונת מצבים (FSM - Finite State Machine) היא מודל חישובי פשוט אך רב עצמה המשמש לייצוג מערכות דינמיות בעלות מספר סופי של מצבים אפשריים. ניתן להשתמש בהן כדי לדמות התנהלות של מגוון רחב של מערכות, החל ממנגלים אלקטרוניים פשוטים ועד לתהליכיים עסקיים מורכבים.

רכיבים עיקריים

מכונת מצבים מורכבת משלושה רכיבים עיקריים:

מצבים: מערכת סופית של מצבים אפשריים עבור המערכת.

מעברים: קבוצות של כללים הקובעים כיצד המערכת עוברת ממצב אחד למצב אחר. כל מעבר מותנה בתנאי קלט ספציפי.

פעולות: קבוצות של פעולות שהמערכת יכולה לבצע בכל מצב.

פעולה

מכונת מצבים פועלת באופן הבא:

המערכת מתחילה במצב התחלתי מוגדר.

קלט נותן למערכת. בהתאם על הקלט הנוכחי וה המצב הנוכחי, מופעל מעבר מתאים. המעבר גורם למערכת לעבור למצב חדש. יתרון שביצוע המעבר יגרום גם לביצוע פעולה אחת או יותר. שלבים 5-2 חוזרים על עצמם עד שהמערכת מנעה למצב סופי או עד שהיא מקבלת פקודה לעצורה.

סוגים של מכונות מצבים

קיימים שני סוגי רכיבים של מכונות מצבים:

מכונות מצבים דטרמיניסטיות (DFA):

בכל מצב נתון, קיים מעבר אחד לכל קלט אפשרי.

מכונות מצבים לא דטרמיניסטיות (NFA):

במצב נתון, ניתן מספר מעברים אפשריים עבור אותו קלט, או שאף לא מעבר כלל

יישומים

מכונות מצבים משמשות בטווח רחב של יישומים, ביניהם:

עיצוב מנגלים אלקטרוניים: ניתן להשתמש במכונות מצבים כדי לתאר את התנהנותם של מנגלים לוגיים וdigitzליים.

פיתוח תוכנה: ניתן להשתמש במכונות מצבים כדי לתוכנת מערכות תוכנה פשוטות, כגון מפרשים ותוכנות בקרה.

שימוש שפה טבעית: ניתן להשתמש במכונות מצבים כדי לנתח טקסט ולזהות דפוסים לשוניים.

תכנון משחקים: ניתן להשתמש במכונות מצבים כדי ליצור משחקים פשוטים, כגון משחקי מחשב ופאזלים.

יתרונות

למכונות מצבים יש מספר יתרונות, ביניהם:
פשטות: הן קלות להבנה ולתכנות.

יעילות: הן יכולות לפעול בצורה יעילה גם במערכות מורכבות.
רב-תכליתיות: ניתן להשתמש בהן במגוון רחב של שימושים.
כח ביטוי: הן מסוגלות לייצג התנהלות מורכבת של מערכות.

חסרונות

למכונות מצבים יש גם מספר חסרונות, ביניהם:
גודל: ניתן שהן יהיו גדולות ומסובלות לייצוג מערכות מורכבות מאוד.
קושי בניהול: קשה לנוול ולהזדקק מכונות מצבים גדולות.
קושי בתכנון: תכנון נכון של מכונת מצבים מורכבת יכול להיות מ затגר.

שפה פורמלית

הנדסה: שפה פורמלית היא מערכת פורמלית המוגדרת על ידי מערכת חוקים וכללים תחביריים שתפקידו לייצג מושגים מתמטיים, לוגיים, או מידע טכני בצורה מדוקقة וברורה.

מאפיינים עיקריים

סינטקטיסטי: מערכת חוקים וכללים תחביריים המגדירים את המבנה התקין של משפטיים וחוקי יצירת ביטויים.

סמנטיקתית: מערכת משמעויות המוחשבת לסימבולים ולביטויים בשפה.
כח ביטוי: יכולת השפה לייצג מושגים מורכבים ומנוגנים.

יתרונות:

דיוק: מבטיחים העברה מדוקقة וחד משמעית של מידע.

בהירות: מקלים על הבנה וניתוח של מידע מורכב.

פורמליות: מאפשרים ניתוח מתמטי ולוגי קפפני.

יעילות: חוסכים זמן ומאפשרות בתקשורת ובהעברת מידע.

הנדרת מטרות ומשימות

מטרת הפרויקט:

בנייה מהדר עبور שפת תכנות דומות C, שתתאים לתוכנים מתחילהם.
למידה عمיקה על תהליך הידור.
יצירת שפה חדשה מאפס.
פיתוח יכולות למידה אישיות.

אפיון שפת התכנות:

תחביר דומה ל-C, תוך פשוטות ונוחות שימוש.
פירוט שגיאות מעמיק.
מיועדת ללימוד עקרונות התכנות הבסיסיים.
מהוות גשר והכנה לקרה שפת C.

מגבליות הפרויקט:

בנייה כל חלק מהדר בצורה ייעלה יחסית.
שימוש בספריות מבני נתונים ב-C++ בלבד, ללא טכנולוגיות חיצונית (flex & bison). (flex & bison)

אתגרים בפרויקט:

מורכבות אלגוריתמית גבוהה.
ניהול מורכבות קוד.
בדיקות מקיפות.
תחזקה לאורך זמן.

הנדרת הבעיה האלגוריתמית

בנית מהדר היא משימה מורכבת מבחינה אלגוריתמית. היא דורשת טיפול במספר רב של בעיות, כגון:

ניתוח תחבירי: ניתוח קוד המקור של התוכנית ויזהו המבנה שלו, תוך יצירת עץ תחביר.

ניתוח סמנטי: הבנת משמעות קוד המקור, תוך זיהוי משתנים, פונקציות, טיפוסים וביטויים.

יצירת קוד BINIIM: הפיכת קוד המקור לייצוג ביניים יעיל, כגון קוד P-code או MVL.

מיוטוב: שיפור יעילות הקוד הביניים, תוך שימוש בטכניות כמו הסרת קוד מיותר, ניתוח זרימה נתוניים ועוד.

יצירת קוד מכונה: הפיכת הקוד הביניים לקוד מכונה ספציפי למעבד.

כל אחת מהבעיות הללו היא בעיה אלגוריתמית קשה בפני עצמה. בנוסף, יש צורך לתאם בין כל השלבים הללו כדי ליצור מהדר ייעיל וקורקט.

אתגרים ספציפיים

בנית מהדר כפרויקט טומנת בחובה מספר אתגרים ספציפיים:

ניהול מרכיבות: פרויקט מהדר מורכב בדרך כלל ממספר רב של קבצים ומודולים. ניהול המרכיבות הזה הוא אתגר ממשמעותי.

בדיקות: בדיקת מהדר היא משימה קשה, כיוון שיש צורך לוודא שהוא עובד בצורה נכונה עבור מנון רחוב של קוד מקור.

תחזקה: מהדרים מתעדכנים באופן קבוע כדי לתרום בשפות תכונות חדשות, תכונות חדשות וטכנולוגיות חדשות. תחזקה של מהדר לאורך זמן היא משימה אתגרית.

פתרונות

קיימות מספר גישות לפתורן הבעיה האלגוריתמית בבניית מהדר:

שימוש בטכניות קיימות: ניתן להשתמש בטכניות קיימות בתחום מדעי המחשב, כגון אלגוריתמי ניתוח תחבירי, אלגוריתמי ניתוח סמנטי וטכנולוגיות אופטימיזציה.

שימוש בכלים תוכנה: קיימים מספר כלים שיכולים לשיעם בבניית מהדר, כגון גנרטורים של עצי תחביר וنمפרשים (למשל `bison & flex`).

ניתוח הבעיות בכל שלב

בנית קומפיילר היא תהליך מורכב הכולל מספר שלבים עיקריים, כאשר בכל שלב ישן בעיות אלגוריתמיות ספציפיות שיש להתמודד איתן:

ניתוח לקסיקלי (Lexical Analysis)

- זיהוי וקידוד של יחידות לקסיקליות (מילים שמורות, מזהים, מספרים וכו') מתוך הקלט הtekstual של התוכנית.
- טיפול בתווים מיוחדים ובסימני פיסוק.

ניתוח תחבירי (Parsing)

- בניית עצ חלוקה תחבירי (Parse Tree) המיצג את המבנה התחבירי של התוכנית.
- זיהוי ופתרון דו-משמעותות תחביריות.
- טיפול בשגיאות תחביריות.

ניתוח סמנטי (Semantic Analysis)

- בדיקת סוג הנתונים של ביטויים ומשתנים והתאמתם לפעולות.
- איתור שגיאות סמנטיות כגון שימוש לא חוקי במשתנים.
- ניהול טבלת סמלים לאיחזור מידע על משתנים, פונקציות ומבנה נתונים.

יצור קוד (Code Generation)

- תרגום התיאור המופשט של התוכנית לקוד מכונה ספציפי.
- בחירת רגיסטרים ומיקום זיכרון לאחסון משתנים.
- אופטימיזציה הקוד המוצר לביצועים טובים יותר.

סקירה של אלגוריתמים בתחום הבעה

תהליך הניתוח התחבירי, ה – Parsing, הוא התהליך המשמעותי והמורכב ביותר מבחן אלגוריתמית וריעונית בתהליך הקומפילציה. כתע אציג שיטות שונות ואלגוריתמים שונים הנפוצים בשלב זה.

מונחים

Derivation

בעברית, גזרה, היא בעצם רצף של rules Production, על מנת לקבל את מהירות הקלט. במתוך תהליכי Parsing אנו בעצם מקבלים שתי החלטות עבור קלט מסוים:
 1. החלטה על ה – Non-terminal אשר יוחלף.
 2. ההחלטה על כלל הייצור, שבאמצעותו יוחלף ה – terminal-Non-terminal.

על מנת להחליט על איזה Non-terminal יוחלף בכלל הייצור, יכולות להיות לנו שתי אפשרויות:

Left-most Derivation

אפשרות זו קובעת כי תמיד ה – terminal השמאלי ביותר הוא זה שיוחלף.

Right-most Derivation

אפשרות זו קובעת כי תמיד ה – terminal הימני ביותר הוא זה שיוחלף.

דוגמא

נתון ה – Grammar הבא:

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$

עבור מחרוזת הקלט " id + id * id " נ"כ יראו שני סוגי ה – Derivation

Left-most derivation

$E \rightarrow E * E$
$E \rightarrow E + E * E$
$E \rightarrow id + E * E$
$E \rightarrow id + id * E$
$E \rightarrow id + id * id$

Right-most derivation

$E \rightarrow E + E$
$E \rightarrow E + E * E$
$E \rightarrow E + E * id$
$E \rightarrow E + id * id$
$E \rightarrow id + id * id$

Left Factoring

אם יותר מ – rule Production אחד מתחילה באותה קידומת, אז ה – Parser לא יכול לבצע הרכעה באיזה מהחוקים הוא צריך לבחור בשבייל לנתח את הקלט הנוכחי.

דוגמא

אם כלל ייצור מסוים נראה כך:

$$A \Rightarrow \alpha\beta \mid \alpha\gamma \mid \dots$$

המנתח לא יודע להחליט איזה חוק לעקוב, כיוון שני החוקים מתחילה באותו Terminal או Non-terminal). על מנת להסיר בעיה זאת משתמשים בטכנית שנקראת Left factoring Left grammar את ה – Grammar כך שלא יהיו חוסר הוודאות האלו. היא עובדת כך שבעור כל קידומת שימושת יותר מפעם אחת יוצרים כלל חדש והמשך של הכלל הישן משורשר לכל החדש.

לדוגמה הכלל הקודם יוכל כתוב להירות כך:

$$\begin{aligned} A' &\Rightarrow \alpha A \\ A' &\Rightarrow \beta \mid \gamma \mid \dots \end{aligned}$$

עכשו למנתח יש רק כלל אחד עבור הקידומת המסויימת הזאת, מה שמקל עליו לקבל החלטות.

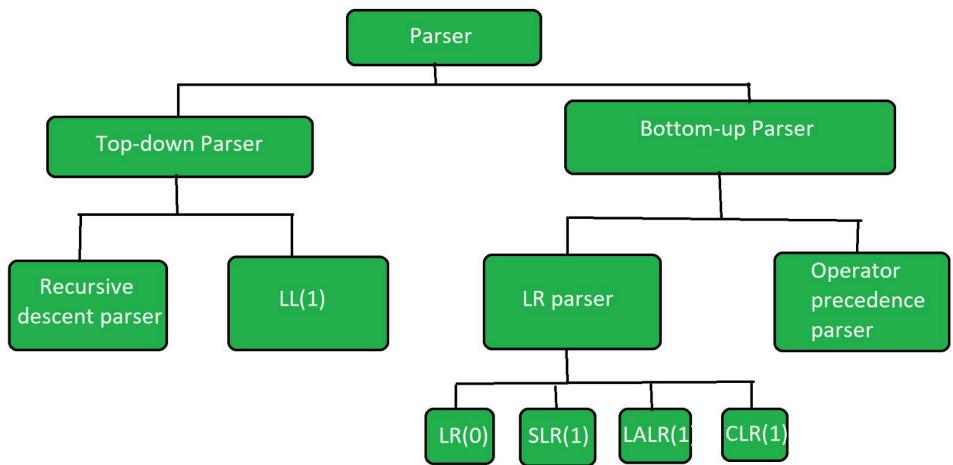
אלגוריתמי פרסר

על מנת ליצור Parse Tree עליו יתבסס תהליך הקומפילציה, ישנו כמה אלגוריתמים הנקראים Algorithms Parsing. אלגוריתמים אלה מתחלקים לשני סוגים עיקריים.

1. Top Down Parsing (TDP)

2. Bottom Up Parsing (BUP)

שפות תכנות הן בדרך כלל כל free-Context languages. נהוג לפרש CFL באמצעות מכונות מצבים, ובאופן יותר ספציפי מצבות מצבים המשמשות במחסנית (machines Pushdown). לכן האלגוריתמים שכעת אציג ישתמשו באופןוט מחסנית לרוב, על מנת לבצע את פעולה זו Parsing.



מכיוון שהחומר להשתמש באלגוריתם LALR אסביר רק במניעה על שאר האלגוריתמים ולא את עכוב עליהם יותר מכך.

Top Down Parsing (TDP)

טכנית בה עוברים מהחלקיםعلילונים החלקים התחתיים של העץ התħבורי, על ידי שימוש בכללי השכטוב של Grammar השפה. עוברים מה – Grammar.

LL Parser

אלגוריתם chow-down-top המתבסס על ניתוח מהשמאל לימין (Left-to-right) ובניה מלמעלה למטה (chow-down). מתאים לשפות תחביר (LL). קלומר שפות שניתן לנתח אותן על ידי הסתכלות קדימה של k טוקנים. האלגוריתם מתחילה מסמל השורש ומנתח את המבנה התħבורי באופן רקורסיבי תוך כדי קידום בטוקנים.

Recursive Decent Parsing

צורה נפוצה של TDP. בשיטה זו, כיוון שהיא שיטה שמתבססת על הגישה של Top-Down, עז הניתוח נוצר מלמעלה למטה, והקלט נקרא משמאליימין. שיטה זו משתמש בפונקציות עבור parser descent Recursive – Non-terminal | Terminal שנמצא ב – Grammar השפה. מה שיכל לגזור לו לSUBOL מ – tracking יוצר את עז הניתוח תוך מעבר רקורסיבי על הקולט, מה שיכל לגזור לו לSUBOL מ – tracking Back – בוגל ה – tracking Back יכול להיות אף אקספוננציאלית

Back tracking

כאשר ה – Parser משתמש בשיטה של parsing decent Recursive (factored Left factored Left), יוצרו מצבים במהלך הניתוח של הקולט בהם המנתה ייעז למבי סתום, וזהת כנראה בוגל שעשה בחירה לא נכונה של כלל מסוים בדרך. לכן, המנתה חוזרת חזרה למקום האחרון בו ביצוע הבחירה, על מנת לבצע הבחירה אחרת, נקראת tracking-Back. רק כאשר נסה את כל האפשרויות ולא הצליח להתקיים את הקולט לכלי השפה, ניתן להבין שהקלט הוא לא תקין. מבחינת השפה.

Bottom Up Parsing (BUP)

טכנית בה עוברים מחלקים התחכוניים לחלקיםعلילונים של העץ התחבירי, על ידי שימוש בכללי השכתב של Grammar השפה. עוברים מהקלט אל ה - Grammar.

Reduce Shift

הוא התהליך של הפחתת מחרוזת הקלט ל – terminal-non Starting step parsing up Bottom . שיטה זו משתמש בשני שלבים הייחודיים ל – Reduce | Shift .step

Shift step

שלב זה מבצע מעבר לסמל, symbol, הבא שמניע מהקלט, והוא נקרא גם **Shifted symbol**. סמל זה נדחף (PUSH) למחרסנית. ה – symbol Shifted מטופל כצומת אחת של עצ הניתוח.

Reduce step

כאשר המנתח מוצא כלל יצור שלם, כלומר הסמלים שעל המחרסנית תואמים לאחד מה – RHS terminal Hand Right side () של כלל יצור של ה – Grammar, ומחליף אותו ב – LHS שנמצא ב – .step Reduce שלב זה נקרא POP. שלב זה בעצם עושה למצא, וודחף למחרסנית את ה – RHS התואם. יעילות האלגוריתם היא $O(n)$

LR Parser

אלגוריתם **bottom-up** המtabסס על ניתוח משמאלי לימין ובניה מלמטה למעלה. מתאים לשפות תחביר k(LR) – שפות שניתן לנתח אותן על ידי הסתכלות אחורה של k טוקנים. האלגוריתם מתחילה מהטוקנים ובונה את עצ החלוקת התחבירי מלמטה למעלה תוך שימוש (SLR) (Simple LR), LALR (Look-Ahead LR) כמו LR ויעוד.

Operator Precedence Parser

אלגוריתם **bottom-up** המנתח ביטויים ארכיטמטיים על בסיס קדימות הפעולות. מתאים במיוחד לשפות תכונות עם ביטויים ארכיטמטיים. האלגוריתם סורק את הטוקנים ומחליט איך לבנות את עצ החלוקת על בסיס טבלת קדימות של האופרטורים.

LALR parser

פועל על ידי בניית עצ ניתוח מטור רצף של סמלים (tokens) המייצגים את הטקסט. עצ הניתוח מתאר את המבנה התחבירי של הטקסט, ומאפשר לבצע עליו פעולות נוספות, כמו בדיקת תקינות, תרגום, ביצוע, ועוד.

:LALR Parser היתרון של

- **יעילות:** LALR Parser הוא יעיל יחסית ביצועים, הוא מבחינת זמן והן מבחינת זיכרון.
- **קלות שימוש:** יחסית לסוגים אחרים של Parsers, LALR Parser קל יותר להבנה ולשימוש.

- כוח ביטוי: LALR Parser מסוגל לנתח תחביר של מגוון רחב של שפות תכנות ופורמטים טקסטואליים.

החסרון של LALR Parser:

- מוגבלות: לא ניתן להשתמש ב-LALR Parser עבור כל דקדוק (Left-to-Left) LL(0).
- מורכבות: השימוש LALR Parser יכול להיות מורכב יותר מאשר סוגים אחרים של Parsers.
- דוגמה לנתח תחביר באמצעות LALR Parser:

נניח שיש לנו את דקדוק LL הבא, המגדיר ביטוי ארכיטמטי פשוט:

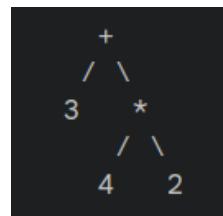
```

expr → term | expr + term | expr - term
term → factor | term * factor | term / factor
factor → num | ( expr )
num → [0-9]*

```

הנה דוגמה לרצף של סמלים המייצנים ביטוי ארכיטמטי:

LALR Parser יבנה את עץ הנתחם הבא מtower רצף הסמלים זהה:



עץ הנתחם הזה מתאר את המבנה התחבירי של הביטוי הארכיטמטי "3 + 4 * 2". ניתן להשתמש בעץ הנתחם הזה כדי לבצע פעולות נוספות על הביטוי, כמו חישוב ערכו.

אופן פעולה של LALR Parser:

LALR Parser פועל על ידי שימוש במטריצת פעולות (Action Table) וטבלת מעברים (Goto Table). מטריצת הפעולות מנדרה איזו פעולה לבצע עבור כל זוג של מצב וטוקן. טבלת המעברים מנדרה את מצב היעד עבור כל זוג של מצב וסימן טרמינלי.

במהלך נתחם תחביר, LALR Parser מתחילה במצב ההתחלה ומעבד את רצף הסמלים אחד לאחר. עבור כל טוקן, Parser משתמש במטריצת הפעולות כדי לקבוע איזו פעולה לבצע. הפעולות האפשריות הן:

Shift: LALR Parser מזיז את ה-pointer על פני רצף הסמלים ומעבר למצב חדש.

Reduce: LALR Parser מפחית את עץ הנתחם על ידי שימוש כלל הפחתה מתאים.

Accept: LALR Parser מקבל את הטקסט.

Error: LALR Parser מזהה שגיאה בתחביר.

LALR Parser משתמש בטבלת המעברים כדי לקבוע את מצב היעד לאחר ביצוע פעולה.

דוגמא לutableות הניתוח של LALR parser:

	ACTION			GOTO	
	a	b	\$	A	S
0	S3	S4		2	1
1			accept		
2	S6	S7		5	
3	S3	S4		8	
4	R3	R3			
5			R1		
6	S6	S7		9	
7			R3		
8	R2	R2			
9			R2		

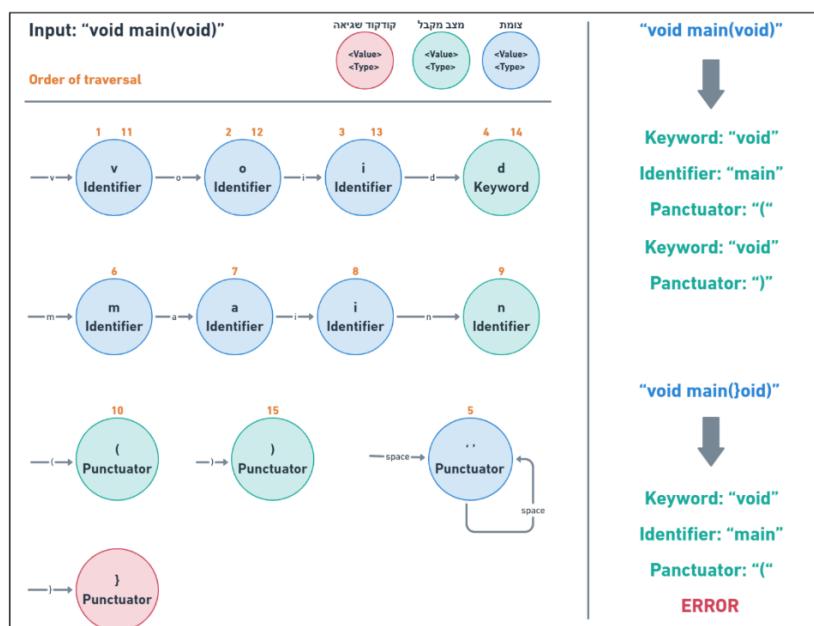
ניתוח שלבי אלגוריתמיקה

בפרק זה אציג את האסטרטגיה והגישה לפתרון הבעיה האלגוריתמית הנכראת Compiler. איזג את ה – Compiler במציאות מכונת מצבים. על מנת לעשות זאת ועל מנת לתרגם את קוד המקור, תוק שאייפה ליעילות זמן ריצה לינארית, משתמש במבנה הנתונים גראף שיציג את מכונת המצבים של ה – Compiler. כת עת אציג כיצד אסטרטגיה זו באה לידי ביטוי בשלבים השונים של תהליך הקומpileציה.

ניתוח מילוני - Lexical Analysis

שוב, מטרתנו לשאוף ליעילות זמן ריצה לינארית. עקב כך אנו לא רוצים להתייחס לקוד המקור אותו אנחנו מתרגם כטיקסט. לכן אנו מתרגם את קוד המקור לאסימונים בעלי משמעות בקוד, כפי שמתואר בחלק של הסקירה התיאורטית. המסלול שיוווצר יתבסס על התווים מקוד המקור. המסלול יסתהים כאשר נגיע לקוד קוד שהוא על, אשר יחויר סוג אסימון (Identifier, Keyword, Operator, Literal) ששמור אצלו. במקרה וקוד המקור יחל רצף תווים לא חוקי, המסלול יוביל לקוד קוד שנייה.

דוגמא:

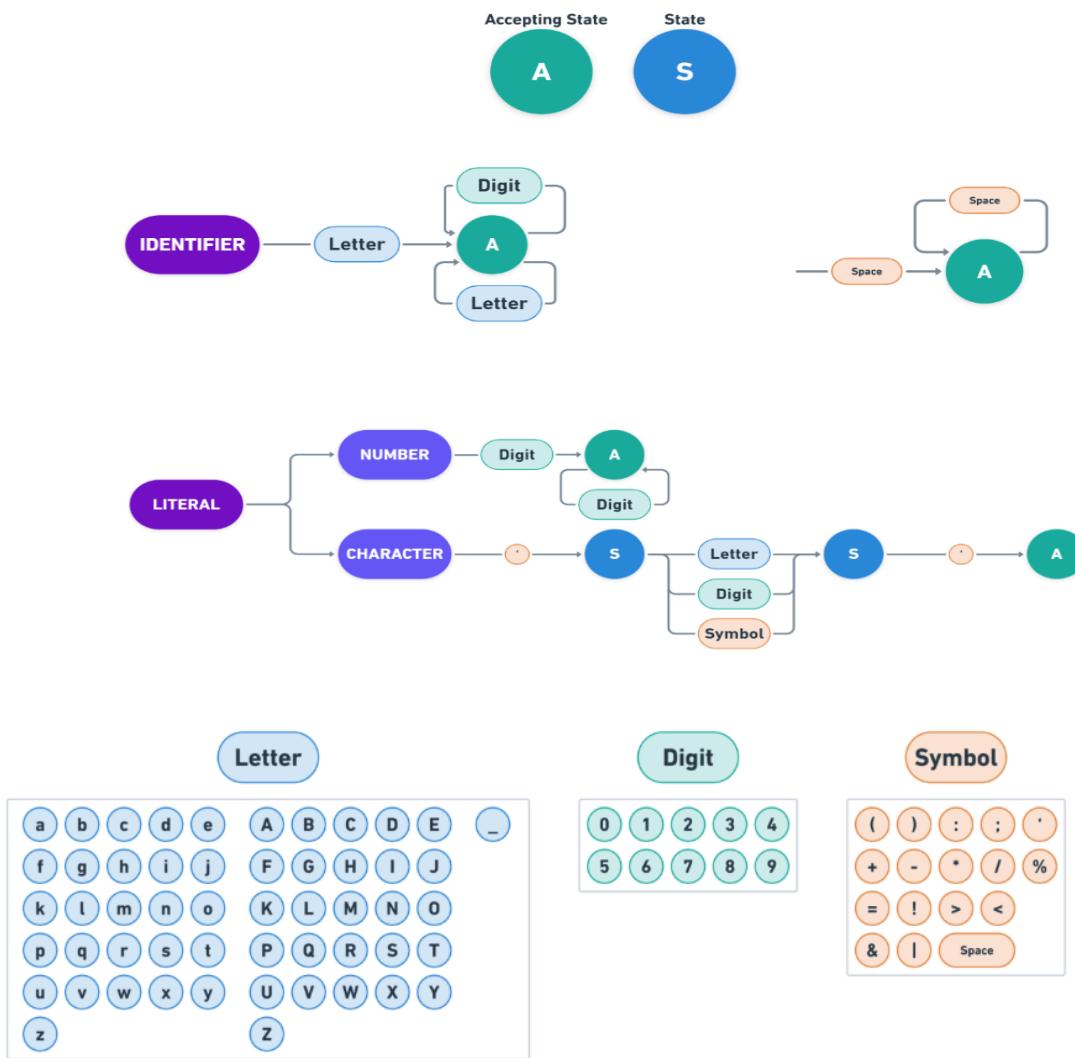


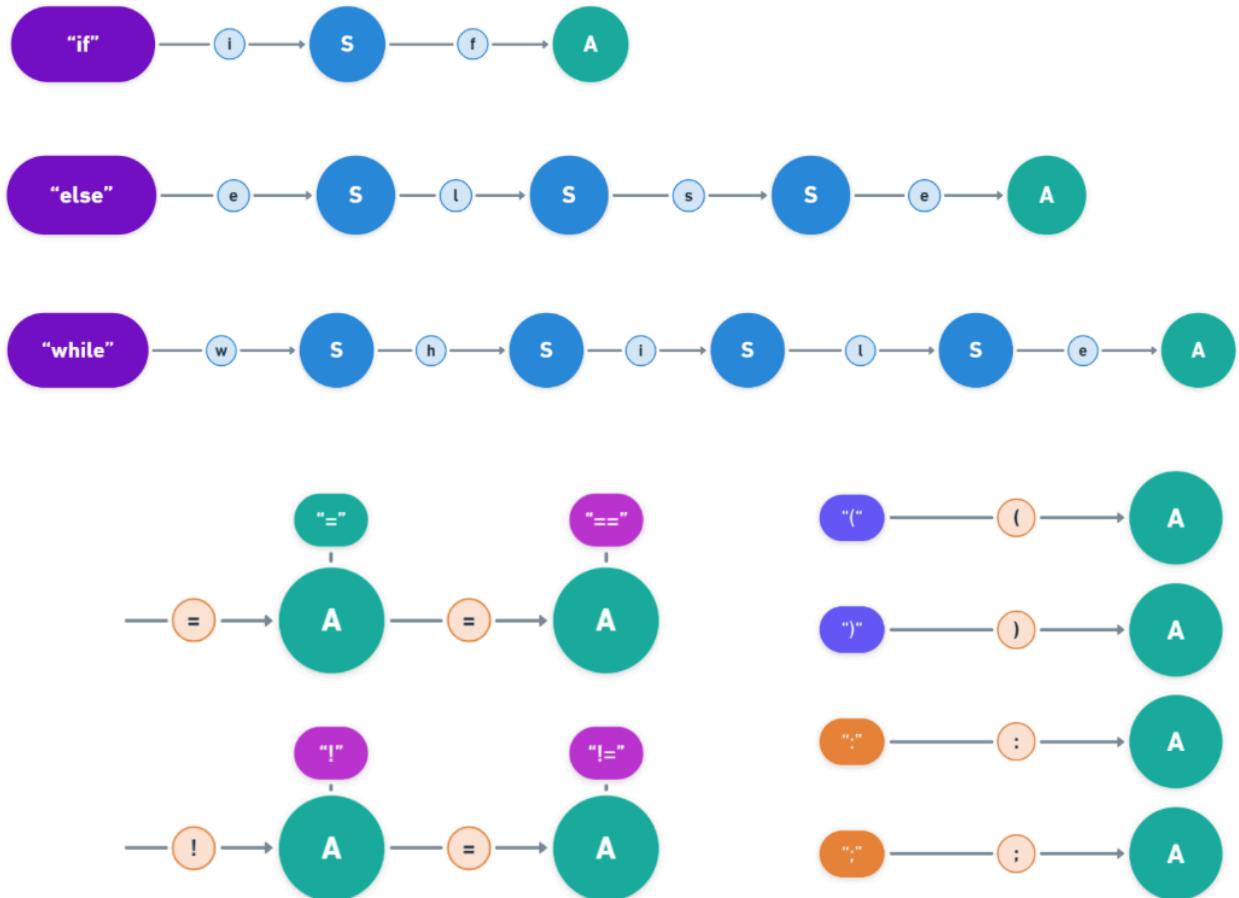
דוגמא עברו מטריצת הסמיוכיות של הנגרף המיציג את מכונת המוצבים של המנתה המילוני.

	„	đ	i	o	v	ó	1)	...
1	9	5	5	5	2				
2	9			3	5	5	5		
3	9	5	4	5	5	5	5		
4	9	7	5			5	5		
5	9	5		5	5	5	5		
6	9					6	6	8	
...									

במטריצה שלහן כל שורה מייצגת קודקוד. כל עמודה מייצגת שכנות של קודקוד. השכנות מסודרת לפי סוג התווים. עברו מצב מסויים ותו נוכחי מקוד המקור, נדע לאיזה מצב אנחנו צריכים לעבור. תא ריק מצין שאין שכנות בין המצב הנוכחי ובין התו הנוכחי מקוד המקור, ככלומר הגענו לעלה זהה סוף האסימונ.

דוגמא לתרשים האוטומט הסופי

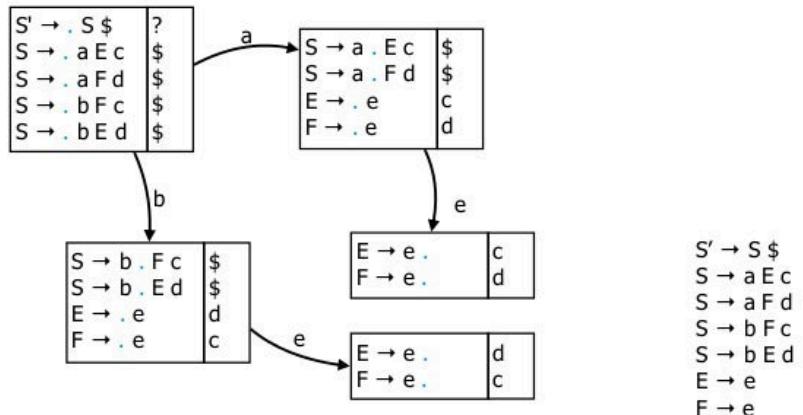




ניתוח תחבירי – Syntax Analysis

בשלב זה נרצה לבדוק האם רצף האסימונים שקיבלנו מהשלב הקודם הוא רצף אסימונים תקין על פי דקדוק שפת המקור אותה אנחנו מתרגמים, ובנוסף נרצה ליצור עץ ניתוח שייצג את התוכנית התקינה. בשלב הקודם, שלב ניתוח המילוני, השתמשנו בטבלה עבור מימוש הנגרף שמייצג את האוטומט הסופי של המנתה המילוני, בה עברו כל מצב ותו נוכחי ידענו לאיזה מצב עליינו לעבור. שלב הניתוח התחבירי מורכב יותר. התחביר של שלב זה הוא תחביר חופשי הקשור ולכן אינו יכול להתקבל על ידי אוטומט סופי. על מנת למשו נשתמש באוטומט מחסנית. כעת שהבנו זאת, טבלה אחת של מצב ותו נוכחי לא תספק לנו לביצוע המטלה. אנו עדין שואפים ליעילות זמן ריצה לינארית ולכן נשתמש באלגוריתם ניתוח מסוג Bottom Up, אלגוריתם ה - Reduce-Shift, ובאופן כללי, Parser LALR. עליו פירטתי בפרק סקירת אלגוריתמים בתחום הבעה.

תרשים פעולות LALR parser



Parsing table & Stack

על מנת למשוך את ה – Parser LR נשתמש במחסנית, ובשתי טבלאות שיחד יקראו table Action table – Goto table. Parsing

Action table

מצינית למנתח איזו פעולה – Shift, Reduce, Accept – הוא צריך לבצע, בהתאם למצב הנוכחי והאיסימון הבא מהקלט.

כל שורה תציג מצב, וכל עמודה תציג איסימון (Token / Terminal) שנמצא בשפה. נגיע לתא בטבלה על ידי המצב הנוכחי והאיסימון הבא מוקוד המקורי. כל תא בטבלה יגיד לנו איזו פעולה علينا לעשות, Shift או Reduce, ולאייה מצב עליינו לעבור לאחר מכן. תא ריק מציין שהגענו במצב שלא יכול להתקיים על פי תחביר השפה, כלומר שפיאה. בטבלה ה – Action ישנו רק תא אחד שמצוין את פעולה – Accept. תא זה הוא התא אליו נגיע לאחר שסיימנו את החוק הראשון בדקדוק במלואו, מה שמצוין תוכנית תקינה.

Goto table

מצינית למנתח לאיזה מצב הוא צריך לעבור לאחר ביצוע פעולה – Reduce. בטבלה ה – Goto כל שורה תציג מצב, וכל עמודה תציג משתנה (Non-terminal) שנמצא בתחביר השפה. נגיע לתא בטבלה על ידי המצב הנוכחי והמשתנה (Non-terminal) אשר נמצא בראש המחסנית. כל תא בטבלה יגיד לנו לאיזה מצב علينا לעבור בהתאם למצב הנוכחי ולמשתנה שנמצא בראש המחסנית לאחר ביצוע פעולה – Shift או – Reduce אשר table Action אמרה לנו לעשות.

State	Action Table						Goto Table		
	id	+	*	()	\$			
0	S5			S4			1	2	3
1		S6				ACC			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4			9	3	
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

המחסנית תחזיק בתוכה זוגות של Terminal או Non-Terminal ומספר מצב. בראש המחסנית תמיד יימצא מספר המצב הבא אליו צריך לעבור. המחסנית תאותחל עם איבר אחד שהוא מספר המצב הראשון, 0. עברו כל פעולה Shift נדוח למחסנית שני איברים: האחד הוא הTerminal הנוכחי מקוד המקור, והשני הוא מספר המצב כפי שצוין ב – table Action. עברו כל פעולה Reduce נוציא מהמחסנית את כמות האיברים שנמצאת ב – RHS של חוק הדקדוק שצוין ליד פעולה – Reduce בטבלת ה – Reduce כפول 2 (משום שעבור כל איבר שנדחף למחסנית אנו גם דוחפים את מספר המצב אליו צריכים לעبور), ונדוח למחסנית את ה – Non-Terminal שנמצא ב – LHS של אותו חוק דקדוק. בעת כל שנותר הוא לדוח את מספר המצב הבא לראש המחסנית. ניתן לדעת מספר זה על ידי התא בטבלת ה – Goto שנמצא בעמודה של ה – RHS של חלק הדקדוק שכעת דחפנו, ובשורה של מספר המצב שנמצא תחתיו במחסנית.

кар תראה המחסנית בהתחלה

0									
---	--	--	--	--	--	--	--	--	--

לאחר שתי פעולות Shift 3 לשואה a

0	a	3	a	3					
---	---	---	---	---	--	--	--	--	--

לאחר פעולה Reduce עברו חוק הדקדוק aa → A

0	A								
---	---	--	--	--	--	--	--	--	--

דחיפת מספר המצב המתאים על פי בטבת ה – Goto במיקום [0, A]

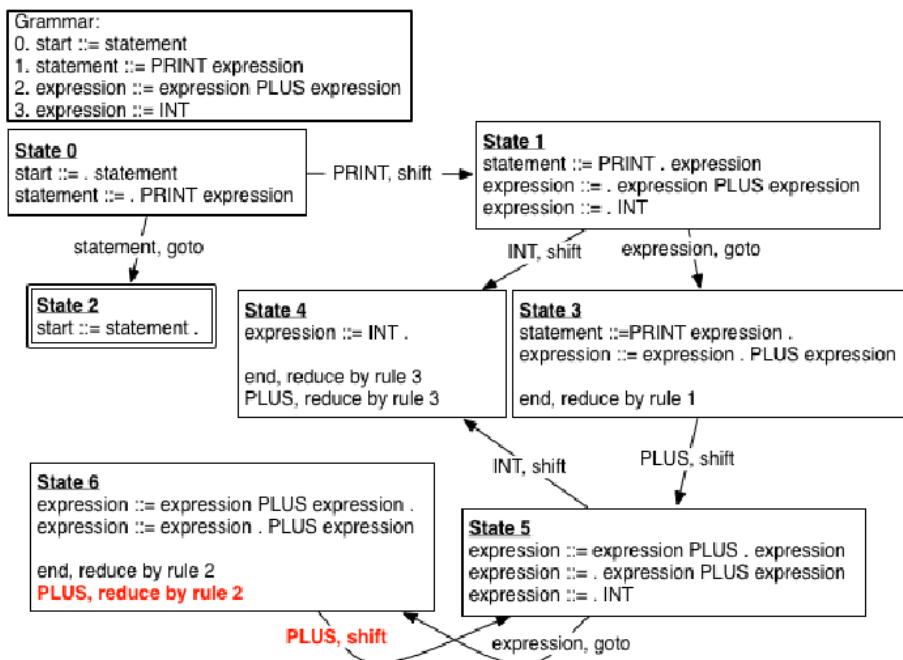
0	A	4							
---	---	---	--	--	--	--	--	--	--

ע"ז ניתוח

במהלך תהליך הניתוח התחבירי של תוכנית, נבנה במקביל ע"ז ניתוח המיצג את המבנה התחבירי של התוכנית. בנייה זו מתבצעת בשלב ה-Reduce של הפרסר, כאשר מיושם חוק דקדוק מסוים. בכל פעולה Reduce, נוסיף לע"ז ניתוח צומת חדשה המיצגת את הסמל האלא-טרמינלי (Non-Terminal) שבעודו שמאל (Left Hand Side - LHS) של חוק הדקדוק שיושם. הבנים של צומת זה יהיו הסמלים הטרמינליים (Terminals) שבעודו ימין (Right Hand Side - RHS) של חוק הדקדוק, כפי שהם נמצאים במחסנית הפרסר. הבנים הללו שמוסאים מהמחסנית הם בעצם עצים קטינים יותר שנבנו בשלבים קודמים של הפרסינג. כך, אט אט נבנה את ע"ז ניתוח המלא בתור המהנסית של הפרסר. בסיום תהליכי הניתוח התחבירי התקין, תישאר במחסנית יחידה אחת בלבד - ע"ז ניתוח המלא של התוכנית. כל צומת בע"ז תכיל את סוג התחבירי שלה (טרמינלי או לא-טרמינלי) ורשימה של הבנים שלה. בנייה מקבילה של ע"ז ניתוח במהלך הפרסינג מאפשרת ייצוג מבני של התוכנית, שיישמש לשלבים הבאים בבניית הקומpileר כגון ניתוח סמנטי ויצור קוד

LALR parser implementation

כפי שהסבירתי, תהליכי המימוש מורכב מכמה שלבים וכעת אציג אותם באמצעות שרטוט



Example parse of 'print 1 + 2'

Stack	Input	Action
[]	'print 1 + 2'	shift to state 1
[(1,PRINT)]	'1 + 2'	shift to state 4
[(1,PRINT),(4,INT)]	'+'	reduce by rule 3 to state 1, goto 3
[(1,PRINT),(3,expression)]	'2'	shift to state 5
[(1,PRINT),(3,expression),(5,+)]	'+'	shift to state 4
[(1,PRINT),(3,expression),(5,+),(4,INT)]	"	reduce by rule 3 to state 5, goto 6
[(1,PRINT),(3,expression),(5,+),(6,expression)]	"	reduce by rule 2 to state 1, goto 3
[(1,PRINT),(3,expression)]	"	reduce by rule 1 to state 0, goto 2
[(2,statement)]	"	accept

כפי שניתן לראות התהיליך מתחילה במצב 0 כאשר בכל שלב על הפרסר להחליט האם לעשות פעולה צמצום או פעולה שיפט או האם לעבור למצב מקבל כלומר האם נגמר הקלט והכל תקין

דוגמאות משפט ACE

עד כה דיברתי על מצבים תיאורתיים ונתקתי דוגמאות כלליות, כתעת ארצתה להציג דוגמאות אמיתיות מהקומpileר שלו

טבלאות goto & action

מפהת גודל הטבלה לא יכולתי למצוא דרך לצלם אותה ולהדביך כאן תצלום, לכן אציג לך לصفיה בקובץ האקסל שישדרתי לנוחות הקוראים

[לינק לטבלה](#)

ניתוח סמנטי

מהו ניתוח סמנטי?

ניתוח סמנטי הוא שלב חינוי בתהיליך הקומפליציה של שפות תכנות. לאחר שהקוד עבר בהצלחה את שלבי הניתוח המילוני (Lexer) והתחבירי (Parser), מניעתו של הניתוח הסמנטי. בשלב זה, מנתח הסמנטי בודק האם לתוכנית יש משמעותם הגיוני בהתאם על כללי השפה.

הבדל בין ניתוח תחבירי לניתוח סמנטי:

ניתוח תחבירי: בודק האם מבנה התוכנית תקין בהתאם לדקדוק השפה. כלומר, האם רצף הסמלים (*tokens*) מרכיב משפטים חוקיים בשפה.

ניתוח סמנטי: בודק האם לתוכנית יש משמעותם הגיוני בהתאם על כללי השפה. כלומר, האם השימוש במסתנים, פונקציות וביטויים תקין בהקשר נתון.

דוגמה:

המשפט הבא תקין מבחינה תחבירית, אך לא מבחינה סמנטית:

10 + "hello" := x

הסיבה לכך היא שימושה במסוג *int* (מספר שלם) לא יכול לקבל ערך מסוג *string* (מחרוזת).

מטרות ניתוח סמנטי:

בדיקות תקינות סוני המשתנים:

ודא שכל משתנה מקבל ערכים תואמים לסוגו.

ודא שביטויים מורכבים מורכבים מסוגים תואמים.

זהוי שימוש במסתנים לא מוגדרים:

ודא שכל משתנה מוגדר לפני השימוש בו.

זהוי ביטויים לא חוקיים:

ודא שפעולות מתבצעות בין סוגים תואמים.

ודא שאין ניסיונות לבצע פעולות לא חוקיות (לדוגמא, חילוק באפס).

דיהוי קוד מיותר:

דיהוי קוד שאינו משפיע על תוצאות התוכנית.

דיהוי קוד חשוד:

דיהוי קוד שעלול להוביל לשגיאות או התנהגות לא רצiosa.

יתרונות ניתוח סמנטי:

דיהוי שגיאות סמנטיות בשלב מוקדם של תהליך הפיתוח חוסך זמן ומאפשר השוואת תיקון שגיאות בשלבים מאוחרים יותר.

דיהוי קוד מיותר וקוד חשוד מסיע בכתיבת קוד נקי, יעיל וקריא יותר.

דיהוי ביטויים לא חוקיים וקוד חשוד מסיע במניעת פגימות אבטחה בתוכנה.

טכניות ניתוח סמנטי:

ניתוח טבלאות סמלים - שמיירה על מידע אודוט כל משתנה, כולל סוגו, ערכו והקשרים שלו.

ניתוח דירימת נתונים - מעקב אחר האופן שבו ערכים זורמים דרך התוכנית.

ניתוח סטטי - ניתוח התוכנית ללא ביצועה בפועל.

ניתוח דינמי - ניתוח התוכנית תוך כדי ביצועה.

בדיקות ייחודיות שלמות המשתנים

שם משתנה לא יכול להיות מוגדר מספר פעמים באותו ה- *Scope*. *Scope*, או תחום, הוא מושן חשוב בתוכנות. הוא מגדיר את האזור שבו משתנה זמני.

אסטרטניה לפתרון

Grammar Attribute הוא צורה מיוחדת של Grammar free-Context Non-terminal מקבל מידע נוסף, תכונה נוספת, Attribute, על מנת לספק לנו מידע תלוי הקשור, sensitive-Context. לכל תכונה כזו יש תחום מסוים של ערכים, לדוגמא, מספר, מחרוזת, ביטוי ועוד. כל Attribute היא בעצם צמד של <name Attribute> >value Attribute. Grammar Attribute הוא דרך לספק סמנטייקה ל- CFG, והוא יכול לעזר לנו להגדיר בד בבד את התחביר והסמנטייקה של שפה. אם נסתכל על ה- Grammar Attribute עצם, בפועל נראה שהוא מסוגל להעביר מידע בין הצמתים השונים בעץ, מה ש- CFG לבדו לא מסוגל לעשות.

הפקת קוד

יצירת קוד הוא השלב האחרון בתהליך הקומpileציה של שפות תכנות. בשלב זה, מתרגמים הקומpileר את עץ הניתוח המופשט (AST) והטבלת סמלים, המייצגים תוכנית תקינה לאחר שעברה את שלבי הניתוח המילוני (Lexer), התחבירי (Parser) והסמנטי (Semantic Analyzer), להוראות בשפת אסמבלי. הוראות אלו הן הוראות מכונה שניתן לבצע על ידי המעבד.

מטרות יצירת קוד:

תרגם AST להוראות בשפת אסמבלי: יצירת קוד לוקחת את AST המיציג את התוכנית ברמת הפשטה גבוהה ומפשיטת אותה להוראות מכונה ברמת הפשטה נמוכה.

יעול הקוד: יצירת קוד יכולה ליעול את הקוד שנוצר על ידי שימוש בטכניקות שונות, כגון הקצאת רישומים, הסרת הוראות מיותרות וסדר חדש של הוראות.

יצירת קובץ אובייקט: יצירת קוד מייצרת קובץ אובייקט המכיל את הוראות המכונה של התוכנית. קובץ אובייקט זה יכול להיקשר עם קבצי אובייקט אחרים על ידי linker כדי ליצור קובץ הפעלה.

אסטרטגיית יצירת קוד:

תרגום מבוסס טבלאות: ניתנת תרגום מבוססת טבלאות משתמש בטבלאות כדי להגדיר את התרגום של כל צומת ב-AST להוראות בשפת אסמבלי.

תרגום מבוסס דקדוק: ניתנת תרגום מבוססת דקדוק משתמש בדקדוק מיוחד כדי להגדיר את התרגום של כל צומת ב-AST להוראות בשפת אסמבלי.

תרגום מבוסס עץ: ניתנת תרגום מבוססת עצ' משתמש בעץ הניתוח המופשט (AST) עצמו כדי להנחות את תהליך יצירת הקוד.

שיקולים ביצירת קוד:

נכונות: הקוד שנוצר צריך להיות שקול מבחינה סמנטית לתוכנית המקורי.

יעילות: הקוד שנוצר צריך להיותiesel מבחינת זמן וזכרון.

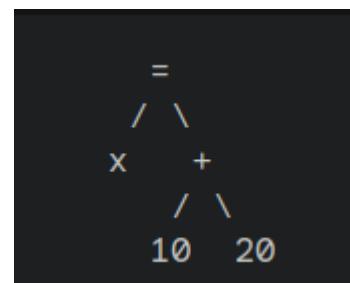
ニידות: הקוד שנוצר צריך להיות ניד בין פלטפורמות שונות.

דוגמה:

נניח שיש לנו את המשפט הבא בשפת C:

$20 + 10 = x$; int;

ה-AST עבור משפט זה עשוי להיראות כך:



יצירת קוד תתרגם את ה-AST זהה להוראות בשפת אסמבלי הבאות:

```

mov eax, 10
add eax, 20
mov [ebp-4], eax
  
```

הוראות אלו יטענו את המספר 10 לeax, את המספר 20 לeax, וישמרו את התוצאה במשתנה x.

טבלת סמלים

טבלת סמלים היא מבנה נתונים חשוב המשמש לאורך כל תהליך הקומpileציה של שפות תכנות. היא מאחסנת מידע על כל משתנה בתוכנית, כגון שמו, סוגו, ערכו ותחום הגישתו (Scope).

מטרות טבלת הסמלים:

שמירה ואחזור מידע על משתנים: טבלת הסמלים מאפשרת לנו לשמר ולמצוא מידע על כל משתנה בתוכנית בצורה ייעילה.

ניהול תחומי גישה (Scope Management): טבלת הסמלים עוזרת לנו לעקוב אחר תחומי הגישה של משתנים, כמו למשל באיזה חלקים בתוכנית כל משתנה זמין. בדיקות סמנטיות: טבלת הסמלים משמשת לבדיקות סמנטיות שונות, כגון בדיקת ייחודיות שמות משתנים ובדיקה סוגית נתונים.

יצוג טבלת הסמלים:

ניתן לציג את טבלת הסמלים באמצעות מבני נתונים שונים, כגון מערך, רשימה או מילון. ייצוג נפוץ הוא שימוש במילון (Dictionary/Hash Table). כיוון שכל משתנה בתוכנית חייב להיות בעל שם ייחודי, ניתן למפות כל משתנה למיקום ספציפי במילון באמצעות פונקציית גיבוב (Hash Function).

 יתרונות שימוש במילון:

גישה ייעילה: שימוש במילון מאפשר גישה מהירה למינימום בנסיבות זמן ריצה קבועה, O(1).

יעילות ذיכרון: שימוש במילון מאפשר ניצול יעיל של ذיכרון, כיוון שהמילון מוקצתה דינמית בהתאם לצורך.

 ניהול תחומי גישה:

ניהול תחומי גישה (Scope Management): הוא חלק חשוב משימוש בטבלת הסמלים. תחום גישה מגדיר את האזור בתוכנית שבו משתנה זמין. טכניקות נפוצות לניהול תחומי גישה: אחסון מילונים מקוונים: ניתן לאחסן מילונים מקוונים בטבלת הסמלים, כאשר כל מילון מייצג תחום גישה אחר.

שימוש בסימון טוקן מיוחד: ניתן להשתמש בסימון טוקן מיוחד (לדוגמה, #) כדי לציין את תחילת ותום תחום גישה.

טבלת סמלים היא כלי חשוב בתהליכי הקומpileציה, המאפשרת ניהול יעיל של מידע על משתנים, ניהול תחומי גישה וביצוע בדיקות סמנטיות. שימוש במילונים מקצועיים מאפשר גישה מהירה ויעילה למינימום, על מנת לא ליצור ניצול יעיל של ذיכרון.

טיפול בשגיאות

טיפול בשגיאות הוא חלק חשוב בתהליכי הקומpileציה של שפות תכנות. מטרתו לזהות ולדוח על טעויות בקוד המקורי שעלותו למנוע מהתוכנית להתבצע בצורה נכונה.

גישות שונות לטיפול בשגיאות

1. יציאה מיידית (Exit on Error):
יתרונות:

פושא לישום.
מנועת ביצוע קוד שניי.
חסרון:
מציגה רק את השגיאה הראשונה שנמצאה.
מחיבת את המשתמש לתקן שגיאות אחת ולקמפל שוב ושוב.

2. מצב חירום (Panic Mode):
יתרונות:
 ממשיכה בkompileציה לאחר גילוי שגיאה.
עשויה לזהות שגיאות נוספות.
חסרון:
 עשויה לדלג על חלקים שימושיים בקוד.
קשה יותר לאבחן את מקור השגיאה.

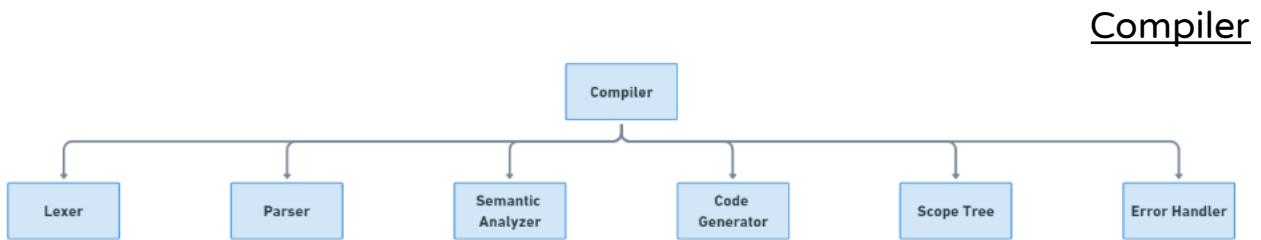
3. מצב משפט (Statement Mode):
יתרונות:
 מנסה לתקן את הקוד באופן אוטומטי.
עשויה לאפשר המשך kompileציה מוצלחת.
חסרון:
 מסובכת יותר לישום.
עלולה ליצור לולאות אינסופיות או שגיאות נוספות.
4. גישות נוספות:

בחירת גישת הטיפול בשגיאות תלואה במספר גורמים, כגון:
 מרכיבות השפה.
 סנון התכונות הרצוי.
 קהיל היעד של הכליל.

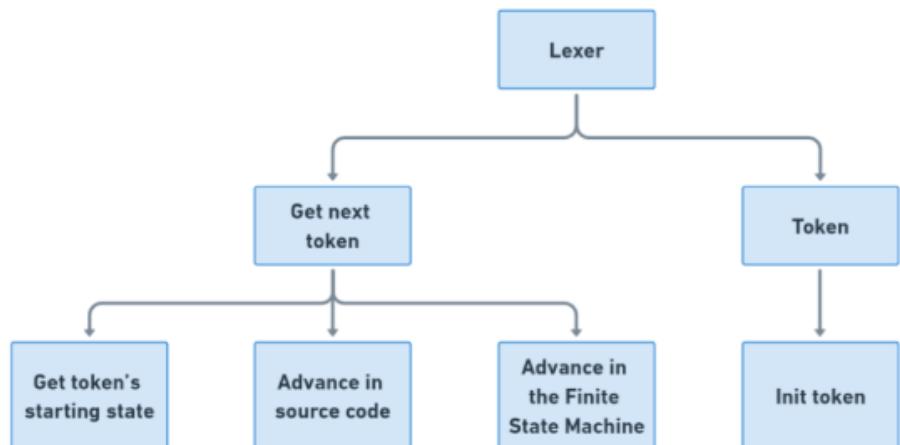
טיפול בשגיאות תחביר הוא נושא חשוב בתכנות. קיימות גישות שונות לטיפול בשגיאות, לכל אחת מהן יתרונות וחסונים משלها. בחירת הגישה המתאימה תלואה במספר גורמים, כגון:
 מרכיבות השפה, סנון התכונות הרצוי וקהיל היעד של הכליל.

במסגרת פרויקט זה, בחרתי בגישה "יציאה מיידית" (Exit on Error) בשל פשוטותה ויעילותה.

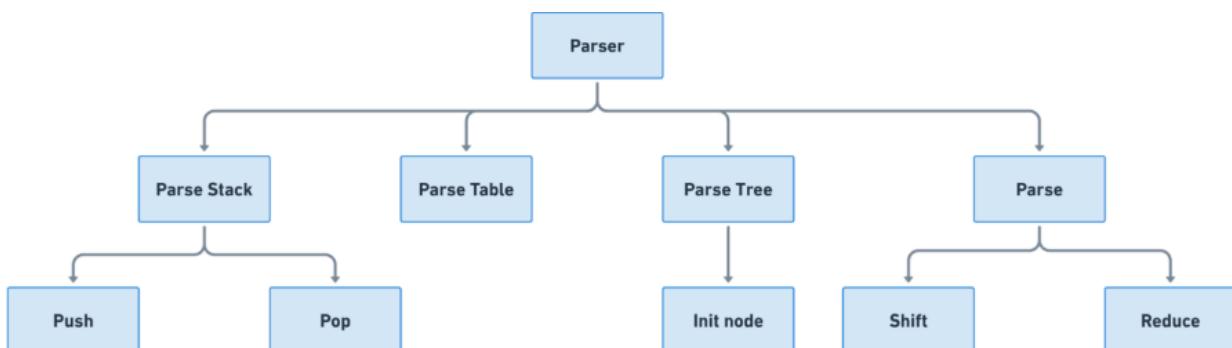
Top down level design



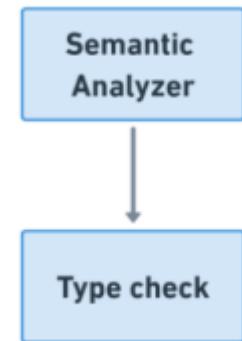
Lexer



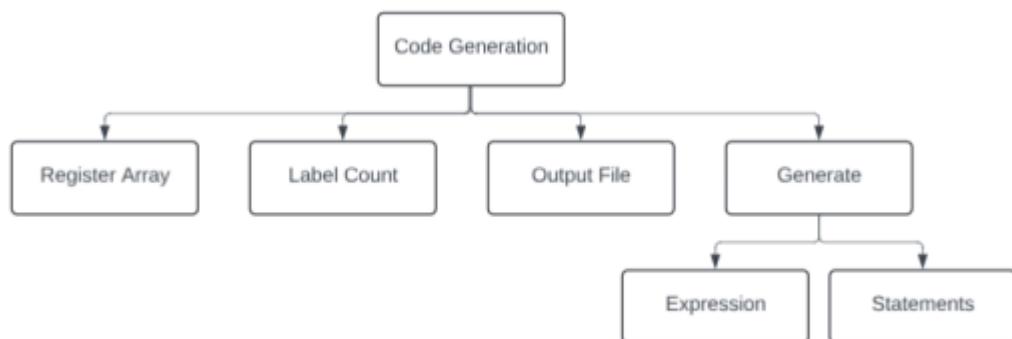
Parser



Semantic analyzer



Code generation



מודולים

כעת ארכחיב על כל מודול ואפרט מה תפקידו, מהם ה – Structs שמכיל, אילו קבצים שייכים לאותו מודול, מהם המודולים הנמצאים תחתיו, מהן הפונקציות שמכיל ומהי היעילות האלגוריתמית של כל אחת מהן.

Token

מודול המיציג אסימון שהוא אבן יסוד ללקסר

```
struct Token
{
    std::string token; //lexeme
    int type; //tokId from enum
    int loc; //line of code
} typedef Token;
```

Lexer

מודול המכיל את כל הֆונקציות הקשורות לשלב ה CodeGeneration בפරסר.

```

class Lexer
{

private:
    std::string _lastCollectedValue; // Last collected value
    list<Token> tokenList;          // List of tokens
    std::string path;               // Path to the file
    DFA *dfa;                      // Deterministic Finite Automata

public:
    // Constructor
    Lexer(std::string path);

    // function to read DFA from file
    void readFromFile();

    /// @brief Function to return the string of the token
    /// @param code
    /// @return
    std::string returnTokenString(int code);

    /// @brief Function to analyze the current word
    /// @param currWord
    /// @param loc line of code
    /// @return Token
    Token analyze(std::string currWord, int loc);

    /// @brief Function to get the token list
    /// @return list<Token>
    list<Token> getTokenList() { return this->tokenList; }

};


```

<u>פונקציה</u>	<u>קלט</u>	<u>פלט</u>	<u>תיאור</u>	<u>יעילות</u>
<u>lexer</u>			<u>אתחול</u>	<u>O(1)</u>
<u>readFromFile</u>			<u>קריאה מקובץ</u>	<u>O(1)</u>
<u>analyze</u>	<u>מילה נוכחית</u> <u>ושורת קוד</u> <u>נוכחית</u>	<u>טוקן</u>	<u>מנתחת מילה</u> <u>ומתיגת אותה</u> <u>כטוקן</u>	<u>(N)</u>
<u>getTokenList</u>		<u>רשימת טוקנים</u>	<u>החזרת רשימת הטוקנים</u>	<u>O(1)</u>
<u>returnTokenString</u>	<u>מספר</u>	<u>מחרוזת</u>	<u>מחזיר מילה</u> <u>עבור טוקן</u> <u>לדיבאגינג</u>	<u>O(1)</u>

Parser

מודל האחראי על המרת רשימת הטוקנים לעץ סינטקס אבסטרקט

```

66 class ParseTree{
67 public:
68     ParseTree* root;
69     int value;
70     Token token;
71     std::vector<ParseTree> children;
72     types type;
73     RegisterEntry reg;
74     void setRoot(Token *root);
75     void setReg(RegisterEntry reg);
76     RegisterEntry getReg();
77
78     ParseTree()
79     {
80         this->root = NULL; // create a new token
81         this->children = {}; // assign an empty vector to children
82     }
83
84     ParseTree(Token *root)
85     {
86         this->root = new ParseTree; // create a new token
87         Token *newRoot = new Token(); // create a new token
88         newRoot->token = root->token; // set the token value
89         newRoot->type = root->type; // set the token type
90         newRoot->loc = root->loc; // set the line number
91         newRoot->type = root->type;
92         this->root->token = *newRoot;
93         this->children = {}; // children of the root
94     }
95
96 };
97
98 class Parser{
99 private:
100    Stack<ParseTree> parStack;           //stack to store the parse tree
101    Stack<int> tokenStack;               //stack to store the tokens
102    std::list<Token> inputList;          //list of tokens
103    std::vector<Production> productions; //list of productions
104    std::vector<std::vector<std::string>> actionGoTable; //action go table
105
106 //to convert between token id and action table keys we can use a hashtable which maps token id to action table keys
107 std::unordered_map<int, actionTableKeys> tokenActions;
108
109 public:
110     /// @brief Constructor
111     Parser();
112
113     /// @brief Function to set the rule list
114     void setRuleList();
115
116     /// @brief Function to set the input list
117     /// @param inputList
118     void setInputList(list<Token> inputList){this->inputList = inputList;}
119
120     /// @brief Function to read the productions from the file
121     void generateParseTables();
122
123     /// @brief Function to parse the input list
124     ParseTree parse();
125
126     /// @brief Function to print the parse tree
127     void makeMap();
128
129     /// @brief Function to print the parse tree
130     /// @param root
131     /// @param prefix
132     /// @param isLast
133     void printAST(ParseTree& root, string prefix, bool isLast);
134
135 };

```

<u>יעילות</u>	<u>תיאור</u>	<u>פלט</u>	<u>קלט</u>	<u>פונקציה</u>
<u>O(1)</u>	<u>אתחול</u>			<u>parser</u>
<u>O(1)</u>	<u>הכרזת חוקי השפה</u>			<u>setRuleList</u>
<u>O(1)</u>	<u>קבלת רשימת הtokנים</u>		<u>רשימת tokנים</u>	<u>setInputList</u>
<u>O(1)</u>	<u>קריאה גוטן ואקשין טיבול מהזיכרנו (קובץ)</u>			<u>generateParseTable</u>
<u>O(1)</u>	<u>פעולה מתחלה</u>			<u>Parse</u>
<u>O(1)</u>	<u>מפה סוגית tokנים לסוגי אקשנים</u>			<u>makeMap</u>
<u>O(N)</u>	<u>מדפיס את העץ</u>		<u>שורש העץ, האם סופי, וקידומת</u>	<u>printAST</u>

Stack

מודל האחראי לשמר את הנתונים בצורה ראשונית פנימה אחרון החוצאה

```

1  #pragma once
2  #include <iostream>
3
4  #define SIZE 1000
5
6  using namespace std;
7
8  template <class T> class Stack {
9  public:
10     Stack() { top = -1; }
11
12     // To add element element k to stack
13     void push(T k){
14         // Checking whether stack is completely filled
15         if (isFull()){
16             cout << "Stack is full\n";
17         }
18
19         // Inserted element
20         top = top + 1;
21         st[top] = k;
22     }
23
24     // To remove the top element from stack
25     T pop(){
26         // Initialising a variable to store popped element
27         T popped_element = st[top];
28         top--;
29         return popped_element;
30     }
31
32     // To get the top element of stack
33     T peek()
34     {
35         // Initialising a variable to store top element
36         T top_element = st[top];
37
38         // Returning the top element
39         return top_element;
40     }
41
42     // To check if the stack is full
43     bool isFull(){
44         if (top == (SIZE - 1))
45             return 1;
46         else
47             return 0;
48     }
49
50
51     // To check if the stack is empty
52     bool isEmpty()
53     {
54         if (top == -1)
55             return 1;
56         else
57             return 0;
58     }
59
60     private:
61     int top;
62     T st[SIZE];
63 };
64

```

<u>פעולות</u>	<u>תיאור</u>	<u>פלט</u>	<u>קלט</u>	<u>פונקציה</u>
O(1)	<u>דחיפת איבר</u> <u>ראש</u> <u>המחסנית</u>		<u>איבר לדחוף</u> <u>לחחסנית</u>	<u>push</u>
O(1)	<u>הסרה</u> <u>וחזרה של</u> <u>האיבר</u> <u>שבראש</u> <u>המחסנית</u>	<u>האיבר</u> <u>שבראש</u> <u>המחסנית</u>		<u>pop</u>
O(1)	<u>מצאה ראש</u> <u>המחסנית</u>	<u>הערך</u> <u>שבראש</u> <u>המחסנית</u>		<u>peek</u>
O(1)	<u>בדיקה</u> <u>מחסנית</u>	<u>ערך בוליאני</u> <u>שמתאר האם</u> <u>המחסנית</u> <u>ריקה</u>		<u>isEmpty</u>
O(1)	<u>פעולה</u> <u>מתחילת</u>			<u>Stack</u>

semanticAnalysis

מודול האחראי על דיווח שגיאות מסוג שניאות התאמת טיפוסים וכדומה

```

1 #pragma once
2 #include "Symbol.hpp"
3 #include <unordered_set>
4
5 class SemanticAnalysis
6 {
7 private:
8     unordered_set<Symbol, Symbol::HashFunction> symbolTable;    // Symbol table
9     ParseTree* parseTree;                                         // Parse tree
10
11    unordered_map<string, int> mapOfTypes = {
12        {"int", INT},
13        {"bool", BOOL},
14        {"char", CHAR},};
15
16 public:
17     /// @brief Constructor
18     /// @param parseTree
19     SemanticAnalysis(ParseTree* parseTree);
20
21     /// @brief Function to create the symbol table
22     /// @param parseTree
23     /// @param scope
24     void createSymbolTable(ParseTree* parseTree, int scope);
25
26     /// @brief getter for the symbol table
27     /// @return std::unordered_set<Symbol, Symbol::HashFunction>
28     unordered_set<Symbol, Symbol::HashFunction> getSymbolTable();
29
30     /// @brief Functions to perform semantic analysis
31     /// @param tree
32     /// @param scope
33     /// @return ParseTree*
34     ParseTree* semantic();
35     ParseTree* semanticHelper(ParseTree *tree, int scope);
36
37     /// @brief Function to get the type of the symbol
38     /// @param type
39     /// @return
40     string getSymbolType(int type);
41
42     /// @brief Function to check if the token is a comparison operator
43     /// @param token
44     /// @return bool value to indicate if the token is a comparison operator
45     bool isCmpOp(string token);
46
47     /// @brief Function to identify the type of the token
48     /// @param token
49     /// @return type of the token
50     int identifyType(Token *token);
51
52     /// @brief Function to print the symbol table
53     void printSymbolTable();
54 };

```

<u>יעילות</u>	<u>תיאור</u>	<u>פלט</u>	<u>קלט</u>	<u>פונקציה</u>
O(1)	אתחול		עץ פרסר	<u>semanticAnalysis</u>
O(N)	יצירת טבלאות הסימנים		עץ פרסר וטוווח	<u>createSymbolTable</u>
O(N)	מבצע ניתוח סמנטי	עץ פרסר		<u>semantic</u>
O(N)	פה מtabצע הניתוח בפועל	עץ פרסר	עץ פרסר וטוווח	<u>semanticHelper</u>
O(1)	הזהרת טבלאות סימנים	טבלת סימנים		<u>getSymbolTable</u>
O(1)	בודק האם טוקן מסוים הוא פועלות השוואה	אמת או שקר	טוקן	<u>isCmpOp</u>
O(1)	בודק טיפוס מסוים של טוקן	מספר	טוקן	<u>identifyType</u>
O(N)	הדפסת טבלה לדיבוג			<u>printSymbolTable</u>

Code generation

מודל האחראי על הפקת הקוד מהעץ המוגמר

```

1  #pragma once
2  #include "Parser.hpp"
3  #include "Symbol.hpp"
4  #include "RegisterEntry.hpp"
5  #include <iostream>
6
7  class CodeGen
8  {
9  private:
10     int labelCount;           // Counter for labels
11     vector<string> code;    // Vector to store the generated code
12     std::string fileName;    // Name of the file
13     unordered_set<Symbol, Symbol::HashFunction> symbolTable; // Symbol table
14     std::vector<RegisterEntry> registers = {
15         {"eax", true}, {"ebx", true}, {"ecx", true},
16         {"edx", true}, {"edi", true}, {"esi", true}, }; // Vector to store the registers
17
18 public:
19
20     /// @brief Constructor
21     /// @param tree
22     CodeGen(std::string fileName, unordered_set<Symbol, Symbol::HashFunction> symbolTable)
23     {this->fileName = fileName; this->symbolTable = symbolTable; labelCount = 0;};
24     ~CodeGen();
25
26     /// @brief Function to generate the code
27     bool generate(ParseTree *tree);
28
29     /// @brief Function to execute the nasm code
30     /// @param outputFileName
31     void executeNasm(string outputFileName);
32
33     /// @brief Function to create the base assembly code
34     /// @param file
35     void createBaseAsm(ofstream &file);
36
37     /// @brief Function to generate the identifiers
38     /// @param file
39     void generateIdentifiers(ofstream &file);
40
41     /// @brief Function to generate the code from the AST
42     /// @param tree
43     void generateCodeFromAST(ParseTree *tree);
44
45     /// @brief Function to generate the code for the statements
46     /// @param tree
47     /// @return RegisterEntry
48     RegisterEntry generateExpression(ParseTree *expression);
49
50     /// @brief Function to generate the code for the statements
51     /// @param tree
52     /// @return RegisterEntry
53     string convert32to8(RegisterEntry reg, int byte);
54
55     /// @brief Function to generate the code for the statements
56     /// @param regs
57     /// @param n
58     RegisterEntry getRegister();
59
60     /// @brief Function to free the register
61     /// @param reg
62     void freeRegister(RegisterEntry reg);
63
64     /// @brief Function to set the register state
65     /// @param reg
66     /// @param state
67     void setRegState(RegisterEntry reg, bool state);
68
69     /// @brief Function to push the registers
70     /// @param regs
71     /// @param n
72     void pushRegs(string *regs, int n);
73
74     /// @brief Function to pop the registers
75     /// @param regs
76     /// @param n
77     void popRegs(string *regs, int n);
78
79     /// @brief Function to add the code to the vector
80     /// @param code
81     void addCode(string code);
82 };
83

```

<u>יעילות</u>	<u>תיאור</u>	<u>פלט</u>	<u>קלט</u>	<u>פונקציה</u>
O(1)	<u>מוסיף שורה</u> <u>קוד לרשימה</u> <u>השורות</u>	<u>אמת או שקר</u>	<u>מחרוזת</u>	<u>addCode</u>
O(N)	<u>קורא לכל</u> <u>פונקציות</u> <u>יצירת הקוד</u>		<u>עץ פרסר</u>	<u>generate</u>
O(1)	<u>מפעיל</u> <u>פקודות</u> <u>להרצת קוד</u> <u>אסמבלי</u>		<u>שם קובץ</u>	<u>executeNas</u> <u>m</u>
O(1)	<u>يוצר בסיס</u> <u>לקוד אסמבלי</u>		<u>קובץ</u>	<u>create</u> <u>BaseAsm</u>
O(N)	<u>יצירת כל</u> <u>המשתנים</u> <u>המוכרזים</u> <u>לזכרו</u>		<u>קובץ</u>	<u>generate</u> <u>Identifiers</u>
O(1)	<u>כואן מתבצעת</u> <u>עיקר היצירה</u> <u>של הקוד</u> <u>מਐיס</u> <u>לאסמבלי</u>		<u>עץ פרסר</u>	<u>generateCo</u> <u>deFromAst</u>
O(N)	<u>מחשבת את</u> <u>הביתוי שבעץ</u>	<u>רנייסטר עם</u> <u>תוצאת</u> <u>הביתוי</u>	<u>עץ פרסר</u>	<u>generateEx</u> <u>pression</u>
O(1)	<u>ההמרה ביתוי</u> <u>ברנייסטר 32</u> <u>בית לרגניסטר</u> <u>8 בית</u> <u>בהתאם לדגל</u>	<u>מחרוזת</u>	<u>רנייסטר, דגל</u>	<u>convert32to</u> <u>8</u>
O(1)	<u>מחזיר</u> <u>רנייסטר</u>	<u>רנייסטר</u>		<u>getRegister</u>

	<u>חופשי</u>			
O(1)	<u>קובע האם רגיסטר בשימוש או לא</u>		<u>רגיסטר, מצב</u>	<u>setRegState</u>
O(1)	<u>דוחיפת כל הרגיסטרים למחסנית</u>		<u>וקטור רגיסטרים וכמויות</u>	<u>pushRegs</u>
O(1)	<u>שליפה כל הרגיסטרים מהמחסנית</u>		<u>וקטור רגיסטרים וכמויות</u>	<u>popRegs</u>
O(1)	<u>שחרור זיכרון הרגיסטר</u>		<u>רגיסטר</u>	<u>freeRegister</u>

אלגוריתם ראשי

Compiler

1. קיבל מהמשתמש את קובץ הקלט ace.

2. אתחל את Lexer.

3. אתחל את parser.

4. הכנס לתוך הרשימה tokens את פלט פונקציית lexer.

a. התחל.

b. אתחל את גרען האוטומט הסופי של המנתה המילוני.

c. עברו על התווים בקובץ הקלט.

i. עברו אל המצב ההתחלה עבור התו הנוכחי בקוד המקור.

ii. כל עוד לא הגעת למצב -1 בצע.

1. עברו למצב הבא על פי המצב הנוכחי והתו הנוכחי בקוד המקור.

2. התקדם تو אחד בקוד המקור.

iii. צור אסימון על פי העלה שהגעת אליו.

iv. הוסף את האסימון לרשימת האסימונים.

d. סיום.

5. אתחל את parser.

6. הכנס לתוך העץ ParseTree את פלט פונקציית parser.

a. התחל.

b. אתחל את המחסניות ואת הפעולות Action & Goto.

c. הכנס למשתנה a את האסימון הראשון מקוד המקור.

d. כל עוד לא עברת על כל קוד המקור בצע:

i. הכנס למשתנה s את המצב שנמצא בראש המחסנית:

בצע s == Action[s, a] אם ii. shift == Action[s, a]:

1. דוחף את a למחסנית העצים.

2. דוחף את [Action[s, a], a] למחסנית המצביעים.

3. הכנס למשתנה a את האסימון הבא מקוד המקור.

בצע s == reduce:

4. קבע את R להיות ה Non-Terminal שנמצא לצד השמאלי של rule n.

5. הוציא ממחסנית העצים ח איברים ובני מהם עץ שהאב שלו הוא R, כ-*s*-*t* הוא

מספר האיברים הימיני בכל n.

6. הוציא ממחסנית המצביעים ח איברים ח זהה.

7. הכנס למשתנה v את המצב הנמצא כתע בראש מחסנית המצביעים.

8. דוחף את העץ החדש למחסנית העצים.

9. דוחף את [A, Goto[v, a]] למחסנית המצביעים.

10. בצע s == accept, אם אחרת iii. action[s, a] == accept:

i. סיום והחזיר את עץ הניתוח שנמצא בראש המחסנית.

ii. אחרת, בצע:

1. דוחה על השגיאה שנמצאה.

e. סיום.

7. אתחל את Symantic.

8. הכנס לתוך AST את העץ החדש שנוצר מפונקציית *semanticic*.

a. עברו על כל התתי עצים בא-AST בסריקה תוכית:

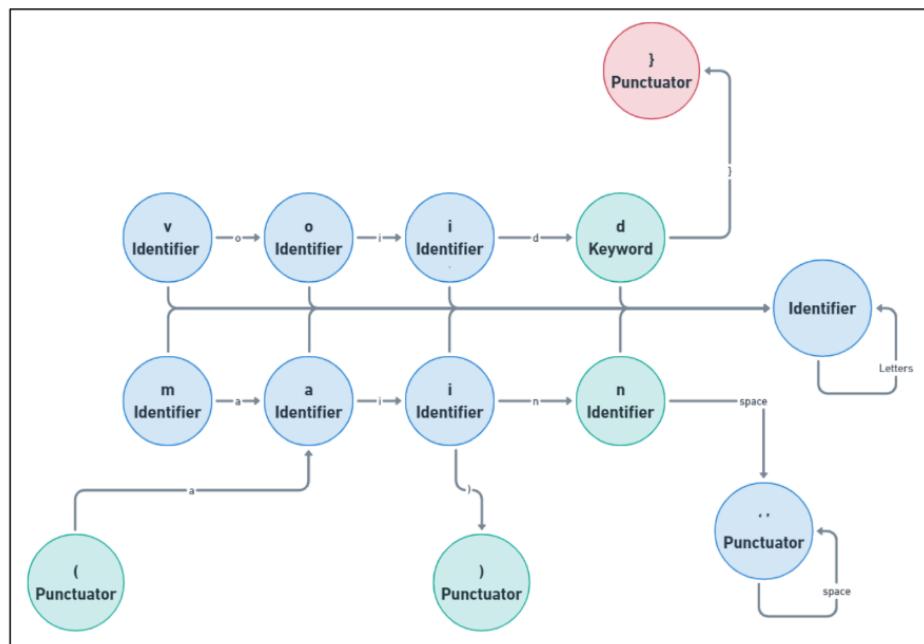
i. הכנס לתוך root את הערך בעלה.

- i. אם ערך root הוא identifier ו term ו expression או literal.
- :assignment
- .table symbol להתחם ל accordance עם type.
1. בצע בדיקת תקינות סמנטית בהתאם ל symbol.
2. הנדר את type העליה לפני הבדיקה הסמנטית.
- b. החזר
9. אתחל את CodeGeneration.
10. בצע את פעולה generate.
- a. סרוק את העץ בסריקה תוכית:
- i. עבור כל statement שלח את העליה לפונקציה המטפלת בסוג ההשارة.
- d. רשום את כל הקוד לקובץ
- c. סיום
11. סיום

מבנה נתונים

מנתח מילוני – Lexical Analysis

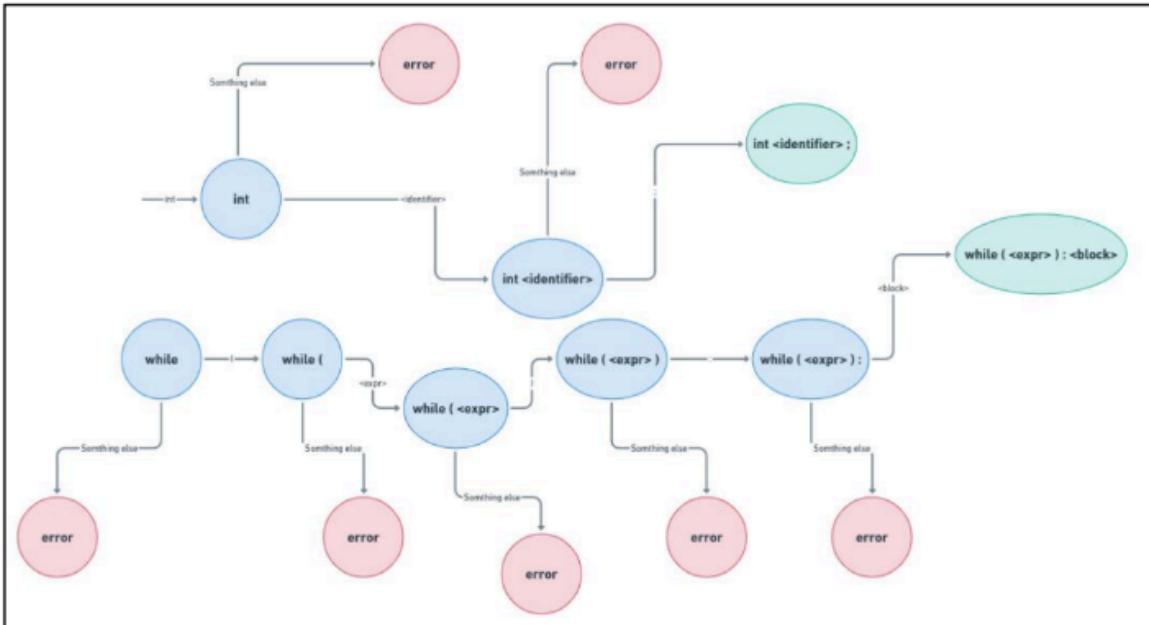
עבור המנתח המילוני השתמשתי במכונת מצבים סופית המייצגת על ידי גրף. משקל הקשרות בגרף הן אוטיות, כך שמעבר כל אות ומצב נכון נעהו למצב ספציפי אחר (אוטומט דטרמיניסטי). כאשר נגע במצב בו אין לאן להתקדם עברו התו הבא מהקלט, סימן שהגענו במצב סופי. ווחזיר את ה – Token המשמר במצב זה. עבורתו לא צפוי נגע במצב שנייה. מבנה נתונים זה בעצם מאפשר לנו לשמור על ייעילות זמן ריצה לינארית, ($O(n)$), משום שהוא לא צריך לשאול שום שאלות, או לבדוק מספר אופציות עבור כל מצב. כל המצבים האפשריים מצויים בתחום מבנה הנתונים וכך בכל מצב אנחנו יכולים להכריע איזו פעולה علينا לעשות.



מנתח תחבירי – Syntax Analysis

עבור המנתח התחבירי השתמשתי באוטומט מחסנית המיוצג על ידי גראף. כל צומת בגרף מייצג מצב נכון, מיקום, בחוק בתחביר השפה, והקשרות בגרף היו ה – Tokens שקיבלו מהתה המילוני. עבור מצב נכון, וה – Token הבא מהקלט נדע איזו פעולה עליינו לעשות בהתאם מטה מעלה (Accept, Error, Reduce, Shift). עבור Token לא צפוי מהקלט נגע במצב שנייה.

מבנה נתונים זה מאפשר לנו לקבל החלטה עבור כל מצב אפשרי במהלך תהליך הקומpileציה. אנו לא צריכים לשאול שום שאלות, או לבדוק מספר אופציות, מה שעוזר לנו לשמר על יעילות זמן ריצה לינארית, (O)n.



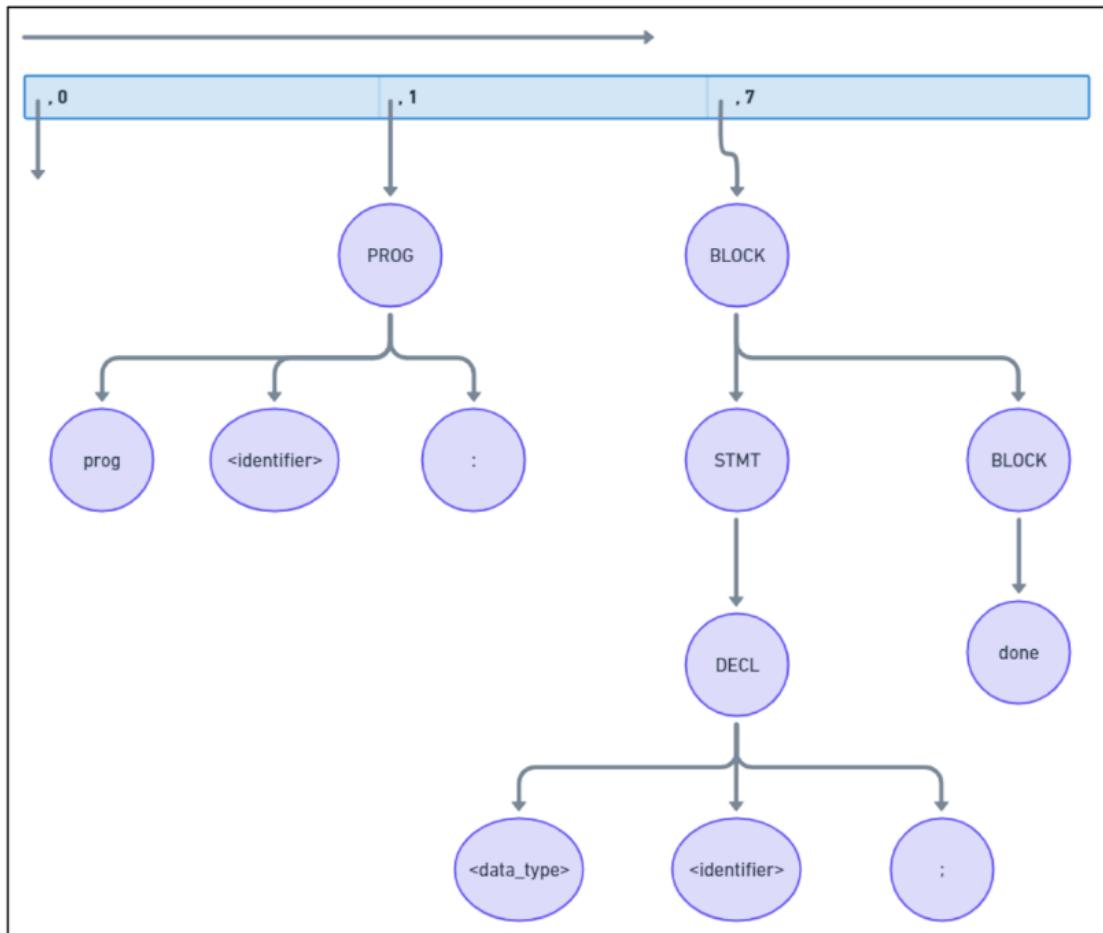
Parse Table

על מנת לממש את הנגרף שמייצג את אוטומט המחסנית השתמשתי בשתי טבלאות, & **Action**, אשר יחד נקראות **Parse Table**. אלו בעצם שתי מטריצות כך שב_TBL ה – **Action** כל תא הוא הפעולה שצרכי לעשות עבור המצב והאיסימון הנוכחי, וב_TBL ה – **Goto** כל תא הוא מספר המצב אליו צריך לעבור עבור המצב הנוכחי וה – **Non-Terminal** שבראש מחסנית הניתוח. הסבר מפורט על טבלאות ה – **Action** וה – **Goto** ניתן למצוא ב – **Stack & table Parsing** בפרק האסטרטגיה. ישנה לתאים השונים במטריצות תחכצע ביעילות אלגוריתמית קבועה, (O)1, ובכך תעדור לנו לשמר על יעילות זמן ריצה לינארית, (O)n, של הקומpileר.

State	Action Table							Goto Table		
	id	+	*	()	\$	E	T	F	
0	S5			S4			1	2	3	
1		S6				ACC				
2		R2	S7		R2	R2				
3		R4	R4		R4	R4				
4	S5			S4			8	2	3	
5		R6	R6		R6	R6				
6	S5			S4				9	3	
7	S5			S4					10	
8		S6			S11					
9		R1	S7		R1	R1				
10		R3	R3		R3	R3				
11		R5	R5		R5	R5				

Parse Stack

עבור אוטומט המחסנית אנו כמובן צריכים מחסנית. עבור המחסנית של האוטומט השתמשתי במחסנית בה כל איבר מכיל את שורש העץ שנבנה עד כה בתא זה במחסנית, ובנוסף כל איבר במחסנית שומר לאיזה מצב צוריך ללקת לאחר מכן. המחסנית מאפשרת לנו "לזכור" את הדברים שראינו כך שבמהלך נוכל להוציאו אוטם מהמחסנית ולהשתמש בהם במקום לחפשם שוב. בכך היא עוזרת לשמר על ייעילות זמן ריצה לינארית, (O)n, של הקומפיילר.



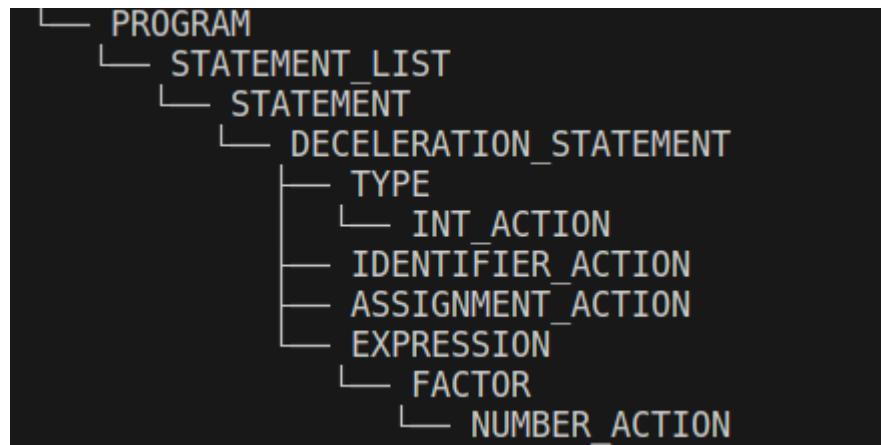
דקדוק השפה – Rules Production

על מנת לשמור את כללי היצירה של השפה נשימוש במערך המכיל עבור כל כלל יצירה את ה – Non-terminal שבס – RHS שלו, ואת מספר הסמלים שיש בו – RHS של כלל היצירה. כך נוכל לדעת בסיבוכיות זמן ריצה קבועה, (1)O, כמה איברים צריך להוציא ממחסנית בכל Reduce, ובנוסף נוכל לדעת איזה Non-terminal צריך להיות בשורש העץ לאחר פעולה הה – Reduce.

עץ הניתוח – Tree Parse

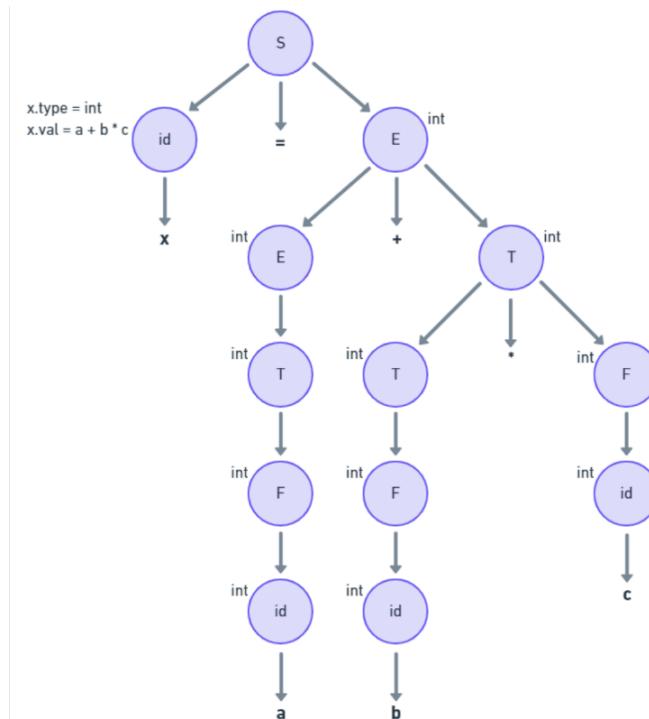
נרצה שהמנתח התחרבוי יחזיר לנו עצם המיציג את התוכנית. כל צומת בעץ יכול Terminal או Non-terminal. לכל צומת המכיל Non-terminal, יהיה מערך של הבנים שלו, וכל עלה, צומת המכיל Terminal, יחזק Token מקוד המקוור. העץ נבנה את כתתי עצים בתוך ה – Stack Parse, והוא נבנה כך שגם נὑבור עליו בסדריקת תוכית, Inorder.

לקבל את תוכנית המקור, העץ מייצג את מבנה התוכנית, ובעצם מתוך המבנה שלו אנו יכולים לקבל החלטות עבור כל מקום בעץ. דבר זה עוזר לנו לשמר על יעילות אלגוריתמית לינארית, ($O(n)$, של הקומpileר).



מנתח סמנטי – Analysis Semantic

עבור המנתח הסמנטי נוסיף לכל צומת בעץ הניתוח Attribute שיאוסיף לנו סמנטיקה נוספת בנוסף למבנה של התוכנית. כך נוכל למצוא ולנתח שגיאות סמנטיות אשר לא נתפסות בשלבים הקודמים של הקומPILEר. התכונות שלעיל נספנות לעצ tor כדי תחילך הניתוח ביעילות זמן ריצה קבועה, ($O(1)$), ובכך לא פוגעות ביעילות זמן הריצה הלינארית, ($O(n)$), של הקומPILEר.



Hash table

מטרתה של טבלת הסמלים היא שטירה ואחזור של מידע אודות המשתנים השונים בתוכנית. על מנת שתבוצע את עבودתה בצורה היעילה ביותר, נשתמש ב – Hash table.

על כל שם משתנה נפעיל פונקציית Hash אשר תיתן לנו את מיקום שם המשתנה בטבלת ה – Hash. וכך נוכל לנשח למקומם המשתנה הספציפי זהה במערך ביעילות זמן ריצה קבועה, ($O(1)$).

סביבת העבודה

עורך קוד

עובדתי בסביבת **visual studio code**



מערכת הפעלה

מערכת הפעלה **linux mint**



שפת תכנות



שפת **C++**

קומpileר

g++ compiler



מדריך למשתמש

דרישות

וודאו תחילה כי מותקן על מכשיר הלינוקס שלכם קומפיילר C++ עדכני וגם תוכנת NASM
מכיוון שקיים בי תסקול רב על הבנת מדריכים ופקודות ארוכים ומסובים החלטתי להפוך את
השימוש בשפה פשוט יותר שביקולתי

1. ראשית העתיקו את הפרויקט מעמוד האינטרנט שלי לתוך תיקייה ריקה
2. לאחר מכן הריצו את הפקודה ace *.cpp -o -++ וען לкопל את הפרויקט
3. אחר כך כתבו להנאתכם קוד בשפת ACE
4. בשלב האחרון הריצו את הפקודה
`./ace <full-file-path.ace>`
על מנת לкопל ולהריץ קובץ עם סימת מתאימה.
בתום התהליך תקבלו גם קובץ הריצה

*הערה: הקומפיילר בניו לעבודה עם מערכת הפעלה מסווג לינוקס בלבד

סיכום אישי

מסע מרתך בעולם קומפיילרים: סיורו של סטודנט צעיר

עמדתי בפני אתגר עצום. פרויקט נגמר בקורס מדעי המחשב, לא פחות. המשימה: בניית קומפיילר משלי. עולם קומפיילרים, תחום מורכב ומרתק, נראה בתחילת בלתי אפשרי. אבל אני, עידן הירש, סטודנט צעיר עם רצון עד ללמידה ולצמוח, לא היסטי. לפתח אל תוך האתגר וצעדי במסע מרתך רצוף גילויים, תובנות והתפתחויות אישיות.

המסע לא היה קל. התמודדתי עם אתגרים תיאורתיים ואלגוריתמיים רבים. הייתה צורך ללמוד כמויות עצומות של חומר מורכב, לפתח פתרונות יצירתיים לביעות מורכבות, ולהתמודד עם אתגרים תכוניים ומעשיים, כמו חלוקת קוד לעיליה, מימוש תאוריית מורכבות, פתרון באגים, ניהול פרויקט בסדר גודל ממשמעותי.

אבל לאור כל הדרכ, פיתחתי כלים רבים שעזרו לי להתמודד עם האתגרים יכולת מיידעה עצמית: גיליתי יכולת מרשימה ללמידה באופן עצמאי נושאים מורכבים, לחקור מקורות מידע שונים, ולהרכיב ידע תיאורטי נרחב. פתרון בעיות יצירתי: פיתחתי יכולת לנתח בעיות מורכבות, לפתח אלגוריתמים יעילים, ולמצוא פתרונות יצירתיים תוך שימוש מבני נתונים מתאימים. ניהול פרויקטים: צברתי ניסיון ניהול משמעותי, תוך תכנון, ניהול זמן, ועבודה לעיליה בסביבה מורכבת.

עולם מדעי המחשב, עולם מלא אפשרויות ויצירתיות, שימש מקור השראה עבורי. מצאתי השראה בדמותות מובילות בתחום, כמו אלן טיורינג, אבי מדעי המחשב, דניס ריצ', מפתח שפת C, ולינוס טורוואלדס, מפתח מערכת ההפעלה לינוקס.

הצלחתי להשיג את כל המטרות שהציבתי לעצמי: פיתוח אישי: פיתחתי את עצמי הנו מבחינה מקצועית והנו מבחינה אישית. רכשתי ידע נרחב, פיתחתי מיומנויות חשובות, וחיזקתי את ביטחונו העצמי. בניית קומפיילר: הצלחתי לבנות קומפיילר פועל, תוך שימוש באלגוריתמים ומבנה נתונים יעילים. שיתוף ידע: כתבתי ספר פרויקט מפורט וברור, המאפשר לקוראים להבין לעומק את התהילה של בניית קומפיילר. הפרויקט שלי מוגמה למסע אישי מרתך בעולם מדעי המחשב. עבדתי מדגישה את חשיבות הלמידה העצמית, ההתמדה, והיצירתיות. הוכחת שעם רצון עד, עבודה קשה, והתמדה, ניתן להגשים חלומות ולפתח יכולות יוצאות דופן.

אבל זה לא נגמר פה.

המסע שלי רק התחליל. אני נרגש להמשיך וללמוד, לחזור ולפתח את עצמי בתחום מדעי המחשב. אני יודע שיש לי עוד הרבה מה ללמידה, ויש לי הרבה מטרות שאני רוצה להגשים. אני בטוח שאמשיך להתמודד עם אתגרים, לפתח פתרונות יצירתיים, ולצמוח כאדם וכמקצוען.

העתיד נראה מרתך.

במידה והיה לי זמן נוסף בפרויקט היהתי רוצה לבצע מספר דברים

אופטימיזציה של קוד:

אקדיש זמן רב לאופטימיזציה של קוד המקור של הקומפיאר. אלמד טכניקות שונות לאופטימיזציה, כגון ניתוח זרימה של נתונים, הסרת קוד מיותר, וחיזוק קוד. אני מאמין שתוצאות אופטימיזציה אלו ישפרו משמעותית את הביצועים של הקומפיאר, הן מבחינת מהירות ההידור והן מבחינת יעילות השימוש בזיכרון.

תמייה בשפות תכנות נוספות:

ארחיב את הקומפיאר כך שייחזור בשפות תכנות נוספות מלבד השפה המקורית ש תמכת | בה. זה יהיה תהליך מأتגר ומלמד, שנדרש ממני להבין לעומק את ההבדלים הסינטקטיים והסמנטיים בין שפות התכנות השונות.

אני מאמין שתהיליך זה יעניק לי ידע רב יותר בתחום שפות התכנות ויאפשר לי לפתח קומפיאר גמיש ורב-תכליתי יותר.

פיתוח ממשק משתמש גרפי:

בנוסף, אפתח ממשק משתמש גרפי (GUI) עבור הקומפיאר. ממשק המשמש לאפשר למשתמשים להזין קוד מקור בצורה נוחה וקלה יותר, לראות את תוצאות ההידור, ולקבל מידע נוסף על תהליכי ההידור. אני מאמין שימוש ממשק משתמש גרפי יהפוך את הפרויקט שלי לנגיש וקל לשימוש עבור משתמשים אחרים, ויאפשר להם להבין טוב יותר את תהליכי ההידור.

השתתפות בפרויקטים Open-Source:

אני מרגיש צורך לחלוק את הידע והניסיון שרכשתי עם הקהילה. לכן, אצטרף לפרויקטים Open-Source בתחום הקומפיארים. אשתף קוד לפרויקטים קיימים, וסייע למפתחים אחרים לפתור בעיות טכניות. אני מאמין שהשתתפות בפרויקטים אלו תאפשר לי ללמידה מפתחים מנוסים אחרים, לשפר את כישורי העבודה הצעות שלי, ולהתרום לקהילה התוכנה.

סיכום:

הרבתה הפרויקט שלי תרמה רבות למידה שלי ולצמיחה שלי כאיש מקצועי. למדתי טכניקות חדשות, פיתחת מיומנויות חשובות, והרחבתי את הידע שלי בתחום מדעי המחשב. יתרה מזאת,

נהניתי מאד מהתהליך ומהאתגרים שהציב בפנוי. אני בטוח שניסיון זה ישמש אותי רבות בעתיד, הנו בלימודים האקדמיים שלי והן בקריירה המקצועית שלי.

ביבליוגרפיה

בפרק זה אפרט את מקורות המידע שלו

מקור 1: [ספר בנושא קומפיילרים](#)

מקור 2: [ספר של האוניברסיטה הפתוחה בנושא אוטומטים](#)

מקור 3: [ספר אינטראקטיב על BNF](#)

מקור 4: [סרטוני יוטיוב](#)

מקור 5: [ניטהאב](#)

מקור 6: [סדרת סרטונים של נברת הודית מיזטיב](#)

מקור 7: [אתר cpp לפקודות](#)

קוד הפרויקט

הרחבת על כל חלק הפרויקט

Dfa

מחלקה האחראית על תפעול וכתיבת מוכנת המ מצבים

```

1 #pragma once
2 #include <iostream>
3 #include <fstream>
4
5 #define CHARS_NUM 128
6 #define NUM_OF_STATES 128
7
8 class DFA
9 {
10 private:
11     int mat[NUM_OF_STATES][CHARS_NUM];
12
13 public:
14     // DFA handle
15     DFA();
16
17     /// @brief Function to make the DFA
18     void makeDFA();
19
20     /// @brief Function to add tokens to the DFA
21     /// @param token
22     void addTokens(const std::string &token);
23
24     // file handle
25
26     /// @brief Function to write the DFA to a file
27     /// @param path
28     /// @return bool value to indicate the success of the operation
29     bool writeToFile(const std::string &path);
30
31     /// @brief Function to read the DFA from a file
32     /// @param path
33     /// @return bool value to indicate the success of the operation
34     bool readFromFile(const std::string &path);
35
36     /// @brief Function to get the DFA matrix
37     /// @return int(*)[CHARS_NUM]
38     int (*getMat())[CHARS_NUM] { return this->mat; }
39 };
40

```

```

1 #include "../headers/DFA.hpp"
2 #include <ctype.h>
3 #include <string>
4 #define MAT_PATH "./DFA/dfa"
5
6 static int freeState = 1;
7
8 DFA::DFA()
9 {
10     makeDFA();
11     writeToFile(MAT_PATH);
12 }
13
14 void DFA::addTokens(const std::string &token)
15 {
16     int i = 0;
17     int currState = 0;
18
19     if (freeState < 2)
20     {
21
22         // Give all letters state of 1 that is identifier
23         for (int row = 0; row < NUM_OF_STATES; row++)
24         {
25             mat[row]['_'] = freeState;
26
27             for(int col = 'a'; col < 'z'; col++)
28             {
29                 mat[row][col] = freeState;
30             }
31             for (int col = 'A'; col < 'Z'; col++)
32             {
33                 mat[row][col] = freeState;
34             }
35         }
36     }
37     freeState++;
38
39
40     //give all numbers state of 2
41     for (int row = 0; row < NUM_OF_STATES; row++)
42     {
43         for(int col = '0'; col < '9'; col++)
44         {
45             mat[row][col] = freeState;
46         }
47     }
48     freeState++;
49 }
50
51 }
52

```

```

52 // Checking if the received token is new
53 // or a prefix of older token
54 while (this->mat[currState][token[i]] != -1 && this->mat[currState][token[i]] != 1)
55 {
56     currState = mat[currState][token[i]];
57     i++;
58 }
59
60
61 // Making sure each state will point to the next
62 // state in the matrix(state machine)
63 for (; i < token.size(); i++)
64 {
65     mat[currState][token[i]] = freeState;
66
67     //for debugging: print all keywords and their state
68     std::cout << token << ":" << freeState << std::endl;
69
70     currState = freeState;
71     freeState++;
72 }
73
74 //if state is not a keyword, turn all states in final state to -1
75 if(!isalpha(token[i-1]))
76 {
77     for(int j = 0; j < CHARS_NUM; j++)
78     {
79         mat[currState][j] = -1;
80     }
81 }
82
83 freeState++;
84 }
85
86 bool DFA::writeToFile(const std::string &path)
87 {
88     std::ofstream file;
89
90     file.open(path);
91
92     if (!file.is_open())
93     {
94         std::cerr << "Error opening file" << std::endl;
95         exit(1);
96     }
97
98     for (int i = 0; i < NUM_OF_STATES; i++)
99     {
100         for (int j = 0; j < CHARS_NUM; j++)
101         {
102             file << this->mat[i][j] << ',';
103         }
104         file << "\n";
105     }
106     return true;
107 }
108

```

```
109  bool DFA::readFromFile(const std::string &path)
110 {
111     std::ifstream file;
112     int i = 0, j = 0;
113
114     if (!file.is_open())
115         file.open(path);
116
117     if (!file.is_open())
118     {
119         std::cerr << "failed creating dfa file..." << std::endl;
120         return false;
121     }
122
123     while (file)
124     {
125         char ch = file.get();
126         if (ch != '\n' && ch != ',')
127         {
128             for (i = 0; i < NUM_OF_STATES; i++)
129             {
130                 for (j = 0; j < CHARS_NUM; j++)
131                 {
132                     this->mat[i][j] = ch;
133                 }
134             }
135         }
136     }
137
138     return true;
139 }
```

```

140
141     void DFA::makeDFA()
142     {
143         for (size_t i = 0; i < NUM_OF_STATES; i++)
144             for (int j = 0; j < CHARs_NUM; j++)
145                 this->mat[i][j] = -1;
146
147         // Conditions
148         addTokens("if");
149         addTokens("else");
150
151         // Loops
152         addTokens("while");
153         addTokens("for");
154
155         // Variables
156         addTokens("char");
157         addTokens("int");
158         addTokens("bool");
159         addTokens("void");
160
161         // Binop
162         addTokens("/");
163         addTokens("+");
164         addTokens("-");
165         addTokens("*");
166         addTokens("&");
167         addTokens("%");
168
169         // Logical gates
170         addTokens("&&");
171         addTokens("||");
172         addTokens("!=");
173         addTokens("==");
174         addTokens("<");
175         addTokens(">");
176         addTokens("<=");
177         addTokens(">=");
178         addTokens("!=");
179
180         // Special
181         addTokens(":=");
182         addTokens(";");
183         addTokens("\\" );
184         addTokens("\\\"");
185         addTokens(",");
186         addTokens("(" );
187         addTokens(")" );
188         addTokens("{");
189         addTokens("}" );
190
191         //return statement
192         addTokens("return");
193
194         //true and false
195         addTokens("true");
196         addTokens("false");
197
198         //for function
199         addTokens("give");
200
201         //for printing
202         addTokens("scream");
203
204         //end of file
205         addTokens("$");
206     }
207 }
```

Lexer

```

1 #pragma once
2 #include <iostream>
3 #include <fstream>
4 #include <stdlib.h>
5 #include <list>
6 #include "Token.hpp"
7 #include "DFA.hpp"
8
9 #define COMMENT_SIGN '#'
10#define QUOTES '\"'
11#define DOUBLE_QUOTES '\"'
12
13 using namespace std;
14
15
16 class Lexer
17 {
18
19 private:
20     std::string _lastCollectedValue; // Last collected value
21     list<Token> tokenList;          // List of tokens
22     std::string path;              // Path to the file
23     DFA *dfa;                     // Deterministic Finite Automata
24
25 public:
26     // Constructor
27     Lexer(std::string path);
28
29     // function to read DFA from file
30     void readFromFile();
31
32     /// @brief Function to return the string of the token
33     /// @param code
34     /// @return
35     std::string returnTokenString(int code);
36
37     /// @brief Function to analyze the current word
38     /// @param currWord
39     /// @param loc line of code
40     /// @return Token
41     Token analyze(std::string currWord, int loc);
42
43     /// @brief Function to get the token list
44     /// @return list<Token>
45     list<Token> getTokenList() { return this->tokenList; }
46 };
47

```

```

1 #include "../headers/Lexer.hpp"
2
3 int LOC = 1;
4
5 Lexer::Lexer(std::string path)
6 {
7     this->path = path;
8     this->dfa = new DFA();
9 }
10
11 void Lexer::readFromFile()
12 {
13     int i = 0;
14     Token tok;
15     std::string skip;
16     std::string data, line;
17     ifstream file;
18
19     if (!file.is_open())
20         file.open(this->path);
21
22     if (!file.is_open())
23     {
24         std::cout << "error opening code file" << std::endl;
25         exit(1);
26     }
27
28     while (std::getline(file, line)) {
29         if(line[0] != '~')
30         {
31             data += line + "\n"; // Append each line to the string
32         }
33         else
34         {
35             LOC++;
36         }
37     }
38
39     // Analyizing each word received from the file
40     // word is a sequence of characters which ends with a space
41     for (i = 0; i < data.size();)
42     {
43         if(data[i] == '\n')
44         {
45             LOC++;
46             data[i] = '\0';
47         }
48         tok = analyze(data.substr(i), LOC);
49         i += tok.token.size();
50     }
51
52     //Printing results for debugging
53     for (Token token : this->tokenList)
54     {
55         std::cout << token.token << " ";
56         std::cout << returnTokenString(token.type) << " ";
57         std::cout << to_string(token.type) << std::endl;
58     }
59 }
```

```

61 Token Lexer::analyze(std::string data, int loc)
62 {
63     int i = 0, tempNext = 0;
64     std::string result = "";
65     Token *t = new Token();
66
67     int(*mat)[CHARS_NUM] = this->dfa->getMat();
68
69     int state = mat[0][data[0]];
70
71     // Getting the token by stat from the matrix
72     // if no matching state and final state is positive identify as identifier
73     // else identify as UNDEFINED
74     while(state != -1)
75     {
76         if(data[i] == '\'' || data[i] == '\"')
77         {
78             result += data[i];
79             i++;
80
81             while(data[i] != '\'' && data[i] != '\"' && data[i] != '\0')
82             {
83                 result += data[i];
84                 i++;
85             }
86
87             if(data[i] != '\'' && data[i] != '\"')
88             {
89                 std::cout << "Lexing error on line: " << loc << " missing closing quote" << std::endl;
90                 t->type = UNDEFINED;
91                 t->token = ' ';
92                 t->loc = loc;
93                 tokenList.push_back(*t);
94                 return (*t);
95             }
96             result += data[i];
97             i++;
98
99             // the type will be literal
100            t->type = ID_LITERAL;
101            t->token = result;
102            t->loc = loc;
103            tokenList.push_back(*t);
104            return (*t);
105        }
106        // the type will be the most recent valid state
107        t->type = state;
108        result += data[i];
109
110        i++;
111        state = mat[state][data[i]];
112    }
113
114    t->token = result;
115
116    if(t->token.size() > 0)
117    {
118        t->loc = loc;
119        tokenList.push_back(*t);
120    }
121    else{
122        t->token += ' ';
123    }
124
125    return (*t);
126 }
127 }
```

```

130 std::string Lexer::returnTokenString(int code)
1/1
172
173     case BIGEER_EQUAL:
174         return "BIGEER_EQUAL";
175
176     case SMALLER_EQUAL:
177         return "SMALLER_EQUAL";
178
179
180     case BINOP_PLUS:
181         return "BINOP_PLUS";
182
183
184     case BINOP_DIV:
185         return "BINOP_DIV";
186
187
188     case BINOP_MINUS:
189         return "BINOP_MINUS";
190
191
192     case BINOP_MULT:
193         return "BINOP_MULT";
194
195
196
197     case AND_OP:
198         return "BINOP_AND";
199
200
201     case LOGIC_AND:
202         return "LOGIC_AND";
203
204
205     case LOGIC_OR:
206         return "LOGIC_OR";
207
208
209     case DEFINE_VAR:
210         return "DEFINE_VAR";
211
212
213     case LPARAN:
214         return "LEFT_PARAN";
215
216
217     case RPARAN:
218         return "RIGHT_PARAN";
219
220
221     case RBRACE:
222         return "RIGHT_BRACE";
223
224     case LBRACE:
225         return "LEFT_BRACE";
226
227     case ID_RETURN:
228         return "ID_RETURN";
229
230
231     case -1:
232         return "NO SUCH TOKEN";
233
234
235     default:
236         return "IDENTIFIER";
237
238 }

```

Parser

```

1 #pragma once
2 #include "Stack.hpp"
3 #include "Token.hpp"
4 #include "Production.hpp"
5 #include "types.hpp"
6 #include "RegisterEntry.hpp"
7 #include <vector>
8 #include <fstream>
9 #include <iostream>
10 #include <unordered_map>
11
12
13 enum actionTableKeys{
14     FOR_ACTION = 0,
15     LPARAN_ACTION,
16     SEMI_COLON_ACTION,
17     RPARAN_ACTION,
18     LBRACE_ACTION,
19     RBRACE_ACTION,
20     IF_ACTION,
21     ELSE_ACTION,
22     WHILE_ACTION,
23     IDENTIFIER_ACTION,
24     ASSIGNMENT_ACTION,
25     INT_ACTION,
26     BOOL_ACTION,
27     CHAR_ACTION,
28     VOID_ACTION,
29     NUMBER_ACTION,
30     LITERAL_ACTION,
31     EQUAL_ACTION,
32     NOTEQ_ACTION,
33     SMALLER_ACTION,
34     BIGGER_ACTION,
35     SMALLER_EQUAL_ACTION,
36     BIGGER_EQUAL_ACTION,
37     MINUS_ACTION,
38     PLUS_ACTION,
39     MULT_ACTION,
40     DIV_ACTION,
41     UNARY_PLUS_ACTION,
42     UNARY_MINUS_ACTION,
43     OR_ACTION,
44     AND_ACTION,
45     SHR_ACTION,
46     SHL_ACTION,
47     RETURN_ACTION,
48     GIVE_ACTION,
49     COMMA_ACTION,
50     TRUE_ACTION,
51     FALSE_ACTION,
52     PRINT_ACTION,
53     EOF_ACTION,
54 }typedef actionTableKeys;
55
56 const string terminal_words[] = {
57     "FOR ACTION", "LPARAN ACTION", "SEMI_COLON ACTION", "RPARAN ACTION", "LBRACE ACTION",
58     "RBRACE ACTION", "IF ACTION", "ELSE ACTION", "WHILE ACTION", "IDENTIFIER ACTION", "ASSIGNMENT ACTION",
59     "INT ACTION", "BOOL ACTION", "CHAR ACTION", "VOID ACTION", "NUMBER ACTION", "LITERAL", "EQUAL ACTION",
60     "NOTEQ ACTION", "SMALLER ACTION", "BIGGER ACTION", "SMALLER_EQUAL ACTION", "BIGGER_EQUAL ACTION",
61     "MINUS ACTION", "PLUS ACTION", "MULT ACTION", "DIV ACTION", "UNARY_PLUS ACTION", "UNARY_MINUS ACTION",
62     "OR ACTION", "AND ACTION", "SHR ACTION", "SHL ACTION", "RETURN ACTION", "GIVE ACTION", "COMMA ACTION",
63     "TRUE ACTION", "FALSE ACTION", "EOF ACTION"};
64

```

```

66 class ParseTree{
67 public:
68     ParseTree* root;
69     int value;
70     Token token;
71     std::vector<ParseTree> children;
72     types type;
73     RegisterEntry reg;
74     void setRoot(Token *root);
75     void setReg(RegisterEntry reg);
76     RegisterEntry getReg();
77
78     ParseTree()
79     {
80         this->root = NULL; // create a new token
81         this->children = {}; // assign an empty vector to children
82     }
83
84     ParseTree(Token *root)
85     {
86         this->root = new ParseTree; // create a new token
87         Token *newRoot = new Token(); // create a new token
88         newRoot->token = root->token; // set the token value
89         newRoot->type = root->type; // set the token type
90         newRoot->loc = root->loc; // set the line number
91         newRoot->type = root->type;
92         this->root->token = *newRoot;
93         this->children = {}; // children of the root
94     }
95
96 };
97
98 class Parser{
99 private:
100     Stack<ParseTree> parStack;           //stack to store the parse tree
101     Stack<int> tokenStack;               //stack to store the tokens
102     std::list<Token> inputList;          //list of tokens
103     std::vector<Production> productions; //list of productions
104     std::vector<std::vector<std::string>> actionGoTable; //action go table
105
106     //to convert between token id and action table keys we can use a hashtable which maps token id to action table keys
107     std::unordered_map<int, actionTableKeys> tokenActions;
108
109 public:
110     /// @brief Constructor
111     Parser();
112
113     /// @brief Function to set the rule list
114     void setRuleList();
115
116     /// @brief Function to set the input list
117     /// @param inputList
118     void setInputList(list<Token> inputList){this->inputList = inputList;}
119
120     /// @brief Function to read the productions from the file
121     void generateParseTables();
122
123     /// @brief Function to parse the input list
124     ParseTree parse();
125
126     /// @brief Function to print the parse tree
127     void makeMap();
128
129     /// @brief Function to print the parse tree
130     /// @param root
131     /// @param prefix
132     /// @param isLast
133     void printAST(ParseTree& root, string prefix, bool isLast);
134
135 };

```

```

1 #include "../headers/Parser.hpp"
2
3 Parser::Parser()
4 {
5     setRuleList();
6     generateParseTables();
7     makeMap();
8 }
9
10 void Parser::setRuleList()
11 {
12     //defining program
13     productions.push_back(Production(PROGRAM, {STATEMENT_LIST}));
14
15     //defining statement list
16     productions.push_back(Production(STATEMENT_LIST, {STATEMENT}));
17     productions.push_back(Production(STATEMENT_LIST, {STATEMENT_LIST, STATEMENT}));
18
19     //defining statements
20     productions.push_back(Production(STATEMENT, {IF_STATEMENT}));
21     productions.push_back(Production(STATEMENT, {WHILE_STATEMENT}));
22     productions.push_back(Production(STATEMENT, {ASSIGNMENT}));
23     productions.push_back(Production(STATEMENT, {DECLARATION_STATEMENT}));
24     productions.push_back(Production(STATEMENT, {FOR_STATEMENT}));
25     productions.push_back(Production(STATEMENT, {EXPRESSION}));
26     productions.push_back(Production(STATEMENT, {RETURN_STATEMENT}));
27     productions.push_back(Production(STATEMENT, {FUNCTION}));
28     productions.push_back(Production(STATEMENT, {PRINT_STATEMENT}));
29
30     //defining loops and conditions
31     productions.push_back(Production(IF_STATEMENT, {IF_ACTION, LPARAN_ACTION, ASSIGNMENT, SEMI_COLON_ACTION, CONDITION, SEMI_COLON_ACTION, EXPRESSION, RPARAN_ACTION, LBRACE_ACTION, STATEMENT_LIST, RBRACE_ACTION}));
32     productions.push_back(Production(IF_STATEMENT, {IF_ACTION, LPARAN_ACTION, CONDITION, RPARAN_ACTION, LBRACE_ACTION, STATEMENT_LIST, RBRACE_ACTION}));
33     productions.push_back(Production(IF_STATEMENT, {IF_ACTION, LPARAN_ACTION, CONDITION, RPARAN_ACTION, ELSE_ACTION, LBRACE_ACTION, STATEMENT_LIST, RBRACE_ACTION}));
34     productions.push_back(Production(WHILE_STATEMENT, {WHILE_ACTION, LPARAN_ACTION, CONDITION, RPARAN_ACTION, LBRACE_ACTION, STATEMENT_LIST, RBRACE_ACTION}));
35
36     //defining declarations
37     productions.push_back(Production(DECLARATION_STATEMENT, {TYPE, IDENTIFIER_ACTION, ASSIGNMENT_ACTION, EXPRESSION}));
38     productions.push_back(Production(DECLARATION_STATEMENT, {TYPE, IDENTIFIER_ACTION}));
39
40     //defining assignments
41     productions.push_back(Production(ASSIGNMENT, {IDENTIFIER_ACTION, ASSIGNMENT_ACTION, EXPRESSION}));
42
43     //defining types
44     productions.push_back(Production(TYPE, {INT_ACTION}));
45     productions.push_back(Production(TYPE, {CHAR_ACTION}));
46     productions.push_back(Production(TYPE, {BOOL_ACTION}));
47     productions.push_back(Production(VOID_TYPE, {VOID_ACTION}));
48
49     //defining expressions
50     productions.push_back(Production(EXPRESSION, {FACTOR}));
51     productions.push_back(Production(EXPRESSION, {EXPRESSION, AROP, FACTOR}));
52     productions.push_back(Production(EXPRESSION, {EXPRESSION, BITOP, FACTOR}));
53     productions.push_back(Production(EXPRESSION, {IDENTIFIER_ACTION, UNARY_EXPRESSION, COMMA_TOKEN}));
54
55     //defining factors
56     productions.push_back(Production(FACTOR, {NUMBER_ACTION}));
57     productions.push_back(Production(FACTOR, {IDENTIFIER_ACTION}));
58     productions.push_back(Production(FACTOR, {LITERAL_ACTION}));
59     productions.push_back(Production(FACTOR, {LPARAN_ACTION, EXPRESSION, RPARAN_ACTION}));
60     productions.push_back(Production(FACTOR, {BOOL_TYPE}));
61
62     //defining conditions
63     productions.push_back(Production(CONDITION, {EXPRESSION, RELATIONAL_OPERATOR, EXPRESSION}));
64
65     //defining relational operators
66     productions.push_back(Production(RELATIONAL_OPERATOR, {EQUAL_ACTION}));
67     productions.push_back(Production(RELATIONAL_OPERATOR, {NOTEQ_ACTION}));
68     productions.push_back(Production(RELATIONAL_OPERATOR, {SMALLER_ACTION}));
69     productions.push_back(Production(RELATIONAL_OPERATOR, {BIGGER_ACTION}));
70     productions.push_back(Production(RELATIONAL_OPERATOR, {SMALLER_EQUAL_ACTION}));
71     productions.push_back(Production(RELATIONAL_OPERATOR, {BIGGER_EQUAL_ACTION}));
72
73     //defining arithmetic operators
74     productions.push_back(Production(AROP, {MINUS_ACTION}));
75     productions.push_back(Production(AROP, {PLUS_ACTION}));
76     productions.push_back(Production(AROP, {MULT_ACTION}));
77     productions.push_back(Production(AROP, {DIV_ACTION}));
78
79     //defining unary operators
80     productions.push_back(Production(UNARY_EXPRESSION, {PLUS_ACTION, PLUS_ACTION}));
81     productions.push_back(Production(UNARY_EXPRESSION, {MINUS_ACTION, MINUS_ACTION}));
82
83     //defining bitwise operators
84     productions.push_back(Production(BITOP, {OR_ACTION}));
85     productions.push_back(Production(BITOP, {AND_ACTION}));
86     productions.push_back(Production(BITOP, {SHR_ACTION}));
87     productions.push_back(Production(BITOP, {SHL_ACTION}));
88
89     //defining return statement
90     productions.push_back(Production(RETURN_STATEMENT, {RETURN_ACTION, FACTOR}));
91
92     //defining functions
93     productions.push_back(Production(FUNCTION, {GIVE_ACTION, TYPE, IDENTIFIER_ACTION, LPARAN_ACTION, INPUT_VAR_LIST,
94     | RPARAN_ACTION, LBRACE_ACTION, STATEMENT_LIST, RETURN_STATEMENT, RBRACE_ACTION}));
95     productions.push_back(Production(FUNCTION, {GIVE_ACTION, VOID_TYPE, IDENTIFIER_ACTION, LPARAN_ACTION, RPARAN_ACTION,
96     | LBRACE_ACTION, STATEMENT_LIST, RBRACE_ACTION}));
97
98     //defining input var list
99     productions.push_back(Production(INPUT_VAR_LIST, {INPUT_VAR}));
100    productions.push_back(Production(INPUT_VAR_LIST, {INPUT_VAR_LIST, COMMA_ACTION, INPUT_VAR}));
101    productions.push_back(Production(INPUT_VAR_LIST, {}));
102
103    //defining input var
104    productions.push_back(Production(INPUT_VAR, {TYPE, IDENTIFIER_ACTION}));
105
106    productions.push_back(Production(BOOL_TYPE, {TRUE_ACTION}));
107    productions.push_back(Production(BOOL_TYPE, {FALSE_ACTION}));
108
109    //define print statement
110    productions.push_back(Production(PRINT_STATEMENT, {PRINT_ACTION, LPARAN_ACTION, EXPRESSION, RPARAN_ACTION}));
111 }
112
113 }
```

```

113 // Function to map token types to action table keys
114 // input: none
115 // output: none
116 void Parser::makeMap()
117 {
118     //mapping token types to action table keys
119     //these lexer tokens are mapped to their respective action table keys
120
121     //factors
122     this->tokenActions[ID_IDENTIFIER] = actionTableKeys::IDENTIFIER_ACTION;
123     this->tokenActions[ID_NUMBER] = actionTableKeys::NUMBER_ACTION;
124
125     //special characters
126     this->tokenActions[LPARAN] = actionTableKeys::LPARAN_ACTION;
127     this->tokenActions[RPARAN] = actionTableKeys::RPARAN_ACTION;
128     this->tokenActions[LBRACE] = actionTableKeys::LBRACE_ACTION;
129     this->tokenActions[RBRACE] = actionTableKeys::RBRACE_ACTION;
130     this->tokenActions[SEMI_COLON] = actionTableKeys::SEMI_COLON_ACTION;
131     this->tokenActions[COMMA_TOKEN] = actionTableKeys::COMMA_ACTION;
132
133     //keywords
134     this->tokenActions[ID_IF_CONDITION] = actionTableKeys::IF_ACTION;
135     this->tokenActions[ID_ELSE_CONDITION] = actionTableKeys::ELSE_ACTION;
136     this->tokenActions[ID_WHILE_LOOP] = actionTableKeys::WHILE_ACTION;
137     this->tokenActions[ID_FOR_LOOP] = actionTableKeys::FOR_ACTION;
138
139     //types
140     this->tokenActions[ID_CHAR] = actionTableKeys::CHAR_ACTION;
141     this->tokenActions[ID_INT] = actionTableKeys::INT_ACTION;
142     this->tokenActions[ID_BOOL] = actionTableKeys::BOOL_ACTION;
143     this->tokenActions[ID_VOID] = actionTableKeys::VOID_ACTION;
144     this->tokenActions[ID_TRUE] = actionTableKeys::TRUE_ACTION;
145     this->tokenActions[ID_FALSE] = actionTableKeys::TRUE_ACTION;
146     this->tokenActions[ID_LITERAL] = actionTableKeys::LITERAL_ACTION;
147
148     //operators
149     this->tokenActions[BINOP_DIV] = actionTableKeys::DIV_ACTION;
150     this->tokenActions[BINOP_PLUS] = actionTableKeys::PLUS_ACTION;
151     this->tokenActions[BINOP_MINUS] = actionTableKeys::MINUS_ACTION;
152     this->tokenActions[BINOP_MULT] = actionTableKeys::MULT_ACTION;
153
154     //logical operators
155     this->tokenActions[AND_OP] = actionTableKeys::AND_ACTION;
156     this->tokenActions[EQUAL] = actionTableKeys::EQUAL_ACTION;
157     this->tokenActions[SMALLER] = actionTableKeys::SMALLER_ACTION;
158     this->tokenActions[BIGEER] = actionTableKeys::BIGGER_ACTION;
159     this->tokenActions[SMALLER_EQUAL] = actionTableKeys::SMALLER_EQUAL_ACTION;
160     this->tokenActions[BIGEER_EQUAL] = actionTableKeys::BIGGER_EQUAL_ACTION;
161     this->tokenActions[NOT_EQ] = actionTableKeys::NOTEQ_ACTION;
162
163
164     this->tokenActions[DEFINE_VAR] = actionTableKeys::ASSIGNMENT_ACTION;
165     this->tokenActions[ID_RETURN] = actionTableKeys::RETURN_ACTION;
166     this->tokenActions[ID_GIVE] = actionTableKeys::GIVE_ACTION;
167     this->tokenActions[ID_PRINT] = actionTableKeys::PRINT_ACTION;
168     this->tokenActions[ID_EOF] = actionTableKeys::EOF_ACTION;
169
170 }
171

```

```

171 // Function to generate parse tables from csv file
172 // input: csv file with action and goto table
173 // output: 2d vector of strings
174 void Parser::generateParseTables()
175 {
176     std::string line;           /* string to hold each line */
177     std::vector<std::string>> array;    /* vector of vector<int> for 2d array */
178     ifstream f("ParserGen/parser_generator.csv"); /* open file */
179
180     while (getline (f, line)) {      /* read each line */
181         std::string val;           /* string to hold value */
182         std::vector<string> row;    /* vector for row of values */
183         std::stringstream s (line); /* stringstream to parse csv */
184         while (getline (s, val, ',')) /* for each value */
185             row.push_back (val); /* add to row */
186         array.push_back (row); /* add row to array */
187     }
188     this->actionGoTable = array;
189     f.close();
190 }
191
192 // Function to parse the input list and generate the AST
193 // input: none
194 // output: ParseTree node that is the root of the AST
195 ParseTree Parser::parse()
196 {
197     int state, tokenCode, nextState, productionNum, nonTerminal;
198     std::string action, nextStateStr;
199     Production production;
200     Token token;
201
202     //pushing initial state to stack
203     this->tokenStack.push(0);
204     inputList.push_back(Token{"$", ID_EOF});
205
206     while (!inputList.empty())
207     {
208         token = inputList.front();
209         state = this->tokenStack.peek();
210         tokenCode = this->tokenActions[token.type];
211         action = this->actionGoTable[state][tokenCode];
212
213         //print state, token, action for debugging
214         cout << "state: " << state << " token: " << tokenCode << " action: " << action << endl;
215
216         if(action == "acc")
217         {
218             cout << "accepted" << endl;
219             ParseTree prog;
220             prog.value = PROGRAM;
221             prog.token = Token("PROGRAM", PROGRAM);
222             prog.children.push_back(parStack.pop());
223             return prog;
224         }
225         else if(action[0] == 's')
226         {
227             nextState = stoi(action.substr(1));
228             this->tokenStack.push(nextState);
229             inputList.pop_front();
230
231             ParseTree node;
232             node.value = this->tokenActions[token.type];
233             node.token = token;
234             parStack.push(node);
235         }
236     }

```

```

235     parStack.push(mode);
236 }
237 else if(action[0] == 'r')
238 {
239     productionNum = stoi(action.substr(1));
240     auto it = std::next(productions.begin(), productionNum);
241     production = *it;
242
243     Token *token = new Token();
244     token->token = non_terminal_words[production.left - NT_OFFSET];
245     token->type = production.left;
246
247     ParseTree *ast = new ParseTree(token);
248     ast->token = *token;
249
250     //poping from stack and adding to children
251     for(int i = 0; i < production.right.size(); i++)
252     {
253         this->tokenStack.pop();
254
255         ParseTree child = parStack.pop();
256         child.root = ast;
257         ast->children.insert(ast->children.begin(), child);
258     }
259     //pushing the node to node stack
260     parStack.push(*ast);
261
262     //getting next state from goto table and pushing it to stack
263     state = this->tokenStack.peek();
264     nonTerminal = production.left;
265     nextStateStr = this->actionGoTable[state][nonTerminal];
266     this->tokenStack.push(stoi(nextStateStr));
267 }
268 else
269 {
270     cout << "Parsing error in line " << token.loc << " - token:\\" " << token.token << "\" is not valid in this position" << endl;
271     return ParseTree();
272 }
273 }
274
275 }
276
277 void ParseTree::setReg(RegisterEntry reg)
278 {
279     this->reg = reg;
280 }
281
282 RegisterEntry ParseTree::getReg()
283 {
284     return this->reg;
285 }
286
287
288 // Function to print the tree in level order
289 // input: ParseTree node, string prefix, bool isLast
290 // output: none
291 void Parser::printAST(ParseTree& root, string prefix, bool isLast) {
292     cout << prefix << (isLast ? "└──" : "├──") << root.token.token << endl;
293     for (int i = 0; i < root.children.size(); i++) {
294         printAST(root.children[i], prefix + (isLast ? "    " : "|   "), i == root.children.size() - 1);
295     }
296 }
297
298

```

Production

```

1 #pragma once
2 #include <vector>
3 #include <iostream>
4 #include "Lexer.hpp"
5
6 #define NT_OFFSET 40
7
8 enum non_terminal{
9     PROGRAM = NT_OFFSET,|
10    STATEMENT_LIST,
11    STATEMENT,
12    FOR_STATEMENT,
13    IF_STATEMENT,
14    WHILE_STATEMENT,
15    DECELERATION_STATEMENT,
16    ASSIGNMENT,
17    TYPE,
18    VOID_TYPE,
19    EXPRESSION,
20    FACTOR,
21    CONDITION,
22    RELATIONAL_OPERATOR,
23    AROP,
24    UNARY_EXPRESSION,
25    BITOP,
26    RETURN_STATEMENT,
27    FUNCTION,
28    INPUT_VAR_LIST,
29    INPUT_VAR,
30    BOOL_TYPE,
31    PRINT_STATEMENT,
32 };
33
34 const string non_terminal_words[] = {
35     "PROGRAM", "STATEMENT_LIST", "STATEMENT", "FOR_STATEMENT", "IF_STATEMENT",
36     "WHILE_STATEMENT", "DECELERATION_STATEMENT", "ASSIGNMENT", "TYPE", "VOID_TYPE", "EXPRESSION",
37     "FACTOR", "CONDITION", "RELATIONAL_OPERATOR", "AROP", "UNARY_EXPRESSION", "BITOP",
38     "RETURN_STATEMENT", "FUNCTION", "INPUT_VAR_LIST", "INPUT_VAR", "BOOL_TYPE", "PRINT_STATEMENT"};
39
40 //example: production for type
41 // left-> Token(NON-TERMINAL) right -> (Token(ID_INT))
42 // left-> Token(NON-TERMINAL) right -> (Token(ID_CHAR))
43 // left-> Token(NON-TERMINAL) right -> (Token(ID_BOOL))
44
45 class Production{
46 public:
47     Production();           //default constructor
48     non_terminal left;    //non-terminal
49     std::vector<int> right; //vector of terminals
50
51     /// @brief Constructor for Production
52     /// @param left
53     /// @param right
54     Production(non_terminal left, std::vector<int> right){
55         this->left = left;
56         this->right = right;
57     }
58 };

```

semanticAnalysis

```

1 #pragma once
2 #include "Symbol.hpp"
3 #include <unordered_set>
4
5 class SemanticAnalysis
6 {
7 private:
8     unordered_set<Symbol, Symbol::HashFunction> symbolTable; // Symbol table
9     ParseTree* parseTree; // Parse tree
10
11    unordered_map<string, int> mapOfTypes = {
12        {"int", INT},
13        {"bool", BOOL},
14        {"char", CHAR},};
15
16 public:
17     /// @brief Constructor
18     /// @param parseTree
19     SemanticAnalysis(ParseTree* parseTree);
20
21     /// @brief Function to create the symbol table
22     /// @param parseTree
23     /// @param scope
24     void createSymbolTable(ParseTree* parseTree, int scope);
25
26     /// @brief getter for the symbol table
27     /// @return std::unordered_set<Symbol, Symbol::HashFunction>
28     unordered_set<Symbol, Symbol::HashFunction> getSymbolTable();
29
30     /// @brief Functions to perform semantic analysis
31     /// @param tree
32     /// @param scope
33     /// @return ParseTree*
34     ParseTree* semantic();
35     ParseTree* semanticHelper(ParseTree *tree, int scope);
36
37     /// @brief Function to get the type of the symbol
38     /// @param type
39     /// @return
40     string getSymbolType(int type);
41
42     /// @brief Function to check if the token is a comparison operator
43     /// @param token
44     /// @return bool value to indicate if the token is a comparison operator
45     bool isCmpOp(string token);
46
47     /// @brief Function to identify the type of the token
48     /// @param token
49     /// @return type of the token
50     int identifyType(Token *token);
51
52     /// @brief Function to print the symbol table
53     void printSymbolTable();
54 };

```

```

57 ParseTree *SemanticAnalysis::semantic(ParseTree *tree, int scope)
58 {
59     if (tree == NULL) // Empty tree
60     {
61         return tree;
62     }
63     for (int i = 0; i < tree->children.size(); i++) // Iterate through the children of the tree
64     {
65         if (tree->children.at(i).token.type == LBRACE) // Increment scope when entering a new block
66         {
67             scope++;
68             semantic(&tree->children.at(i), scope); // Recursively perform symantic analysis on the children
69         }
70     }
71     Token* token = &tree->token;
72
73     if (token->type == ID_IDENTIFIER && tree->root->token.type != FUNCTION && tree->root->token.type)
74     {
75         auto symbol = symbolTable.find(Symbol(token->token, 0, 0, 0)); // Find the identifier in the symbol table
76         if (symbol == symbolTable.end() || symbol->scope > scope || symbol->line > token->loc) // Check if the identifier is undeclared
77         {
78             std::cerr << "Undeclared variable " + token->token + " in line: " << token->loc << std::endl;
79             this->errors = true;
80         }
81         else{
82             token->type = symbol->type; // Set the type code of the identifier
83         }
84     }
85     else if (token->type == LITERAL_ACTION)
86     {
87         token->type = identifyType(token); // Identify the type of the literal
88     }
89     else if (token->type == FACTOR || token->type == EXPRESSION)
90     {
91         if (tree->children.size() == 1 || token->type == EXPRESSION) // Set the type code of the term or expression
92             token->type = tree->children.at(0).token.type;
93         else
94             token->type = tree->children.at(1).token.type; // Set the type code of the term or expression
95     }
96     else if (token->type == ASSIGNMENT)
97     {
98         if(symbolTable.find(Symbol(tree->children.at(0).token.token, 0,0,0)) == symbolTable.end()) // Check if the identifier is declared
99         {
100             this->errors = true;
101         }
102         else
103         {
104             int varType = symbolTable.find(Symbol(tree->children.at(0).token.token, 0,0,0))->type;
105             if (varType != tree->children.at(2).token.type) // Check for type errors in the assignment
106             {
107                 std::cerr << "Invalid Type Error: Cannot assign " +
108                 getSymbolType(tree->children.at(2).token.type) + " to identifier " + tree->children.at(0).token.token + " (" +
109                 getSymbolType(varType) + ")" + " in line: " << tree->children.at(0).token.loc << std::endl;
110                 this->errors = true;
111             }
112         }
113     }
114     else if (token->type == DECLARATION_STATEMENT)
115     {
116         if (tree->children.size() == 4 && (mapOfTypes[tree->children.at(0).children.at(0).token.token] != tree->children.at(3).token.type)) // Check for type error
117         {
118             std::cerr << "Invalid Type Error: Cannot assign " +
119             getSymbolType(tree->children.at(3).token.type) + " to identifier " + tree->children.at(1).token.token + " (" <<
120             getSymbolType(mapOfTypes[tree->children.at(0).children.at(0).token.token]) << ") in line: " << tree->children.at(1).token.loc << std::endl;
121             this->errors = true;
122         }
123         tree->children.at(1).token.type = mapOfTypes[tree->children.at(0).children.at(0).token.token];
124     }
125     else if(token->type == FUNCTION)
126     {
127         if(tree->children.size() == 10) // Check if the function has a return statement
128         {
129             if(tree->children.at(8).children.size() < 2) // Check if the function has a return type
130             {
131                 std::cerr << "Function " + tree->children.at(0).token.token + " has no return type in line: " << tree->children.at(0).token.loc << std::endl;
132                 this->errors = true;
133             }
134         }

```

```

1 #include "../headers/SemanticAnalysis.hpp"
2
3 SemanticAnalysis::SemanticAnalysis(ParseTree* root)
4 {
5     this->parseTree = root;
6     this->symbolTable = unordered_set<Symbol, Symbol::HashFunction>();
7     this->createSymbolTable(parseTree, 0);
8 }
9
10
11 void SemanticAnalysis::createSymbolTable(ParseTree *tree, int scope)
12 {
13     if (tree == NULL) // Empty tree
14     {
15         return;
16     }
17     // Iterate through the children of the tree
18     for (int i = 0; i < tree->children.size(); i++)
19     {
20         if (tree->children.at(i).token.type == LBRACE) // Increment scope when entering a new block
21             scope++;
22
23         createSymbolTable(&tree->children.at(i), scope); // Recursively create symbol table for the children
24     }
25     Token token = tree->token; // Get the token of the tree
26     if (token.type == DECLARATION_STATEMENT)
27     {
28         string type = tree->children.at(0).children.at(0).token.token; // Get the type of the variable
29         int typeCode = mapOfTypes[type]; // Get the type code of the variable
30         string identifier = tree->children.at(1).token.token; // Get the identifier of the variable
31         if (symbolTable.find(Symbol(identifier, 0, 0, 0)) != symbolTable.end()) // Check if the variable is already declared
32         {
33             std::cerr << "Variable " + identifier + " already declared, line: " << tree->children.at(0).token.loc << std::endl;
34         }
35         symbolTable.insert(Symbol(identifier, typeCode, scope, tree->children.at(1).token.loc)); // Insert the variable into the symbol table
36     }
37 }
38
39 void SemanticAnalysis::printSymbolTable()
40 {
41     cout << "Symbol Table" << endl;
42     cout << "-----" << endl;
43     cout << "Identifier\tType\tScope" << endl;
44     for (auto symbol : this->symbolTable)
45     {
46         cout << symbol.name << "\t" << symbol.type << "\t" << symbol.scope << endl;
47     }
48 }
49
50 unordered_set<Symbol, Symbol::HashFunction> SemanticAnalysis::getSymbolTable()
51 {
52     return this->symbolTable;
53 }
54

```

```

112     }
113     else if(token->type == FUNCTION)
114     {
115         if(tree->children.size() == 10) // Check if the function has a return statement
116         {
117             if(tree->children.at(8).children.size() < 2) // Check if the function has a return type
118             {
119                 std::cerr << 'Function ' + tree->children.at(0).token.token + " has no return type in line: " << tree->children.at(0).token.loc << std::endl;
120             }
121             if (mapOfTypes[tree->children.at(1).children.at(0).token.token] != tree->children.at(8).token.type) // Check if the return type of the function is the same as the return statement
122             {
123                 std::cerr << "Invalid Type Error: Cannot return " +
124                 getSymbolType(tree->children.at(8).token.type) + " from function with return type " +
125                 getSymbolType(mapOfTypes[tree->children.at(1).children.at(0).token.token]) + " in line: " << tree->children.at(8).token.loc << std::endl;
126             }
127         }
128     }
129 }
130 }
131 }
132 }
133 }
134 else if (token->type == CONDITION) // Check if the token is a comparison operator
135 {
136     if (tree->children.at(0).token.type != tree->children.at(2).token.type) // Check if the operands of the comparison operator are of the same type
137     {
138         std::cerr << "Invalid Type Error: Cannot compare " +
139         getSymbolType(tree->children.at(0).token.type) + " with " + getSymbolType(tree->children.at(2).token.type) + " in line: " << tree->children.at(2).token.loc << std::endl;
140     }
141     token->type = BOOL_ACTION; // Set the type code of the comparison
142 }
143 else if (token->type == RETURN_STATEMENT)
144 {
145     if(tree->children.size() < 2) // Check if the return statement is valid
146     {
147         std::cerr << "Invalid Return Statement in line: " << token->loc << std::endl;
148     }
149     tree->token.type = tree->children.at(1).token.type; // Set the type code of the return statement
150 }
151 else if (token->type == PRINT_STATEMENT)
152 {
153     auto symbol = symbolTable.findSymbol(tree->children.at(2).children.at(0).children.at(0).token.token, 0, 0, 0); // Find the identifier in the symbol table
154     if ((symbol == symbolTable.end()) || symbol->scope > scope || symbol->line > token->loc) && tree->children.at(2).token.type != ID_LITERAL) // Check if the identifier is undeclared
155     {
156         std::cerr << "Undeclared variable " + tree->children.at(2).children.at(0).children.at(0).token.token + " in line: " << token->loc << std::endl;
157     }
158     else
159     {
160         token->type = tree->children.at(2).token.type; // Set the type code of the identifier
161     }
162 }
163 }
164 //tree->root->token = "token"; // Set the root of the tree
165 return tree;
166 }
167 }
168 }
169 string SemanticAnalysis::getSymbolType(int type)
170 {
171     switch (type)
172     {
173     case CHAR:
174         return "char";
175     case INT:
176         return "int";
177     case BOOL:
178         return "bool";
179     default:
180         return "unknown";
181     }
182 }
183 }
184
185 bool SemanticAnalysis::isCmpOp(string token)
186 {
187     return token == "=" || token == "!=" || token == "<" || token == ">" || token == "<=" || token == ">=" || token == "&&" || token == "||";
188 }
189
190 int SemanticAnalysis::identifyType(Token *token)
191 {
192     string literal = token->token;
193     if (literal[0] == '\'')
194     {
195         // Check if the char literal is valid
196         if (literal.size() != 3)
197         {
198             std::cerr << "Invalid char literal " + literal + " in line: " << token->loc << std::endl;
199         }
200         if(literal[3] != '\'')
201         {
202             std::cerr << "No closing apostrophe for char: " + literal + " in line: " << token->loc << std::endl;
203         }
204         return CHAR;
205     }
206     else if (literal == "true" || literal == "false")
207     {
208         return BOOL;
209     }
210     else
211     {
212         return INT;
213     }
214 }
215

```

Stack.hpp

```

1 #pragma once
2 #include <iostream>
3
4 #define SIZE 1000
5
6 using namespace std;
7
8 template <class T> class Stack {
9 public:
10     Stack() { top = -1; }
11
12     // To add element element k to stack
13     void push(T k){
14         // Checking whether stack is completely filled
15         if (isFull()) {
16             cout << "Stack is full\n";
17         }
18
19         // Inserted element
20         top = top + 1;
21         st[top] = k;
22     }
23
24     // To remove the top element from stack
25     T pop(){
26         // Initialising a variable to store popped element
27         T popped_element = st[top];
28         top--;
29         return popped_element;
30     }
31
32     // To get the top element of stack
33     T peek(){
34     {
35         // Initialising a variable to store top element
36         T top_element = st[top];
37
38         // Returning the top element
39         return top_element;
40     }
41
42     // To check if the stack is full
43     bool isFull(){
44         if (top == (SIZE - 1))
45             return 1;
46         else
47             return 0;
48     }
49
50
51     // To check if the stack is empty
52     bool isEmpty(){
53     {
54         if (top == -1)
55             return 1;
56         else
57             return 0;
58     }
59
60 private:
61     int top;
62     T st[SIZE];
63 }
```

Symbol

```

1 #pragma once
2 #include "Parser.hpp"
3 #include <stdio.h>
4 #include <unordered_set>
5
6 class Symbol
7 {
8 public:
9     string name;      // Name of the symbol
10    int type;        // Type of the symbol
11    int scope;       // Scope of the symbol
12    int line;        // Line of the symbol
13
14    /// @brief Function to compare two symbols
15    /// @param other
16    /// @return
17    bool operator==(const Symbol &other) const;
18    Symbol(string name, int type, int scope, int line);
19
20    // hash function for the symbol table
21    struct HashFunction
22    {
23        size_t operator()(const Symbol &symbol) const
24        {
25            return hash<string>()(symbol.name);
26        }
27    };
28
29 };

```

```

1 #include "../headers/Symbol.hpp"
2
3 Symbol::Symbol(string name, int type, int scope, int line)
4 {
5     this->name = name;
6     this->type = type;
7     this->scope = scope;
8     this->line = line;
9 }
10
11 bool Symbol::operator==(const Symbol &other) const
12 {
13     return this->name == other.name;
14 }
15

```

Token.hpp

```

1 #pragma once
2 #include <string>
3
4 enum tokId{
5     UNDEFINED = -999,
6     ID_IDENTIFIER = 1,
7     ID_NUMBER = 2,
8     ID_IF_CONDITION = 4,
9     ID_ELSE_CONDITION = 9,
10    ID_WHILE_LOOP = 15,
11    ID_LITERAL = 17,
12    ID_FOR_LOOP = 19,
13    ID_CHAR = 24,
14    ID_INT = 27,
15    ID_BOOL = 32,
16    ID_VOID = 37,
17    BINOP_DIV = 39,
18    BINOP_PLUS = 41,
19    BINOP_MINUS = 43,
20    BINOP_MULT = 45,
21    AND_OP = 47,
22    LOGIC_AND = 51,
23    LOGIC_OR = 54,
24    EQUAL = 59,
25    SMALLER = 61,
26    BIGEER = 63,
27    SMALLER_EQUAL = 65,
28    BIGEER_EQUAL = 67,
29    NOT_EQ = 70,
30    DEFINE_VAR = 73,
31    SEMI_COLON = 75,
32    APOSTROPHE = 77,
33    QUOTES_TOKEN = 79,
34    COMMA_TOKEN = 81,
35    LPARAN = 83,
36    RPARAN = 85,
37    LBRACE = 87,
38    RBRACE = 89,
39    ID_RETURN = 96,
40    ID_TRUE = 101,
41    ID_FALSE = 106,
42    ID_GIVE = 111,
43    ID_PRINT = 118,
44    ID_EOF = 120,
45 }typedef tokId;
46
47 struct Token
48 {
49     std::string token; //lexeme
50     int type; //tokId from enum
51     int loc; //line of code
52 } typedef Token;
53

```

Types.hpp

```

1 #pragma once
2
3 //*****
4 /* */
5 /* Expression type enum */
6 /* */
7 //*****
8 enum types
9 {
10     INT = 2,
11     BOOL = 60,
12     CHAR = 17,
13 };

```

Dfa

```

1 #pragma once
2 #include <iostream>
3 #include <fstream>
4
5 #define CHARS_NUM 128
6 #define NUM_OF_STATES 128
7
8 class DFA
9 {
10 private:
11     int mat[NUM_OF_STATES][CHARS_NUM];
12
13 public:
14     // DFA handle
15     DFA();
16
17     /// @brief Function to make the DFA
18     void makeDFA();
19
20     /// @brief Function to add tokens to the DFA
21     /// @param token
22     void addTokens(const std::string &token);
23
24     // file handle
25
26     /// @brief Function to write the DFA to a file
27     /// @param path
28     /// @return bool value to indicate the success of the operation
29     bool writeToFile(const std::string &path);
30
31     /// @brief Function to read the DFA from a file
32     /// @param path
33     /// @return bool value to indicate the success of the operation
34     bool readFromFile(const std::string &path);
35
36     /// @brief Function to get the DFA matrix
37     /// @return int(*)[CHARS_NUM]
38     int (*getMat())[CHARS_NUM] { return this->mat; }
39 };
40

```

```

1  #include "../headers/DFA.hpp"
2  #include <ctype.h>
3  #include <string>
4  #define MAT_PATH "./DFA/dfa"
5
6  static int freeState = 1;
7
8  DFA::DFA()
9  {
10    makeDFA();
11
12    writeToFile(MAT_PATH);
13 }
14
15 void DFA::addTokens(const std::string &token)
16 {
17    int i = 0;
18    int currState = 0;
19
20    if (freeState < 2)
21    {
22
23        // Give all letters state of 1 that is identifier
24        for (int row = 0; row < NUM_OF_STATES; row++)
25        {
26            mat[row]['_'] = freeState;
27
28            for(int col = 'a'; col < 'z'; col++)
29            {
30                mat[row][col] = freeState;
31            }
32            for (int col = 'A'; col < 'Z'; col++)
33            {
34                mat[row][col] = freeState;
35            }
36
37        }
38        freeState++;
39
40
41        //give all numbers state of 2
42        for (int row = 0; row < NUM_OF_STATES; row++)
43        {
44            for(int col = '0'; col < '9'; col++)
45            {
46                mat[row][col] = freeState;
47            }
48        }
49        freeState++;
50
51    }
52
53    // Checking if the received token is new
54    // or a prefix of older token
55    while (this->mat[currState][token[i]] != -1 && this->mat[currState][token[i]] != 1)
56    {
57        currState = mat[currState][token[i]];
58        i++;
59    }
60

```

```

60
61 // Making sure each state will point to the next
62 // state in the matrix(state machine)
63 for (; i < token.size(); i++)
64 {
65     mat[currState][token[i]] = freeState;
66
67     //for debugging: print all keywords and their state
68     std::cout << token << ":" << freeState << std::endl;
69
70     currState = freeState;
71     freeState++;
72 }
73
74 //if state is not a keyword, turn all states in final state to -1
75 if(!isalpha(token[i-1]))
76 {
77     for(int j = 0; j < CHARS_NUM; j++)
78     {
79         mat[currState][j] = -1;
80     }
81 }
82
83 freeState++;
84 }
85
86 bool DFA::writeToFile(const std::string &path)
87 {
88     std::ofstream file;
89
90     file.open(path);
91
92     if (!file.is_open())
93     {
94         std::cerr << "Error opening file" << std::endl;
95         exit(1);
96     }
97
98     for (int i = 0; i < NUM_OF_STATES; i++)
99     {
100         for (int j = 0; j < CHARS_NUM; j++)
101         {
102             file << this->mat[i][j] << ',';
103         }
104         file << "\n";
105     }
106 }
107 }
```

```

108     bool DFA::readFromFile(const std::string &path)
109 {
110     std::ifstream file;
111     int i = 0, j = 0;
112
113     if (!file.is_open())
114         file.open(path);
115
116     if (!file.is_open())
117     {
118         std::cerr << "failed creating dfa file..." << std::endl;
119         return false;
120     }
121
122     while (file)
123     {
124         char ch = file.get();
125         if (ch != '\n' && ch != ',')
126         {
127             for (i = 0; i < NUM_OF_STATES; i++)
128             {
129                 for (j = 0; j < CHARS_NUM; j++)
130                 {
131                     this->mat[i][j] = ch;
132                 }
133             }
134         }
135     }
136
137
138     return true;
139 }
140
141 void DFA::makeDFA()
142 {
143     for (size_t i = 0; i < NUM_OF_STATES; i++)
144     for (int j = 0; j < CHARS_NUM; j++)
145         this->mat[i][j] = -1;
146
147     // Conditions
148     addTokens("if");
149     addTokens("else");
150
151     // Loops
152     addTokens("while");
153     addTokens("for");
154
155     // Variables
156     addTokens("char");
157     addTokens("int");
158     addTokens("bool");
159     addTokens("void");
160
161     // Binop
162     addTokens("/");
163     addTokens("+");
164     addTokens("-");
165     addTokens("**");
166     addTokens("&");
167     addTokens("%");
168 }
```

```

169     // Logical gates
170     addTokens("||");
171     addTokens("||");
172     addTokens("~");
173     addTokens("==");
174     addTokens("<=");
175     addTokens(">=");
176     addTokens("<=");
177     addTokens(">=");
178     addTokens("!=");
179
180
181     // Special
182     addTokens(":=");
183     addTokens(";");
184     addTokens("\\");
185     addTokens("\\\"");
186     addTokens(",");
187     addTokens("(");
188     addTokens(")");
189     addTokens("{");
190     addTokens("}");
191
192     //return statment
193     addTokens("return");
194
195     //true and false
196     addTokens("true");
197     addTokens("false");
198
199     //for function
200     addTokens("give");
201
202     //for printing
203     addTokens("scream");
204
205     //end of file
206     addTokens("$");
207 }
208

```

Code generation

```

1  #pragma once
2  #include "Parser.hpp"
3  #include "Symbol.hpp"
4  #include "RegisterEntry.hpp"
5  #include <iostream>
6
7  class CodeGen
8  {
9  private:
10     int labelCount;           // Counter for labels
11     vector<string> code;    // Vector to store the generated code
12     std::string fileName;    // Name of the file
13     unordered_set<Symbol, Symbol::HashFunction> symbolTable; // Symbol table
14     std::vector<RegisterEntry> registers = {
15         {"eax", true}, {"ebx", true}, {"ecx", true},
16         {"edx", true}, {"edi", true}, {"esi", true}, }; // Vector to store the registers
17
18 public:
19
20     /// @brief Constructor
21     /// @param tree
22     CodeGen(std::string fileName, unordered_set<Symbol, Symbol::HashFunction> symbolTable)
23     {this->fileName = fileName; this->symbolTable = symbolTable; labelCount = 0;};
24
25     ~CodeGen();
26
27     /// @brief Function to generate the code
28     bool generate(ParseTree *tree);
29
30     /// @brief Function to execute the nasm code
31     /// @param outputFileName
32     void executeNasm(string outputFileName);
33
34     /// @brief Function to create the base assembly code
35     /// @param file
36     void createBaseAsm(ofstream &file);
37
38     /// @brief Function to generate the identifiers
39     /// @param file
40     void generateIdentifiers(ofstream &file);
41
42     /// @brief Function to generate the code from the AST
43     /// @param tree
44     void generateCodeFromAST(ParseTree *tree);
45
46     /// @brief Function to generate the code for the statements
47     /// @param tree
48     /// @return RegisterEntry
49     RegisterEntry generateExpression(ParseTree *expression);
50
51     /// @brief Function to generate the code for the statements
52     /// @param tree
53     /// @return RegisterEntry
54     string convert32to8(RegisterEntry reg, int byte);
55
56     /// @brief Function to generate the code for the statements
57     /// @param regs
58     /// @param n
59     RegisterEntry getRegister();
60
61     /// @brief Function to free the register
62     /// @param reg
63     void freeRegister(RegisterEntry reg);
64
65     /// @brief Function to set the register state
66     /// @param reg
67     /// @param state
68     void setRegState(RegisterEntry reg, bool state);
69
70     /// @brief Function to push the registers
71     /// @param regs
72     /// @param n
73     void pushRegs(string *regs, int n);
74
75     /// @brief Function to pop the registers
76     /// @param regs
77     /// @param n
78     void popRegs(string *regs, int n);
79
80     /// @brief Function to add the code to the vector
81     /// @param code
82     void addCode(string code);
83 }

```

```

1 #include "../headers/CodeGen.hpp"
2
3 bool CodeGen::generate(ParseTree *parseTree)
4 {
5     std::string outputFileName = this->fileName.substr(0, fileName.size() - 4) + ".asm";
6     std::ofstream outputFile(outputFileName);
7
8     if (outputFile.is_open())
9     {
10         createBaseAsm(outputFile);
11         generateCodeFromAST(parseTree);
12         outputFile << "\n_start:\n";
13         for (string line : code)
14         {
15             outputFile << line;
16         }
17         generateExit(outputFile);
18         generateIdentifiers(outputFile);
19     }
20     else
21     {
22         std::cerr << "Error: Unable to open file" << std::endl;
23         return false;
24     }
25     outputFile.close();
26     executeNasm(outputFileName);
27
28     return true;
29 }

30 void CodeGen::executeNasm(string outputFileName)
31 {
32     //prints here are for debugging purposes
33     //std::cout << "\nRunning following commands to run the code: "<<std::endl;
34     // Command to execute (replace with your actual command)
35     std::string assemble = "nasm -f elf32 " + outputFileName + " -o " + this->fileName.substr(0, fileName.size() - 4) + ".o";
36     std::string link = "ld -m elf_i386 -s -o " + this->fileName.substr(0, fileName.size() - 4) + " " + this->fileName.substr(0, fileName.size() - 4) + ".o";
37     std::string run = "./" + this->fileName.substr(0, fileName.size() - 4);
38     std::string remove = "rm " + this->fileName.substr(0, fileName.size() - 4) + ".o" + " " + this->fileName.substr(0, fileName.size() - 4) + ".asm";
39     //std::cout << assemble << std::endl << link << std::endl << run << std::endl;
40
41     // Execute the command using system
42     bool returnCode = system(assemble.c_str());
43     bool returnCode2 = system(link.c_str());
44     bool returnCode3 = system(run.c_str());
45     bool returnCode4 = system(remove.c_str());
46
47     // if (!(returnCode || returnCode2 || returnCode3 || returnCode4)) {
48     //     std::cout << "\nCommands executed successfully." << std::endl;
49     // } else {
50     //     std::cerr << "\nError executing one or more commands " << std::endl;
51     // }
52 }
53

54
55 void CodeGen::createBaseAsm(ofstream &file)
56 {
57     file << "BITS 32\nsection .text\nglobal _start\n" << std::endl;
58 }
59
60 void CodeGen::generateIdentifiers(ofstream &file)
61 {
62     file << "\nsection .data\n";
63     file << "\tsavedMemoryForPrinting db 100 dup(0)\n"; // write the temporary variable
64     file << "\ttenDividerForPrintingNumbers dw 10\n";           // write the divider
65
66     addData("newLineSavedMemory db 10, 0");
67     for(string str : data) //add the predefined data
68     {
69         file << "\t" << str << "\n";
70     }
71     for (Symbol s : symbolTable) // iterate over the symbol table
72     {
73         file << "\t" << s.name << (s.type == INT ? " dd 0\n" : " db 0\n"); // write the identifier
74     }
75 }
76

```

```

77 void CodeGen::generateCodeFromAST(ParseTree *tree)
78 {
79     Token *token = &tree->token; // get the root of the AST
80     if (tree->value == ASSIGNMENT) // if the root is an assignment
81     {
82         Token *identifier = &tree->children.at(0).token;
83         RegisterEntry reg = generateExpression(&tree->children.at(2)); // get a register
84         addCode("\tmov [" + identifier->token + "], " + reg.name + "\n");
85         freeRegister(reg); // free the register
86         return;
87     }
88     if (tree->value == DECLARATION_STATEMENT) // if the root is a declaration
89     {
90         if (tree->children.size() == 4) {
91             RegisterEntry reg = generateExpression(&tree->children.at(3));
92             addCode("\tmov [" + tree->children.at(1).token.token + "], " + reg.name + "\n");
93             freeRegister(reg);
94         }
95         else
96         {
97             addCode("\tmov byte [" + tree->children.at(1).token.token + "], 0\n");
98         }
99         return;
100    }
101    if (tree->value == EXPRESSION) // if the root is an expression
102    {
103        return;
104    }
105    if (tree->value == PRINT_STATEMENT) // if the root is an out function
106    {
107        RegisterEntry reg = generateExpression(&tree->children.at(2)); // generate the expression
108        if(tree->children.at(2).token.type == ID_CHAR )
109        {
110            addCode("\tmov [savedMemoryForPrinting], " + reg.name + "\n"); // move the result to the temporary variable
111            addCode("\tpusha\n");
112            addCode("\tmov eax, 4\n");
113            addCode("\tmov ebx, 1\n");
114            addCode("\tmov ecx, savedMemoryForPrinting\n");
115            addCode("\tmov edx, 2\n");
116            addCode("\tint 0x80\n");
117            addCode("\tpopaf\n");
118        }
119        else if (tree->children.at(2).token.type == ID_NUMBER)
120        {
121            //prepare the number for printing
122            addCode("\n\tmov eax, " + reg.name + "\n");
123            addCode("\tcall print_num\n");
124        }
125        else if(tree->children.at(2).token.type == ID_LITERAL)
126        {
127            //for printing string literals
128            // calculate the length of string
129            addCode("\n\tmov eax,messageToPrint" + to_string(dataCount) + "\n");
130            addCode("\tmov ecx,-1\n");
131            addCode("print_loop"+ to_string(labelCount) + ":\n");
132            addCode("\tinc ecx\n");
133            addCode("\tcmp byte [eax + ecx], 0\n");
134            addCode("\tjne print_loop"+ to_string(labelCount++) + "\n");

```

```

136
137     // print the string
138     addCode("\tmov edx,ecx\n");           // arg3, length of string to print
139     addCode("\tmov ecx,messageToPrint"+ to_string(dataCount) + "\n");      // arg2, pointer to string
140     addCode("\tmov ebx,1\n");             // arg1, where to write, screen
141     addCode("\tmov eax,4\n");            // write sysout command to int 80 hex
142     addCode("\int 0x80\n");              // interrupt 80 hex, call kernel
143 }
144     addCode("\tcall print_new_line_label\n");
145     return;
146 }
147 if (tree->value == WHILE_STATEMENT) // if the root is a loop
148 {
149     addCode("while_loop_label" + to_string(whileLabels) + ":\n");
150     RegisterEntry reg = generateExpression(&tree->children.at(2)); // get a register
151     addCode("\tcmp " + reg.name + ", 0\n");
152     addCode("\tjz while_statement_end" + to_string(whileLabels) + "\n");
153     generateCodeFromAST(&tree->children.at(5)); // generate code for the body
154     addCode("\tjmp while_loop_label" + to_string(whileLabels) + "\n");
155     addCode("while_statement_end" + to_string(whileLabels++) + ":\n");
156     freeRegister(reg); // free the register
157     return;
158 }
159 if (tree->value == FOR_STATEMENT)
160 {
161     Token *count = &tree->children.at(2).root->token;
162     int typeCode = count->type;
163     RegisterEntry reg = getRegister();
164     addCode("\tmov " + reg.name + ", " + count->token + "\n");
165     addCode("for_loop_label" + to_string(forLabels) + ":\n");
166     addCode("\tpush " + reg.name + "\n");
167     generateCodeFromAST(&tree->children.at(5)); // generate code for the list of expressions
168     addCode("\tpop " + reg.name + "\n");
169     addCode("\tdec " + reg.name + "\n");
170     addCode("\tjnz for_loop_label" + to_string(forLabels++) + "\n");
171     return;
172 }

173 if (tree->value == IF_STATEMENT) // if the root is a conditional
174 {
175     if(tree->children.size() == 7) //if statement without else statement
176     {
177         RegisterEntry reg = generateExpression(&tree->children.at(2)); // get a register
178         addCode("\tcmp " + reg.name + ", 0\n");
179         addCode("\tjz if_statement_end" + to_string(ifLabels) + "\n");
180         generateCodeFromAST(&tree->children.at(5)); // generate code for the body
181         addCode("if_statement_end" + to_string(ifLabels++) + ":\n");
182         freeRegister(reg); // free the register
183     }
184     else
185     {
186         //if statement with else statement
187         ParseTree *cond = &tree->children.at(0);
188         RegisterEntry reg = generateExpression(&cond->children.at(2).children.at(0));
189         addCode("\tcmp " + reg.name + ", 0\n");
190         addCode("\tjz else_" + to_string(ifLabels) + "\n");
191         generateCodeFromAST(&cond->children.at(5)); // generate code for the if body
192         addCode("\tjmp else_statemet_end" + to_string(ifLabels) + "\n");
193         addCode("else_" + to_string(ifLabels) + ":\n");
194         generateCodeFromAST(&tree->children.at(3)); // generate code for the else body
195         addCode("else_statemet_end" + to_string(ifLabels++) + ":\n");
196         freeRegister(reg); // free the register
197     }
198     return;
199 }
200 for (ParseTree child : tree->children) // generate code for each child
201 {
202     generateCodeFromAST(&child); // generate code from the child
203 }
204 }
205 }
```

```

205 RegisterEntry CodeGen::generateExpression(ParseTree *expression)
206 {
207     // ParseTree *root = expression;           // get the root of the expression
208     if (expression->value == IDENTIFIER_ACTION || expression->value == NUMBER_ACTION) // if the expression is an identifier or literal
209     {
210         RegisterEntry reg = getRegister();
211         addCode("\tmov " + reg.name + ", " + (expression->value == IDENTIFIER_ACTION ? "[" + expression->token.token + "]" : expression->token.token) + "\n");
212         expression->setReg(reg);
213     }
214     else if(expression->value == LITERAL_ACTION)
215     {
216         RegisterEntry reg = getRegister();
217         addData("messageToPrint" + to_string(++dataCount) + " db " + expression->token.token + ", 0");
218         expression->setReg(reg);
219     }
220     else if (expression->value == EXPRESSION || expression->value == CONDITION) // if the expression is an expression
221     {
222         for (ParseTree& child : expression->children) // generate code for each child
223         {
224             generateExpression(&child);
225         }
226         if (expression->children.size() == 3) // if the expression has 3 children
227         {
228             RegisterEntry lreg = expression->children.at(0).getReg();      // get the left register
229             RegisterEntry rreg = expression->children.at(2).getReg();      // get the right register
230             string op = expression->children.at(1).children.at(0).token.token; // get the operator
231
232             if (op == "+")                                              // if the operator is addition
233             {
234                 addCode("\taddl " + lreg.name + ", " + rreg.name + "\n");
235                 freeRegister(rreg);
236                 expression->setReg(lreg);
237             }
238             else if (op == "-") // if the operator is subtraction
239             {
240                 addCode("\tsubl " + lreg.name + ", " + rreg.name + "\n");
241                 freeRegister(rreg);
242                 expression->setReg(lreg);
243             }
244             else if (op == "*") // if the operator is multiplication
245             {
246                 addCode("\timul " + lreg.name + ", " + rreg.name + "\n");
247                 freeRegister(rreg);
248                 expression->setReg(lreg);
249             }
250             else if (op == "/") // if the operator is division
251             {
252                 addCode("\tpush edx\n");
253                 addCode("\tpush eax\n");
254                 addCode("\tmov eax, " + lreg.name + "\n");
255                 RegisterEntry reg = getRegister();
256                 addCode("\tmov " + reg.name + ", " + rreg.name + "\n");
257                 addCode("\txor edx, edx\n");
258                 addCode("\tidiv " + reg.name + "\n");
259                 addCode("\tmov " + reg.name + ", eax\n");
260                 addCode("\tpop eax\n");
261                 addCode("\tpop edx\n");
262                 addCode("\tmov " + lreg.name + ", " + reg.name + "\n");
263                 freeRegister(reg);
264                 freeRegister(rreg);
265                 expression->setReg(lreg);
266             }
267         }
268     }
269 }
```

```

268     else if (op == "%") // if the operator is modulo
269     {
270         addCode("\tpush edx\n");
271         addCode("\tpush eax\n");
272         addCode("\tmov eax, " + lreg.name + "\n");
273         setRegState({"edx", true}, false);
274         RegisterEntry reg = getRegister();
275         addCode("\tmov " + reg.name + ", " + rreg.name + "\n");
276         addCode("\txor edx, edx\n");
277         setRegState({"edx", false}, true);
278         addCode("\tidiv " + reg.name + "\n");
279         addCode("\tmov " + reg.name + ", edx\n");
280         addCode("\tpop eax\n");
281         addCode("\tpop edx\n");
282         addCode("\tmov " + lreg.name + ", " + reg.name + "\n");
283         freeRegister(reg);
284         freeRegister(rreg);
285         expression->setReg(lreg);
286     }
287     else if (op == "<") // if the operator is less than
288     {
289         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");
290         string reg8 = convert32to8(lreg, 1);
291         addCode("\tsetl " + reg8 + "\n");
292         freeRegister(rreg);
293         expression->setReg(lreg);
294     }
295     else if (op == ">") // if the operator is greater than
296     {
297         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");
298         string reg8 = convert32to8(lreg, 1);
299         addCode("\tsetg " + reg8 + "\n");
300         freeRegister(rreg);
301     }
302 }
```

```

295     else if (op == ">") // if the operator is greater than
296     {
297         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");
298         string reg8 = convert32to8(lreg, 1);
299         addCode("\tsetg " + reg8 + "\n");
300         freeRegister(rreg);
301         expression->setReg(lreg);
302     }
303     else if (op == "==") // if the operator is equal to
304     {
305         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");
306         string reg8 = convert32to8(lreg, 1);
307         addCode("\tsetz " + reg8 + "\n");
308         freeRegister(rreg);
309         expression->setReg(lreg);
310     }
311     else if (op == "!=") // if the operator is not equal to
312     {
313         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");
314         string reg8 = convert32to8(lreg, 1);
315         addCode("\tsetz " + reg8 + "\n");
316         addCode("\tnot " + reg8 + "\n");
317         addCode("\tand " + lreg.name + ", 1\n");
318         freeRegister(rreg);
319         expression->setReg(lreg);
320     }
321     else if (op == "<=") // if the operator is less than or equal to
322     {
323         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");
324         string reg8 = convert32to8(lreg, 1);
325         addCode("\tsetle " + reg8 + "\n");
326         freeRegister(rreg);
327         expression->setReg(lreg);
328     }
329     else if (op == ">=") // if the operator is greater than or equal to

```

```

328
329     }  

330     else if (op == ">=") // if the operator is greater than or equal to  

331     {  

332         addCode("\tcmp " + lreg.name + ", " + rreg.name + "\n");  

333         string reg8 = convert32to8(lreg, 1);  

334         addCode("\tsetge " + reg8 + "\n");  

335         freeRegister(rreg);  

336         expression->setReg(lreg);  

337     }  

338     else if (op == "&") // if the operator is bitwise AND  

339     {  

340         addCode("\tand " + lreg.name + ", " + rreg.name + "\n");  

341         freeRegister(rreg);  

342         expression->setReg(lreg);  

343     }  

344     else if (op == "|") // if the operator is bitwise OR  

345     {  

346         addCode("\tor " + lreg.name + ", " + rreg.name + "\n");  

347         freeRegister(rreg);  

348         expression->setReg(lreg);  

349     }  

350     else if (op == "^") // if the operator is bitwise XOR  

351     {  

352         addCode("\txor " + lreg.name + ", " + rreg.name + "\n");  

353         freeRegister(rreg);  

354         expression->setReg(lreg);  

355     }  

356     else if (op == "!=") // if the operator is logical NOT  

357     {  

358         addCode("\tnot " + lreg.name + "\n");  

359         expression->setReg(lreg);  

360     }  

361     else if (op == "&&") // if the operator is logical AND  

362     {  

363         int count = this->labelCount++;

```

```

361     {
362         int count = this->labelCount++;
363
364         // Evaluate the first expression and jump to falseLabel if it's false
365         addCode("\tcmp " + lreg.name + ", 0\n");
366         addCode("\tje falseLabel_" + to_string(count) + "\n");
367
368         // Evaluate the second expression
369         addCode("\tcmp " + rreg.name + ", 0\n");
370         addCode("\tje falseLabel_" + to_string(count) + "\n");
371
372         // If we're here, the first expression was true. The result of the AND operation
373         addCode("\tmov " + lreg.name + ", 1\n");
374         addCode("\tjmp endLabel_" + to_string(count) + "\n");
375
376         // Label for the case where the first expression is false
377         addCode("falseLabel_" + to_string(count) + ":\n");
378         addCode("\tmov " + lreg.name + ", 0\n"); // Set result to false
379
380         // End label
381         addCode("endLabel_" + to_string(count) + ":\n");
382
383         freeRegister(rreg);
384         expression->setReg(lreg);
385     }
386     else if (op == "||") // if the operator is logical OR
387     {
388         int count = this->labelCount++;
389
390         // Evaluate the first expression and jump to trueLabel if it's true
391         addCode("\tcmp " + lreg.name + ", 0\n");
392         addCode("\tjne trueLabel_" + to_string(count) + "\n");
393
394         // Evaluate the second expression
395         addCode("\tcmp " + rreg.name + ", 0\n");
396         addCode("\tjne trueLabel_" + to_string(count) + "\n");
397
398         // If we're here, the first expression was false. The result of the OR operation
399         addCode("\tmov " + lreg.name + ", 0\n");

```

```

401         // Label for the case where the first expression is true
402         addCode("trueLabel_" + to_string(count) + ":\n");
403         addCode("\tmov " + lreg.name + ", 1\n"); // Set result to true
404
405         // End label
406         addCode("endLabel_" + to_string(count) + ":\n");
407
408         freeRegister(rreg);
409         expression->setReg(lreg);
410     }
411 }
412 }
413 else // if the expression has only one child
414 {
415     expression->setReg(expression->children.at(0).getReg());
416 }
417 }
418 else if (expression->value == FACTOR) // if the expression is a factor
419 {
420     if (expression->children.size() == 1) // if the expression has only one child
421     {
422         RegisterEntry reg = generateExpression(&expression->children.at(0));
423         expression->setReg(reg);
424     }
425     else
426     {
427         RegisterEntry reg = generateExpression(&expression->children.at(1));
428         expression->setReg(reg);
429     }
430 }
431 return expression->getReg();
432 }
433

```

```

433
434     string CodeGen::convert32to8(RegisterEntry reg, int byte)
435     {
436         string reg8;
437         if (byte == 1) // low byte
438         {
439             if (reg.name == "eax")
440                 reg8 = "al";
441             else if (reg.name == "ebx")
442                 reg8 = "bl";
443             else if (reg.name == "ecx")
444                 reg8 = "cl";
445             else if (reg.name == "edx")
446                 reg8 = "dl";
447         }
448         else // high byte
449         {
450             if (reg.name == "eax")
451                 reg8 = "ah";
452             else if (reg.name == "ebx")
453                 reg8 = "bh";
454             else if (reg.name == "ecx")
455                 reg8 = "ch";
456             else if (reg.name == "edx")
457                 reg8 = "dh";
458         }
459         return reg8;
460     }

```

```

462     RegisterEntry CodeGen::getRegister()
463     {
464         for (int i = 0; i < 6; i++) // iterate over the registers
465         {
466             if (registers[i].isFree) // if the register is free
467             {
468                 registers[i].isFree = false;
469                 return registers[i];
470             }
471         }
472         return registers[0];
473     }
474
475     void CodeGen::freeRegister(RegisterEntry reg)
476     {
477         for (int i = 0; i < 6; i++)
478         {
479             if (registers[i].name == reg.name)
480             {
481                 registers[i].isFree = true;
482                 break;
483             }
484         }
485     }
486
487     void CodeGen::setRegState(RegisterEntry reg, bool state)
488     {
489         for (int i = 0; i < 6; i++)
490         {
491             if (registers[i].name == reg.name)
492             {
493                 registers[i].isFree = state;
494                 break;
495             }
496         }
497     }

```

```
98 void CodeGen::pushRegs(string *regs, int n)
99 {
100     for (int i = 0; i < n; i++)
101     {
102         addCode("\tpush " + regs[i] + "\n");
103     }
104 }
105
106 void CodeGen::popRegs(string *regs, int n)
107 {
108     for (int i = n - 1; i >= 0; i--)
109     {
110         addCode("\tpop " + regs[i] + "\n");
111     }
112 }
113
114 void CodeGen::addCode(string code)
115 {
116     this->code.push_back(code); // add the code to the code vector
117 }
118
119 void CodeGen::addData(string data)
120 {
121     this->data.push_back(data); // add the data to the data vector
122 }
```

```

526 void CodeGen::generateExit(ofstream &file)
527 {
528     file << "\tmov eax, 1\n";
529     file << "\txor ebx, ebx\n";
530     file << "\tint 0x80\n";
531
532
533     //function to print numbers
534     file << "\nprint_num:\n";
535     file << "\txor ecx, ecx\n";
536     file << "counter_label:\n";
537     file << "\tinc ecx\n";
538     file << "\txor edx, edx\n";
539
540     //dividing the number by 10
541     file << "\tdiv word [tenDividerForPrintingNumbers]\n";
542     file << "\tpush edx\n";
543     file << "\tcmp eax, 0\n";
544     file << "\tjg counter_label\n";
545     file << "\nputChar:\n";
546     file << "\tpop edx\n";
547     file << "\tadd edx, '0'\n";
548     file << "\tmov [savedMemoryForPrinting], edx\n";
549     file << "\tpusha\n";
550
551     //printing the number
552     file << "\tmov eax, 4\n";
553     file << "\tmov ebx, 1\n";
554     file << "\tmov ecx, savedMemoryForPrinting\n";
555     file << "\tmov edx, 2\n";
556     file << "\tint 0x80\n";
557
558     //pop the registers
559     file << "\tpopa\n";
560     file << "\tloop putChar\n";
561     file << "\tret\n";
562
563

```

```
563
564     //after each print command we will print a new line
565     file <<"print_new_line_label:\n";
566     file <<"\tmov    edx,1\n";
567     file <<"\tmov    ecx, newLineSavedMemory\n"; // to print newline
568     file <<"\tmov    ebx,1\n";    // arg1, where to write, screen
569     file <<"\tmov    eax,4\n";    // write sysout command to int 80 hex
570     file <<"\tint    0x80\n";
571     file <<"\tret\n";
572 }
```