עבודת גמר

לקבלת תואר טכנאי תוכנה

תמונה שמכילה אומנות קליפיפם, גרפיקה, לוגו, סמל

התיאור נוצר באופן אוטומטי

נושא: The Easy Compiler

מהדר לשפת Easy

המגיש: אביב אש

ת.ז המגיש: 214887556

שם המנחה: מיכאל צ'רנובילסקי

אפריל 2025 תשפ"ה

# תוכן עניינים

[תוכן עניינים 2](#_Toc194622096)

[תקציר 4](#_Toc194622097)

[מושגים 5](#_Toc194622098)

[מושגים כללים 5](#_Toc194622099)

[תיאור הפרויקט 6](#_Toc194622100)

[תכולת השפה 6](#_Toc194622101)

[תיאור אבני השפה 6](#_Toc194622102)

[אופרטורים בינארים: 6](#_Toc194622103)

[משתנים בסיסיים: 7](#_Toc194622104)

[השמת נתונים 8](#_Toc194622105)

[השמה מורכבת 8](#_Toc194622106)

[השוואות 9](#_Toc194622107)

[תנאים לוגים 9](#_Toc194622108)

[תנאים 10](#_Toc194622109)

[פונקציות 11](#_Toc194622110)

[קריאה לפונקציה 11](#_Toc194622111)

[מצביעים 12](#_Toc194622112)

[תוכנית לדוגמא בשפת easy 13](#_Toc194622113)

[דקדוק השפה 14](#_Toc194622114)

[רקע תאורטי בתחום הפרויקט 18](#_Toc194622115)

[קומפיילר/מהדר – הגדרה כללית: 18](#_Toc194622116)

[סוגים שונים של מהדרים: 20](#_Toc194622117)

[סוגים שונים של שפות תכנות: 20](#_Toc194622118)

[שלבי הקומפילציה: 20](#_Toc194622119)

[Front end: 20](#_Toc194622120)

[ניתוח לקסיקלי(Lexical Analysis) 20](#_Toc194622121)

[ניתוח תחבירי (Syntax Analysis) 21](#_Toc194622122)

[דקדוק חסר הקשר (Context free grammar) 21](#_Toc194622123)

[**סוגי מנתחים תחביריים** 23](#_Toc194622124)

[מלמעלה למטה (**top - down**) 23](#_Toc194622125)

[**מנתחי bottom-up** 24](#_Toc194622126)

[מלמטה למעלה (**bottom - up)** 24](#_Toc194622127)

[ניתוח סמנטי (Semantic Analysis) – 25](#_Toc194622128)

[Back end- 25](#_Toc194622129)

[אופטימיזציה – 25](#_Toc194622130)

[יצירת קוד הסף – 25](#_Toc194622131)

[טיפול בשגיאות- 25](#_Toc194622132)

[תיאור הבעיה האלגוריתמית 26](#_Toc194622133)

[אסטרטגיה נבחרת לפתרון 27](#_Toc194622134)

[האסטרטגיה לפיתוח המנתח הסמנטי 27](#_Toc194622135)

[**האוטומט של tec** 28](#_Toc194622136)

[האסטרטגיה לפיתוח המנתח התחבירי 31](#_Toc194622137)

[**הפעולות האפשריות** 31](#_Toc194622138)

[האסטרטגיה לפיתוח המנתח הסמנטי 34](#_Toc194622139)

[האסטרטגיה לפיתוח שלב כתיבת קוד הסף 34](#_Toc194622140)

[ארכיטקטורה של הפתרון המוצע בפורמט של Top-Down Level Design 37](#_Toc194622141)

[תרשים Top-down מפורט 38](#_Toc194622142)

[מבני נתונים 39](#_Toc194622143)

[מטריצה 39](#_Toc194622144)

[מפת hash 40](#_Toc194622145)

[מחסנית 40](#_Toc194622146)

[עץ 41](#_Toc194622147)

[מערך 41](#_Toc194622148)

[תיאור סביבת העבודה ושפות התכנות 42](#_Toc194622149)

# תקציר

בחרתי לעשות את פרויקט הגמר שלי בנושא פיתוח מהדר (compiler)

כבר בתחילת לימודיי במדעי המחשב מצאתי את המהדר כרכיב מסתורי וקסום, איך זה ייתכן שאני כותב קוד באנגלית והמחשב יודע לפרש זאת ולהריץ אותו, זאת אחת הסיבות שבגללן בחרתי את נושא זה כי איזו דרך יותר טובה להבין משהו מללכלך את הידיים ולעשות אותו בעצמך.

במהלך הפרויקט חקרתי מספר כלי הקומפילציה ותרגום (אינטרפטציה) שונים ביניהם (**cpython**,**gcc**) בשביל להבין יותר על הנושא וקריאה יותר עם הנושאים האלו עניינה אותי מאוד ועזרה לי בכתיבת והבנת הפרויקט

הפרויקט פיתח אותי מאוד אני מרגיש שהשתפרתי מאוד כמתכנת. למדתי לכתוב קוד יותר קריא ודינמי וזהו יתרון גדול. במהלך הכתיבה הבנתי שהרבה פעמים ארצה לשנות או להוסיף פונקציונליות ברכיב בקוד שלי ואם הקוד לא כתוב כראוי. למדתי שפת תכנות חדשה, החלטתי לכתוב את הפרויקט שלי בשפת c++ בשל גמישות השפה והיותה גם מונחת עצמים וגם זמן הריצה המהיר שלה וגישה לזיכרון, תהליך למידת השפה היה משמעותי ומקדם עבורי.

וזהו הפרויקט הגדול והמורכב ביותר שלי עד כה, פגשתי אתגרים רבים ולמדתי המון.

לפני שאנחנו קופצים למים נשאל את השאלה – **מהי שפה?**

**שפה פורמלית** (במדעי המחשב ובתורת האוטומטים) מוגדרת כקבוצה של L מחרוזות (strings) מעל אלפבית נתון. Σ כלומר:

L  ⊆ ∗ Σ

**ומה זה קומפיילר?** – תוכנה אשר לוקחת שפה ומתרגמת אותה לקוד אשר רץ על מחשב. זה הכל!

מקווה שתהנו 😁

# מושגים

בפרק זה אציג את המושגים השונים הקשורים ביצירת מהדר

## מושגים כללים

**מהדר** (compiler) - רכיב תוכנה שמטרתו לתרגם קוד מקור (Source Code) לשפת יעד, כמו קובץ הרצה (Executable) או שפת ביניים (Intermediate Representation - IR). מהדרים נפוצים: gcc, clang

**אסימון** (token) – אסימון הוא חלק בעל משמעות בקוד, לכל אסימון יש סוג ושם. לדוגמא בשורת הקוד while(x) נמצאים שלושה אסימונים:

**while** : keyWord] , [**(**:openParen] , [**x**: identifier] ,[**(**,closedParen] ]

**עץ ניתוח תחבירי** (AST) –עץ המורכב מאסימונים המתאר את המבנה הסמנטי של התוכנית

**טבלת הסימנים** (Symbol table) - מבנה נתונים שמנהל מידע על משתנים, פונקציות, וטיפוסים בקוד.

**דקדוק חופשי-הקשר** (CFG) הינו חלק מענף במתמטיקה הנקרא הוא חלק מענף המתמטיקה שנקרא **תורת השפות הפורמליות** (Formal language Theory) בעזרת דקדוק זה נבטא את המבנה של התוכנית (הסבר מפורט יותר בחלק **הרקע התאורטי**)

# תיאור הפרויקט

## תכולת השפה

השפה easy שפה יחסית מינימליסטית והיא נבנתה כשמה, היא אמורה להיות קלה ללמידה ואינטואיטיבית לאנשים עם רקע בתכנות ולוקחת הרבה השראה מC למרות שהיא כוללת גם מאפיינים משפות אחרות כמו **JS**ו **Java**

היא שפה **פרוצדורלית** ולכן מכילה את המבני השפה הנחוצים לה

השפה מבנים בסיסיים לולאות, תנאים, והגדרת ביטויים מתמטיים אך היא מכילה מבנים נוספים כגון: פונקציות, מצביעים ומערכים

## תיאור אבני השפה

### אופרטורים בינארים:

כמו בהרבה שפות גם easy מכילה מגוון אופרטורים בינארים

אופרטור בינארי הינו אופרטור המפעיל פעולה בין שני ביטויים

**<ביטוי 1> <אופרטור> <ביטוי 2>**

**האופרטורים:**

+ **חיבור**

- **חיסור**

\* **כפל**

/**חילוק**

~ פעולת ביטים **not**

| פעולת ביטים **or**

& פעולת ביטים **and**

^ פעולת ביטים **xor**

### משתנים בסיסיים:

השפה כוללת מספר סוגי משתנים בסיסיים:**int**, **char**, **float**

הגדרת משתנה תתבצע בצורה הבאה (בדומה לC):

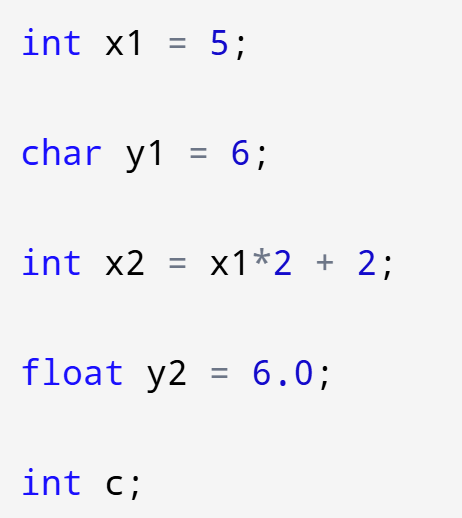
<**ערך אתחול ראשוני**> = <**שם המשתנה**> <**סוג המשתנה**>

**או:**

;<**שם המשתנה**> <**סוג המשתנה**>

יש לשים לבכי בדרך האתחול השנייה המשתנה יאותחל לערך "זבל" ויכלול את תכולת הזכרון הקודם במקום שהוא תפס מבלי לדרוס אותו.

דוגמאות לאתחול נתונים בשפה

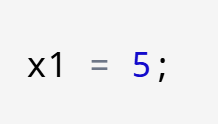


### השמת נתונים

ההשמה מתבצעת בדומה לאתחול רק ללא שם המשתנה בתחילת ההשמה

צד ימין של ההשמה לא יכול להיות ריק

<**ערך אתחול ראשוני**> = <**שם המשתנה**>



דוגמה להשמה תקנית:

תמונה שמכילה גופן, לבן, גרפיקה, עיצוב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

דוגמה להשמה לא תקנית:

### השוואות

השוואה מחזירה אחד משתי ערכים **0** (שקר) או **1** (אמת)

השוואה מתבצעת בעזרת האופרטורים **==** (שווה) או **=!** (לא שווה)

דוגמא:



תחזיר **שקר** כי **0 הוא לא 3!**

### תנאים לוגים

בין שני ביטויים לניתן לשים אופרטור לוגי בצורה הבאה

**<ביטוי 1> && <ביטוי 2>**

**או**

**<ביטוי 1> || <ביטוי 2>**

&& - מסמן **"ו-" לוגי** והוא יחזיר 1 (אמת) רק **אם שתי הביטויים שונים מאפס** (אמת)

&& - מסמן **"או" לוגי** והוא יחזיר 1 (אמת) **אם אחד או יותר משתי הביטויים שונים מאפס** (אמת)

תמונה שמכילה גופן, טקסט, קו, גרפיקה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.לדוגמא:

יחזיר האם המספר מתחלק בשלוש ללא שארית **וגם** מתחלק בשתיים ללא שארים

### תנאים

תנאים בשפה יכתבו בצורה הבאה:

**if(<Expression>) <body>**

**אפשר להוסיף**

**Else <body>**

כאשר תוצאת הביטוי מניבה מספר

**<Body>** הינו או ביטוי אחר או מספר ביטויים בתוך סוגריים מסולסלות

**אם המספר לא שווה אפס** התוכנית תמשיך למה שקיים בתוך גוף התנאי

**כשערכו של המספר הוא אפס** התוכנית לא תכנס לגוף התנאי ותכנס לגוף **הElse** (אם יש)

לדוגמא:



עם **x** שונה מ-0 ערכו ישתנה ל5 אחרת - ישאר אותו הדבר

## פונקציות

**מהי פונקציה?** פונקציה היא מבנה בשפת תכנות המאפשר לבצע קטע קוד מסוים באופן חוזר תוך מתן אפשרות להעברת פרמטרים והחזרת ערך.

פונקציות בשפה easy נכתבות כך –

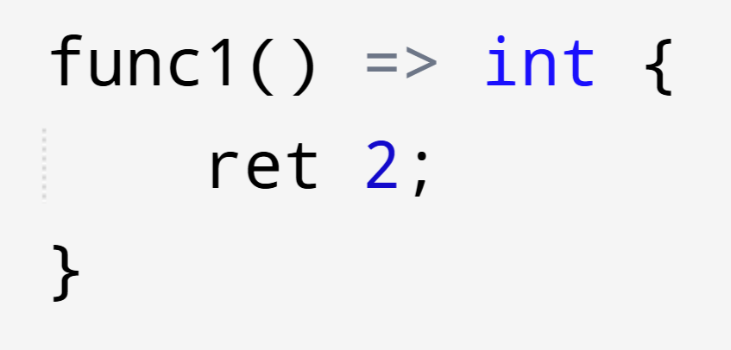
**{<גוף הפונקצייה>} <סוג משתנה> <= (<רשימת פרמטרים>)<שם הפונקצייה>**

גוף הפונקצייה הינו אפס או יותר ביטויים אשר אחד מהביטויים הינו **ביטוי** **ret**

השפה מכילה את מילת המפתח - **ret**כאשר התוכנית מגיעה אל המילה הזאת היא תצא ותחזיר את הערך אשר נמצא אחריה

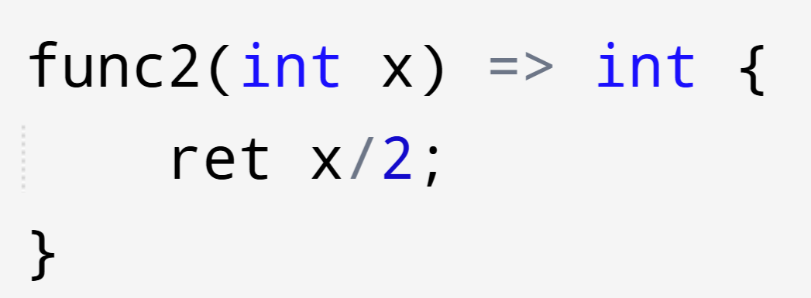
**דוגמאות לפונקציות:**

**דוגמא 1:**

פונקציה שלא מקבלת כלום ומחזירה את הערך 2

**דוגמא 2:**

פונקציה המקבלת מספר ומחזירה את החצי שלו



### קריאה לפונקציה

קריאה לפונקציה תראה כך

**(<פרמטרים>)<שם הפונקציה>**

תמונה שמכילה גופן, גרפיקה, לוגו, טקסט

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**לדוגמא:**

הפונקצייה func2 תקרא עם הערך **6** ותחזיר את הערך **2**

### מצביעים

עוד בונוס קטן לשפה, **מצביעים**.

בשביל שמירה על הפשטות בשפה easy המצביעים מגיעים עד לרמה אחת של ריחוק

**השפה לא מאפשרת מצביע למצביע** או כל מבציע ברמה גבוהה יותר מהצבעה לערך

מצביעים כתובים כך:

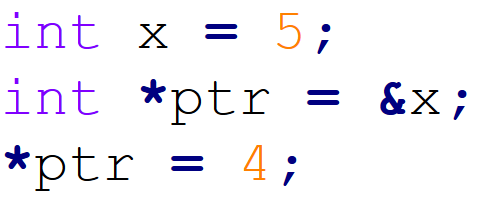
**הגדרת מצביע:**

**< ערך אתחול ראשוני > = <סוג משתנה> \* <שם משתנה>**

**גזירת מצביעה (dereferencing)**

**<שם מצביע>&**

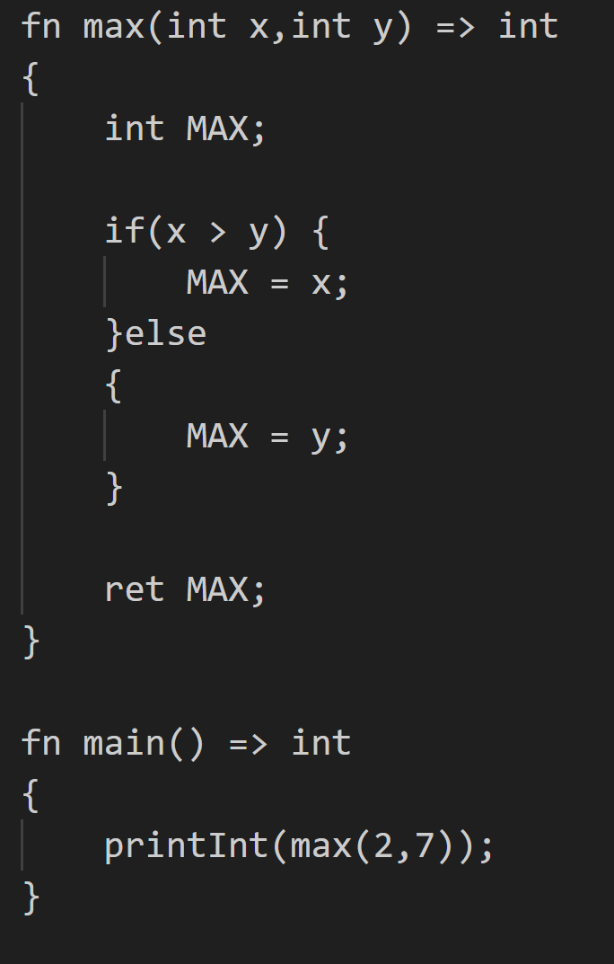
**דוגמא לשימוש:**



בסיום ריצת קטע הקוד הנ"ל ערכו של x יהיה 4

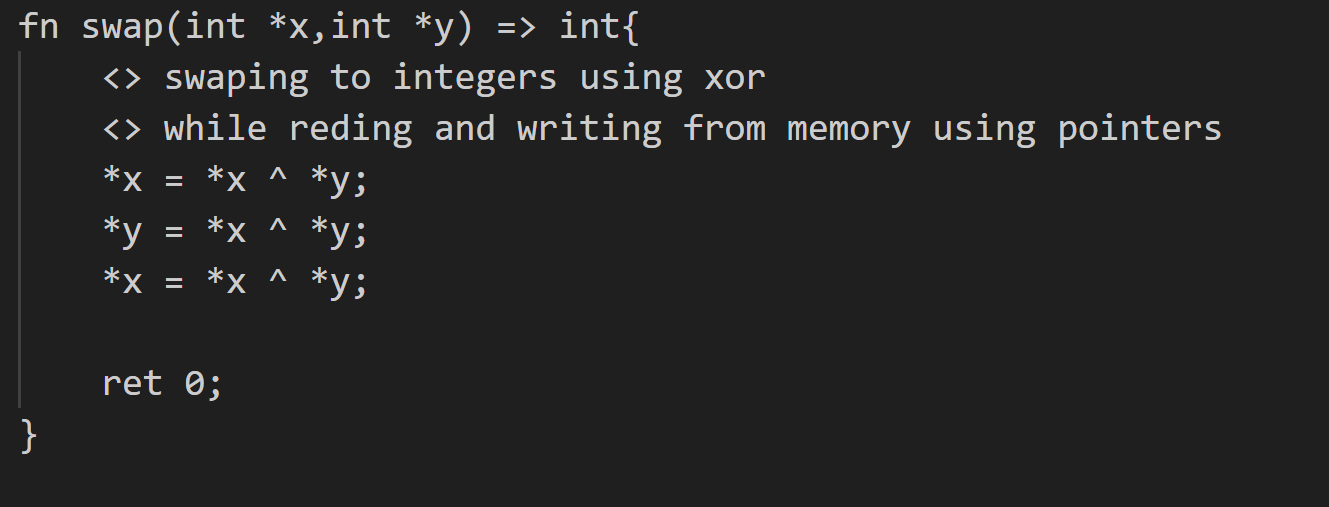
וערכו של ptr הינו הכתובת של x

## תוכניות לדוגמא בשפת easy

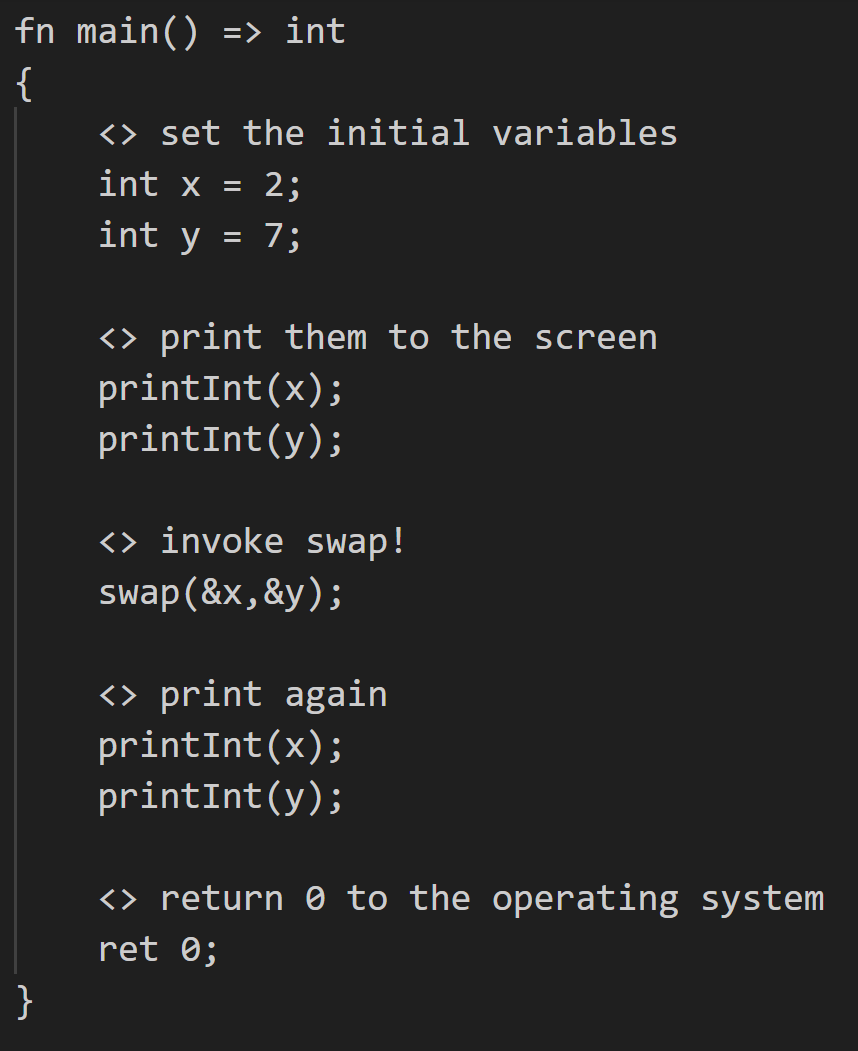


להלן תוכנית המחזירה את המספר הגדול ביותר :

פונקציה המקבלת שתי מצביעים ומחליפה בין הערכים בתוכם:



ופונקציית main אשר קוראת לה :



## דקדוק השפה

מה הוא דקדוק?

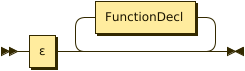
דקדוק הינו אוסף של חוקים המגדירים את השפה.

להלן הדקדוק מוצג בשתי אופנים שונים.

בBNF אשר הינו פורמט סטנדרטי לייצוג שפות

וב **Railroad Diagram**אשר גם הוא דרך סטנדרטית להצגת דקדוק בצורה הממחישה את זרימת השפה

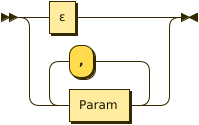
**Program:**



**FunctionDecl:**

FunctionDecl

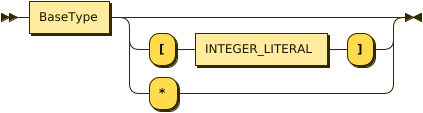
**ParamList:**



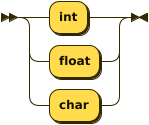
**Param:**

Param

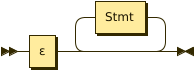
**Type:**



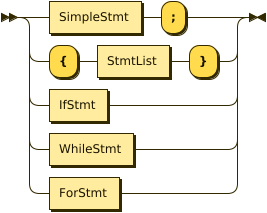
**BaseType:**



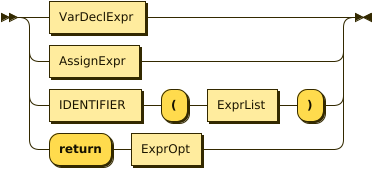
**StmtList:**



**Stmt:**



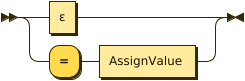
**SimpleStmt:**



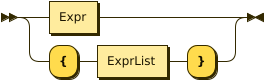
**VarDeclExpr:**

VarDeclExpr

**InitOpt:**



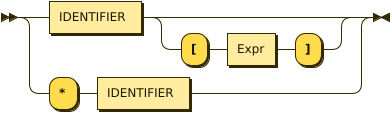
**AssignValue:**



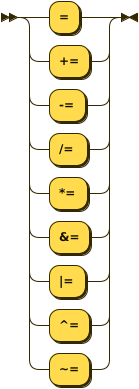
**AssignExpr:**

AssignExpr

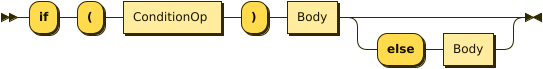
**AssignTarget:**



**AssignOp:**



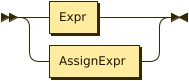
**IfStmt:**



**WhileStmt:**

WhileStmt

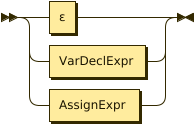
**ConditionOp:**



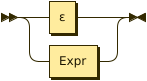
**ForStmt:**

ForStmt

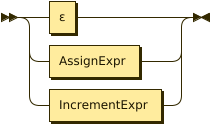
**ForInit:**



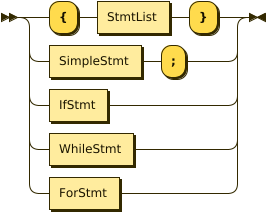
**ExprOpt:**



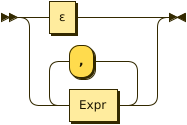
**ForUpdate:**



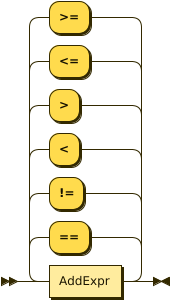
**Body:**

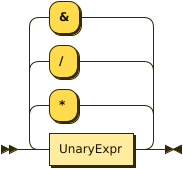


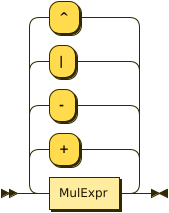
**ExprList:** **Expr:**



**RelationalExpr:**

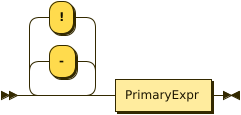


**AddExpr:** **MulExpr:**

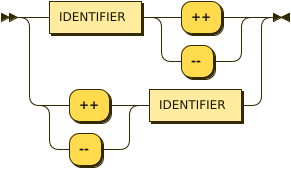


**:**

**UnaryExpr:**



**IncrementExpr:**



**AddressExpr:**

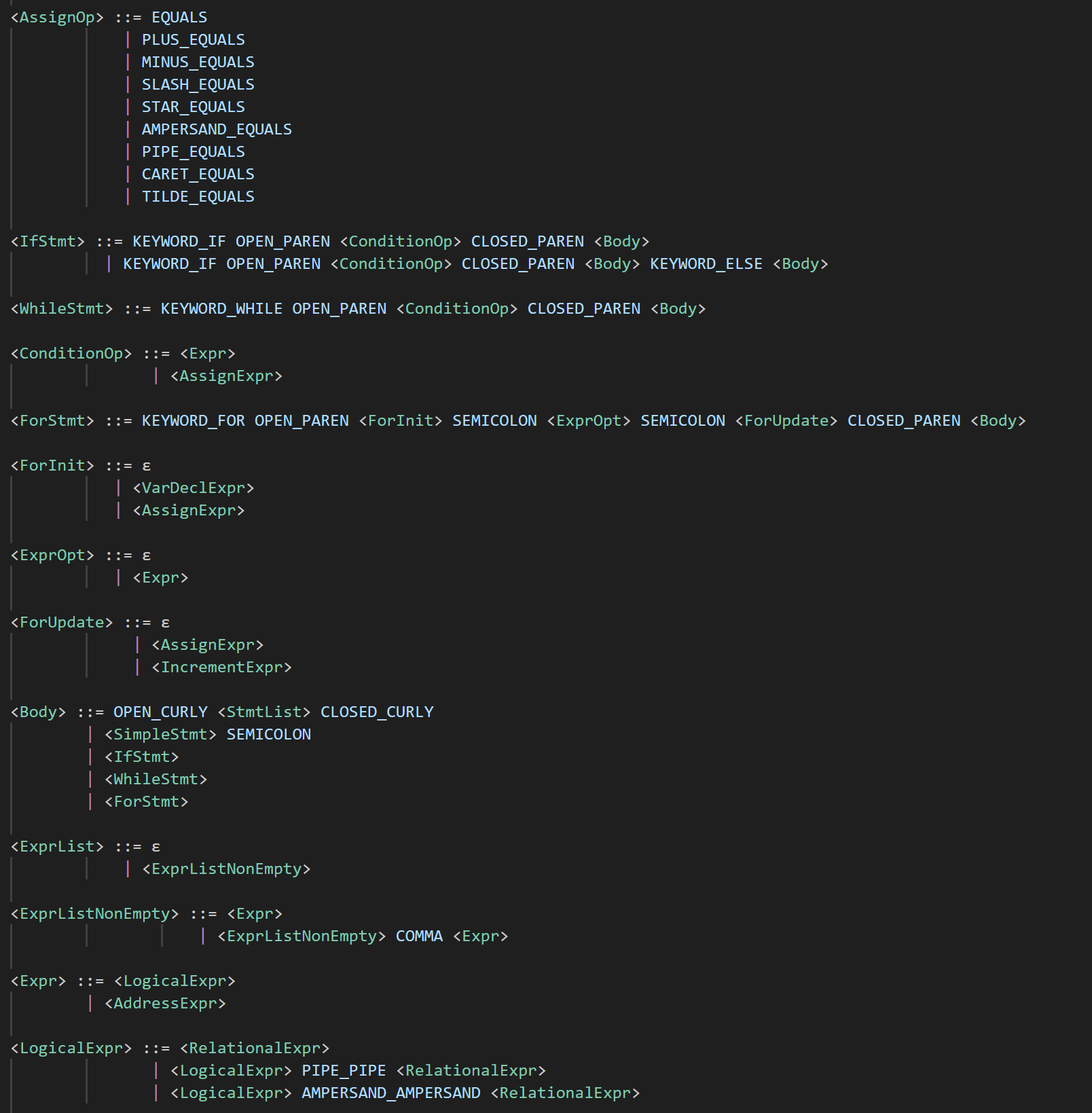
AddressExpr

DereferenceExpr**DereferenceExpr:**

***PrimaryExpr*:**

![PrimaryExpr](data:image/png;base64;base64,)

## **תצוגת bnf**



תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

# רקע תאורטי בתחום הפרויקט

### קומפיילר/מהדר – הגדרה כללית:

בעולם המחשבים קומפיילר (או בשמו העברי: מהדר) הוא רכיב תוכנתי שתפקידו הוא להמיר בין שתי שפות תכנה, המהדר הקלאסי ימיר בין שפה עילית לשפת מכונה

**תמונה שמכילה טקסט, קו, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

ניקח לדוגמא את הקוד הבא הנכתב בשפה העילית "C"אשר נרצה להריץ:

**תמונה שמכילה טקסט, צילום מסך, גופן, מספר

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**

שפות עליות כמו "C" ורבות אחרות עוצבו בצורה הדומה לשפה האנגלית המדוברת ושיהיה למתכנת נוח וקל להשתמש בהן אך המעבד לא בנוי בצורה שהוא יכול להבין בקלות את הוראות אלו. תפקידו של המעבד הוא להמיר את השפה העילית לשפת סף הבנויה בצורה קרובה לארכיטקטורת המעבד שאיתו אנחנו עובדים.

לצורך ההדגמה המהדר יצור לנו את קובץ הassembly הבא:

**תמונה שמכילה טקסט, צילום מסך, מספר, תוכנה

התיאור נוצר באופן אוטומטי**

אז נשתמש בעוד מהדר הממיר קוד סף אל שפת מכונה

תמונה שמכילה טקסט, צילום מסך, גופן, מספר

התיאור נוצר באופן אוטומטי( דוגמאות למהדרים כאלו: NASM לארכיטקטורת x86\_64 TASM לארכיטקטורת x86)

### סוגים שונים של מהדרים:

**Source compiler** – ממיר את הקוד הראשוני (בשפה העילית) היישר לשפת מכונה לדוגמא -רוב המעבדים של השפות C וCPP משתמשים בצורת הידור זאת

**Intermediate Compiler** - ממיר את הקוד הראשוני לשפת ביניים, אז שפת הביניים עוברת מהדר נוסף הממיר אותה לשפת מכונה. לדוגמא – המהדר של C# ממיר את קוד המקור לשפת CIL ומשם עובר עוד מהדר הממיר את הקוד לשפת מכונה

**Transpiler** – ממיר קוד משפה עילית אחת לשנייה לדוגמא – קוד בTypeScript יומר תחילה לJS בשלב הראשוני בהידורו (וזה נעשה בעזרת Transpiler)

### סוגים שונים של שפות תכנות:

שפות תכנות פרוצדורליות - שפות כמו C ופסקל, בשפות אלו, התוכנית מורכבת מרשימת פקודות המבוצעות בסדר שלב אחרי שלב.

שפות תכנות מונחות עצמים -שפות כמו C# וJAVA. בשפות אלו, התוכנית בנויה ממודולים הנקראים אובייקטים, שכל אחד מהם כולל נתונים (תכונות) ופעולות (מתודולוגיות).

שפות תכנות פונקציונליות – שפות כמו Haskell וErlang. בשפות אלו, הפוקנציה היא יחידת התכנות העיקרית. התכנות נעשה דרך קריאה לפונקציות, ולא על ידי שינוי מצב של משתנים.

### שלבי הקומפילציה:

תהליך הקומפילציה מתחלק לשני שלבים מרכזיים,

הfront end- והbeck end-.

### Front end:

שלב זה מתעסק בניתוח הקוד והמרתו לתצורת ביניים (IR) שאיתה השלב השני (beck end) יכול לעבוד. הוא עוסק בניתוח לוגי ובהבנה של המבנה והמשמעות של הקוד, וגם הוא מתחלק לכמה חלקים.

## ניתוח לקסיקלי(Lexical Analysis)

השלב הראשון בfront end עוסק בחילוק הקוד הנתון לטוקנים, אשר כל טוקן הוא יחידה בעלת משמעות כמו מילים שמורות, משתנים, מספרים, אופרטורים וכדומה.

לדוגמה, פיסת הקוד int x = 7; תוכל להיות מתורגמת לטוקנים

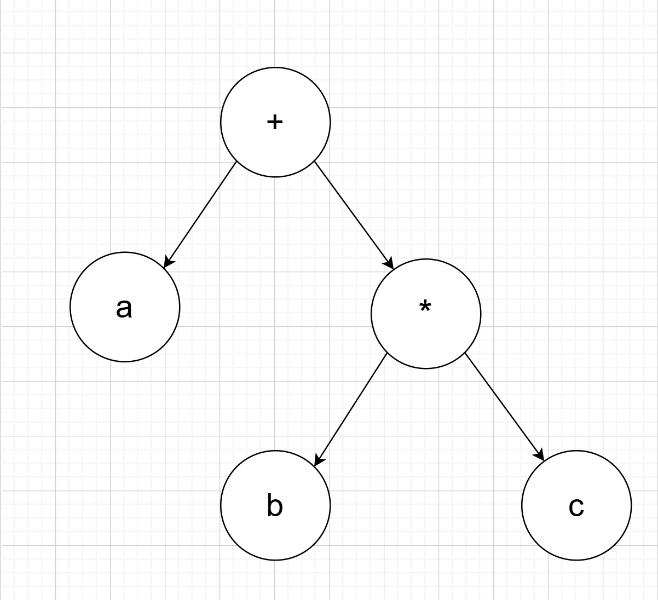
* **Int**  (מילה שמורה)
* **x** (מזהה)
* **=** (אופרטור)
* **7** (מספר)
* **;** (נקודה פסיק)

## ניתוח תחבירי (Syntax Analysis)

מטרה: השלב הבא הוא לבדוק אם רצף הטוקנים שנוצר בתהליך הקודם תואם לחוקי התחביר של השפה (הדקדוק שלה). השלב הזה יוצר את ה-עץ תחבירי או ה-עץ סמנטי (abstract syntax tree - AST), שמייצג את מבנה הקוד לפי כללי השפה.

דוגמה: אם הקוד הוא **a + b \* c**, האנליזר התחבירי ייצור עץ תחבירי שמייצג את ההצהרה על המשתנה וההקצאה.

תמונה שמכילה עיגול, תרשים, קו

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.דוגמא לעץ סמנטי תקין: דוגמא לעץ סמנטי לא תקין:

במהלך השלב הזה, אם יש בעיות תחביריות בקוד (כגון חוסר בסוגריים או סדר לא נכון של רכיבי השפה), יופיעו הודעות שגיאה.

## דקדוק חסר הקשר (Context free grammar)

דקדוק חסר הקשר הינו ענף במתמטיקה אשר עוזר לנו להבין את המבנה של השפה והוא השלד שלפיו נבנה את העץ

**סימני הדקדוק** מתחלקים לשתי קבוצות **סופיים** ו**לא סופיים**

**סופיים**: סימנים אשר ניתן להרחיב אותם לפי כלל דקדוק

**לא סופיים**: סימנים אשר לא ניתנים להרחבה

**כללי דקדוק:** דקדוקים בכללי ודקדוקים חסרי הקשר בפרט מורכבים מאוסף של חוקים אשר מגדירים את הדקדוק

**כלל דקדוק בדקדוק חסר הקשר נראה ככה**:

**<רצף סימנים (סופיים או לא)> → <סימן לא סופי>**

ניקח לדוגמא את הדקדוק הבא:

**הסימנים הסופיים בדקדוק: S**,**E**,**T**,**F**

**הסימנים הלא סופיים בדקדוק: id** , **=** , **;** , **+** , **\***

אם היינו מקבלים את הקלט הבא: **x = a + b \* c;**

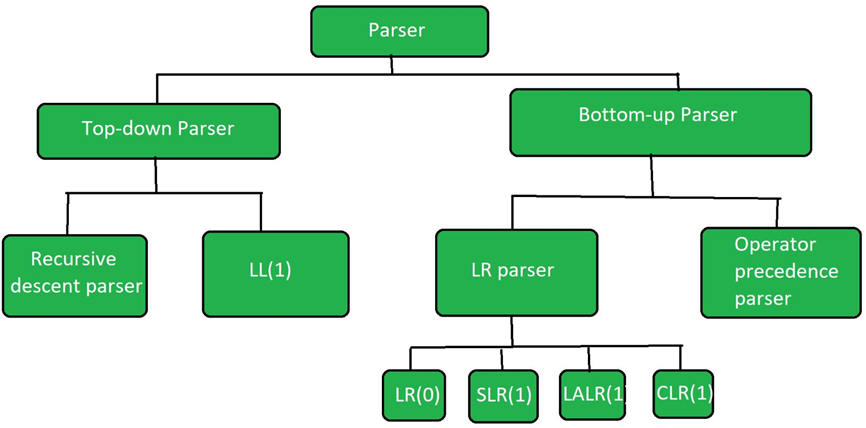
לפי הדקדוק הנתון יתקבל העץ:

תמונה שמכילה קו, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

## **סוגי מנתחים תחביריים**

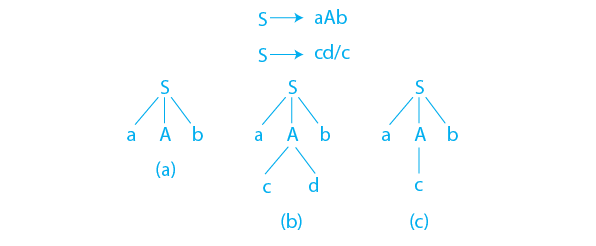
המנתח התחבירי הוא הרכיב המסובך ביותר במהדר. עקב זאת הוא בעל הכי הרבה תאוריה ומחקר בין כל הרכיבים

המנתחים התחביריים המתחלקים לשתי קבוצות עיקריות: **bottom up** ו- **Top down**

## מלמעלה למטה (**top - down**)

ניתוח תחבירי מלמעלה למטה מתחיל עם כלל התחבירי הראשון ומנסה לפתח אותו אל הnon terminals השונים בשפה

אם ניקח לדוגמא את הgrammar הבא:

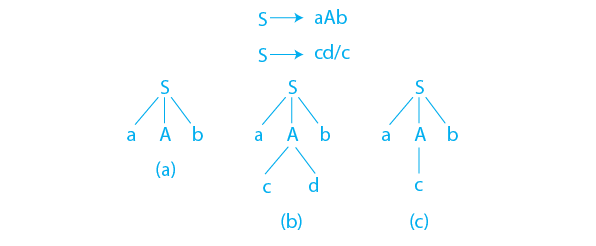


ניתוח top – down יוכל לפתח את העץ בצורה הבאה:

**שלב 1**

**שלב 3**

**שלב 2**



## **מנתחי bottom-up**

מנתחי LR (left-right)

משפחה של מנתחם תחביריים המנתחים מלמטה למעלה ומשתמשים באוטומט מחסנית pda

**(0 LR** – מנתח פשוט המשתמש הלא יודע להתמודד עם דקדוקים מסובכים ומורכבים מידי

**(1)LR**– עובד בדומה ל**(0)LR** אך מסתכל גם על האסימון הנוכחי וגם על האחד שאחריו וכך יכול להתמודד על דקדוקים מסובכים יותר

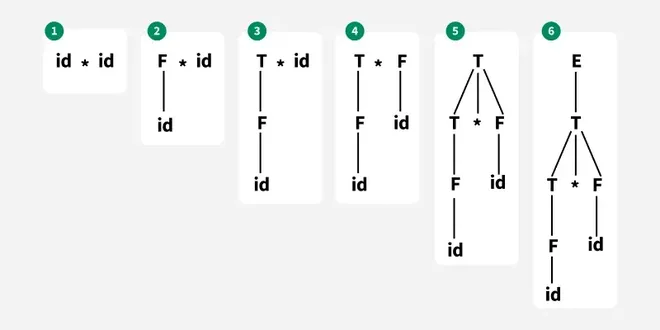
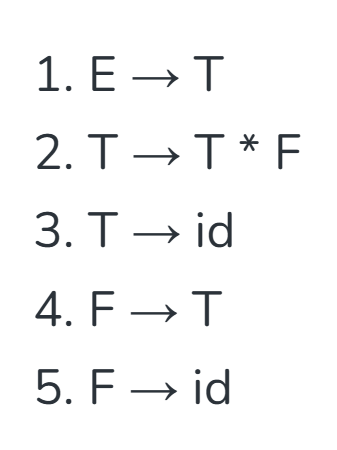
**(1)LALR (look-ahead, left-to-right)** – עובד בדומה ל **(1)LR** אך יוצר טבלה קטנה יותר וכך חוסך מקום זיכרון

**(1)CLR** **(Canonical, left-to-right )** – המנטח הרובסטי יותר מכולם ויודע להתמודד עם כל הדקדוקים חסרי ההקשר שהם

## מלמטה למעלה (**bottom - up)**

ניתוח תחבירי מלמטה למעלה ינסה לקחת את כל האסימונים ולצמצם אותם עד לכדי הגעה לכלל הראשון של הדקדוק

**הgrammar הבא: ינותח בצורה הבאה:**



**Recursive Descent** - מנתח המשתמש בפונקציה לכל כלל דקדוק

– **LL(k) Parsing**משתמש בטבלת ניתוח תחבירי ובמבט קדימה של k סמלים. יכול להיות ממומש עם מחסנית

## ניתוח סמנטי (Semantic Analysis) –

שלב הניתוח הסמנטי הוא שלב שמטרתו לעבור על העץ הסמנטי ולוודא שאין בקוד שגיאות סמנטיות כגון, בדיקת טיפוסים, וידוא התאמה של פרמטרים לפונקציות ועוד....

## Back end-

שלב זה עוסק בניתוח קוד הביניים שהתקבל בשלב הfront end. המטרה בשלב זה היא לא "הבנה" או "ניתוח " של קוד המקור אלה לעבור על קוד הביניים ולהעביר אותו בצורה אופטימלית לשפת הסף

## אופטימיזציה –

שלב האופטימיזציה יעבור על קוד הביניים וייעל אותו בעזרת כמה שיטות, מחיקת שורות ללא משמעות, קיצור הקוד בדרכים שישאירו לו את אותה המשמעות וכו...

## יצירת קוד הסף –

לקיחת קוד הביניים שעבר אופטימיזציה ולהמיר אותו לקוד סף

## טיפול בשגיאות-

בכל שלבי ההידור נרצה לתפוס שגיאות.

לדוגמא, כתיבת שם משתנה לא תקין (שגיאה תחבירית) או פנייה למשנה לא מוצהר (שגיאה סמנטית). טיפול בשגיאות הוא חלק קריטי בכל תהליך פיתוח תוכנה, ובפרט בקומפיילר. המטרה המרכזית היא להבטיח שהתוכנית שכתבת, או במקרה שלנו, הקוד שהקומפיילר מייצר, יתפקד בצורה נכונה, יציבה ובטוחה.נרצה להודיע למתכנת כשהקוד לא תקין ואיפה נמצאת השגיאה לנוחות ולפיתוח מהיר.

# תיאור הבעיה האלגוריתמית

הבעיה האלגוריתמית טמונה במימוש ששת השלבי פיתוח הקומפיילר. כל שלב מציג בעיה אלגוריתמית שונה.

ניתוח לקסיקלי – חלוקת קוד המקור לטוקנים בעזרת **אוטומט דטרמיניסטי סופי**.

ניתוח תחבירי – שלב זה הוא מהעמוסים ביותר מבחינה אלגוריתמית בפיתוח המהדר, הוא כולל את יצירת העץ הסמנטי ועבודה אתו הכוללת מספר אלגוריתמים על עצים והכרעת מצבים בין בניית עצים שונים. שלב זה גם פועל על **אוטומט דטרמיניסטי סופי**.

ניתוח סמנטי – עבירה על העץ הסמנטי ובדיקה של **התקינות הסמנטית** שלו

יצירת קוד הביניים – עבירה על העץ הסמנטי ותרגומו להצגת הביניים.

אופטימיזציה – עבירה על קוד הביניים ובדיקה על איזה מקודות ניתן לעשות לו אופטימיזציה בדרכים שונות.

יצירת קוד הסף – תרגום קוד הביניים אל קוד הסף.

# אסטרטגיה נבחרת לפתרון

האסטרטגיה – לפרק את הפרויקט הגדול לחלקים, כל חלק בעל פונקציונליות משל עצמו.

בסוף פיתוח כל חלק נשתמש בו כקופסא שחורה (black box) כך נוכל להפסיק להתעסק בחלק זה ולהתחיל לפתח את החלק השני בלי להתייחס לתאוריה של החלק הראשון, כך הפיתוח יהיה ממוקד יותר ומוציא קוד דינמי וקריא יותר.

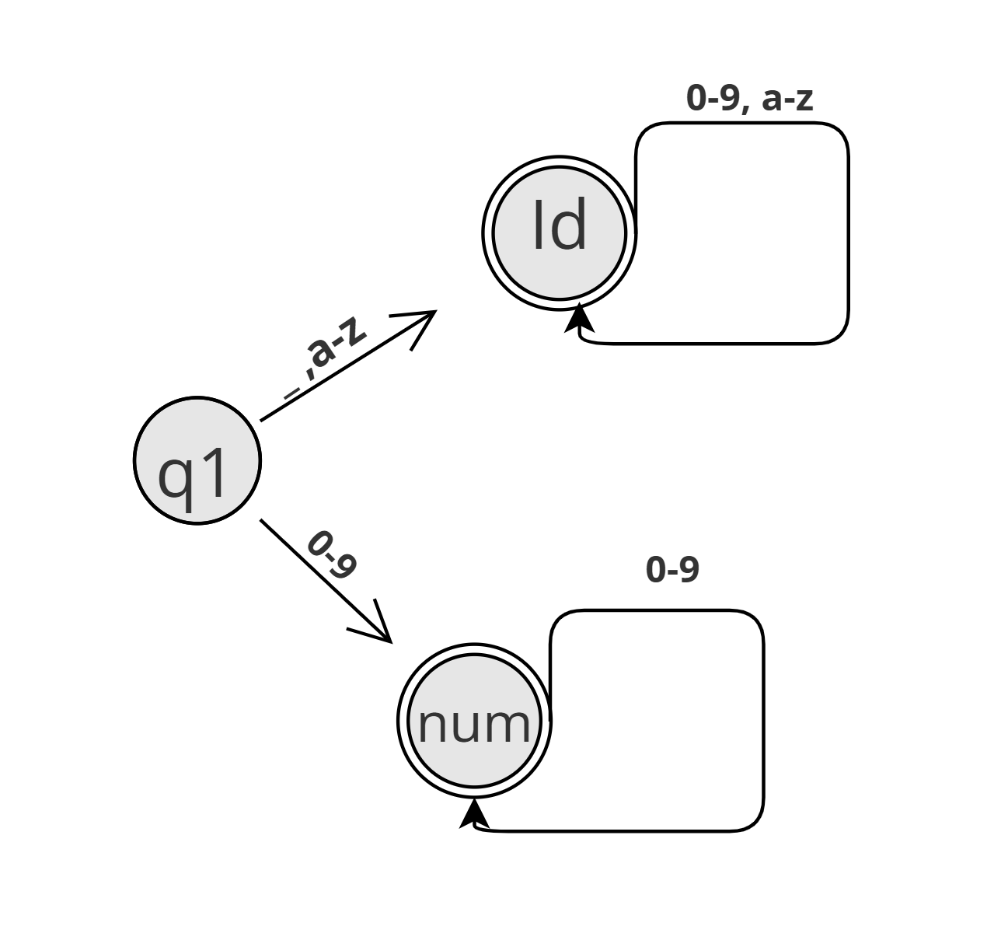
להלן האסטרטגיה לפיתוח כל חלק:

## האסטרטגיה לפיתוח המנתח הסמנטי

בחרתי לפתח את המנתח הסמנטי בעזרת **אוטומט סופי דטרמיניסטי (DFA)**. המנתח עובר תו תו על הקלט ומחלק אותו לאסימונים

ניתח לדוגמא שנראה לחלק את השפה לארבעה סוגים שונים של אסימונים:

**מזהה , מספר, ואסימון לא תקין (לא שייך לשפה)**

נבנה את האוטומט הבא:

כפי שאנו רואים האוטומט למעלה יצליח לזהות מספרים ומזהים . כלומר: אם ניקח לדוגמה את המזהה sum נראה כי האוטומט יסיים בצומתId וכך נדע שהוא מזהה. ציין כי עם קיבלנו קלט שהוא לא אחד מהקלטים האפשריים בצומת הנוכחית, נשלח לצומת מלכודת (Trap state) המציינת אסימון לא תקין

### **האוטומט של tec**

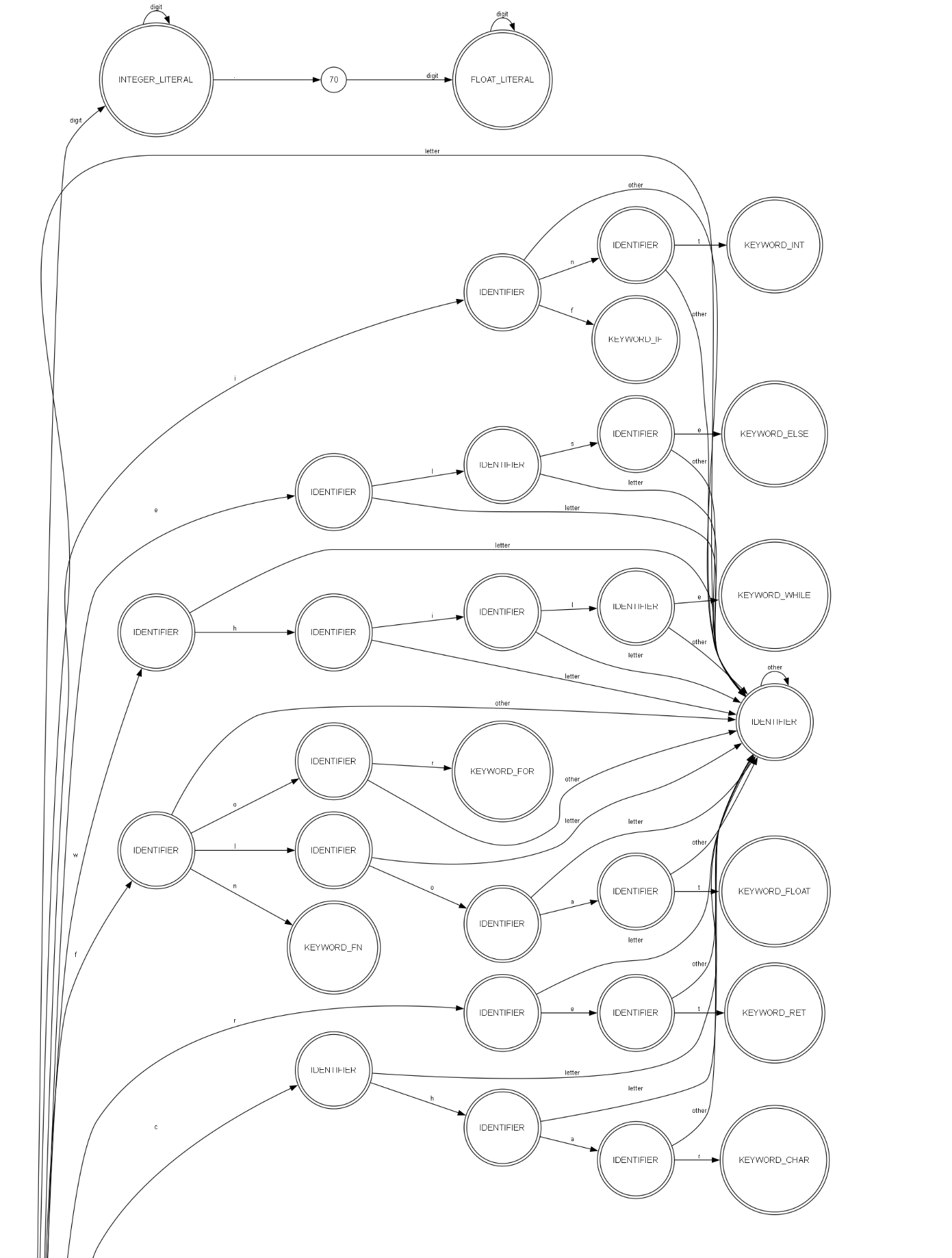
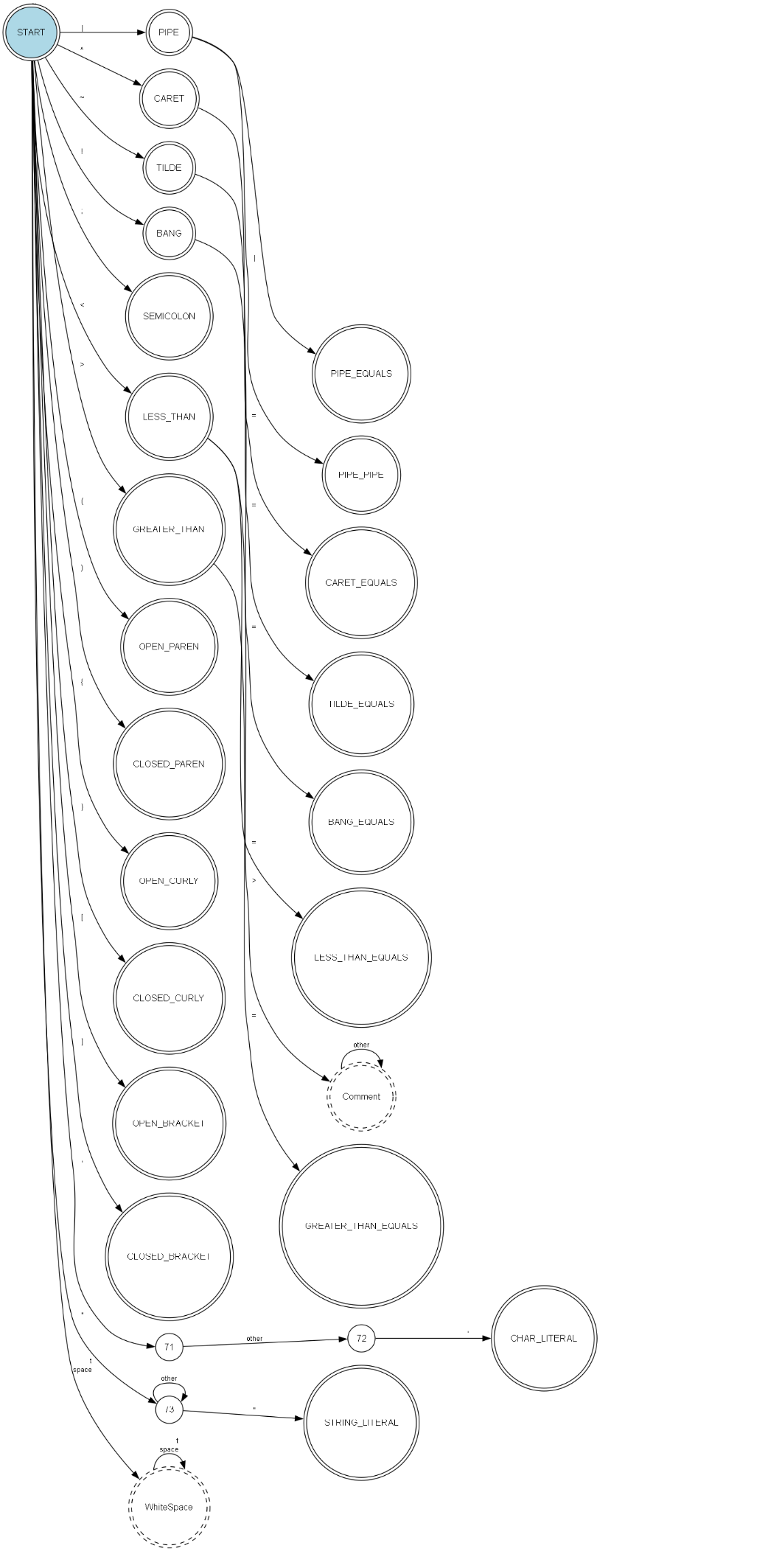
הסבר על התרשים:

מצב התחלתי (מסומן ב**S**-) – מסמל את המצב הראשוני לסריקת אסימון לפני קבלת כל אות

מצבי ביניים (מסומנים ב**מספרים**) – מסמלים מעבריי ביניים באוטומט. סיום הסריקה בהם תסתמן כשגיאה במהדר

מצבי סיום (מסומנים בעיגול כפול) – מסמלים מצביי קבלה/סיום של האוטומט. סיום הסריקה בהם תסתמן כאסימן תקין

מצבי דילוג (מסומנים בעיגול מקווקו) – מסמלים מצביי דילוג של האוטומט. סיום הסריקה בהם לא תחשב ככלום במהדר

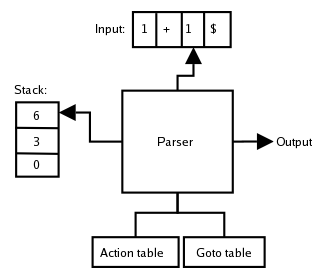
חלק האוטומט הקולט מזהים (שמות משתנים ופונקציות),מחרוזות,מספרים שלמים, מספרים עשרוניים ו חלק האוטומט הקולט את שלל סימני השפה, רווחים והערות:

מילות מפתח:

## האסטרטגיה לפיתוח המנתח התחבירי

בחרתי להשתמש ב **LR parser**לקומפיילר שלי

איך הוא בנוי?



המנתח התחבירי מסתמך על שתי טבלאות – **Goto**, **Action**

ומבנה נתונים – **מחסנית**.

הוא בנוי על מכונת מצבים גם כן

המחסנית – **מאחסנת את סימני הדקדוק ומספרי המצבים**

הטבלאות:

**Goto** – מסמנת לאיזה מצב לעבור על פי מצב הקלט והמחסנית

**Action** – מסמן איזה פעולה לבצע לפי המצב הנוכחי

### **הפעולות האפשריות**

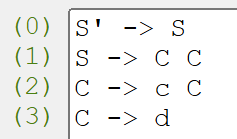
**Shift –** הכנס את האסימון הבא מהקלט אל המחסנית ביחד עם מספר המצב המצויין

**Reduce – צמצם** את הקלטים בראש המחסנית לפי חוק דקדוק מצוין

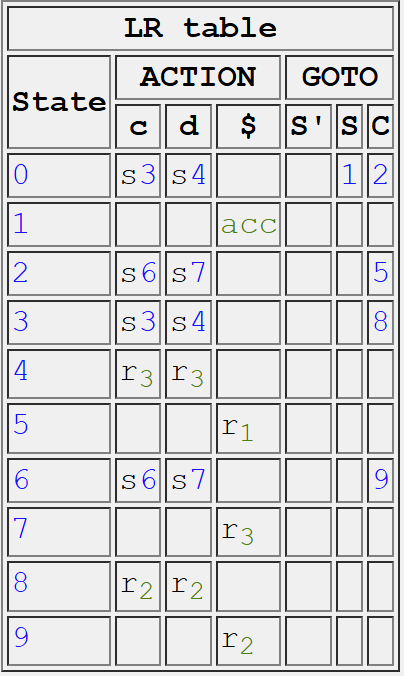
**לפי מה נבנה את הטבלאות ?**

ישנם כמה שיטות לבנות את הטבלאות אך אני בחרתי לבנות את הטבלה לפי אלגוריתם **IELR**

**למה בחרתי באלגוריתם זה?** כי הוא בונה את הטבלה יחסית מצומצמת ויכול לקבל הרבה סוגים שונים של חוקי דקדוק

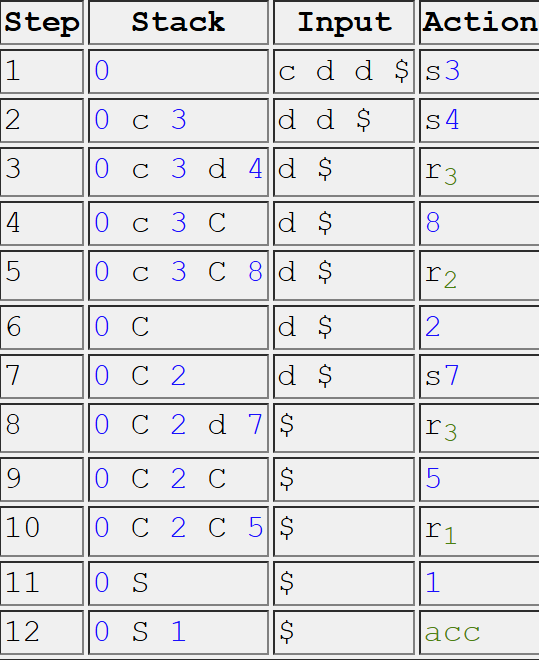
ניקח לדוגמא את **הCFG** הבא:

בעזרתו נבנה את טבלאות **הGOTOו** **ACTION**



ואם נרצה לנתח לדוגמא את הקלט: **c d d**

המעקב יראה ככה:



תמונה שמכילה מלבן, עיצוב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.והעץ הסופי יראה ככה:

## 

## האסטרטגיה לפיתוח המנתח הסמנטי

בשביל לממש שלב זה נצטרך:

**לעבור על טבלת הסימנים** ולראות ש-

* אין כפילויות.
* כל משתנה משומש לפי הscope שלו

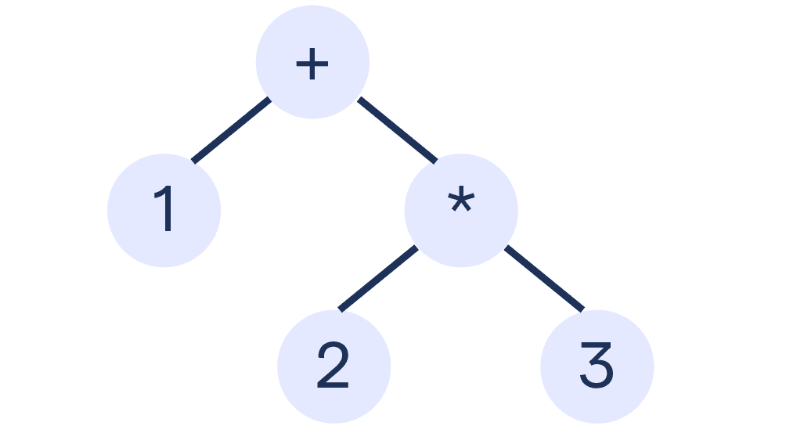
לעבור על העץ בצורה רקורסיבית, ולבדוק את התקינות של דברים שונים כגון:

**האם בהכרזת וייסום משתנים סוגי המשתנים תואמים?**

**האם הפונקציה מחזירה את מה שנמצא בהכרזה שלה?**

**האם קריאה לפונקציה תואמת את הפרמטרים בהכרזה שלה?**

גם כן יש ליזכור שבשפה ישנם טיפוסים שונים ונרצה לבדוק אם הם תואמים בכדי שנוכל לבצע עליהם פעולת. בשביל זה ניתן לכל חולייה בעץ טיפוס, החל מלמטה במעלה העץ ניתן לכל החוליה טיפוס



**int**

**int**

**int**

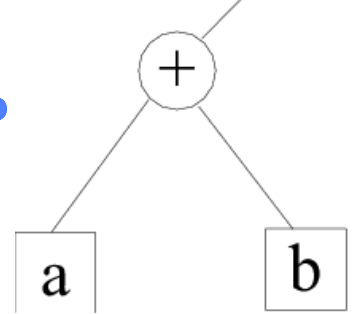
## 

**int**

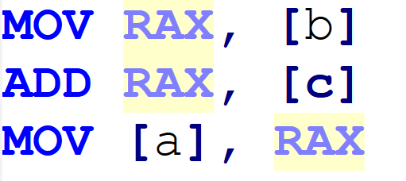
**int**

## האסטרטגיה לפיתוח שלב כתיבת קוד הסף

בשלב זה גם נעבור על העץ אך הפעם נסתכל על כל צומת בעץ ולפי סוגה נמיר לקוד סף.

**פעולות אריתמטיות**

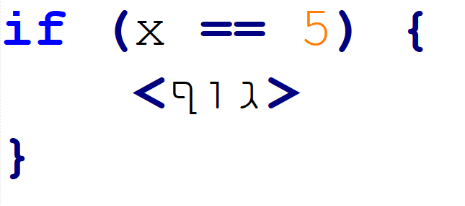
**החלק בעץ:**



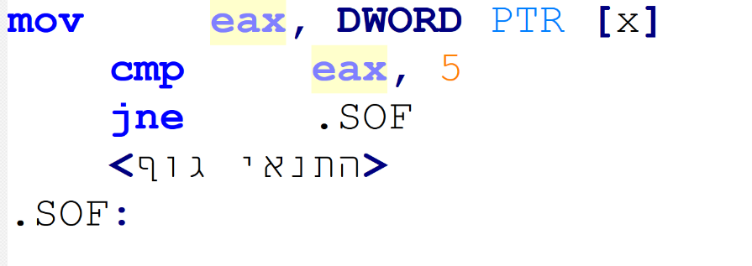
**יהפוך ל:**

**תנאים**

נשתמש במבנה בסיסי של שפת סף לתנאים

**הביטוי:**

**יהפוך ל:**



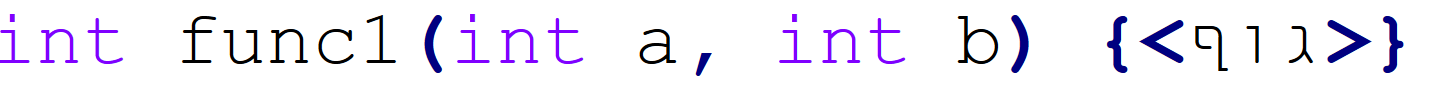
## פונקציות:

**יצירה:**

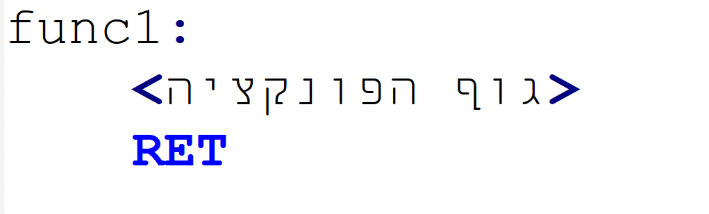
נדחוף למחסנית את כל המשתנים של הScope הנוכחי (כולל הפרמטרים)

נאתחל את הפרמטרים עם הערכים שנתנו להם בזימון

ניקח לדוגמא פונקציה.

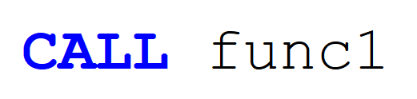


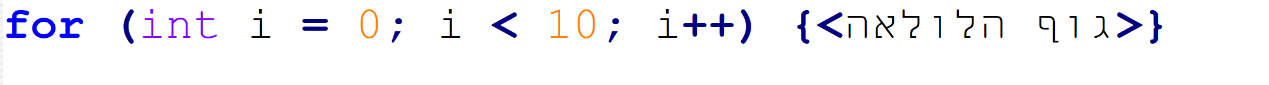
נוכל ליצור אותה באמצעות יצירת label חדש וסיום בשימוש הפקודה ret



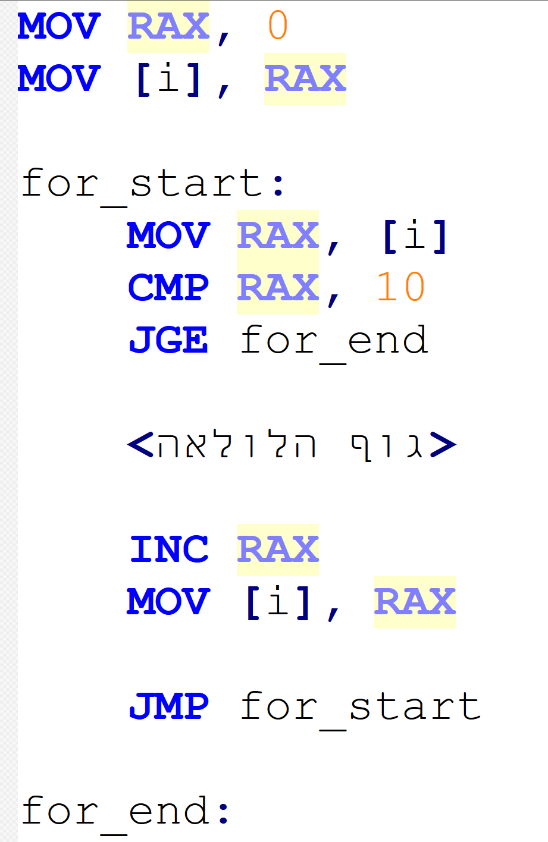
כאשר **a**ו**b**- שמורים ב**[bp - 8]**ו **[bp - 16]-**במחסנית

**קריאה לפונקציה:**



**לולאת for**

**יתורגם ל:**



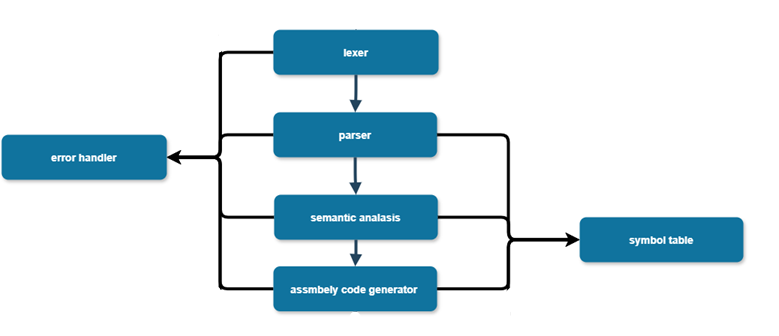
**לולאות while**



**יתורגם ל:**

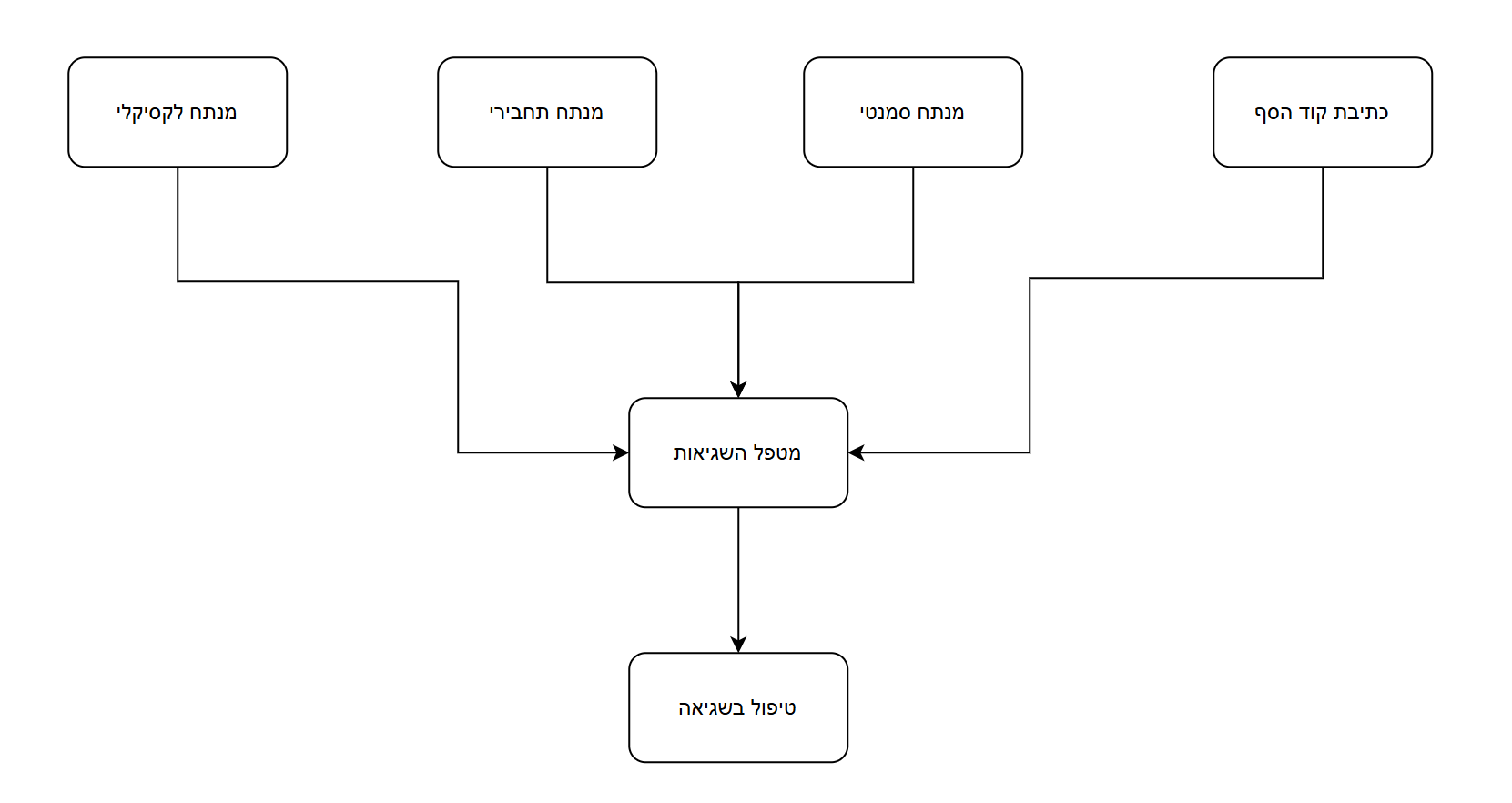


# ארכיטקטורה של הפתרון המוצע בפורמט של Top-Down Level Design

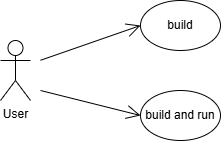
להלן תרשים זרימה המתאר את החלקים השונים ואת התקשורת ביניהם בפורמט השקפת על (High level):

## תמונה שמכילה תרשים, טקסט, תוכנית, שרטוט טכני תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תרשים Top-down מפורט

כל חלק אשר גזור ישירות מהקומפוננטה הראשית מקושר ישירות אל מטפל השגיאות



# תרשים מקרי שימוש UML Use cases



# מבני נתונים

## מטריצה

פיתוח מהדר כולל עבודה עם כלל מבני נתונים שונים ומגוונים. הרי אנחנו עובדים עם קלט אשר אנו רוצים לפרק ולעבד.

ובשביל לעשות זאת נשתמש **במבנה נתונים מתאים** לכל בעיה

כפי שראינו בפרקים הקודמים כתיבת המנתח הלקסיקלי כרוכה בעבודה עם **אוטומט סופי דטרמיניסטי (dfa)**. אך איך נמיר אותו לקוד.

ישנם שתי דכים מקובלות לעשות זאת.

**מפת hash** – אשר שומרת כמפתח את ה**מצב ההתחלתי** ובערך את הצמד של **תו המעבר** **והמצב הסופי**

**מטריצה –** אשר שומרת את אותם שלושה ערכים בפורמט שבו **השורות** מייצגות את ה**מצב ההתחלתי העמודות** את **תו המעבר** **והחיתוך בינהן** הינו **המצב הסופי**

בחרתי להשתמש ב**מטריצה**

מטריצה הינו מבנה נתונים הבנוי כמערך דו ממדי של ערכים (בעל עמודות ושורות)

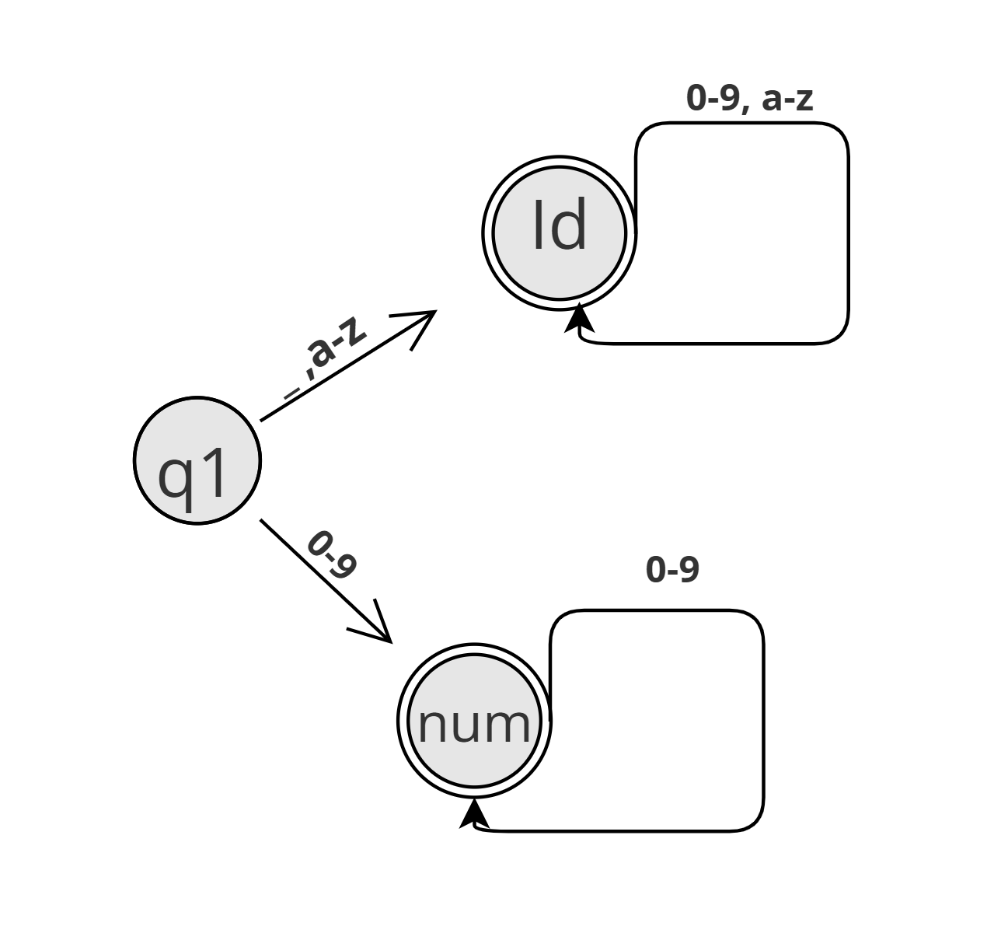
למה? – זהו מבנה נתונים אשר קל לעבוד איתו ולהכניס אילו נתונים.

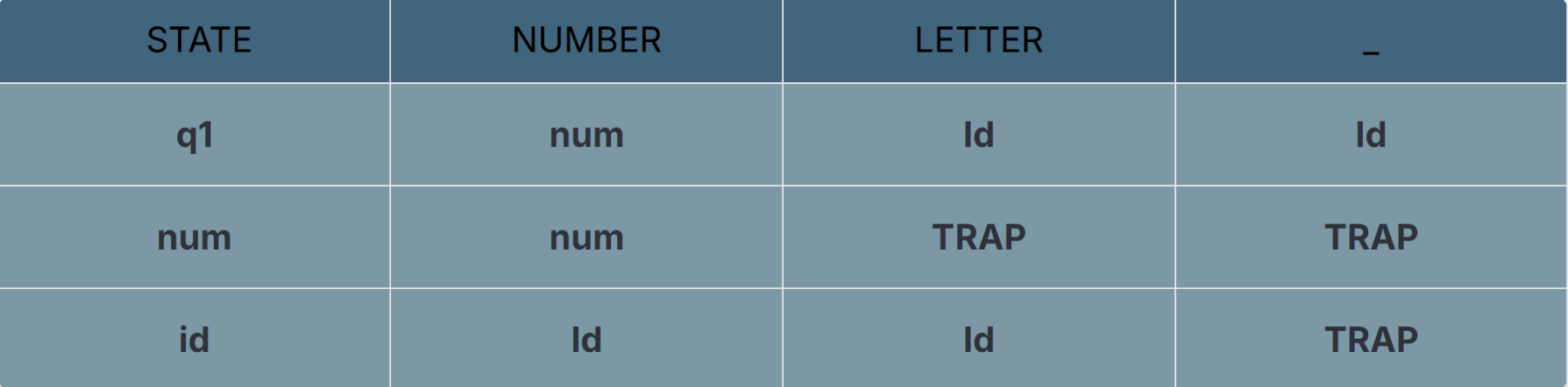
שליפת **מצב הבא** בהינתן מצב נוכחי ותו מעבר הינו **O(1)** לעומת **מפת הhash** ששם היא **O(N)** אך יש לשים לב שהמטריצה לא יותר טובה בצורה אבסולוטית מהמפה מכיוון שיעילות המקום של שהמטריצה תופסת בהתייחס לכמות המצבים שיש הינה **O(N^2)** לעומת המפה אשר תופסת **O(N)** מקום.

אך למרות זאת בחרתי במטריצה מכיוון שאעדיף להקריב **מקום** מאשר **זמן ריצה**

**דוגמא להמרת אוטומט למטריצה**

ניקח לדוגמא את האוטומט הבא:



נוכל להמירו למטריצה הזאת:

קל מאוד לראות שאם לדוגמא מצבו הנוכחי של האוטומט הוא q1 והקלט הוא מספר נעבור למצב **num**

אנו נשתמש במטריצה בעוד מספר מקומות בקוד שבהם השימוש בא יותר טריוויאלי. כמו טבלאות (**symbol table**, **goto** **table**, **action table**)

## מפת hash

Hash map הינו מבנה נתונים חשוב בתכנות אשר מאפשר לשמור צמדים של נתונים key,value בצורה כזאת שניתנן אפשרות לשלוף את ערך הvalue בעזרת המפתח ביעילות זמן ריצה **O(1)**

השתמשתי במבנה נתונים זה במקומות רבים בקוד שלי בעיקר לנוחות.

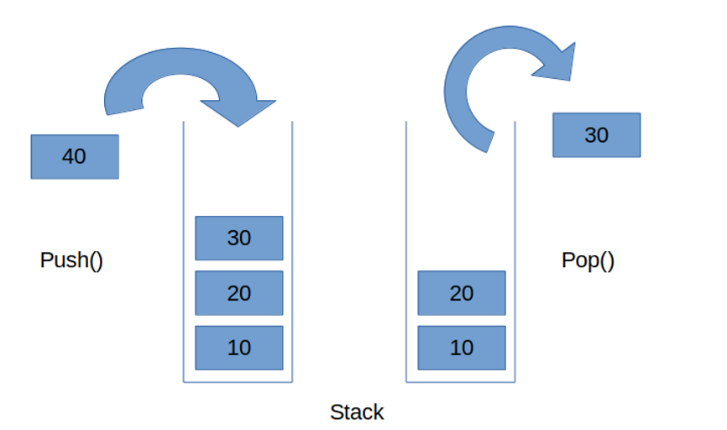
לדוגמא:

**בטבלת הסימנים –** שם משתנה\פונקציה (**key**) רשומה בטבלה (**value**)

**באוטומטDFA -** סימן בשפה (**key**) מיקומו במערך הסימנים (**value**)

## מחסנית

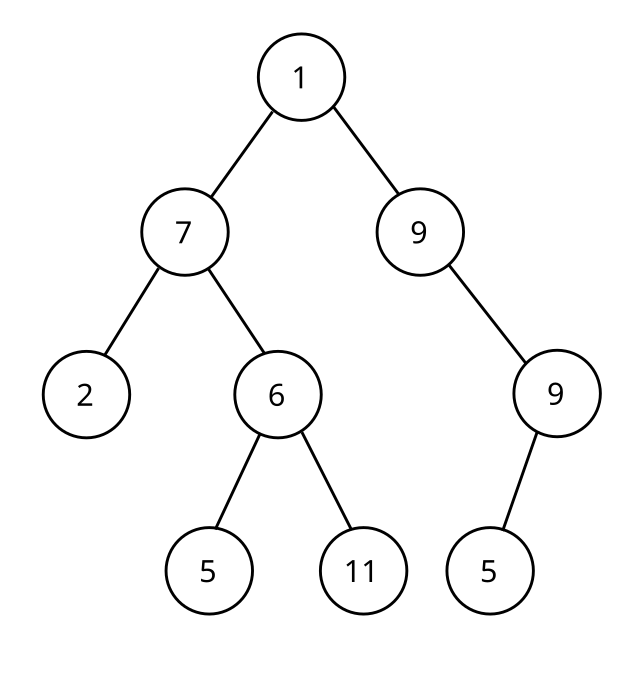
מחסנית הינו מבנה נתונים העובד בשיטת LIFO – Last in first out (הראשון בפנים האחרון בחוץ)

הוא מכיל שתי פעולות עיקריות push ו pop אשר משמשות להוצאה והכנסה של איברים מהמחסנית

**נשתמש מחסנית כחלק ממימוש המנתח המילוני**

## עץ

עץ בינארי הוא מבנה נתונים הבנוי מצמתים, לכל צומת יש אפס "בנים" או יותר המחוברים אילו בקשת כך שלא יכולה להיות קשת המובילה מהבן בחזרה לאב (**או לדרגה קדומה יותר**)



**דוגמא לעץ:**

**איך זה שימושי לנו?**

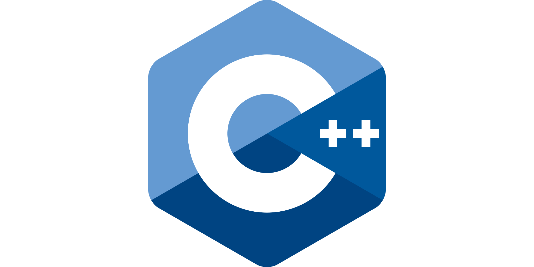
החלק המרכזי של המנתח התחבירי הוא ליצור עץ המתאר את המבנה התחבירי של הקוד. הקוד בשפה בנוי בצורה היררכית ועץ זאת דרך מעולה לשמור את הנתונים בצורה בעלת ב=המשמעות שאנחנו רוצים

## מערך

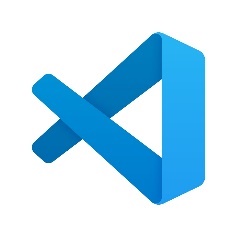
מערך הינו מבנה הנתונים הנפוץ ביותר. ובכללי בתכנות, נשתמש בו בלי סוף

מדובר באיחסון הנתונים בצורה רציפה בזיכרון, דבר המאפשר התעסקות קלה ושליפת איברים ב**O(1)**

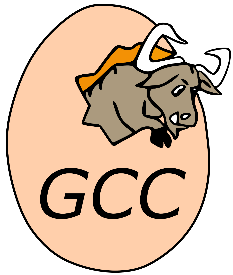
# תיאור סביבת העבודה ושפות התכנות



לפרויקט זה, החלטתי להשתמש בשפת ++C מכיוון שהיא מאפשרת גם עבודת Low **Level והתעסקות ישירה עם הזיכרון** וגם שימוש **בתכנות מונחה עצמים (OOP)** אשר נותן לכל ישות בקוד להיות עצם משל עצמו בעל תכונות ופונקציות.

ככה אוכל לשמור על ביצועים גבוהים בעוד שימוש בתכנות מונחה עצמים וכלים מודרניים יותר

את הקוד עצמו כתבתי בתוכנת **Visual Studio Code Version: 1.95.2**



הידרתי אותו באמצעות **GCC** (gnu C compiler) אשר באופן אירוני בהכנת הפרויקט יצא לי לחקור עליו בעצמי וללמוד עליו

# אלגוריתם ראשי

בפרק זה אתאר **בפסיאודו קוד** את האלגוריתם

## **אלגוריתם העל**

1. **אתחל את המנתח הלקסיקלי**
2. **אתחל את מדווח השגיאות**
3. **ייצר רשימת אסימונים מהקובץ הנתון**
4. **אם יש אסימון שגיאה דווח.**
5. **אתחל את המנתח המילוני**
6. **ייצר את העץ התחבירי**
7. **אתחל את המנתח סמנטי**
8. **עבירה על העץ הסמנטי**
9. **אתחל את מכולל קוד הסף**
10. **ייצר את קוד הסף**

## **אלגוריתם המנתח הלקסיקלי**

1. **שמור את שם הקובץ**
2. **אתחל את טבלת האוטומט**
3. **עד שלא קיבלנו אסימון עם סוג סוף קובץ** 
   1. **הוסף לרשימת האסימונים את האסימון הבא**

## מציאת אסימון הבא

1. **פתח את הקובץ**
2. **אתחל את המצב ההתחלתי**
3. **אם הגענו לסוף הקובץ** 
   1. **החזר אסימון סוף קובץ**
4. **עשה** 
   1. **קרא את התו הבא בקובץ**
   2. **קבל את המצב הבא**
   3. **אם המצב שקיבלנו לא תקין \ הגענו לסוף הקובץ**
      1. **צא מהלולאה**
   4. **אחרת**
      1. **עדכן את הפוזיציה של האסימון ואת תכולתו**
5. **חזור ל 5.2.**
6. **אם המצב שהגענו אילו הינו מצב דילוג** 
   1. **קרא את האסימון הבא**
7. **אחרת**
   1. **אם זהו מצב סופי תקין של האוטומט** 
      1. **החזר אסימון חדש אם סוג מתאים למצב הסיום ועם התכולה שנצברה**
   2. **אחרת** 
      1. **דווח שגיאה לקסיקלית למדווח השגיאות והחזר אסימון חדש עם סוג שגיאה**

## **אלגוריתם המנתח התחבירי**

1. **אתחל את הטבלה**
2. **אתחל את חוקי הדקדוק**
3. **אתחל את קבוצת הFOLLOW**
4. **אתחל את המחסנית**
5. **הוסף את מצב 1 לראש המחסנית**
6. **עשה**
   1. **הסתכל על המצב שבראש המחסנית**
   2. **קבל את הפעולה לפי האסימון הנוכחי והמצב שבראש המחסנית בטבלת הACTION**
   3. **אם פעולה הינה SHIFT**
      1. **בצע פעולת SHIFT**
   4. **אחרת אם הפעולה הינה REDUCE**
      1. **בצע פעולת REDUCE**
7. **כל עוד הפעולה היא לא ACCEPT חזור ל 6**
8. **הוצא מהמחסנית את האיבר הראשון (שורש העץ)**
9. **החזר את השורש**

### Shift

1. **דחוף את האסימון למחסנית**
2. **שנה את המצב הנוכחי לפי המספר בפעולה**
3. **עבור לאסימון הבא**

### Reduce

1. **טעינת חוק הדקדוק התואם לפעולה**
2. **הוצאת מספר איברים לפי גודל חוק הדקדוק**
3. **ייצר צומת חדשה שכל ילדיה הם האיברים שהוצאנו והערך שלו הוא הסימן בצד שמאל של החוק**
4. **אם הצומת החדשה מסוג יצירת משתנה הוסף רשומה לטבלת הסימנים**
5. **דחוף את הצומת החדשה למחסנית**
6. **חשב את מצב החדש לפי סוג הצומת והצומת הקודמת עם טבלת הGOTO**
7. **דחוף את המספר המתקבל**

## אלגוריתם המנתח הסמנטי

1. **לכל צומת בעץ**
   1. **אם הצומת הינה מסוג השמה**
      1. **בדוק האם הסוגים בהשמה מתאימים**
      2. **בדוק בטבלת הסימנים האם הסימן קיים בScope הנוכחי**
   2. **אם הצומת מסוג קריאה לפונקציה**
      1. **בדוק כי הפונקציה קיימת**
      2. **בדוק כי הפרמטרים טועמים**
   3. **אם הצומת מסוג הכרזת פונקציה** 
      1. **בדוק שהפונקציה לא קיימת**
   4. **אם הצומת מסוג return statement**
      1. **בדוק שסוג הצומת תואם להחזרה של הפונקצייה**
   5. **לכל שאר הצמתים**
      1. **התאם להם טיפוס בהתאם**
      2. **אם ישנה שגיה - דווח**

## אלגוריתם מכולל קוד הסף

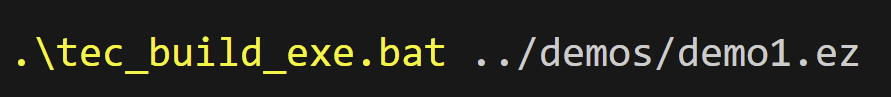
1. **לכל צומת בעץ**
   1. **ייצר קוד בהתאם**

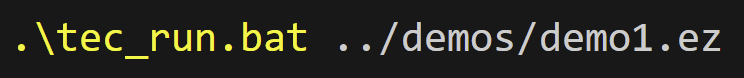
# תיאור ממשקים חיצוניים

קיימות שתי פקודות למשתמש:

**tec\_build\_exe.bat –** אשר יוצר קובץ exe מקובץ המקור הנתון לו

**tec\_run.bat** – אשר מריץ את קובץ המקור ומדפיס אותו למסך







# PlantUML Diagram

# תרשים מחלקות - UML

# מחלקות\פונקציות מרכזיות בפרויקט

## המחלקה DFA

מחלקה המייצגת אוטומט דטרמיניסטי סופי

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| DFA(int stateCount, vector<char> \*alphabet) | בנאי האוטומט, המאתחל אותו בעזרת מספר המצבים והשפה של האוטומט | O(1) | אתחל את מספר המצבים, ואת האלף-בית |
| DFA(string DFAConfigFile) | בנאי המאתחל את האוטומט לפי קובץ | O(n) | פירוק קובץ הconfig לחלקים ולהכניסן למטריצה |
| void insertTransition(int from, char alpha, int to) | מכניס מעבר למטריצה | O | **mat[from][char] = to** |

## המחלקה Lexer

מחלקה אשר ייעודה לחלק את קטע המקור אל אסימונים בעלי משמעות

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| Lexer(string srcFile, string DFAConfigFile, ErrorHandler \*handler) | בנאי הרכיב, הוא מקבל את קובץ המקור, קובץ הconfig של האוטומט ומצביע אל מדווח השגיאות | O(n) | שמור את שם קובץ המקור  אתחל את האוטומט  שמור את המצביע את מדווח השגיאות |
| SyntaxToken \*getNextToken() | מחזיר את האסימון הבא מהקלט | O(n) | פסואודו קוד מלא נכתב בפרק "אלגוריתם העל"  [כאן](#_אלגוריתם_המנתח_הלקסיקלי) |
| vector<SyntaxToken \*> getTokens(); | מחזיר את כל האסימונים מהקלט הנתון | O(n) | כל עוד לא הגענו לסוף הקובץ  הוסף לרשימה את האסימון הבא |

## המחלקה Parser

מחלקה אשר ייעודה לחלק את קטע המקור אל אסימונים בעלי משמעות

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| void shift(action currAction) | דוחף את האסימון הבא למחסנית | O(1) | צור חולייה בעץ בשביל האסימון  מצא את המספר של המצב שמתאים לאסימון לפי המספר הנתון מהטבלאה  אתחל את החוליה אם הנתונים האלו ודחוף למחסנית |
| void reduce(action currAction) | מצמצם את החוליות בראש המחסנית לחוליה אחת לפי כלל ייצור נתון | O(n) | מצא את הכלל לפי הפעולה הנתונה  הוצא מהמחסנית כמספר האיברים בצד ימין של הכלל  הפוך אותם לילים של החוליה החדשה  דחוף אותה למחסנית |
| ASTNode \*parse() | מחזיר את העץ ניתוח מופשט AST מהאסימונים הנוכחים | O(n) | פסואודו קוד מלא נכתב בפרק "אלגוריתם העל"  [כאן](#_אלגוריתם_המנתח_התחבירי) |

## המחלקה CodeGen

מחלקה אשר ייעודה לייצר את קוד הסף

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| void genCode() | ייצר את הקוד של התוכנית | O(n) | עבור כל חולייה יצר את הקוד שלה |
| int layoutLocals(scope \*s, int runningOffset) | מקצה זיכרון במחסנית למשתנים שבScope הנוכחי | O(n) | חשב לכל משתנה את הכתובת שלו  הוסף לoffset הכללי  עשה זאת לscopes הילדים |
| string getVarAddr(const &name) const; | מחזירה את הכתובת של המשתנה בזיכרון | O(n) | חפש בScope הנוכחי. אם המשתנה לא שם, חפש בScope האב |

## המחלקה Semantic

מחלקה אשר ייעודה הוא לבדוק את תקינות העץ התחבירי ולעדכן את טבלאת הסימנים

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| void updateSybolTable(ASTNode \*node) | מעדכן את טבלאת הסימנים ובודק שהסימן לא כבר קיים. אם כן ,ידווח על שגיאה | O(n) | לפי כל חולייה עדכן את הטבלאה כראוי |
| void updateScope(SyntaxToken\*currToken) | מעדכן את הscope הנוכחי לפי החולייה הנתונה | O(1) | אם הסימון הינו {   צור scope חדר  אם הסימן הינו }  צא מהscope הנוכחי |
| void assignNodeType(ASTNode\*node) | נותן לחולייה סוג ערך | O(1) | לכל חולייה בנפרד בחן אותה והבן איזה ערך היא צריכה לקבל |
| valType checkCompatibilityBinaryOp(valType leftOp, valType rightOp, SyntaxToken \*opToken) | בדוק האם שני סוגי הערכים הנתונים יכולים לתפקד ביחד עם הפעולה הנתונה | O(1) | זהה את אופרטור ההשוואה ואתה האופרנדים ובדוק את תיאומם |
| valType checkCompatibilityAssignExp(valType leftOp, valType rightOp, SyntaxToken \*opToken) | בדוק אם יכולה להתבצע השמה של צד ימיד לשמאל | O(1) | בדוק האם ההשמה יכולה להתבצע באזרת שני האופרנדים |
| void checkForMainFunction() | בודק את תקינות פונקציית הmain | O(1) | בדוק אם פונקציית הmain קיימת ומחזירה Int |
| void checkReturnStatements(functionEntry \*funcEntry) | בודק את תקינות משפטי ההחזרה מפונקציה | O(1) | בודק את תאימות מה שהמשפט מחזיר למה שהפונקציה מחזירה |

## המחלקה ErrorHandler

מחלקה אשר ייעודה לאגור רשימת שגיאות שיצטברו במהלך הקומפילציה

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| void addError(Error \*error) | הוספת שגיאה | O(1) | הוספת שגיאה למערך השגיאות |

## מחלקה Error

מחלקה אשר מייצגת שגיאה בקוד

למחלקה זאת משמשת כמחלקת אב להרבה מחלקות שגיאה אחרות אשר כל אחת מתייחסת לשגיאה מסוימת

|  |  |  |  |
| --- | --- | --- | --- |
| כותרת הפונקציה | הסבר | סיבוכיות | פסאודו קוד |
| Error(string body) | ייצר שגיאה חדשה | O(1) | אתחל את גוף השגיאה למה שניתן בקלט |
| virtual string toString() const = 0; | פונקציה אבסטרקטית אשר מחזירה את הייצוג המחרוזתי של השגיאה | O(1) | הפוך את השגיאה למחרוזת |

# מדריך למשתמש

אז איך משתמשים?

דבר ראשון צריך להתחיל לכתוב קוד! וכשהקוד מוכן, מצוין כל מה שיש לעשות הוא רק לקרוא לScript **tec\_run.bat** ופלט תוכנית תודפס למסך!



יש

תמונה שמכילה לוגו, גרפיקה, סמל, אומנות קליפיפם

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

# רפלקציה

אני יכול להגיד בלב שלב שהפרויקט הזה שיפר אותי כמתכנת וכתלמיד. הגעתי לפרויקט הזה מסקרנות מוחלטת ולא ידעתי למה אני מכניס את עצמי.

במסגרתי למדתי שלל דברים חדשים -

שפת תכנות חדשה, הבנתי שאני רוצה לבנות את הפרויקט בשפה שתיתן לי שליטת low-level אך הבנתי גם שאני רוצה גישה לעצמים. אז התשובה הייתה ברורה c++ אך ישנה בעיה אחת. לא ידעתי את השפה. למידת השפה לא רק הביאה לי כלים בפיתוח הפרויקט אך כלים בפיתוח ובעולם התוכנה בכללי. ++C הינה שפה אהובה מאוד בקרב מתכנתים ואנשי פיתוח ועקב זה יוצא לי להיתקל בא הרבה בין עם זה בהרצאות או סרטוני הסבר ועוד ועכשיו אני סוף סוף יכול להבין על מה הם מדברים. בנוסף לכך שנוספה עוד שפה לארגז הכלים שלי שזה תמיד דבר טוב

תאוריה, כמות התאוריה שאפשר ללמוד בתחום פיתוח הקומפיילרים הוא בלתי נגמר. זה נושא אחר נושא אחר נושא. הלמידה של דברים אלו פיתחה אותי כתלמיד, למדתי לספוג חומר מהר יותר וליישם אותו בצורה חלקה יותר והתעשרתי בידע רב בתחום הזה

אך נתקלתי גם בהרבה בעיות, היו הרבה פעמים שנתקעתי לפעמים אפילו לשבועות, אך תמיד הצלחתי למצוא את הטעות ולהמשיך אלאה עם זה בכוחות עצמי או בעזרה עם חברים. חוויה זו בהחלט נתנה לי כלים להמשיך גם אם יש כישלון ולא הכל נראה טוב

לסיכום. אני שמח שבחרתי בפרויקט זה והחוויה בפיתוחו הייתה מהנה ומעשירה.

# ביבליוגרפיה

**Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D.** (2006). *Compilers: Principles, techniques, and tools* (2nd ed.). Addison-Wesley.

**GeeksforGeeks.** (n.d.). *Compiler design tutorials*. <https://www.geeksforgeeks.org/compiler-design-tutorials/>

**Landwerth, I.** [terrajobst]. (n.d.). *Building a compiler* [Video playlist]. YouTube. <https://www.youtube.com/watch?v=wgHIkdUQbp0&list=PLRAdsfhKI4OWNOSfS7EUu5GRAVmze1t2y>

**Neso Academy.** (n.d.). *Compiler design* [Video playlist, 42 lessons]. YouTube. <https://www.youtube.com/playlist?list=PLBlnK6fEyqRjT3oJxFXRgjPNzeS-LFY-q>

**Thain, D.** (n.d.). *Compilers: Code generation* [Video]. YouTube. <https://www.youtube.com/watch?v=-ti07Z0xKKg&t=2778s&pp=ygUZY29tcGlsZXJzIGNvZGUgZ2VuYXJhdGlvbg%3D%3D>

## codeGen.cpp

#include "codeGen.hpp"  
   
 CodeGenarator::CodeGenarator(string outputFile, ASTNode \*root, SymbolTable \*symbolTable)  
 : \_outputFile(outputFile), \_root(root), \_functions(symbolTable->getFunctions()), \_currFunctionIndex(0) {}  
   
 void CodeGenarator::genCode()  
 {  
 floatMacroCodeGen();  
   
 \_outputFile << ".data\n";  
 \_outputFile << "format\_int db \"%d \", 10, 0\n";  
 \_outputFile << "format\_float db \"%lf \", 10, 0\n";  
 \_outputFile << "\n.code\n";  
 \_outputFile << "extern ExitProcess : proc\n";  
   
 \_outputFile << "extern printf : proc\n";  
   
 \_outputFile << "\nmain PROC\n";  
 \_outputFile << "\tcall function\_main\n";  
 \_outputFile << "\tmov ecx, eax\n";  
 \_outputFile << "\tcall ExitProcess\n";  
 \_outputFile << " main ENDP\n";  
 printIntFuncCodeGen();  
 printFloatFuncCodeGen();  
 vector<NonTerminalNode \*> funcDeclNodes = getFunctionDeclNodes((NonTerminalNode \*)\_root);  
 int j = 0;  
 for (int i = NUM\_OF\_BUILT\_IN\_FUNCTIONS; i < \_functions.size(); i++)  
 {  
 \_currFunctionIndex = i;  
   
 funcCodeGen(funcDeclNodes[j++]);  
 }  
   
 \_outputFile << "\n\nEND";  
 \_outputFile.close();  
 }  
   
 void CodeGenarator::funcCodeGen(NonTerminalNode \*node)  
 {  
 \_scopeStack.push(getCurrFunctionEntry()->getInnerScope());  
 string funcName = getCurrFunctionEntry()->getName();  
 \_outputFile << "\n"  
 << \_lableManager.getFunctionLable(funcName) << ":\n";  
 \_outputFile << "\tpush rbp\n";  
 \_outputFile << "\tmov rbp, rsp\n";  
   
 int offset = assignStackOffset(getCurrFunctionEntry()->getInnerScope());  
   
 \_outputFile << "\tsub rsp, " << offset << "\n";  
 loadFunctionVariables(getCurrFunctionEntry());  
   
 stmtListCodeGen((NonTerminalNode \*)node->GetChildren()[8]);  
   
 \_outputFile << \_lableManager.getFunctionEpilogueLable(funcName) << ":\n";  
 \_outputFile << "\tmov rsp, rbp\n";  
 \_outputFile << "\tpop rbp\n";  
 \_outputFile << "\tret\n";  
 \_scopeStack.pop();  
 }  
   
 void CodeGenarator::stmtListCodeGen(NonTerminalNode \*node)  
 {  
 vector<NonTerminalNode \*> children = getStmtNodes(node);  
 for (NonTerminalNode \*child : children)  
 {  
 stmtCodeGen((NonTerminalNode \*)child);  
 }  
 }  
   
 void CodeGenarator::stmtCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*firstChild = (NonTerminalNode \*)children[0];  
 if (firstChild->getNonTerminalKind() == SIMPLE\_STMT)  
 {  
 simpleStmtCodeGen(firstChild);  
 }  
   
 else if (firstChild->getNonTerminalKind() == IF\_STMT)  
 {  
 ifStmtCodeGen(firstChild);  
 }  
   
 else if (firstChild->getNonTerminalKind() == WHILE\_STMT)  
 {  
 whileStmtCodeGen(firstChild);  
 }  
   
 else if (firstChild->getNonTerminalKind() == FOR\_STMT)  
 {  
 forStmtCodeGen(firstChild);  
 }  
 }  
   
 void CodeGenarator::simpleStmtCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*firstChild = (NonTerminalNode \*)children[0];  
   
 if (isReturnStatement(node))  
 {  
 ASTNode \*exprOptNode = node->GetChildren()[1];  
 if (!exprOptNode->GetChildren().empty())  
 {  
 exprOptCodeGen((NonTerminalNode \*)exprOptNode);  
 \_outputFile << "\tmov rax, " << \_scratchManager.getName(exprOptNode->GetRegister()) << "\n";  
 \_scratchManager.free(exprOptNode->GetRegister());  
 }  
 \_outputFile << "\tjmp " << \_lableManager.getFunctionEpilogueLable(getCurrFunctionEntry()->getName()) << "\n";  
 \_outputFile << endl;  
 }  
   
 if (firstChild->getNonTerminalKind() == VAR\_DECL\_EXPR)  
 {  
 varDeclExprCodeGen(firstChild);  
 }  
   
 if (isFuncCall(node))  
 {  
 callExprCodeGen(node);  
 \_scratchManager.free(node->GetRegister());  
 }  
   
 if (firstChild->getNonTerminalKind() == ASSIGN\_EXPR)  
 {  
 assignExpressionCodeGen((NonTerminalNode \*)firstChild);  
   
 int reg = firstChild->GetRegister();  
 \_scratchManager.free(reg);  
 }  
 }  
   
 void CodeGenarator::varDeclExprCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> varDeclChildren = node->GetChildren();  
 NonTerminalNode \*initOptNode = (NonTerminalNode \*)varDeclChildren[2];  
   
 if (initOptNode->GetChildren().size() != 0)  
 {  
 NonTerminalNode \*assignValueNode = (NonTerminalNode \*)(initOptNode->GetChildren()[1]);  
 NonTerminalNode \*exprNode = (NonTerminalNode \*)(assignValueNode->GetChildren()[0]);  
   
 exprCodeGen(exprNode);  
 int reg = exprNode->GetRegister();  
 storeMem(getVarAddr(((TerminalNode \*)varDeclChildren[1])->getToken()->val), reg);  
 \_scratchManager.free(reg);  
 }  
 }  
   
 void CodeGenarator::ifStmtCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*conditionOpNode = (NonTerminalNode \*)children[2];  
   
 int elseLabel = \_lableManager.create();  
 int endLabel = \_lableManager.create();  
   
 conditionOptionCodeGen(conditionOpNode);  
 compareToZero(conditionOpNode->GetRegister());  
 \_outputFile << "\tjz " << \_lableManager.getName(elseLabel) << "\n";  
 bodyCodeGen((NonTerminalNode \*)children[4]);  
 \_outputFile << "\tjmp " << \_lableManager.getName(endLabel) << "\n";  
 \_outputFile << \_lableManager.getName(elseLabel) << ":\n";  
   
 if (children.size() == 7)  
 {  
 bodyCodeGen((NonTerminalNode \*)children[6]);  
 }  
   
 \_outputFile << \_lableManager.getName(endLabel) << ":\n";  
 }  
   
 void CodeGenarator::conditionOptionCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*firstChild = (NonTerminalNode \*)children[0];  
   
 if (firstChild->getNonTerminalKind() == EXPR)  
 {  
 exprCodeGen(firstChild);  
 }  
 else if (firstChild->getNonTerminalKind() == ASSIGN\_EXPR)  
 {  
 assignExpressionCodeGen(firstChild);  
 }  
   
 node->SetRegister(firstChild->GetRegister());  
 }  
   
 void CodeGenarator::whileStmtCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*conditionOpNode = (NonTerminalNode \*)children[2];  
   
 int startLabel = \_lableManager.create();  
 int endLabel = \_lableManager.create();  
   
 \_outputFile << \_lableManager.getName(startLabel) << ":\n";  
   
 conditionOptionCodeGen(conditionOpNode);  
   
 compareToZero(conditionOpNode->GetRegister());  
 \_outputFile << "\tjz " << \_lableManager.getName(endLabel) << "\n";  
 bodyCodeGen((NonTerminalNode \*)children[4]);  
 \_outputFile << "\tjmp " << \_lableManager.getName(startLabel) << "\n";  
 \_outputFile << \_lableManager.getName(endLabel) << ":\n";  
 }  
   
 void CodeGenarator::forStmtCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*forIntNode = (NonTerminalNode \*)children[2];  
 NonTerminalNode \*exprOpNode = (NonTerminalNode \*)children[4];  
 NonTerminalNode \*incrementOptNode = (NonTerminalNode \*)children[6];  
   
 int startLabel = \_lableManager.create();  
 int endLabel = \_lableManager.create();  
   
 forInitCodeGen(forIntNode);  
   
 \_outputFile << \_lableManager.getName(startLabel) << ":\n";  
   
 exprOptCodeGen(exprOpNode);  
 compareToZero(exprOpNode->GetRegister());  
 \_outputFile << "\tjz " << \_lableManager.getName(endLabel) << "\n";  
 bodyCodeGen((NonTerminalNode \*)children[8]);  
   
 forUpdateCodeGen(incrementOptNode);  
   
 \_outputFile << "\tjmp " << \_lableManager.getName(startLabel) << "\n";  
 \_outputFile << \_lableManager.getName(endLabel) << ":\n";  
 }  
   
 void CodeGenarator::forInitCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 if (children.size() != 0)  
 {  
 NonTerminalNode \*firstChild = (NonTerminalNode \*)children[0];  
 if (firstChild->getNonTerminalKind() == VAR\_DECL\_EXPR)  
 {  
 exprCodeGen(firstChild);  
 \_scratchManager.free(firstChild->GetRegister());  
 }  
 else if (firstChild->getNonTerminalKind() == ASSIGN\_EXPR)  
 {  
 assignExpressionCodeGen(firstChild);  
 }  
 }  
 }  
   
 void CodeGenarator::forUpdateCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 if (children.size() != 0)  
 {  
 NonTerminalNode \*firstChild = (NonTerminalNode \*)children[0];  
 if (firstChild->getNonTerminalKind() == ASSIGN\_EXPR)  
 {  
 assignExpressionCodeGen(firstChild);  
 }  
   
 else if (firstChild->getNonTerminalKind() == INCREMENT\_EXPR)  
 {  
 incrementExprCodeGen(firstChild);  
 }  
 }  
 }  
   
 void CodeGenarator::bodyCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*firstChild = (NonTerminalNode \*)children[0];  
   
 if (children.size() == 3)  
 {  
 stmtListCodeGen((NonTerminalNode \*)children[1]);  
 }  
 else if (children.size() == 2)  
 {  
 simpleStmtCodeGen((NonTerminalNode \*)children[0]);  
 }  
 else if (firstChild->getNonTerminalKind() == IF\_STMT)  
 {  
 ifStmtCodeGen((NonTerminalNode \*)firstChild);  
 }  
 else if (firstChild->getNonTerminalKind() == WHILE\_STMT)  
 {  
 whileStmtCodeGen((NonTerminalNode \*)firstChild);  
 }  
 else if (firstChild->getNonTerminalKind() == FOR\_STMT)  
 {  
 forStmtCodeGen((NonTerminalNode \*)firstChild);  
 }  
 }  
 void CodeGenarator::assignExpressionCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 NonTerminalNode \*leftExprNode = (NonTerminalNode \*)children[0];  
 NonTerminalNode \*rightExprNode = (NonTerminalNode \*)children[2];  
   
 exprCodeGen(rightExprNode);  
 int reg = rightExprNode->GetRegister();  
   
 if(leftExprNode->GetChildren().size() == 2)  
 {  
 storeMemPtrValue(getVarAddr(((TerminalNode \*)leftExprNode->GetChildren()[1])->getToken()->val), reg);  
 }else  
 {  
 storeMem(getVarAddr(((TerminalNode \*)leftExprNode->GetChildren()[0])->getToken()->val), reg);  
 }  
   
 node->SetRegister(reg);  
 }  
   
 void CodeGenarator::exprOptCodeGen(NonTerminalNode \*node)  
 {  
 if (!node->GetChildren().empty())  
 {  
 exprCodeGen(node->GetChildren()[0]);  
 node->SetRegister(node->GetChildren()[0]->GetRegister());  
 }  
 }  
   
 void CodeGenarator::exprCodeGen(ASTNode \*node)  
 {  
 ASTNode \*child = node->GetChildren()[0];  
 if (child->GetType() == NON\_TERMINAL) {  
 auto nt = (NonTerminalNode\*)child;  
 switch (nt->getNonTerminalKind()) {  
 case LOGICAL\_EXPR:  
 logicalExprCodeGen(nt);  
 break;  
 case ADDRESS\_EXPR:  
 addressExprCodeGen(nt);  
 break;  
 case PRIMARY\_EXPR:  
 primaryExprCodeGen(nt);  
 break;  
 }  
 }  
 node->SetRegister(child->GetRegister());  
 }  
   
   
 void CodeGenarator::logicalExprCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
   
 if (children.size() == 1)  
 {  
 relationalExprCodeGen((NonTerminalNode \*)children[0]);  
 node->SetRegister(children[0]->GetRegister());  
 return;  
 }  
   
 logicalExprCodeGen((NonTerminalNode \*)children[0]);  
 relationalExprCodeGen((NonTerminalNode \*)children[2]);  
   
 int leftReg = children[0]->GetRegister();  
 int rightReg = children[2]->GetRegister();  
   
 string left = \_scratchManager.getName(leftReg);  
 string right = \_scratchManager.getName(rightReg);  
   
 SyntaxKind opKind = ((TerminalNode \*)children[1])->getTerminalKind();  
   
 string resultReg = left;  
   
 // Compare both sides to zero  
 \_outputFile << "\tcmp " << left << ", 0\n";  
 \_outputFile << "\tsetne al\n";  
 \_outputFile << "\tmovzx " << left << ", al\n";  
   
 \_outputFile << "\tcmp " << right << ", 0\n";  
 \_outputFile << "\tsetne al\n";  
 \_outputFile << "\tmovzx " << right << ", al\n";  
   
 if (opKind == PIPE\_PIPE)  
 {  
 \_outputFile << "\tor " << left << ", " << right << "\n";  
 }  
 else if (opKind == AMPERSAND\_AMPERSAND)  
 {  
 \_outputFile << "\tand " << left << ", " << right << "\n";  
 }  
   
 node->SetRegister(leftReg);  
 \_scratchManager.free(rightReg);  
 }  
   
 void CodeGenarator::relationalExprCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 if (children.size() == 1)  
 {  
 addExprCodeGen((NonTerminalNode \*)children[0]);  
 node->SetRegister(children[0]->GetRegister());  
 return;  
 }  
   
 relationalExprCodeGen((NonTerminalNode \*)children[0]);  
 addExprCodeGen((NonTerminalNode \*)children[2]);  
   
 int leftReg = children[0]->GetRegister();  
 int rightReg = children[2]->GetRegister();  
   
 \_outputFile << "\tcmp " << \_scratchManager.getName(leftReg) << ", " << \_scratchManager.getName(rightReg) << "\n";  
   
 SyntaxKind opKind = ((TerminalNode \*)children[1])->getTerminalKind();  
 string instr;  
 if (opKind == LESS\_THAN)  
 instr = "setl";  
 else if (opKind == LESS\_THAN\_EQUALS)  
 instr = "setle ";  
 else if (opKind == GREATER\_THAN)  
 instr = "setg";  
 else if (opKind == GREATER\_THAN\_EQUALS)  
 instr = "setge";  
 else if (opKind == EQUALS\_EQUALS)  
 instr = "sete";  
 else if (opKind == BANG\_EQUALS)  
 instr = "setne";  
   
 \_outputFile << "\t" << instr << " al\n";  
 \_outputFile << "\tmovzx " << \_scratchManager.getName(leftReg) << ", al\n";  
   
 node->SetRegister(leftReg);  
 \_scratchManager.free(rightReg);  
 }  
   
 void CodeGenarator::addExprCodeGen(NonTerminalNode \*node)  
 {  
 auto &children = node->GetChildren();  
 if (children.size() == 1)  
 {  
 mulExprCodeGen((NonTerminalNode \*)children[0]);  
 node->SetRegister(children[0]->GetRegister());  
 return;  
 }  
   
 addExprCodeGen((NonTerminalNode \*)children[0]);  
 mulExprCodeGen((NonTerminalNode \*)children[2]);  
   
 int leftReg = children[0]->GetRegister();  
 int rightReg = children[2]->GetRegister();  
   
 SyntaxKind opKind = ((TerminalNode \*)children[1])->getTerminalKind();  
 string op;  
 if (opKind == PLUS)  
 op = "add";  
 else if (opKind == MINUS)  
 op = "sub";  
 else if (opKind == PIPE)  
 op = "or";  
 else if (opKind == CARET)  
 op = "xor";  
   
 \_outputFile << "\t" << op << " " << \_scratchManager.getName(leftReg) << ", " << \_scratchManager.getName(rightReg) << "\n";  
   
 node->SetRegister(leftReg);  
 \_scratchManager.free(rightReg);  
 }  
   
 void CodeGenarator::mulExprCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 if (children.size() == 1)  
 {  
 unaryExprCodeGen((NonTerminalNode \*)children[0]);  
 node->SetRegister(children[0]->GetRegister());  
 return;  
 }  
   
 mulExprCodeGen((NonTerminalNode \*)children[0]);  
 unaryExprCodeGen((NonTerminalNode \*)children[2]);  
   
 int leftReg = children[0]->GetRegister();  
 int rightReg = children[2]->GetRegister();  
   
 SyntaxKind opKind = ((TerminalNode \*)children[1])->getTerminalKind();  
 if (opKind == STAR)  
 {  
 \_outputFile << "\timul " << \_scratchManager.getName(leftReg) << ", " << \_scratchManager.getName(rightReg) << "\n";  
 }  
 else if (opKind == SLASH)  
 {  
 \_outputFile << "\tmov rax, " << \_scratchManager.getName(leftReg) << "\n";  
 \_outputFile << "\tcqo\n";  
 \_outputFile << "\tidiv " << \_scratchManager.getName(rightReg) << "\n";  
 \_outputFile << "\tmov " << \_scratchManager.getName(leftReg) << ", rax\n";  
 }  
 else if (opKind == AMPERSAND)  
 {  
 \_outputFile << "\tand " << \_scratchManager.getName(leftReg) << ", " << \_scratchManager.getName(rightReg) << "\n";  
 }  
   
 node->SetRegister(leftReg);  
 \_scratchManager.free(rightReg);  
 }  
   
 void CodeGenarator::unaryExprCodeGen(NonTerminalNode \*node)  
 {  
 vector<ASTNode \*> children = node->GetChildren();  
 if (children.size() == 1)  
 {  
 primaryExprCodeGen((NonTerminalNode \*)children[0]);  
 node->SetRegister(children[0]->GetRegister());  
 return;  
 }  
   
 unaryExprCodeGen((NonTerminalNode \*)children[1]);  
 int reg = children[1]->GetRegister();  
 node->SetRegister(reg);  
   
 SyntaxKind op = ((TerminalNode \*)children[0])->getTerminalKind();  
 if (op == MINUS)  
 neg(reg);  
 else if (op == BANG)  
 {  
 compareToZero(reg);  
 \_outputFile << "\tsete al\n";  
 \_outputFile << "\tmovzx " << \_scratchManager.getName(reg) << ", al\n";  
 }  
 }  
   
   
 void CodeGenarator::primaryExprCodeGen(NonTerminalNode \*node)  
 {  
 std::vector<ASTNode\*> children = node->GetChildren();  
 ASTNode \*first = children[0];  
   
 // 1) if it's a nested non-terminal for \* or ++/--, delegate immediately  
 if (first->GetType() == NON\_TERMINAL) {  
 NonTerminalNode \*nt = (NonTerminalNode\*) first;  
 if (nt->getNonTerminalKind() == DEREFERENCE\_EXPR) {  
 dereferenceExprCodeGen(nt);  
 node->SetRegister(nt->GetRegister());  
 return;  
 }  
 if (nt->getNonTerminalKind() == INCREMENT\_EXPR) {  
 incrementExprCodeGen(nt);  
 node->SetRegister(nt->GetRegister());  
 return;  
 }  
 }  
   
 TerminalNode \*tNode = (TerminalNode\*) first;  
 SyntaxToken \*tok = tNode->getToken();  
   
 // integer literal  
 if (tok->kind == INTEGER\_LITERAL) {  
 int r = \_scratchManager.alloc();  
 \_outputFile << "\tmov " << \_scratchManager.getName(r)  
 << ", " << tok->val << "\n";  
 node->SetRegister(r);  
 }  
 // float literal  
 else if (tok->kind == FLOAT\_LITERAL) {  
 int r = \_scratchManager.allocFloat();  
 \_outputFile << "\tmovsd " << \_scratchManager.getName(r)  
 << ", FP8(" << tok->val << ")\n";  
 node->SetRegister(r);  
 }  
 // simple variable load  
 else if (tok->kind == IDENTIFIER && children.size() == 1) {  
 int r = \_scratchManager.alloc();  
 loadMem(getVarAddr(tok->val), r);  
 node->SetRegister(r);  
 }  
 // parenthesized expression: ( Expr )  
 else if (tok->kind == OPEN\_PAREN && children.size() == 3) {  
 exprCodeGen(children[1]);  
 node->SetRegister(children[1]->GetRegister());  
 }  
 // function call: IDENTIFIER OPEN\_PAREN ExprList CLOSED\_PAREN  
 else if (isFuncCall(node)) {  
 callExprCodeGen(node);  
 }  
   
 }  
   
   
 void CodeGenarator::callExprCodeGen(NonTerminalNode \*node)  
 {  
 string funcName = ((TerminalNode \*)node->GetChildren()[0])->getToken()->val;  
 vector<NonTerminalNode \*> args = getFunctionCallArgsNodes((NonTerminalNode \*)node->GetChildren()[2]);  
   
 pushArgs(args);  
   
 \_outputFile << "\tcall " << \_lableManager.getFunctionLable(funcName) << "\n";  
   
 valType funcRetType = getCurrFunctionEntry()->getReturnType();  
 int reg;  
   
 if (funcRetType.type == FLOAT)  
 {  
 reg = \_scratchManager.allocFloat();  
 \_outputFile << "\tmovsd " << \_scratchManager.getName(reg) << ", xmm0\n";  
 }  
 else  
 {  
 reg = \_scratchManager.alloc();  
 \_outputFile << "\tmov " << \_scratchManager.getName(reg) << ", rax\n";  
 node->SetRegister(reg);  
 }  
   
 node->SetRegister(reg);  
 }  
   
 void CodeGenarator::pushArgs(vector<NonTerminalNode \*> args)  
 {  
 for (int i = args.size() - 1; i >= 0; i--)  
 {  
 NonTerminalNode \*arg = args[i];  
 exprCodeGen(arg);  
 push(arg->GetRegister());  
 \_scratchManager.free(arg->GetRegister());  
 }  
 }  
 void CodeGenarator::incrementExprCodeGen(NonTerminalNode \*node)  
 {  
 auto kids = node->GetChildren();  
 TerminalNode \*opNode = ((TerminalNode \*)kids[0])->getTerminalKind() != IDENTIFIER ? (TerminalNode \*)kids[0] : (TerminalNode \*)kids[1];  
 TerminalNode \*idNode = ((TerminalNode \*)kids[0])->getTerminalKind() == IDENTIFIER ? (TerminalNode \*)kids[0] : (TerminalNode \*)kids[1];  
 SyntaxKind op = opNode->getTerminalKind();  
   
 string opCommand = (op == PLUS\_PLUS) ? "inc" : "dec";  
 bool isPost = ((TerminalNode \*)kids[0])->getTerminalKind() == IDENTIFIER;  
   
 tableEntry varEntry = \_scopeStack.top()->getEntry(idNode->getToken()->val);  
 int reg = \_scratchManager.alloc();  
 int tempReg = \_scratchManager.alloc();  
 loadMem(varEntry.addr, tempReg);  
   
 if (isPost)  
 {  
 mov(\_scratchManager.getName(reg), \_scratchManager.getName(tempReg));  
 \_outputFile << "\t" << opCommand << " " << \_scratchManager.getName(tempReg) << "\n";  
 }  
 else  
 {  
 \_outputFile << "\t" << opCommand << " " << \_scratchManager.getName(tempReg) << "\n";  
 mov(\_scratchManager.getName(reg), \_scratchManager.getName(tempReg));  
 }  
   
 storeMem(varEntry.addr, tempReg);  
   
 \_scratchManager.free(tempReg);  
 node->SetRegister(reg);  
 }  
   
 void CodeGenarator::addressExprCodeGen(NonTerminalNode \*node)  
 {  
 std::string var = ((TerminalNode \*)node->GetChildren()[1])->getToken()->val;  
 std::string addr = getVarAddr(var);  
 int r = \_scratchManager.alloc();  
 \_outputFile << "\tlea " << \_scratchManager.getName(r) << ", " << addr << "\n";  
 node->SetRegister(r);  
 }  
   
 void CodeGenarator::dereferenceExprCodeGen(NonTerminalNode \*node)  
 {  
 int reg = \_scratchManager.alloc();  
 TerminalNode \*idNode = (TerminalNode \*)node->GetChildren()[1];  
   
 string idAddr = getVarAddr(idNode->getToken()->val);  
 loadMemPtrValue(idAddr,reg);  
   
 node->SetRegister(reg);  
 }  
   
 void CodeGenarator::castToFloat(int intReg, int xmmReg)  
 {  
 \_outputFile << "\tcvtsi2ss " << \_scratchManager.getName(xmmReg) << ", " << \_scratchManager.getName(intReg) << "\n";  
 }  
   
 int CodeGenarator::assignStackOffset(scope \*root)  
 {  
 int bytes = layoutLocals(root, 0) + 8; // add rbp  
 if (bytes & 0xF)  
 bytes = (bytes + 15) & ~0xF; // keep 16???byte align  
 return bytes;  
 }  
   
 int CodeGenarator::sizeOfType(const valType &t) const  
 {  
 int base = 8;  
 if (t.type == CHAR)  
 base = 1;  
   
 return base \* t.size; // arrays: len ?? elem  
 }  
   
 string CodeGenarator::nameOfType(const valType &t) const  
 {  
 string res = "QWORD";  
   
 if (t.type == CHAR)  
 res = "BYTE";  
   
 return res;  
 }  
   
 int CodeGenarator::layoutLocals(scope \*s, int offsetSoFar)  
 {  
 for (auto &e : s->getEntries())  
 {  
 int sizeInStack = sizeOfType(e.type);  
 offsetSoFar += sizeInStack;  
 e.addr = nameOfType(e.type) + " ptr [rbp - " + to\_string(offsetSoFar) + "]";  
 e.offset = offsetSoFar;  
 }  
 for (scope \*inner : s->getInnerScopes())  
 offsetSoFar = layoutLocals(inner, offsetSoFar);  
 return offsetSoFar;  
 }  
   
 void CodeGenarator::loadFunctionVariables(functionEntry \*func)  
 {  
 const vector<valType> &params = func->getParamTypes();  
 int startOfParams = 16;  
 int offset = 0;  
 int reg;  
   
 valType currParam;  
 stringstream dstName;  
 stringstream srcName;  
   
 for (size\_t i = 0; i < params.size(); i++)  
 {  
 currParam = params[i];  
 reg = \_scratchManager.alloc();  
   
 srcName << nameOfType(currParam) << " PTR [rbp+" << startOfParams + offset << "]";  
 dstName << nameOfType(currParam) << " PTR [rbp-" << offset + 8 << "]";  
   
 if (currParam.type == FLOAT)  
 reg = \_scratchManager.allocFloat();  
   
 loadFunctionVar(srcName.str(), dstName.str(), reg);  
   
 offset += SIZE\_OF\_STACK\_VAR;  
   
 \_scratchManager.free(reg);  
   
 // clear the string streams  
 srcName.str("");  
 srcName.clear();  
 dstName.str("");  
 dstName.clear();  
 }  
 }  
   
 void CodeGenarator::loadFunctionVar(string srcAddr, string dstAddr, int reg)  
 {  
 loadMem(srcAddr, reg);  
 storeMem(dstAddr, reg);  
 }  
   
 void CodeGenarator::push(int reg)  
 {  
 if (\_scratchManager.isFloat(reg))  
 {  
 \_outputFile << "\tsub rsp, 8\n";  
 \_outputFile << "\tmovsd QWORD PTR [rsp], " << \_scratchManager.getName(reg) << "\n";  
 }  
 else  
 {  
 \_outputFile << "\tpush " << \_scratchManager.getName(reg) << "\n";  
 }  
 }  
   
 void CodeGenarator::mov(string leftReg, string rightReg)  
 {  
 string movCommand = "mov ";  
 if (leftReg.find("xmm") != string::npos)  
 movCommand = "movsd ";  
   
 \_outputFile << "\t" << movCommand << leftReg << ", " << rightReg << "\n";  
 }  
   
   
 void CodeGenarator::neg(int reg)  
 {  
 if (\_scratchManager.isFloat(reg))  
 {  
 // -x ==> 0 - x  
   
 int zeroReg = \_scratchManager.allocFloat();  
 \_outputFile << "xorps " << \_scratchManager.getName(zeroReg) << ", " << \_scratchManager.getName(zeroReg) << "\n";  
 \_outputFile << "\tpsubd " << \_scratchManager.getName(zeroReg) << ", " << \_scratchManager.getName(reg) << "\n";  
 \_outputFile << "\tmovsd " << \_scratchManager.getName(reg) << ", " << \_scratchManager.getName(zeroReg) << "\n";  
 \_scratchManager.free(zeroReg);  
 }  
 else  
 {  
 \_outputFile << "\tneg " << \_scratchManager.getName(reg) << "\n";  
 }  
 }  
   
 void CodeGenarator::compareToZero(int reg)  
 {  
 if (\_scratchManager.isFloat(reg))  
 {  
 int zeroReg = \_scratchManager.allocFloat();  
 \_outputFile << "xorps " << \_scratchManager.getName(zeroReg) << ", " << \_scratchManager.getName(zeroReg) << "\n";  
 \_outputFile << "\tcomisd " << \_scratchManager.getName(reg) << ", " << \_scratchManager.getName(zeroReg) << "\n";  
 \_scratchManager.free(zeroReg);  
 }  
 else  
 {  
 \_outputFile << "\tcmp " << \_scratchManager.getName(reg) << ", 0\n";  
 }  
 }  
   
 void CodeGenarator::loadMemPtrValue(string srcAddr, int reg)  
 {  
 string movCommand = "mov ";  
 string regName = \_scratchManager.getName(reg);  
   
   
 if (srcAddr.find("BYTE") != string::npos)  
 regName = \_scratchManager.getLowerByteName(reg);  
   
   
 \_outputFile << "\t" << "mov " << regName << ", " << srcAddr << "\n";  
 \_outputFile << "\t" << "mov " << regName << ", [" << regName << "]\n";   
 }  
   
 void CodeGenarator::loadMem(string srcAddr, int reg)  
 {  
 string movCommand = "mov ";  
   
 if (\_scratchManager.isFloat(reg))  
 movCommand = "movsd ";  
   
 if (srcAddr.find("BYTE") != string::npos)  
 movCommand = "movzx ";  
   
 \_outputFile << "\t" << movCommand << \_scratchManager.getName(reg) << ", " << srcAddr << "\n";  
 }  
   
 void CodeGenarator::storeMemPtrValue(string srcAddr, int reg)  
 {  
 string movCommand = "mov ";  
 string regName = \_scratchManager.getName(reg);  
   
 if (\_scratchManager.isFloat(reg))  
 movCommand = "movsd ";  
   
 if (srcAddr.find("BYTE") != string::npos)  
 regName = \_scratchManager.getLowerByteName(reg);  
   
 int tempReg = \_scratchManager.alloc();  
 loadMem(srcAddr, tempReg);  
   
 string targetAddr = "QWORD PTR [" + \_scratchManager.getName(tempReg) + "]";  
   
 storeMem(targetAddr, reg);  
 \_scratchManager.free(tempReg);  
 }  
 void CodeGenarator::storeMem(string srcAddr, int reg)  
 {  
 string movCommand = "mov ";  
 string regName = \_scratchManager.getName(reg);  
   
 if (\_scratchManager.isFloat(reg))  
 movCommand = "movsd ";  
   
 if (srcAddr.find("BYTE") != string::npos)  
 regName = \_scratchManager.getLowerByteName(reg);  
   
 \_outputFile << "\t" << movCommand << srcAddr << ", " << regName << "\n";  
 }  
   
 string CodeGenarator::getVarAddr(const string &name) const  
 {  
 scope \*it = \_scopeStack.top();  
 string res = "";  
 while (it)  
 {  
 for (const auto &e : it->getEntries())  
 if (e.name == name && !e.addr.empty())  
 res = e.addr;  
 it = it->getParentScope();  
 }  
   
 return res;  
 }  
   
 void CodeGenarator::printIntFuncCodeGen()  
 {  
 \_outputFile << "function\_printInt:\n";  
 \_outputFile << "\tpush rbp\n";  
 \_outputFile << "\tmov rbp, rsp\n";  
 \_outputFile << "\tsub rsp, 16\n"; // stack space for alignment  
   
 \_outputFile << "\tsub rsp, 32\n"; // shadow space (Windows x64 ABI)  
   
 \_outputFile << "\tmov rdx, QWORD PTR [rbp+16]\n"; // parameter -> rdx  
 \_outputFile << "\tlea rcx, format\_int\n"; // format string -> rcx  
 \_outputFile << "\tcall printf\n";  
   
 \_outputFile << "\tadd rsp, 32\n";  
 \_outputFile << "\tmov eax, 0\n";  
 \_outputFile << "\tmov rsp, rbp\n";  
 \_outputFile << "\tpop rbp\n";  
 \_outputFile << "\tret\n";  
 }  
   
 void CodeGenarator::printFloatFuncCodeGen()  
 {  
 \_outputFile << "function\_printFloat:\n";  
 \_outputFile << "\tpush rbp\n";  
 \_outputFile << "\tmov rbp, rsp\n";  
   
 \_outputFile << "\tsub rsp, 32\n"; // Allocate shadow space for Windows x64 ABI  
   
 \_outputFile << "\tmovsd xmm1, QWORD PTR [rbp+16]\n"; // Move float parameter into xmm1  
 \_outputFile << "\tlea rcx, format\_float\n"; // Load address of format string into rcx  
 \_outputFile << "\tcall printf\n";  
   
 \_outputFile << "\tadd rsp, 32\n";  
 \_outputFile << "\tmov eax, 0\n";  
 \_outputFile << "\tmov rsp, rbp\n";  
 \_outputFile << "\tpop rbp\n";  
 \_outputFile << "\tret\n";  
 }  
   
 void CodeGenarator::floatMacroCodeGen()  
 {  
 \_outputFile << "; Floating-point constant macros\n";  
   
 // 64-bit double macro  
 \_outputFile << "FP8 MACRO value\n";  
 \_outputFile << " LOCAL vname\n";  
 \_outputFile << " .const\n";  
 \_outputFile << " align 8\n";  
 \_outputFile << " vname REAL8 value\n";  
 \_outputFile << " .code\n";  
 \_outputFile << " EXITM <vname>\n";  
 \_outputFile << "ENDM\n\n";  
 }

## codeGen.hpp

#ifndef CODEGEN\_HPP  
 #define CODEGEN\_HPP  
   
 #include <iostream>  
 #include <fstream>  
 #include <vector>  
 #include <stack>  
 #include <string>  
 #include <sstream>  
 #include "../nodes/nodes.hpp"  
 #include "../nodeAnalyzer/nodeAnalyzer.hpp"  
 #include "../symbolTable/symbolTable.hpp"  
 #include "../symbolTable/functionEntry/functionEntry.hpp"  
 #include "../symbolTable/tableEntry/tableEntry.hpp"  
 #include "scratch/scratch.hpp"  
 #include "lableManager/lableManager.hpp"  
 #include <stack>  
 #include <fstream>  
   
 #define NUM\_OF\_BUILT\_IN\_FUNCTIONS 2  
 #define SIZE\_OF\_STACK\_VAR 8  
   
 using namespace std;  
   
 class CodeGenarator  
 {  
 private:  
 ofstream \_outputFile;  
 ASTNode \*\_root;  
 vector<functionEntry \*> \_functions;  
 int \_currFunctionIndex;  
 stack<scope \*> \_scopeStack;  
   
 ScratchManager \_scratchManager;  
 LabelManager \_lableManager;  
   
 functionEntry \*getCurrFunctionEntry() { return \_functions[\_currFunctionIndex]; }  
 int assignStackOffset(scope \*currScope);  
   
 void funcCodeGen(NonTerminalNode \*node);  
 void stmtListCodeGen(NonTerminalNode \*node);  
 void stmtCodeGen(NonTerminalNode \*node);  
 void simpleStmtCodeGen(NonTerminalNode \*node);  
 void varDeclExprCodeGen(NonTerminalNode \*node);  
 void ifStmtCodeGen(NonTerminalNode \*node);  
 void conditionOptionCodeGen(NonTerminalNode \*node);  
 void whileStmtCodeGen(NonTerminalNode \*node);  
 void forStmtCodeGen(NonTerminalNode \*node);  
 void forInitCodeGen(NonTerminalNode \*node);  
 void forUpdateCodeGen(NonTerminalNode \*node);  
 void bodyCodeGen(NonTerminalNode \*node);  
 void assignExpressionCodeGen(NonTerminalNode \*node);  
 void exprOptCodeGen(NonTerminalNode \*node);  
 void exprCodeGen(ASTNode \*node);  
 void addExprCodeGen(NonTerminalNode \*node);  
 void mulExprCodeGen(NonTerminalNode \*node);  
 void unaryExprCodeGen(NonTerminalNode \*node);  
 void primaryExprCodeGen(NonTerminalNode \*node);  
 void callExprCodeGen(NonTerminalNode \*node);  
 void logicalExprCodeGen(NonTerminalNode \*node);  
 void relationalExprCodeGen(NonTerminalNode \*node);  
 void pushArgs(vector<NonTerminalNode \*> args);  
 void incrementExprCodeGen(NonTerminalNode \*node);  
 void addressExprCodeGen(NonTerminalNode \*node);  
 void dereferenceExprCodeGen(NonTerminalNode \*node);  
   
 void castToFloat(int intReg, int xmmReg);  
   
 int sizeOfType(const valType &t) const;  
 string nameOfType(const valType &t) const;  
 int layoutLocals(scope \*s, int runningOffset);  
 void loadFunctionVariables(functionEntry \*func);  
 void loadFunctionVar(string srcAddr, string dstAddr, int reg);  
 void push(int reg);  
 void mov(string leftReg, string rightReg);  
 void neg(int reg);  
 void compareToZero(int reg);  
 void loadMemPtrValue(string srcAddr, int reg);  
 void loadMem(string srcAddr, int reg);  
 void storeMemPtrValue(string srcAddr, int reg);  
 void storeMem(string srcAddr, int reg);  
 string getVarAddr(const string &name) const;  
   
 void printIntFuncCodeGen();  
 void printFloatFuncCodeGen();  
 void floatMacroCodeGen();  
   
 public:  
 CodeGenarator(string outputFile, ASTNode \*root, SymbolTable \*symbolTable);  
 void genCode();  
 };  
   
 #endif

## lableManager.cpp

#include "lableManager.hpp"  
   
 using namespace std;  
   
 LabelManager::LabelManager() : \_lableNum(0) {}  
   
 int LabelManager::create()  
 {  
 return \_lableNum++;  
 }  
   
 string LabelManager::getName(int lableNum)  
 {  
 stringstream res;  
 res << "L" << lableNum;  
 return res.str();  
 }  
   
 string LabelManager::getFunctionLable(string funcName)  
 {  
 return "function\_" + funcName ;  
 }  
   
 string LabelManager::getFunctionPreambleLable(string funcName)  
 {  
 return "function\_" + funcName + "\_preamble";  
 }  
   
 string LabelManager::getFunctionEpilogueLable(string funcName)  
 {  
 return "function\_" + funcName + "\_epilogue";  
 }

## lableManager.hpp

#ifndef \_\_LABLE\_MANAGER  
 #define \_\_LABLE\_MANAGER  
   
 #include <sstream>  
 #include <string>  
   
 using namespace std;  
   
 class LabelManager  
 {  
 private:  
 int \_lableNum;  
   
 public:  
 LabelManager();  
 int create();  
 string getFunctionLable(string funcName);  
 string getFunctionPreambleLable(string funcName);  
 string getFunctionEpilogueLable(string funcName);  
 string getName(int lableNum);  
 };  
   
 #endif

## scratch.cpp

#include "scratch.hpp"  
 #include <iostream>  
   
 using namespace std;  
   
 ScratchManager::ScratchManager()  
 {  
 initRegArr();  
 }  
   
 void ScratchManager::initRegArr()  
 {  
 int regIndex = 0;  
   
 // int/char registers  
 \_regArr[regIndex++] = scratchRegister{"rbx", false};  
 \_regArr[regIndex++] = scratchRegister{"r10", false};  
 \_regArr[regIndex++] = scratchRegister{"r11", false};  
 \_regArr[regIndex++] = scratchRegister{"r12", false};  
 \_regArr[regIndex++] = scratchRegister{"r13", false};  
 \_regArr[regIndex++] = scratchRegister{"r14", false};  
 \_regArr[regIndex++] = scratchRegister{"r15", false};  
   
 // float registers  
 \_regArr[regIndex++] = scratchRegister{"xmm1", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm2", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm3", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm4", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm5", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm6", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm7", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm8", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm9", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm10", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm11", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm12", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm13", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm14", false};  
 \_regArr[regIndex++] = scratchRegister{"xmm15", false};  
 }  
   
 // allocates a new scratch register  
 int ScratchManager::alloc()  
 {  
 int res = -1;  
 for (int i = 0; i < NUM\_OF\_SCRATCH\_REGISTERS; i++)  
 {  
 if (!\_regArr[i].inUse && res == -1)  
 res = i;  
 }  
   
 if (res == -1)  
 {  
 cerr << "Fatal error: out of registers";  
 exit(-1);  
 }  
   
 \_regArr[res].inUse = true;  
 return res;  
 }  
   
 // allocates a new scratch register  
 int ScratchManager::allocFloat()  
 {  
 int res = -1;  
 for (int i = NUM\_OF\_SCRATCH\_REGISTERS; i < NUM\_OF\_SCRATCH\_REGISTERS + NUM\_OF\_FLOAT\_SCRATCH\_REGISTERS; i++)  
 {  
 if (!\_regArr[i].inUse && res == -1)  
 res = i;  
 }  
   
 if (res == -1)  
 {  
 cerr << "Fatal error: out of registers";  
 exit(-1);  
 }  
   
 \_regArr[res].inUse = true;  
 return res;  
 }  
   
 bool ScratchManager::isFloat(int i)  
 {  
 return i >= NUM\_OF\_SCRATCH\_REGISTERS && i < NUM\_OF\_FLOAT\_SCRATCH\_REGISTERS;  
 }  
   
 // frees an occupied register  
 void ScratchManager::free(int i)  
 {  
 if (i >= 0 && i < NUM\_OF\_FLOAT\_SCRATCH\_REGISTERS)  
 \_regArr[i].inUse = false;  
 }  
   
 // returns the name of the register  
 string ScratchManager::getName(int i)  
 {  
 string res;  
   
 if (i >= 0 && i < NUM\_OF\_FLOAT\_SCRATCH\_REGISTERS)  
 res = \_regArr[i].name;  
   
 return res;  
 }  
   
 string ScratchManager::getLowerByteName(int i)  
 {  
 string res = \_regArr[i].name;  
   
 // rbx  
 if (res[1] == 'b')  
 {  
 res = "bl";  
 }  
 else  
 {  
 res = res + "l";  
 }  
   
 return res;  
 }

## scratch.hpp

#ifndef \_\_SCRATCH  
 #define \_\_SCRATCH  
   
 #define NUM\_OF\_SCRATCH\_REGISTERS 7  
 #define NUM\_OF\_FLOAT\_SCRATCH\_REGISTERS 16  
   
 #include <string>  
   
 using namespace std;  
 struct scratchRegister  
 {  
 string name;  
 bool inUse;  
 };  
   
 class ScratchManager  
 {  
 private:  
 scratchRegister \_regArr[NUM\_OF\_SCRATCH\_REGISTERS + NUM\_OF\_FLOAT\_SCRATCH\_REGISTERS];  
 void initRegArr();  
   
 public:  
 ScratchManager();  
 int alloc();  
 int allocFloat();  
 bool isFloat(int i);  
 void free(int i);  
 string getName(int i);  
 string getLowerByteName(int i);  
 bool isAllocated(int i) { return \_regArr[i].inUse; }  
 };  
   
 #endif

## DFA.cpp

#include "../DFA/DFA.hpp"  
 #include <fstream>  
 #include <iomanip>  
 #include <string>  
 using namespace std;  
   
 DFA::DFA(string DFAConfigFile)  
 {  
 \_stateCount = 0;  
   
 ifstream file(DFAConfigFile);  
 string currLine;  
 char TransitionLine[13];  
 stringstream ss;  
 int currentNum;  
 int from;  
 char currSymbol[2];  
 int to;  
   
 if (!file.is\_open())  
 {  
 throw runtime\_error("Failed to open file " + DFAConfigFile);  
 }  
   
 // initialising states  
 getline(file, currLine);  
 ss.clear();  
 ss.str(currLine);  
 ss >> \_stateCount;  
   
 // initialising start state  
 getline(file, currLine);  
 ss.clear();  
 ss.str(currLine);  
 ss >> currentNum;  
 \_startState = currentNum;  
   
 // initialising end states  
 getline(file, currLine);  
 ss.clear();  
 ss.str(currLine);  
   
 while (ss >> currentNum)  
 {  
 \_endStates.push\_back(currentNum);  
 }  
   
 // initialising alphabet  
 int newLineCount = 0;  
 while (file.read(currSymbol, 2) && !(currSymbol[0] == '$' && currSymbol[1] == '$'))  
 {  
 addAlpha(currSymbol[0]);  
 }  
   
 getline(file, currLine);  
   
 // initialising matrix  
 initMatrix();  
   
 // initialising transitions  
 while (file.read(TransitionLine, 12))  
 {  
 TransitionLine[12] = '\0';  
 currLine.assign(TransitionLine);  
 insertTransitionString(currLine);  
 }  
 }  
   
 DFA::DFA(int stateCount, vector<char> \*alphabet)  
 {  
 \_stateCount = stateCount;  
   
 for (char alpha : \*alphabet)  
 {  
 addAlpha(alpha);  
 }  
   
 initMatrix();  
 }  
   
 DFA::DFA() : \_mat(nullptr), \_stateCount(0), \_startState(0) {}  
   
 // adding new symbol to alphabet  
 void DFA::addAlpha(char alpha)  
 {  
 isSymbolExistsErr(alpha, false);  
   
 int nextIndex = \_alphabetToIndex.size();  
 \_alphabetToIndex[alpha] = nextIndex;  
 }  
   
 void DFA::setStateCount(int numOfStates)  
 {  
 if (numOfStates > 0)  
 {  
 \_stateCount = numOfStates;  
 }  
 else  
 {  
 throw runtime\_error("invalid number of states!");  
 }  
 }  
   
 // sets a start state  
 // if the state dosent exists the function throws an error  
 void DFA::setStartState(int state)  
 {  
 isStateExistsWErr(state, true);  
   
 \_startState = state;  
 }  
   
 // returns start state  
 int DFA::getStartState() const  
 {  
 return \_startState;  
 }  
   
 // adding an end state  
 // if the state dosent exists the function throws an error  
 void DFA::addEndState(int state)  
 {  
 //if the number is not a state return an error  
 isStateExistsWErr(state, true);  
   
 \_endStates.push\_back(state);  
 }  
   
 //returns a read only vector of end states  
 const vector<int> &DFA::getEndStates() const  
 {  
 return \_endStates;  
 }  
   
 //returns a vector of the alphabet  
 vector<char> DFA::getAlphabet() const  
 {  
 vector<char> alphabet;  
   
 for (const auto& pair : \_alphabetToIndex) {  
 alphabet.push\_back(pair.first);  
 }  
   
 return alphabet;  
 }  
   
 // initialising an empty dfa sized according to alphabet and number of states  
 void DFA::initMatrix()  
 {  
 int length = \_stateCount;  
 int width = \_alphabetToIndex.size();  
 if (length > 0 && width > 0)  
 {  
 \_mat = new int \*[length];  
 for (int i = 0; i < length; i++)  
 {  
 \_mat[i] = new int[width];  
 }  
   
 // initing all transitions to -1 (trap state)  
 for (int i = 0; i < length; i++)  
 {  
 for (int j = 0; j < width; j++)  
 {  
 \_mat[i][j] = -1;  
 }  
 }  
 }  
 }  
   
 //inserts a new transition to the table  
 //if the transition with (from,alpha) allready exists it overrides it   
 void DFA::insertTransition(int from, char alpha, int to)  
 {   
 isStateExistsWErr(from, true);  
   
 // check if alpha exists in alphabet  
 isSymbolExistsErr(alpha, true);  
   
 // check if the state exists  
 isStateExistsWErr(to, true);  
   
 // adding to the transition matrix  
 int alphaIndex = \_alphabetToIndex[alpha];  
   
 \_mat[from][alphaIndex] = to;  
 }  
   
 void DFA::insertTransitionString(string &transition)  
 {  
 int from;  
 char alpha;  
 int to;  
   
 from = stoi(transition.substr(0, 4)); // convert ascii to int  
 alpha = transition[5];  
 to = stoi(transition.substr(7, 4)); // convert ascii to int  
   
 insertTransition(from, alpha, to);  
 }  
   
 int DFA::getState(int state, char alpha) const  
 {  
 // check if the state exists  
 isStateExistsWErr(state, true);  
   
 // check if alpha exists in alphabet go to trap state  
 if (!isSymbolExists(alpha))  
 return -1;  
   
 int res = \_mat[state][\_alphabetToIndex.at(alpha)];  
   
 return res;  
 }  
   
 // Helper function to check if the state exists, returns an error message if not  
 bool DFA::isStateExsists(int state) const  
 {  
 return state < \_stateCount;  
 }  
   
 // Helper function to check if the symbol exists, returns an error message if not  
 bool DFA::isSymbolExists(char alpha) const  
 {  
 auto alphabetIt = \_alphabetToIndex.find(alpha);  
 return (alphabetIt != \_alphabetToIndex.end());  
 }  
   
 // checks if the state exists and compare the result with the wantedResult  
 // if there is no matching it prints an corresponding error  
 void DFA::isStateExistsWErr(int state, bool wantedResult) const  
 {  
 if (isStateExsists(state) != wantedResult)  
 {  
 stringstream ss;  
 ss << "state " << state << " does not exist!";  
 throw runtime\_error(ss.str());  
 }  
 }  
   
 // checks if the symbol exists and compare the result with the wantedResult  
 // if there is no matching it prints an corresponding error  
 void DFA::isSymbolExistsErr(char alpha, bool wantedResult) const  
 {  
 if (isSymbolExists(alpha) != wantedResult)  
 {  
 stringstream ss;  
 ss << "Symbol " << alpha  
 << (wantedResult ? "does not" : "allready exists")  
 << " in the alphabet!";  
 throw runtime\_error(ss.str());  
 }  
 }  
   
 void DFA::printMatrix() const  
 {  
 unordered\_map<int, char> indexToAlphabet;  
 for (const auto &pair : \_alphabetToIndex)  
 {  
 indexToAlphabet[pair.second] = pair.first; // Reverse key-value  
 }  
   
 // Print the alphabet (column headers)  
 cout << "Symbols: ";  
 for (int i = 0; i < indexToAlphabet.size(); i++)  
 {  
 cout << indexToAlphabet.at(i) <<'\t'<< " ";  
 }  
 cout << endl;  
   
 int length = \_stateCount;  
 int width = \_alphabetToIndex.size();  
 // Print each row (transition for each state)  
 for (int i = 0; i < length; i++)  
 {  
 cout << " State " << i << ": "; // Print the state index at the start of the row  
 for (int j = 0; j < width; j++)  
 {  
 // Print transitions, replacing -1 (trap states) with 'T'  
 if (\_mat[i][j] == -1)  
 {  
 cout << '\t' <<'T';  
 }  
 else  
 {  
 cout << '\t' <<\_mat[i][j];  
 }  
 cout << " ";  
 }  
 cout << endl;  
 }  
 }  
   
 void DFA::writeDFAToFile(string dstFile)  
 {  
 ofstream destf(dstFile);  
   
 if (!destf)  
 {  
 std::cerr << "Error opening file: " << dstFile << std::endl;  
 return;  
 }  
   
 destf << \_stateCount << endl; // write state count  
   
 destf << \_startState << endl; // write start state  
   
 // write end states  
 for (int i = 0; i < \_endStates.size(); i++)  
 {  
 destf << \_endStates[i] << " ";  
 }  
   
 destf << endl;  
   
 // write alphabet  
 unordered\_map<int, char> indexToAlphabet;  
 for (const auto &pair : \_alphabetToIndex)  
 {  
 indexToAlphabet[pair.second] = pair.first; // Reverse key-value  
 }  
   
 for (int i = 0; i < indexToAlphabet.size(); i++)  
 {  
 destf << indexToAlphabet.at(i) << " ";  
 }  
 destf << "$$" << endl;  
   
 // write transitions  
 for (int i = 0; i < \_stateCount; i++)  
 {  
 for (int j = 0; j < indexToAlphabet.size(); j++)  
 {  
 int from = i;  
 char alpha = indexToAlphabet.at(j);  
 int to = getState(from, alpha);  
   
 if (to != -1)  
 {  
 destf << left << setw(4) << setfill(' ') << i << " "  
 << alpha << " "  
 << left << setw(4) << setfill(' ') << getState(i, alpha) << endl;  
 }  
 }  
 }  
   
 destf.close();  
 }  
   
 // checks if a certain word is in the language  
 // returns a pair of values  
 // val 1-> is the word in the language  
 // val 2-> ending state  
 pair<bool, int> DFA::inLanguage(string &word) const  
 {  
 int currState = \_startState;  
 int i = 0;  
   
 while (currState != -1 && i < word.length())  
 {  
 currState = getState(currState, word[i]);  
 i++;  
 }  
   
 bool isInLang = find(\_endStates.begin(), \_endStates.end(), currState) != \_endStates.end();  
 return make\_pair(isInLang, currState);  
 }

## DFA.hpp

#ifndef \_\_DFA  
 #define \_\_DFA  
   
 #include <string>  
 #include <vector>  
 #include <sstream>  
 #include <algorithm>  
 #include <iostream>  
 #include <unordered\_map>  
   
 using namespace std;  
   
 class DFA  
 {  
 private:  
 int \*\*\_mat;  
 int \_stateCount;  
 int \_startState;  
   
 vector<int> \_endStates;  
   
 unordered\_map<char, int> \_alphabetToIndex;  
   
 void isStateExistsWErr(int state, bool wantedResult) const;  
 void isSymbolExistsErr(char alpha, bool wantedResult) const;  
   
 public:  
 DFA(int stateCount, vector<char> \*alphabet);  
 DFA(string DFAConfigFile);  
 DFA();  
 void addAlpha(char alpha);  
 void setStateCount(int numOfStates);  
 void setStartState(int state);  
 int getStartState() const;  
 void addEndState(int state);  
 const vector<int> &getEndStates() const;  
 vector<char> getAlphabet() const;  
 int getState(int state, char alpha) const;  
 void initMatrix();  
 void insertTransition(int from, char alpha, int to);  
 void insertTransitionString(string &transition);  
 bool isStateExsists(int state) const;  
 bool isSymbolExists(const char alpha) const;  
 pair<bool, int> inLanguage(string &word) const;  
 void writeDFAToFile(string dstFile);  
 void printMatrix() const;  
 };  
   
 #endif

## errorHandler.cpp

#include "../errorHandler/errorHandler.hpp"  
   
 int ErrorHandler::getErrorCount() const  
 {  
 return \_errors.size();  
 }  
   
 void ErrorHandler::addError(Error \*error)  
 {  
 \_errors.push\_back(error);  
 }  
   
 void ErrorHandler::printErrors()  
 {  
 for (int i = 0; i < \_errors.size(); i++)  
 {  
 cout << \_errors[i] -> toString() << endl;  
 }  
 }

## errorHandler.hpp

#ifndef \_\_ERROR\_HANDLER  
 #define \_\_ERROR\_HANDLER  
   
 class ErrorHandler;  
   
 #include "../errors/errors.hpp"  
 #include <vector>  
 #include <iostream>  
   
 class ErrorHandler  
 {  
 private:  
 vector<Error\*> \_errors;  
   
 public:  
 ErrorHandler() {}  
   
 int getErrorCount() const;  
   
 void addError(Error \*error);  
   
 void printErrors();  
   
 };  
   
 #endif

## errors.hpp

#ifndef \_\_ERRORS  
 #define \_\_ERRORS  
   
 #include "../token/token.hpp"  
 #include <string>  
 using namespace std;  
   
 class Error  
 {  
 protected:  
 string \_body;  
   
 public:  
 Error(string body) : \_body(body) {}  
 Error() : \_body("") {}  
   
 virtual string toString() const = 0;   
 };  
   
 class SyntaxError : public Error  
 {  
 private:  
 int \_line;  
 int \_column;  
   
 public:  
 SyntaxError(string body, int line, int column) : Error(body), \_line(line), \_column(column) {}  
   
 string toString() const override  
 {  
 return "Syntax Error: " + \_body + " in " + to\_string(\_line) + ":" + to\_string(\_column);  
 }  
 };  
   
 class SyntacticError : public Error  
 {  
 public:  
 SyntacticError(SyntaxToken \* token) {  
 int line = token -> line;  
 int column = token ->column;  
   
 \_body = "unvalid placment of token " + syntaxTokenToString(\*token) + "on" + "{" + to\_string(line) +":" + to\_string(column) + "}\n";  
 }  
   
 SyntacticError(SyntaxToken \* errorToken, SyntaxKind replacment) {  
 int line = errorToken -> line;  
 int column = errorToken ->column;  
   
 \_body = "Parser error on {" + to\_string(line) +":" + to\_string(column) + "} with token " + syntaxTokenToString(\*errorToken) + "\n mabe try using " + syntaxKindToString(replacment) + " instead\n";  
 }  
   
 SyntacticError() {  
 \_body = "broken code structure";  
 }  
   
 string toString() const override  
 {  
 return "SyntaticError: " + \_body;  
 }  
 };  
   
 class semanticError : public Error  
 {  
 public:  
 semanticError(string body) {\_body = body;}  
   
 semanticError(string body, SyntaxToken \* token) {  
 int line = token -> line;  
 int column = token ->column;  
   
 \_body = "semantic error on {" + to\_string(line) +":" + to\_string(column) + "} with token " + syntaxTokenToString(\*token) + "\n" + body;  
 }  
   
 string toString() const override  
 {  
 return "semanticError: " + \_body;  
 }  
   
   
   
 };  
   
 #endif

## lexer.cpp

#include "../lexer/lexer.hpp"  
 #include "../DFA/DFA.hpp"  
 #include "../errors/errors.hpp"  
 #include "../token/token.hpp"  
 #include <string>  
 #include <vector>  
 #include <sstream>  
 #include <algorithm>  
 #include <iostream>  
 #include <fstream>  
   
 // #define LEXER\_DEBUG  
   
 using namespace std;  
   
 SyntaxKind endStateToSyntaxKind[] = {  
 SyntaxKind::UNEXPECTED\_TOKEN, SyntaxKind::INTEGER\_LITERAL, SyntaxKind::FLOAT\_LITERAL, SyntaxKind::CHAR\_LITERAL,  
 SyntaxKind::STRING\_LITERAL, SyntaxKind::IDENTIFIER, SyntaxKind::KEYWORD\_IF,SyntaxKind::KEYWORD\_ELSE,  
 SyntaxKind::KEYWORD\_WHILE, SyntaxKind::KEYWORD\_FOR, SyntaxKind::KEYWORD\_FN,  
 SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT, SyntaxKind::KEYWORD\_CHAR,  
 SyntaxKind::EQUALS, SyntaxKind::COMMA, SyntaxKind::PLUS, SyntaxKind::MINUS,  
 SyntaxKind::SLASH, SyntaxKind::STAR, SyntaxKind::AMPERSAND, SyntaxKind::PIPE,  
 SyntaxKind::CARET, SyntaxKind::TILDE,SyntaxKind::BANG, SyntaxKind::SEMICOLON, SyntaxKind::LESS\_THAN,  
 SyntaxKind::GREATER\_THAN, SyntaxKind::OPEN\_PAREN, SyntaxKind::CLOSED\_PAREN, SyntaxKind::OPEN\_CURLY,  
 SyntaxKind::CLOSED\_CURLY, SyntaxKind::OPEN\_BRACKET, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::RIGHT\_ARROW, SyntaxKind::PLUS\_PLUS, SyntaxKind::MINUS\_MINUS,  
 SyntaxKind::PLUS\_EQUALS, SyntaxKind::MINUS\_EQUALS, SyntaxKind::SLASH\_EQUALS,  
 SyntaxKind::STAR\_EQUALS, SyntaxKind::AMPERSAND\_EQUALS,SyntaxKind::PIPE\_EQUALS, SyntaxKind::CARET\_EQUALS,  
 SyntaxKind::TILDE\_EQUALS, SyntaxKind::EQUALS\_EQUALS, SyntaxKind::LESS\_THAN\_EQUALS,  
 SyntaxKind::GREATER\_THAN\_EQUALS, SyntaxKind::AMPERSAND\_AMPERSAND, SyntaxKind::PIPE\_PIPE,  
 SyntaxKind::BANG\_EQUALS, SyntaxKind::END\_OF\_FILE, SyntaxKind::UNEXPECTED\_TOKEN  
 };  
   
 Lexer::Lexer(string srcFile, string DFAConfigFile, ErrorHandler \*handler)  
 : \_srcFile(srcFile), \_dfa(DFAConfigFile), \_errorHandler(handler), \_cursor(0), \_currLine(1), \_currColumn(1)  
 {  
 ifstream src(\_srcFile);  
   
 src.seekg(0, ios::end);  
 \_fileSize = src.tellg();  
 src.close();  
 }  
   
 vector<SyntaxToken \*> Lexer::getTokens()  
 {  
 vector<SyntaxToken \*> tokens;  
 SyntaxToken \*currToken;  
   
 while ((currToken = getNextToken())->kind != SyntaxKind::END\_OF\_FILE)  
 {  
 tokens.push\_back(currToken);  
 }  
   
 tokens.push\_back(currToken); // pushing the EOF token  
   
 #ifdef LEXER\_DEBUG  
 for (int i = 0; i < tokens.size(); i++)  
 {  
 cout << syntaxTokenToString(\*tokens[i]) << endl;  
 }  
 #endif   
   
 return tokens;  
 }  
   
 SyntaxToken \*Lexer::getNextToken()  
 {  
 char currentChar;  
 stringstream val;  
 int currentState = \_dfa.getStartState();  
 int prevState;  
 SyntaxToken \*resToken = new SyntaxToken();  
 ifstream src(\_srcFile);  
 src.seekg(\_cursor, ios::beg);  
   
 if (\_cursor >= \_fileSize)  
 {  
 resToken->kind = SyntaxKind::END\_OF\_FILE;  
 return resToken;  
 }  
   
 if (!src.is\_open())  
 {  
 runtime\_error("Error: Unable to open sorce file");  
 return resToken;  
 }  
   
 do  
 {  
 // get the next character  
 src.get(currentChar);  
 \_cursor++;  
   
 // get the corresponding next state  
 prevState = currentState;  
 currentState = \_dfa.getState(currentState, currentChar);  
   
 // if we didnt finish lexing the token, recognize it.  
 if (currentState != -1 && \_cursor <= \_fileSize)  
 {  
 val << currentChar;  
 }  
   
 // if we dident pocess the token, update the position  
 if (currentState != -1 || prevState == \_dfa.getStartState())  
 {  
 updatePosition(currentChar);  
 }  
   
 } while (currentState != -1 && \_cursor <= \_fileSize);  
   
 \_cursor--;  
 currentState = prevState;  
 src.close();  
   
 // if the state is a skip state, take the token after it  
 if (isSkipState(currentState))  
 {  
 delete resToken;  
 resToken = getNextToken();  
 }  
 else  
 {  
 // else if the state is an end state, return the token  
 resToken->line = \_currLine;  
 resToken->column = \_currColumn;  
   
 vector<int> endStates = \_dfa.getEndStates();  
 if (find(endStates.begin(), endStates.end(), currentState) != endStates.end())  
 {  
 resToken->kind = getSyntaxKind(currentState);  
 resToken->val = val.str();  
 }  
   
 // else the token is invalid so return an unexpected token  
 else  
 {  
 // if the token is invalid, add an error to the handler and skip it  
 \_cursor++;  
 resToken->kind = SyntaxKind::UNEXPECTED\_TOKEN;  
 \_errorHandler->addError(new SyntaxError("Unexpected token error", \_currLine, \_currColumn));  
 }  
 }  
   
   
 return resToken;  
 }  
   
 void Lexer::updatePosition(char ch)  
 {  
 if (ch == '\t')  
 {  
 \_currColumn += 4; // (tab is 4 spaces)  
 }  
 else if (ch == 0x0A)  
 {  
 \_currColumn = 0;  
 \_currLine++;  
   
 \_cursor++; // skip the carriage return in the src code  
 }  
 else if (ch != 0x0D) // Skip carriage return characters  
 {  
 \_currColumn++;  
 }  
 }  
   
 // print & print helper funcs:  
 SyntaxKind getSyntaxKind(int state)  
 {  
 if (state < SyntaxKind::UNEXPECTED\_TOKEN)  
 return endStateToSyntaxKind[state];  
   
 return SyntaxKind::IDENTIFIER;  
 }  
   
 void Lexer::printTransitionMatrix() const  
 {  
 \_dfa.printMatrix();  
 }  
   
 bool isSkipState(int state)  
 {  
 return state == NUM\_OF\_STATES - 1 || state == NUM\_OF\_STATES - 2;  
 }  
   
 bool isWhitespace(char ch)  
 {  
 return ch == ' ' || ch == '\n' || ch == '\t' || ch == '\r';  
 }

## lexer.hpp

#ifndef \_\_LEXER  
 #define \_\_LEXER  
   
 struct SyntaxToken;  
   
 #include "../DFA/DFA.hpp"  
 #include "../errorHandler/errorHandler.hpp"  
 #include "../errors/errors.hpp"  
 #include "../token/token.hpp"  
 #include "../parser/grammerSymbol/grammerSymbol.hpp"  
 #include <string>  
 #include <vector>  
 #include <sstream>  
   
 using namespace std;  
   
 #define NUM\_OF\_STATES 107  
   
 extern SyntaxKind endStateToSyntaxKind[];  
   
 class Lexer  
 {  
 private:  
 string \_srcFile;  
 DFA \_dfa;  
 int \_cursor;  
 int \_fileSize;  
   
 ErrorHandler \*\_errorHandler;  
   
 int \_currLine;  
 int \_currColumn;  
   
 public:  
 Lexer(string srcFile, string DFAConfigFile, ErrorHandler \*handler);  
 vector<SyntaxToken \*> getTokens();  
 SyntaxToken \*getNextToken();  
 void updatePosition(char ch);  
 void printTransitionMatrix() const;  
 };  
   
 SyntaxKind getSyntaxKind(int state);  
 string syntaxTokenToString(SyntaxToken token);  
 bool isSkipState(int state);  
 bool isWhitespace(char ch);  
 #endif

## main.cpp

#include "../lexer/lexer.hpp"  
 #include "../token/token.hpp"  
 #include "../errorHandler/errorHandler.hpp"  
 #include "../parser/parser.hpp"  
 #include "../semantic/semantic.hpp"  
 #include "../codeGenarator/codeGen.hpp"  
 #include <string>  
 #include <sstream>  
 #include <iostream>  
 #include <fstream>  
   
 using namespace std;  
 string getFileBaseName(string fileName);  
   
 int main(int argc, char \*\*argv)  
 {  
 string filePath;  
   
 if (argc != 2)  
 {  
 cout << "ERROR: Please enter the input file";  
 return 1;  
 }  
   
 filePath = argv[1];  
 string baseName = getFileBaseName(filePath);  
   
 ErrorHandler errorHandler;  
 SymbolTable symbolTable;  
 Lexer lex(filePath, "..\\src\\lexerDFAConfig.txt", &errorHandler);  
 vector<SyntaxToken \*> tokens = lex.getTokens();  
   
 if (errorHandler.getErrorCount() > 0)  
 {  
 errorHandler.printErrors();  
 }  
 else  
 {  
 SemanticAnalyzer semantic(&errorHandler, &symbolTable);  
 Parser parser(tokens, 185, &errorHandler, &semantic);  
   
 ASTNode \*root = parser.parse();  
   
 if (errorHandler.getErrorCount() > 0)  
 {  
 cout << "Parsing failed with errors!" << endl;  
 errorHandler.printErrors();  
 }  
 else  
 {  
 cout << "Parsing completed successfully!" << endl;  
 cout << "Generating code to " << baseName << ".asm" << endl;  
 CodeGenarator codeGen(baseName + ".asm", root, &symbolTable);  
 codeGen.genCode();  
 }  
   
 cout << "compiling finished!";  
 }  
 }  
   
 string getFileBaseName(string fileName)  
 {  
   
 size\_t lastSlash = fileName.find\_last\_of("\\/");  
 if (lastSlash == string::npos)  
 lastSlash = -1;  
   
 size\_t lastDot = fileName.find\_last\_of('.');  
 if (lastDot == string::npos)  
 lastDot = fileName.length();  
   
 string baseName = fileName.substr(lastSlash + 1, lastDot - lastSlash - 1);  
   
 return baseName;  
 }

## nodeAnalyzer.cpp

#include "nodeAnalyzer.hpp"  
 #include <algorithm>  
   
 bool isFuncCall(ASTNode \*node)  
 {  
 bool res = false;  
   
 if (node->GetType() == NON\_TERMINAL)  
 {  
 NonTerminalNode \*ntNode = ((NonTerminalNode \*)node);  
 if (ntNode->getNonTerminalKind() == PRIMARY\_EXPR || ntNode->getNonTerminalKind() == SIMPLE\_STMT)  
 {  
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 if (children.size() == 4)  
 {  
 if (((TerminalNode \*)children[1])->getTerminalKind() == OPEN\_PAREN)  
 res = true;  
 }  
 }  
 }  
   
 return res;  
 }  
   
   
 bool isArrDeref(ASTNode \*node)  
 {  
 bool res = false;  
   
 if (node->GetType() == NON\_TERMINAL)  
 {  
 NonTerminalNode \*ntNode = ((NonTerminalNode \*)node);  
 if (ntNode->getNonTerminalKind() == PRIMARY\_EXPR)  
 {  
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 if (children.size() == 4)  
 {  
 if (((TerminalNode \*)children[1])->getTerminalKind() == OPEN\_BRACKET)  
 res = true;  
 }  
 }  
 }  
   
 return res;  
 }  
   
 bool isReturnStatement(ASTNode \*node)  
 {  
 bool res = false;  
   
 if (node->GetType() == NON\_TERMINAL)  
 {  
 NonTerminalNode \*ntNode = ((NonTerminalNode \*)node);  
 if (ntNode->getNonTerminalKind() == SIMPLE\_STMT)  
 {  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 if (children.size() == 2)  
 {  
 res = true;  
 }  
 }  
 }  
   
 return res;  
 }  
   
 void getFunctionParamNodesHelper(NonTerminalNode \*paramListNonEmptyNode, vector<NonTerminalNode \*> \*paramNodes)  
 {  
 vector<ASTNode \*> childern = paramListNonEmptyNode->GetChildren();  
 int numOfChildren = childern.size();  
   
 NonTerminalNode \*paramNode;  
 if (numOfChildren == 1)  
 {  
 paramNode = ((NonTerminalNode \*)(childern[0]));  
 paramNodes->push\_back(paramNode);  
 }  
   
 if (numOfChildren == 3)  
 {  
 paramNode = ((NonTerminalNode \*)(childern[2]));  
 paramNodes->push\_back(paramNode);  
 getFunctionParamNodesHelper((NonTerminalNode \*)childern[0], paramNodes);  
 }  
 }  
   
 int getExprListNonEmptySize(NonTerminalNode \*exprListNonEmptyNode)  
 {  
 vector<ASTNode \*> childern = exprListNonEmptyNode->GetChildren();  
 int numOfChildren = childern.size();  
   
 int res = 0;  
   
 if (numOfChildren == 1)  
 {  
 res = 1;  
 }  
   
 if (numOfChildren == 3)  
 {  
 res = 1 + getExprListNonEmptySize((NonTerminalNode \*)childern[0]);  
 }  
   
 return 0;  
 }  
   
 vector<NonTerminalNode \*> getFunctionParamNodes(NonTerminalNode \*paramListNode)  
 {  
 vector<ASTNode \*> childern = paramListNode->GetChildren();  
 vector<NonTerminalNode \*> res;  
   
 if (childern.size() > 0)  
 {  
 getFunctionParamNodesHelper((NonTerminalNode \*)childern[0], &res);  
 }  
   
 std::reverse(res.begin(), res.end());  
 return res;  
 }  
   
 void getFunctionCallArgsNodesHelper(NonTerminalNode \*exprListNonEmptyNode, vector<NonTerminalNode \*> \*exprNodes)  
 {  
 vector<ASTNode \*> childern = exprListNonEmptyNode->GetChildren();  
 int numOfChildren = childern.size();  
   
 NonTerminalNode \*exprNode;  
 if (numOfChildren == 1)  
 {  
 exprNode = ((NonTerminalNode \*)(childern[0]));  
 exprNodes->push\_back(exprNode);  
 }  
   
 if (numOfChildren == 3)  
 {  
 exprNode = ((NonTerminalNode \*)(childern[2]));  
 exprNodes->push\_back(exprNode);  
 getFunctionCallArgsNodesHelper((NonTerminalNode \*)childern[0], exprNodes);  
 }  
 }  
   
 vector<NonTerminalNode \*> getFunctionCallArgsNodes(NonTerminalNode \*exprListNode)  
 {  
 vector<ASTNode \*> childern = exprListNode->GetChildren();  
 vector<NonTerminalNode \*> res;  
   
 if (childern.size() > 0)  
 {  
 getFunctionParamNodesHelper((NonTerminalNode \*)childern[0], &res);  
 }  
   
 std::reverse(res.begin(), res.end());  
 return res;  
 }  
   
 void getFunctionDeclNodesHelper(NonTerminalNode \*programNode, vector<NonTerminalNode \*> \*funcDeclNodes)  
 {  
 const vector<ASTNode \*> &children = programNode->GetChildren();  
   
 if (children.size() == 2)  
 {  
 funcDeclNodes->push\_back((NonTerminalNode \*)children[1]);  
 getFunctionDeclNodesHelper((NonTerminalNode \*)children[0], funcDeclNodes);  
 }  
 }  
   
 vector<NonTerminalNode \*> getFunctionDeclNodes(NonTerminalNode \*programNode)  
 {  
 vector<NonTerminalNode \*> res;  
 if (!programNode->GetChildren().empty())  
 {  
 getFunctionDeclNodesHelper(programNode, &res);  
 }  
   
 std::reverse(res.begin(), res.end());  
 return res;  
 }  
   
 void getStmtNodesHelper(NonTerminalNode \*stmtListNonEmptyNode, vector<NonTerminalNode \*> \*stmtNodes)  
 {  
 const vector<ASTNode \*> &children = stmtListNonEmptyNode->GetChildren();  
   
 if (children.size() == 2)  
 {  
 stmtNodes->push\_back((NonTerminalNode \*)children[1]);  
 getStmtNodesHelper((NonTerminalNode \*)children[0], stmtNodes);  
 }  
 }  
   
 vector<NonTerminalNode \*> getStmtNodes(NonTerminalNode \*stmtListNode)  
 {  
 vector<NonTerminalNode \*> res;  
 if (!stmtListNode->GetChildren().empty())  
 {  
 getStmtNodesHelper(stmtListNode, &res);  
 }  
   
 std::reverse(res.begin(), res.end());  
 return res;  
 }

## nodeAnalyzer.hpp

#ifndef \_\_NODE\_ANALYZER  
 #define \_\_NODE\_ANALYZER  
   
 #include "../nodes.hpp"  
 #include "../../parser/grammerSymbol/grammerSymbol.hpp"  
 #include "../../token/token.hpp"  
   
 bool isFuncCall(ASTNode \*node);  
 bool isArrDeref(ASTNode \*node);  
 bool isReturnStatement(ASTNode \*node);  
 int getExprListNonEmptySize(NonTerminalNode \*exprListNode);  
 vector<NonTerminalNode \*> getFunctionParamNodes(NonTerminalNode \*paramListNode);  
 vector<NonTerminalNode \*> getFunctionCallArgsNodes(NonTerminalNode \*paramListNode);  
 vector<NonTerminalNode \*> getFunctionDeclNodes(NonTerminalNode \*programNode);  
 vector<NonTerminalNode \*> getStmtNodes(NonTerminalNode \*stmtListNode);  
   
 #endif

## nodes.cpp

#include "Nodes.hpp"  
 #include "../token/token.hpp"  
 #include "../parser/grammerSymbol/grammerSymbol.hpp"  
 #include <vector>  
 #include <string>  
   
 using namespace std;  
   
 TerminalNode::TerminalNode(SyntaxToken \*token) : \_token(token)  
 {  
 \_syntaxKind = token->kind;  
 \_type = GrammarSymbolType::TERMINAL;  
 }  
   
 SyntaxKind TerminalNode::getTerminalKind() const  
 {  
 return \_syntaxKind;  
 }  
   
 SyntaxToken \*TerminalNode::getToken() const  
 {  
 return \_token;  
 }  
   
 const vector<ASTNode \*> &TerminalNode::GetChildren() const  
 {  
 static const vector<ASTNode \*> empty;  
 return empty;  
 }  
   
 //--------------------------------non terminal node--------------------------------  
   
 NonTerminalNode::NonTerminalNode(NonTerminal nonTerminalKind)  
 : \_nonTerminalKind(nonTerminalKind), \_children()  
 {  
 \_type = GrammarSymbolType::NON\_TERMINAL;  
 }  
   
 void NonTerminalNode::AddChildToEnd(ASTNode \*child)  
 {  
 \_children.push\_back(child);  
 }  
   
 void NonTerminalNode::AddChildToFront(ASTNode \*child)  
 {  
 \_children.insert(\_children.begin(), child);  
 }  
   
 NonTerminal NonTerminalNode::getNonTerminalKind() const  
 {  
 return \_nonTerminalKind;  
 }  
   
 const std::vector<ASTNode \*> &NonTerminalNode::GetChildren() const  
 {  
 return \_children;  
 }  
   
 //--------------------------------debugging--------------------------------  
   
 string AstNodeToString(ASTNode \*node)  
 {  
 string res = "";  
   
 if (node->GetType() == GrammarSymbolType::TERMINAL)  
 {  
 res = syntaxTokenToString(\*((TerminalNode \*)node)->getToken());  
 }  
 else  
 {  
 res = nonTerminalToString(((NonTerminalNode \*)node)->getNonTerminalKind());  
 }  
   
 res += " (";  
 res += valTypeToString(node->GetValType());  
 res += ")";  
   
 return res;  
 }  
   
 void printAstNode(ASTNode \*node)  
 {  
 cout << AstNodeToString(node) << endl;  
 }  
   
 void PrintParseTree(ASTNode \*node)  
 {  
 PrintParseTreeHelper(node,"",true);  
 }  
   
 void PrintParseTreeHelper(ASTNode \*parent, const string &prefix, bool isLast)  
 {  
 if (parent == nullptr)  
 {  
 return;  
 }  
   
 cout << prefix;  
   
 cout << (isLast ? "|\_\_ " : "|-- ") << AstNodeToString(parent) << endl;  
   
 vector<ASTNode \*> children = parent->GetChildren();  
   
 int childernCount = children.size();  
   
 for (int i = 0; i < childernCount; i++)  
 {  
 string newPrefix = prefix + (isLast ? " " : "| ");  
 PrintParseTreeHelper(children.at(i), newPrefix, i == childernCount - 1);  
 }  
 }

## nodes.hpp

#ifndef \_\_NODES  
 #define \_\_NODES  
   
 #include "../token/token.hpp"  
 #include "../parser/grammerSymbol/grammerSymbol.hpp"  
 #include "../symbolTable/tableEntry/tableEntry.hpp"  
 #include <vector>  
 #include <string>  
 #include <iostream>  
   
 using namespace std;  
   
 class ASTNode  
 {  
 protected:  
 GrammarSymbolType \_type;  
 valType \_valType; //be used for semantic analysis  
 int \_register; //be used for code gen phase   
   
 public:  
 const GrammarSymbolType GetType() const  
 {  
 return \_type;  
 }  
   
 const valType GetValType() const  
 {  
 return \_valType;  
 }  
   
 void SetValType(valType valType)  
 {  
 \_valType = valType;  
 }  
   
 int GetRegister() const  
 {  
 return \_register;  
 }  
   
 void SetRegister(int regIndex)  
 {  
 \_register = regIndex;  
 }  
   
 const virtual vector<ASTNode \*> &GetChildren() const = 0;  
 };  
   
 class TerminalNode : public ASTNode  
 {  
 private:  
 SyntaxKind \_syntaxKind;  
 SyntaxToken \*\_token;  
   
 public:  
 TerminalNode(SyntaxToken \*token);  
   
 SyntaxKind getTerminalKind() const;  
 SyntaxToken \*getToken() const;  
 const vector<ASTNode \*> &GetChildren() const override;  
 };  
   
 class NonTerminalNode : public ASTNode  
 {  
 private:  
 NonTerminal \_nonTerminalKind;  
 vector<ASTNode \*> \_children;  
   
 public:  
 NonTerminalNode(NonTerminal nonTerminalKind);  
   
 void AddChildToFront(ASTNode \*child);  
 void AddChildToEnd(ASTNode \*child);  
   
 NonTerminal getNonTerminalKind() const;  
 const std::vector<ASTNode \*> &GetChildren() const override;  
 };  
   
 string AstNodeToString(ASTNode \*node);  
 void printAstNode(ASTNode \*node);  
 void PrintParseTree(ASTNode \*parent);  
 void PrintParseTreeHelper(ASTNode \*parent, const std::string &prefix, bool isLast);  
 #endif

## grammerSymbol.cpp

#include "grammerSymbol.hpp"  
 #include <string>  
 #include <sstream>  
   
 using namespace std;  
   
 string nonTerminalEnumToString[] = {  
 "START",  
 "PROGRAM",  
 "FUNCTION\_DECL",  
 "PARAM\_LIST",  
 "PARAM\_LIST\_NON\_EMPTY",  
 "PARAM",  
 "TYPE",  
 "BASE\_TYPE",  
 "STMT\_LIST",  
 "STMT",  
 "SIMPLE\_STMT",  
 "VAR\_DECL\_EXPR",  
 "INIT\_OPT",  
 "ASSIGN\_VALUE",  
 "ASSIGN\_EXPR",  
 "ASSIGN\_TARGET",  
 "ASSIGN\_OP",  
 "IF\_STMT",  
 "WHILE\_STMT",  
 "CONDITION\_OP",  
 "FOR\_STMT",  
 "FOR\_INIT",  
 "EXPR\_OPT",  
 "FOR\_UPDATE",  
 "BODY",  
 "EXPR\_LIST",  
 "EXPR\_LIST\_NON\_EMPTY",  
 "EXPR",  
 "LOGICAL\_EXPR",  
 "RELATIONAL\_EXPR",  
 "ADD\_EXPR",  
 "MUL\_EXPR",  
 "UNARY\_EXPR",  
 "INCREMENT\_EXPR",  
 "ADDRESS\_EXPR",  
 "DEREFERENCE\_EXPR",  
 "PRIMARY\_EXPR"  
 };  
   
   
 string nonTerminalToString(NonTerminal kind)  
 {  
 stringstream res;  
 res << "\e[1;34m<"  
 << nonTerminalEnumToString[(int)kind]  
 << ">\033[0m";  
   
 return res.str();  
 }

## grammerSymbol.hpp

#ifndef \_\_GAMMER\_SYMBOL  
 #define \_\_GAMMER\_SYMBOL  
   
 #include <string>  
   
 using namespace std;  
   
 enum GrammarSymbolType  
 {  
 TERMINAL,  
 NON\_TERMINAL  
 };  
   
 enum NonTerminal  
 {  
 START,  
 PROGRAM,  
 FUNCTION\_DECL,  
 PARAM\_LIST,  
 PARAM\_LIST\_NON\_EMPTY,  
 PARAM,  
 TYPE,  
 BASE\_TYPE,  
 STMT\_LIST,  
 STMT,  
 SIMPLE\_STMT,  
 VAR\_DECL\_EXPR,  
 INIT\_OPT,  
 ASSIGN\_VALUE,  
 ASSIGN\_EXPR,  
 ASSIGN\_TARGET,  
 ASSIGN\_OP,  
 IF\_STMT,  
 WHILE\_STMT,  
 CONDITION\_OP,  
 FOR\_STMT,  
 FOR\_INIT,  
 EXPR\_OPT,  
 FOR\_UPDATE,  
 BODY,  
 EXPR\_LIST,  
 EXPR\_LIST\_NON\_EMPTY,  
 EXPR,  
 LOGICAL\_EXPR,  
 RELATIONAL\_EXPR,  
 ADD\_EXPR,  
 MUL\_EXPR,  
 UNARY\_EXPR,  
 INCREMENT\_EXPR,  
 ADDRESS\_EXPR,  
 DEREFERENCE\_EXPR,  
 PRIMARY\_EXPR,  
 NON\_TERMINAL\_COUNT  
 };  
   
   
 string nonTerminalToString(NonTerminal kind);  
   
 #endif

## initParserFunctions.cpp

#include "parser.hpp"  
 #include "productionRule/productionRule.hpp"  
 #include "../token/token.hpp"  
 void Parser::initProductionRules()  
 {  
 productionRule rule;  
   
 // Rule 0: $accept -> Program $end  
 rule.setLeft(START);  
 rule.addSymbol(PROGRAM).addSymbol(END\_OF\_FILE);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 1: Program -> ??  
 rule.setLeft(PROGRAM);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 2: Program -> Program FunctionDecl  
 rule.setLeft(PROGRAM);  
 rule.addSymbol(PROGRAM).addSymbol(FUNCTION\_DECL);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 3: FunctionDecl -> KEYWORD\_FN IDENTIFIER OPEN\_PAREN ParamList CLOSED\_PAREN RIGHT\_ARROW Type OPEN\_CURLY StmtList CLOSED\_CURLY  
 rule.setLeft(FUNCTION\_DECL);  
 rule.addSymbol(KEYWORD\_FN).addSymbol(IDENTIFIER).addSymbol(OPEN\_PAREN)  
 .addSymbol(PARAM\_LIST).addSymbol(CLOSED\_PAREN).addSymbol(RIGHT\_ARROW)  
 .addSymbol(TYPE).addSymbol(OPEN\_CURLY).addSymbol(STMT\_LIST).addSymbol(CLOSED\_CURLY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 4: ParamList -> ??  
 rule.setLeft(PARAM\_LIST);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 5: ParamList -> ParamListNonEmpty  
 rule.setLeft(PARAM\_LIST);  
 rule.addSymbol(PARAM\_LIST\_NON\_EMPTY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 6: ParamListNonEmpty -> Param  
 rule.setLeft(PARAM\_LIST\_NON\_EMPTY);  
 rule.addSymbol(PARAM);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 7: ParamListNonEmpty -> ParamListNonEmpty COMMA Param  
 rule.setLeft(PARAM\_LIST\_NON\_EMPTY);  
 rule.addSymbol(PARAM\_LIST\_NON\_EMPTY).addSymbol(COMMA).addSymbol(PARAM);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 8: Param -> Type IDENTIFIER  
 rule.setLeft(PARAM);  
 rule.addSymbol(TYPE).addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 9: Type -> BaseType  
 rule.setLeft(TYPE);  
 rule.addSymbol(BASE\_TYPE);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 10: Type -> BaseType OPEN\_BRACKET INTEGER\_LITERAL CLOSED\_BRACKET  
 rule.setLeft(TYPE);  
 rule.addSymbol(BASE\_TYPE).addSymbol(OPEN\_BRACKET).addSymbol(INTEGER\_LITERAL).addSymbol(CLOSED\_BRACKET);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 11: Type -> BaseType STAR  
 rule.setLeft(TYPE);  
 rule.addSymbol(BASE\_TYPE).addSymbol(STAR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 12: BaseType -> KEYWORD\_INT  
 rule.setLeft(BASE\_TYPE);  
 rule.addSymbol(KEYWORD\_INT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 13: BaseType -> KEYWORD\_FLOAT  
 rule.setLeft(BASE\_TYPE);  
 rule.addSymbol(KEYWORD\_FLOAT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 14: BaseType -> KEYWORD\_CHAR  
 rule.setLeft(BASE\_TYPE);  
 rule.addSymbol(KEYWORD\_CHAR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 15: StmtList -> ??  
 rule.setLeft(STMT\_LIST);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 16: StmtList -> StmtList Stmt  
 rule.setLeft(STMT\_LIST);  
 rule.addSymbol(STMT\_LIST).addSymbol(STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 17: Stmt -> SimpleStmt SEMICOLON  
 rule.setLeft(STMT);  
 rule.addSymbol(SIMPLE\_STMT).addSymbol(SEMICOLON);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 18: Stmt -> OPEN\_CURLY StmtList CLOSED\_CURLY  
 rule.setLeft(STMT);  
 rule.addSymbol(OPEN\_CURLY).addSymbol(STMT\_LIST).addSymbol(CLOSED\_CURLY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 19: Stmt -> IfStmt  
 rule.setLeft(STMT);  
 rule.addSymbol(IF\_STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 20: Stmt -> WhileStmt  
 rule.setLeft(STMT);  
 rule.addSymbol(WHILE\_STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 21: Stmt -> ForStmt  
 rule.setLeft(STMT);  
 rule.addSymbol(FOR\_STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 22: SimpleStmt -> VarDeclExpr  
 rule.setLeft(SIMPLE\_STMT);  
 rule.addSymbol(VAR\_DECL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 23: SimpleStmt -> AssignExpr  
 rule.setLeft(SIMPLE\_STMT);  
 rule.addSymbol(ASSIGN\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 24: SimpleStmt -> IDENTIFIER OPEN\_PAREN ExprList CLOSED\_PAREN  
 rule.setLeft(SIMPLE\_STMT);  
 rule.addSymbol(IDENTIFIER).addSymbol(OPEN\_PAREN).addSymbol(EXPR\_LIST).addSymbol(CLOSED\_PAREN);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 25: SimpleStmt -> KEYWORD\_RET ExprOpt  
 rule.setLeft(SIMPLE\_STMT);  
 rule.addSymbol(KEYWORD\_RET).addSymbol(EXPR\_OPT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 26: VarDeclExpr -> Type IDENTIFIER InitOpt  
 rule.setLeft(VAR\_DECL\_EXPR);  
 rule.addSymbol(TYPE).addSymbol(IDENTIFIER).addSymbol(INIT\_OPT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 27: InitOpt -> ??  
 rule.setLeft(INIT\_OPT);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 28: InitOpt -> EQUALS AssignValue  
 rule.setLeft(INIT\_OPT);  
 rule.addSymbol(EQUALS).addSymbol(ASSIGN\_VALUE);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 29: AssignValue -> Expr  
 rule.setLeft(ASSIGN\_VALUE);  
 rule.addSymbol(EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 30: AssignValue -> OPEN\_CURLY ExprList CLOSED\_CURLY  
 rule.setLeft(ASSIGN\_VALUE);  
 rule.addSymbol(OPEN\_CURLY).addSymbol(EXPR\_LIST).addSymbol(CLOSED\_CURLY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 31: AssignExpr -> AssignTarget AssignOp Expr  
 rule.setLeft(ASSIGN\_EXPR);  
 rule.addSymbol(ASSIGN\_TARGET).addSymbol(ASSIGN\_OP).addSymbol(EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 32: AssignTarget -> IDENTIFIER  
 rule.setLeft(ASSIGN\_TARGET);  
 rule.addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 33: AssignTarget -> IDENTIFIER OPEN\_BRACKET Expr CLOSED\_BRACKET  
 rule.setLeft(ASSIGN\_TARGET);  
 rule.addSymbol(IDENTIFIER).addSymbol(OPEN\_BRACKET).addSymbol(EXPR).addSymbol(CLOSED\_BRACKET);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 34: AssignTarget -> STAR IDENTIFIER  
 rule.setLeft(ASSIGN\_TARGET);  
 rule.addSymbol(STAR).addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 35: AssignOp -> EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 36: AssignOp -> PLUS\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(PLUS\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 37: AssignOp -> MINUS\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(MINUS\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 38: AssignOp -> SLASH\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(SLASH\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 39: AssignOp -> STAR\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(STAR\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 40: AssignOp -> AMPERSAND\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(AMPERSAND\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 41: AssignOp -> PIPE\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(PIPE\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 42: AssignOp -> CARET\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(CARET\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 43: AssignOp -> TILDE\_EQUALS  
 rule.setLeft(ASSIGN\_OP);  
 rule.addSymbol(TILDE\_EQUALS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 44: IfStmt -> KEYWORD\_IF OPEN\_PAREN ConditionOp CLOSED\_PAREN Body  
 rule.setLeft(IF\_STMT);  
 rule.addSymbol(KEYWORD\_IF).addSymbol(OPEN\_PAREN).addSymbol(CONDITION\_OP)  
 .addSymbol(CLOSED\_PAREN).addSymbol(BODY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 45: IfStmt -> KEYWORD\_IF OPEN\_PAREN ConditionOp CLOSED\_PAREN Body KEYWORD\_ELSE Body  
 rule.setLeft(IF\_STMT);  
 rule.addSymbol(KEYWORD\_IF).addSymbol(OPEN\_PAREN).addSymbol(CONDITION\_OP)  
 .addSymbol(CLOSED\_PAREN).addSymbol(BODY).addSymbol(KEYWORD\_ELSE).addSymbol(BODY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 46: WhileStmt -> KEYWORD\_WHILE OPEN\_PAREN ConditionOp CLOSED\_PAREN Body  
 rule.setLeft(WHILE\_STMT);  
 rule.addSymbol(KEYWORD\_WHILE).addSymbol(OPEN\_PAREN).addSymbol(CONDITION\_OP)  
 .addSymbol(CLOSED\_PAREN).addSymbol(BODY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 47: ConditionOp -> Expr  
 rule.setLeft(CONDITION\_OP);  
 rule.addSymbol(EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 48: ConditionOp -> AssignExpr  
 rule.setLeft(CONDITION\_OP);  
 rule.addSymbol(ASSIGN\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 49: ForStmt -> KEYWORD\_FOR OPEN\_PAREN ForInit SEMICOLON ExprOpt SEMICOLON ForUpdate CLOSED\_PAREN Body  
 rule.setLeft(FOR\_STMT);  
 rule.addSymbol(KEYWORD\_FOR).addSymbol(OPEN\_PAREN).addSymbol(FOR\_INIT).addSymbol(SEMICOLON)  
 .addSymbol(EXPR\_OPT).addSymbol(SEMICOLON).addSymbol(FOR\_UPDATE).addSymbol(CLOSED\_PAREN).addSymbol(BODY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 50: ForInit -> ??  
 rule.setLeft(FOR\_INIT);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 51: ForInit -> VarDeclExpr  
 rule.setLeft(FOR\_INIT);  
 rule.addSymbol(VAR\_DECL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 52: ForInit -> AssignExpr  
 rule.setLeft(FOR\_INIT);  
 rule.addSymbol(ASSIGN\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 53: ExprOpt -> ??  
 rule.setLeft(EXPR\_OPT);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 54: ExprOpt -> Expr  
 rule.setLeft(EXPR\_OPT);  
 rule.addSymbol(EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 55: ForUpdate -> ??  
 rule.setLeft(FOR\_UPDATE);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 56: ForUpdate -> AssignExpr  
 rule.setLeft(FOR\_UPDATE);  
 rule.addSymbol(ASSIGN\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 57: ForUpdate -> IncrementExpr  
 rule.setLeft(FOR\_UPDATE);  
 rule.addSymbol(INCREMENT\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 58: Body -> OPEN\_CURLY StmtList CLOSED\_CURLY  
 rule.setLeft(BODY);  
 rule.addSymbol(OPEN\_CURLY).addSymbol(STMT\_LIST).addSymbol(CLOSED\_CURLY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 59: Body -> SimpleStmt SEMICOLON  
 rule.setLeft(BODY);  
 rule.addSymbol(SIMPLE\_STMT).addSymbol(SEMICOLON);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 60: Body -> IfStmt  
 rule.setLeft(BODY);  
 rule.addSymbol(IF\_STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 61: Body -> WhileStmt  
 rule.setLeft(BODY);  
 rule.addSymbol(WHILE\_STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 62: Body -> ForStmt  
 rule.setLeft(BODY);  
 rule.addSymbol(FOR\_STMT);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 63: ExprList -> ??  
 rule.setLeft(EXPR\_LIST);  
 rule.addSymbol(); // empty  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 64: ExprList -> ExprListNonEmpty  
 rule.setLeft(EXPR\_LIST);  
 rule.addSymbol(EXPR\_LIST\_NON\_EMPTY);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 65: ExprListNonEmpty -> Expr  
 rule.setLeft(EXPR\_LIST\_NON\_EMPTY);  
 rule.addSymbol(EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 66: ExprListNonEmpty -> ExprListNonEmpty COMMA Expr  
 rule.setLeft(EXPR\_LIST\_NON\_EMPTY);  
 rule.addSymbol(EXPR\_LIST\_NON\_EMPTY).addSymbol(COMMA).addSymbol(EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 67: Expr -> LogicalExpr  
 rule.setLeft(EXPR);  
 rule.addSymbol(LOGICAL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 68: Expr -> AddressExpr  
 rule.setLeft(EXPR);  
 rule.addSymbol(ADDRESS\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 69: LogicalExpr -> RelationalExpr  
 rule.setLeft(LOGICAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 70: LogicalExpr -> LogicalExpr PIPE\_PIPE RelationalExpr  
 rule.setLeft(LOGICAL\_EXPR);  
 rule.addSymbol(LOGICAL\_EXPR).addSymbol(PIPE\_PIPE).addSymbol(RELATIONAL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 71: LogicalExpr -> LogicalExpr AMPERSAND\_AMPERSAND RelationalExpr  
 rule.setLeft(LOGICAL\_EXPR);  
 rule.addSymbol(LOGICAL\_EXPR).addSymbol(AMPERSAND\_AMPERSAND).addSymbol(RELATIONAL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 72: RelationalExpr -> AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 73: RelationalExpr -> RelationalExpr EQUALS\_EQUALS AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR).addSymbol(EQUALS\_EQUALS).addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 74: RelationalExpr -> RelationalExpr BANG\_EQUALS AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR).addSymbol(BANG\_EQUALS).addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 75: RelationalExpr -> RelationalExpr LESS\_THAN AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR).addSymbol(LESS\_THAN).addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 76: RelationalExpr -> RelationalExpr GREATER\_THAN AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR).addSymbol(GREATER\_THAN).addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 77: RelationalExpr -> RelationalExpr LESS\_THAN\_EQUALS AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR).addSymbol(LESS\_THAN\_EQUALS).addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 78: RelationalExpr -> RelationalExpr GREATER\_THAN\_EQUALS AddExpr  
 rule.setLeft(RELATIONAL\_EXPR);  
 rule.addSymbol(RELATIONAL\_EXPR).addSymbol(GREATER\_THAN\_EQUALS).addSymbol(ADD\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 79: AddExpr -> MulExpr  
 rule.setLeft(ADD\_EXPR);  
 rule.addSymbol(MUL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 80: AddExpr -> AddExpr PLUS MulExpr  
 rule.setLeft(ADD\_EXPR);  
 rule.addSymbol(ADD\_EXPR).addSymbol(PLUS).addSymbol(MUL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 81: AddExpr -> AddExpr MINUS MulExpr  
 rule.setLeft(ADD\_EXPR);  
 rule.addSymbol(ADD\_EXPR).addSymbol(MINUS).addSymbol(MUL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 82: AddExpr -> AddExpr PIPE MulExpr  
 rule.setLeft(ADD\_EXPR);  
 rule.addSymbol(ADD\_EXPR).addSymbol(PIPE).addSymbol(MUL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 83: AddExpr -> AddExpr CARET MulExpr  
 rule.setLeft(ADD\_EXPR);  
 rule.addSymbol(ADD\_EXPR).addSymbol(CARET).addSymbol(MUL\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 84: MulExpr -> UnaryExpr  
 rule.setLeft(MUL\_EXPR);  
 rule.addSymbol(UNARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 85: MulExpr -> MulExpr STAR UnaryExpr  
 rule.setLeft(MUL\_EXPR);  
 rule.addSymbol(MUL\_EXPR).addSymbol(STAR).addSymbol(UNARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 86: MulExpr -> MulExpr SLASH UnaryExpr  
 rule.setLeft(MUL\_EXPR);  
 rule.addSymbol(MUL\_EXPR).addSymbol(SLASH).addSymbol(UNARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 87: MulExpr -> MulExpr AMPERSAND UnaryExpr  
 rule.setLeft(MUL\_EXPR);  
 rule.addSymbol(MUL\_EXPR).addSymbol(AMPERSAND).addSymbol(UNARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 88: UnaryExpr -> PrimaryExpr  
 rule.setLeft(UNARY\_EXPR);  
 rule.addSymbol(PRIMARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 89: UnaryExpr -> MINUS UnaryExpr  
 rule.setLeft(UNARY\_EXPR);  
 rule.addSymbol(MINUS).addSymbol(UNARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 90: UnaryExpr -> BANG UnaryExpr  
 rule.setLeft(UNARY\_EXPR);  
 rule.addSymbol(BANG).addSymbol(UNARY\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 91: IncrementExpr -> IDENTIFIER PLUS\_PLUS  
 rule.setLeft(INCREMENT\_EXPR);  
 rule.addSymbol(IDENTIFIER).addSymbol(PLUS\_PLUS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 92: IncrementExpr -> IDENTIFIER MINUS\_MINUS  
 rule.setLeft(INCREMENT\_EXPR);  
 rule.addSymbol(IDENTIFIER).addSymbol(MINUS\_MINUS);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 93: IncrementExpr -> PLUS\_PLUS IDENTIFIER  
 rule.setLeft(INCREMENT\_EXPR);  
 rule.addSymbol(PLUS\_PLUS).addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 94: IncrementExpr -> MINUS\_MINUS IDENTIFIER  
 rule.setLeft(INCREMENT\_EXPR);  
 rule.addSymbol(MINUS\_MINUS).addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 95: AddressExpr -> AMPERSAND IDENTIFIER  
 rule.setLeft(ADDRESS\_EXPR);  
 rule.addSymbol(AMPERSAND).addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 96: DereferenceExpr -> STAR IDENTIFIER  
 rule.setLeft(DEREFERENCE\_EXPR);  
 rule.addSymbol(STAR).addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 97: PrimaryExpr -> IDENTIFIER  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(IDENTIFIER);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 98: PrimaryExpr -> IDENTIFIER OPEN\_BRACKET Expr CLOSED\_BRACKET  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(IDENTIFIER).addSymbol(OPEN\_BRACKET).addSymbol(EXPR).addSymbol(CLOSED\_BRACKET);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 99: PrimaryExpr -> INTEGER\_LITERAL  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(INTEGER\_LITERAL);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 100: PrimaryExpr -> FLOAT\_LITERAL  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(FLOAT\_LITERAL);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 101: PrimaryExpr -> CHAR\_LITERAL  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(CHAR\_LITERAL);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 102: PrimaryExpr -> OPEN\_PAREN Expr CLOSED\_PAREN  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(OPEN\_PAREN).addSymbol(EXPR).addSymbol(CLOSED\_PAREN);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 103: PrimaryExpr -> IDENTIFIER OPEN\_PAREN ExprList CLOSED\_PAREN  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(IDENTIFIER).addSymbol(OPEN\_PAREN).addSymbol(EXPR\_LIST).addSymbol(CLOSED\_PAREN);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 104: PrimaryExpr -> DereferenceExpr  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(DEREFERENCE\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
   
 // Rule 105: PrimaryExpr -> IncrementExpr  
 rule.setLeft(PRIMARY\_EXPR);  
 rule.addSymbol(INCREMENT\_EXPR);  
 addProductionRule(rule);  
 rule.reset();  
 }  
   
   
 void Parser::fillTables() {  
 // State 0  
 \_actionTable.addDefault(0, {REDUCE, 1});  
 \_gotoTable.add(0, PROGRAM, 1);  
   
 // State 1  
 \_actionTable.add(1, END\_OF\_FILE, {SHIFT, 2});  
 \_actionTable.add(1, KEYWORD\_FN, {SHIFT, 3});  
 \_gotoTable.add(1, FUNCTION\_DECL, 4);  
   
 // State 2  
 \_actionTable.addDefault(2, {ACCEPT, 0});  
   
 // State 3  
 \_actionTable.add(3, IDENTIFIER, {SHIFT, 5});  
   
 // State 4  
 \_actionTable.addDefault(4, {REDUCE, 2});  
   
 // State 5  
 \_actionTable.add(5, OPEN\_PAREN, {SHIFT, 6});  
   
 // State 6  
 \_actionTable.add(6, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(6, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(6, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.addDefault(6, {REDUCE, 4});  
 \_gotoTable.add(6, PARAM\_LIST, 10);  
 \_gotoTable.add(6, PARAM\_LIST\_NON\_EMPTY, 11);  
 \_gotoTable.add(6, PARAM, 12);  
 \_gotoTable.add(6, TYPE, 13);  
 \_gotoTable.add(6, BASE\_TYPE, 14);  
   
 // State 7  
 \_actionTable.addDefault(7, {REDUCE, 12});  
   
 // State 8  
 \_actionTable.addDefault(8, {REDUCE, 13});  
   
 // State 9  
 \_actionTable.addDefault(9, {REDUCE, 14});  
   
 // State 10  
 \_actionTable.add(10, CLOSED\_PAREN, {SHIFT, 15});  
   
 // State 11  
 \_actionTable.add(11, COMMA, {SHIFT, 16});  
 \_actionTable.addDefault(11, {REDUCE, 5});  
   
 // State 12  
 \_actionTable.addDefault(12, {REDUCE, 6});  
   
 // State 13  
 \_actionTable.add(13, IDENTIFIER, {SHIFT, 17});  
   
 // State 14  
 \_actionTable.add(14, OPEN\_BRACKET, {SHIFT, 18});  
 \_actionTable.add(14, STAR, {SHIFT, 19});  
 \_actionTable.addDefault(14, {REDUCE, 9});  
   
 // State 15  
 \_actionTable.add(15, RIGHT\_ARROW, {SHIFT, 20});  
   
 // State 16  
 \_actionTable.add(16, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(16, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(16, KEYWORD\_CHAR, {SHIFT, 9});  
 \_gotoTable.add(16, PARAM, 21);  
 \_gotoTable.add(16, TYPE, 13);  
 \_gotoTable.add(16, BASE\_TYPE, 14);  
   
 // State 17  
 \_actionTable.addDefault(17, {REDUCE, 8});  
   
 // State 18  
 \_actionTable.add(18, INTEGER\_LITERAL, {SHIFT, 22});  
   
 // State 19  
 \_actionTable.addDefault(19, {REDUCE, 11});  
   
 // State 20  
 \_actionTable.add(20, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(20, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(20, KEYWORD\_CHAR, {SHIFT, 9});  
 \_gotoTable.add(20, TYPE, 23);  
 \_gotoTable.add(20, BASE\_TYPE, 14);  
   
 // State 21  
 \_actionTable.addDefault(21, {REDUCE, 7});  
   
 // State 22  
 \_actionTable.add(22, CLOSED\_BRACKET, {SHIFT, 24});  
   
 // State 23  
 \_actionTable.add(23, OPEN\_CURLY, {SHIFT, 25});  
   
 // State 24  
 \_actionTable.addDefault(24, {REDUCE, 10});  
   
 // State 25  
 \_actionTable.addDefault(25, {REDUCE, 15});  
 \_gotoTable.add(25, STMT\_LIST, 26);  
   
 // State 26  
 \_actionTable.add(26, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(26, OPEN\_CURLY, {SHIFT, 28});  
 \_actionTable.add(26, CLOSED\_CURLY, {SHIFT, 29});  
 \_actionTable.add(26, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(26, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(26, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(26, STAR, {SHIFT, 30});  
 \_actionTable.add(26, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(26, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(26, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(26, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(26, TYPE, 35);  
 \_gotoTable.add(26, BASE\_TYPE, 14);  
 \_gotoTable.add(26, STMT, 36);  
 \_gotoTable.add(26, SIMPLE\_STMT, 37);  
 \_gotoTable.add(26, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(26, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(26, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(26, IF\_STMT, 41);  
 \_gotoTable.add(26, WHILE\_STMT, 42);  
 \_gotoTable.add(26, FOR\_STMT, 43);  
   
 // State 27  
 \_actionTable.add(27, OPEN\_PAREN, {SHIFT, 44});  
 \_actionTable.add(27, OPEN\_BRACKET, {SHIFT, 45});  
 \_actionTable.addDefault(27, {REDUCE, 32});  
   
 // State 28  
 \_actionTable.addDefault(28, {REDUCE, 15});  
 \_gotoTable.add(28, STMT\_LIST, 46);  
   
 // State 29  
 \_actionTable.addDefault(29, {REDUCE, 3});  
   
 // State 30  
 \_actionTable.add(30, IDENTIFIER, {SHIFT, 47});  
   
 // State 31  
 \_actionTable.add(31, OPEN\_PAREN, {SHIFT, 48});  
   
 // State 32  
 \_actionTable.add(32, OPEN\_PAREN, {SHIFT, 49});  
   
 // State 33  
 \_actionTable.add(33, OPEN\_PAREN, {SHIFT, 50});  
   
 // State 34  
 \_actionTable.add(34, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(34, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(34, STAR, {SHIFT, 53});  
 \_actionTable.add(34, MINUS, {SHIFT, 54});  
 \_actionTable.add(34, BANG, {SHIFT, 55});  
 \_actionTable.add(34, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(34, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(34, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(34, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(34, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(34, MINUS\_MINUS, {SHIFT, 61});  
 \_actionTable.addDefault(34, {REDUCE, 53});  
 \_gotoTable.add(34, EXPR\_OPT, 62);  
 \_gotoTable.add(34, EXPR, 63);  
 \_gotoTable.add(34, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(34, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(34, ADD\_EXPR, 66);  
 \_gotoTable.add(34, MUL\_EXPR, 67);  
 \_gotoTable.add(34, UNARY\_EXPR, 68);  
 \_gotoTable.add(34, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(34, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(34, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(34, PRIMARY\_EXPR, 72);  
   
 // State 35  
 \_actionTable.add(35, IDENTIFIER, {SHIFT, 73});  
   
 // State 36  
 \_actionTable.addDefault(36, {REDUCE, 16});  
   
 // State 37  
 \_actionTable.add(37, SEMICOLON, {SHIFT, 74});  
   
 // State 38  
 \_actionTable.addDefault(38, {REDUCE, 22});  
   
 // State 39  
 \_actionTable.addDefault(39, {REDUCE, 23});  
   
 // State 40  
 \_actionTable.add(40, EQUALS, {SHIFT, 75});  
 \_actionTable.add(40, PLUS\_EQUALS, {SHIFT, 76});  
 \_actionTable.add(40, MINUS\_EQUALS, {SHIFT, 77});  
 \_actionTable.add(40, SLASH\_EQUALS, {SHIFT, 78});  
 \_actionTable.add(40, STAR\_EQUALS, {SHIFT, 79});  
 \_actionTable.add(40, AMPERSAND\_EQUALS, {SHIFT, 80});  
 \_actionTable.add(40, PIPE\_EQUALS, {SHIFT, 81});  
 \_actionTable.add(40, CARET\_EQUALS, {SHIFT, 82});  
 \_actionTable.add(40, TILDE\_EQUALS, {SHIFT, 83});  
 \_gotoTable.add(40, ASSIGN\_OP, 84);  
   
 // State 41  
 \_actionTable.addDefault(41, {REDUCE, 19});  
   
 // State 42  
 \_actionTable.addDefault(42, {REDUCE, 20});  
   
 // State 43  
 \_actionTable.addDefault(43, {REDUCE, 21});  
   
 // State 44  
 \_actionTable.add(44, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(44, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(44, STAR, {SHIFT, 53});  
 \_actionTable.add(44, MINUS, {SHIFT, 54});  
 \_actionTable.add(44, BANG, {SHIFT, 55});  
 \_actionTable.add(44, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(44, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(44, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(44, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(44, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(44, MINUS\_MINUS, {SHIFT, 61});  
 \_actionTable.addDefault(44, {REDUCE, 63});  
 \_gotoTable.add(44, EXPR\_LIST, 85);  
 \_gotoTable.add(44, EXPR\_LIST\_NON\_EMPTY, 86);  
 \_gotoTable.add(44, EXPR, 87);  
 \_gotoTable.add(44, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(44, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(44, ADD\_EXPR, 66);  
 \_gotoTable.add(44, MUL\_EXPR, 67);  
 \_gotoTable.add(44, UNARY\_EXPR, 68);  
 \_gotoTable.add(44, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(44, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(44, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(44, PRIMARY\_EXPR, 72);  
   
 // State 45  
 \_actionTable.add(45, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(45, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(45, STAR, {SHIFT, 53});  
 \_actionTable.add(45, MINUS, {SHIFT, 54});  
 \_actionTable.add(45, BANG, {SHIFT, 55});  
 \_actionTable.add(45, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(45, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(45, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(45, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(45, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(45, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(45, EXPR, 88);  
 \_gotoTable.add(45, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(45, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(45, ADD\_EXPR, 66);  
 \_gotoTable.add(45, MUL\_EXPR, 67);  
 \_gotoTable.add(45, UNARY\_EXPR, 68);  
 \_gotoTable.add(45, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(45, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(45, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(45, PRIMARY\_EXPR, 72);  
   
 // State 46  
 \_actionTable.add(46, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(46, OPEN\_CURLY, {SHIFT, 28});  
 \_actionTable.add(46, CLOSED\_CURLY, {SHIFT, 89});  
 \_actionTable.add(46, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(46, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(46, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(46, STAR, {SHIFT, 30});  
 \_actionTable.add(46, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(46, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(46, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(46, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(46, TYPE, 35);  
 \_gotoTable.add(46, BASE\_TYPE, 14);  
 \_gotoTable.add(46, STMT, 36);  
 \_gotoTable.add(46, SIMPLE\_STMT, 37);  
 \_gotoTable.add(46, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(46, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(46, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(46, IF\_STMT, 41);  
 \_gotoTable.add(46, WHILE\_STMT, 42);  
 \_gotoTable.add(46, FOR\_STMT, 43);  
   
 // State 47  
 \_actionTable.addDefault(47, {REDUCE, 34});  
   
 // State 48  
 \_actionTable.add(48, IDENTIFIER, {SHIFT, 90});  
 \_actionTable.add(48, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(48, STAR, {SHIFT, 91});  
 \_actionTable.add(48, MINUS, {SHIFT, 54});  
 \_actionTable.add(48, BANG, {SHIFT, 55});  
 \_actionTable.add(48, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(48, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(48, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(48, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(48, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(48, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(48, ASSIGN\_EXPR, 92);  
 \_gotoTable.add(48, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(48, CONDITION\_OP, 93);  
 \_gotoTable.add(48, EXPR, 94);  
 \_gotoTable.add(48, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(48, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(48, ADD\_EXPR, 66);  
 \_gotoTable.add(48, MUL\_EXPR, 67);  
 \_gotoTable.add(48, UNARY\_EXPR, 68);  
 \_gotoTable.add(48, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(48, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(48, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(48, PRIMARY\_EXPR, 72);  
   
 // State 49  
 \_actionTable.add(49, IDENTIFIER, {SHIFT, 90});  
 \_actionTable.add(49, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(49, STAR, {SHIFT, 91});  
 \_actionTable.add(49, MINUS, {SHIFT, 54});  
 \_actionTable.add(49, BANG, {SHIFT, 55});  
 \_actionTable.add(49, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(49, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(49, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(49, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(49, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(49, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(49, ASSIGN\_EXPR, 92);  
 \_gotoTable.add(49, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(49, CONDITION\_OP, 95);  
 \_gotoTable.add(49, EXPR, 94);  
 \_gotoTable.add(49, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(49, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(49, ADD\_EXPR, 66);  
 \_gotoTable.add(49, MUL\_EXPR, 67);  
 \_gotoTable.add(49, UNARY\_EXPR, 68);  
 \_gotoTable.add(49, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(49, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(49, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(49, PRIMARY\_EXPR, 72);  
   
 // State 50  
 \_actionTable.add(50, IDENTIFIER, {SHIFT, 96});  
 \_actionTable.add(50, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(50, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(50, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(50, STAR, {SHIFT, 30});  
 \_actionTable.addDefault(50, {REDUCE, 50});  
 \_gotoTable.add(50, TYPE, 35);  
 \_gotoTable.add(50, BASE\_TYPE, 14);  
 \_gotoTable.add(50, VAR\_DECL\_EXPR, 97);  
 \_gotoTable.add(50, ASSIGN\_EXPR, 98);  
 \_gotoTable.add(50, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(50, FOR\_INIT, 99);  
   
 // State 51  
 \_actionTable.add(51, OPEN\_PAREN, {SHIFT, 100});  
 \_actionTable.add(51, OPEN\_BRACKET, {SHIFT, 101});  
 \_actionTable.add(51, PLUS\_PLUS, {SHIFT, 102});  
 \_actionTable.add(51, MINUS\_MINUS, {SHIFT, 103});  
 \_actionTable.addDefault(51, {REDUCE, 97});  
   
 // State 52  
 \_actionTable.add(52, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(52, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(52, STAR, {SHIFT, 53});  
 \_actionTable.add(52, MINUS, {SHIFT, 54});  
 \_actionTable.add(52, BANG, {SHIFT, 55});  
 \_actionTable.add(52, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(52, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(52, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(52, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(52, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(52, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(52, EXPR, 104);  
 \_gotoTable.add(52, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(52, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(52, ADD\_EXPR, 66);  
 \_gotoTable.add(52, MUL\_EXPR, 67);  
 \_gotoTable.add(52, UNARY\_EXPR, 68);  
 \_gotoTable.add(52, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(52, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(52, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(52, PRIMARY\_EXPR, 72);  
   
 // State 53  
 \_actionTable.add(53, IDENTIFIER, {SHIFT, 105});  
   
 // State 54  
 \_actionTable.add(54, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(54, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(54, STAR, {SHIFT, 53});  
 \_actionTable.add(54, MINUS, {SHIFT, 54});  
 \_actionTable.add(54, BANG, {SHIFT, 55});  
 \_actionTable.add(54, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(54, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(54, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(54, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(54, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(54, UNARY\_EXPR, 106);  
 \_gotoTable.add(54, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(54, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(54, PRIMARY\_EXPR, 72);  
   
 // State 55  
 \_actionTable.add(55, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(55, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(55, STAR, {SHIFT, 53});  
 \_actionTable.add(55, MINUS, {SHIFT, 54});  
 \_actionTable.add(55, BANG, {SHIFT, 55});  
 \_actionTable.add(55, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(55, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(55, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(55, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(55, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(55, UNARY\_EXPR, 107);  
 \_gotoTable.add(55, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(55, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(55, PRIMARY\_EXPR, 72);  
   
 // State 56  
 \_actionTable.add(56, IDENTIFIER, {SHIFT, 108});  
   
 // State 57  
 \_actionTable.addDefault(57, {REDUCE, 99});  
   
 // State 58  
 \_actionTable.addDefault(58, {REDUCE, 100});  
   
 // State 59  
 \_actionTable.addDefault(59, {REDUCE, 101});  
   
 // State 60  
 \_actionTable.add(60, IDENTIFIER, {SHIFT, 109});  
   
 // State 61  
 \_actionTable.add(61, IDENTIFIER, {SHIFT, 110});  
   
 // State 62  
 \_actionTable.addDefault(62, {REDUCE, 25});  
   
 // State 63  
 \_actionTable.addDefault(63, {REDUCE, 54});  
   
 // State 64  
 \_actionTable.add(64, PIPE\_PIPE, {SHIFT, 111});  
 \_actionTable.add(64, AMPERSAND\_AMPERSAND, {SHIFT, 112});  
 \_actionTable.addDefault(64, {REDUCE, 67});  
   
 // State 65  
 \_actionTable.add(65, LESS\_THAN, {SHIFT, 113});  
 \_actionTable.add(65, GREATER\_THAN, {SHIFT, 114});  
 \_actionTable.add(65, LESS\_THAN\_EQUALS, {SHIFT, 115});  
 \_actionTable.add(65, GREATER\_THAN\_EQUALS, {SHIFT, 116});  
 \_actionTable.add(65, EQUALS\_EQUALS, {SHIFT, 117});  
 \_actionTable.add(65, BANG\_EQUALS, {SHIFT, 118});  
 \_actionTable.addDefault(65, {REDUCE, 69});  
   
 // State 66  
 \_actionTable.add(66, MINUS, {SHIFT, 119});  
 \_actionTable.add(66, PLUS, {SHIFT, 120});  
 \_actionTable.add(66, PIPE, {SHIFT, 121});  
 \_actionTable.add(66, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(66, {REDUCE, 72});  
   
 // State 67  
 \_actionTable.add(67, STAR, {SHIFT, 123});  
 \_actionTable.add(67, SLASH, {SHIFT, 124});  
 \_actionTable.add(67, AMPERSAND, {SHIFT, 125});  
 \_actionTable.addDefault(67, {REDUCE, 79});  
   
 // State 68  
 \_actionTable.addDefault(68, {REDUCE, 84});  
   
 // State 69  
 \_actionTable.addDefault(69, {REDUCE, 105});  
   
 // State 70  
 \_actionTable.addDefault(70, {REDUCE, 68});  
   
 // State 71  
 \_actionTable.addDefault(71, {REDUCE, 104});  
   
 // State 72  
 \_actionTable.addDefault(72, {REDUCE, 88});  
   
 // State 73  
 \_actionTable.add(73, EQUALS, {SHIFT, 126});  
 \_actionTable.addDefault(73, {REDUCE, 27});  
 \_gotoTable.add(73, INIT\_OPT, 127);  
   
 // State 74  
 \_actionTable.addDefault(74, {REDUCE, 17});  
   
 // State 75  
 \_actionTable.addDefault(75, {REDUCE, 35});  
   
 // State 76  
 \_actionTable.addDefault(76, {REDUCE, 36});  
   
 // State 77  
 \_actionTable.addDefault(77, {REDUCE, 37});  
   
 // State 78  
 \_actionTable.addDefault(78, {REDUCE, 38});  
   
 // State 79  
 \_actionTable.addDefault(79, {REDUCE, 39});  
   
 // State 80  
 \_actionTable.addDefault(80, {REDUCE, 40});  
   
 // State 81  
 \_actionTable.addDefault(81, {REDUCE, 41});  
   
 // State 82  
 \_actionTable.addDefault(82, {REDUCE, 42});  
   
 // State 83  
 \_actionTable.addDefault(83, {REDUCE, 43});  
   
 // State 84  
 \_actionTable.add(84, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(84, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(84, STAR, {SHIFT, 53});  
 \_actionTable.add(84, MINUS, {SHIFT, 54});  
 \_actionTable.add(84, BANG, {SHIFT, 55});  
 \_actionTable.add(84, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(84, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(84, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(84, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(84, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(84, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(84, EXPR, 128);  
 \_gotoTable.add(84, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(84, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(84, ADD\_EXPR, 66);  
 \_gotoTable.add(84, MUL\_EXPR, 67);  
 \_gotoTable.add(84, UNARY\_EXPR, 68);  
 \_gotoTable.add(84, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(84, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(84, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(84, PRIMARY\_EXPR, 72);  
   
 // State 85  
 \_actionTable.add(85, CLOSED\_PAREN, {SHIFT, 129});  
   
 // State 86  
 \_actionTable.add(86, COMMA, {SHIFT, 130});  
 \_actionTable.addDefault(86, {REDUCE, 64});  
   
 // State 87  
 \_actionTable.addDefault(87, {REDUCE, 65});  
   
 // State 88  
 \_actionTable.add(88, CLOSED\_BRACKET, {SHIFT, 131});  
   
 // State 89  
 \_actionTable.addDefault(89, {REDUCE, 18});  
   
 // State 90  
 \_actionTable.add(90, OPEN\_PAREN, {SHIFT, 100});  
 \_actionTable.add(90, OPEN\_BRACKET, {SHIFT, 132});  
 \_actionTable.add(90, PLUS\_PLUS, {SHIFT, 102});  
 \_actionTable.add(90, MINUS\_MINUS, {SHIFT, 103});  
 \_actionTable.add(90, EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, PLUS\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, MINUS\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, SLASH\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, STAR\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, AMPERSAND\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, PIPE\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, CARET\_EQUALS, {REDUCE, 32});  
 \_actionTable.add(90, TILDE\_EQUALS, {REDUCE, 32});  
 \_actionTable.addDefault(90, {REDUCE, 97});  
   
 // State 91  
 \_actionTable.add(91, IDENTIFIER, {SHIFT, 133});  
   
 // State 92  
 \_actionTable.addDefault(92, {REDUCE, 48});  
   
 // State 93  
 \_actionTable.add(93, CLOSED\_PAREN, {SHIFT, 134});  
   
 // State 94  
 \_actionTable.addDefault(94, {REDUCE, 47});  
   
 // State 95  
 \_actionTable.add(95, CLOSED\_PAREN, {SHIFT, 135});  
   
 // State 96  
 \_actionTable.add(96, OPEN\_BRACKET, {SHIFT, 45});  
 \_actionTable.addDefault(96, {REDUCE, 32});  
   
 // State 97  
 \_actionTable.addDefault(97, {REDUCE, 51});  
   
 // State 98  
 \_actionTable.addDefault(98, {REDUCE, 52});  
   
 // State 99  
 \_actionTable.add(99, SEMICOLON, {SHIFT, 136});  
   
 // State 100  
 \_actionTable.add(100, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(100, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(100, STAR, {SHIFT, 53});  
 \_actionTable.add(100, MINUS, {SHIFT, 54});  
 \_actionTable.add(100, BANG, {SHIFT, 55});  
 \_actionTable.add(100, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(100, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(100, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(100, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(100, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(100, MINUS\_MINUS, {SHIFT, 61});  
 \_actionTable.addDefault(100, {REDUCE, 63});  
 \_gotoTable.add(100, EXPR\_LIST, 137);  
 \_gotoTable.add(100, EXPR\_LIST\_NON\_EMPTY, 86);  
 \_gotoTable.add(100, EXPR, 87);  
 \_gotoTable.add(100, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(100, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(100, ADD\_EXPR, 66);  
 \_gotoTable.add(100, MUL\_EXPR, 67);  
 \_gotoTable.add(100, UNARY\_EXPR, 68);  
 \_gotoTable.add(100, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(100, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(100, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(100, PRIMARY\_EXPR, 72);  
   
 // State 101  
 \_actionTable.add(101, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(101, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(101, STAR, {SHIFT, 53});  
 \_actionTable.add(101, MINUS, {SHIFT, 54});  
 \_actionTable.add(101, BANG, {SHIFT, 55});  
 \_actionTable.add(101, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(101, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(101, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(101, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(101, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(101, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(101, EXPR, 138);  
 \_gotoTable.add(101, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(101, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(101, ADD\_EXPR, 66);  
 \_gotoTable.add(101, MUL\_EXPR, 67);  
 \_gotoTable.add(101, UNARY\_EXPR, 68);  
 \_gotoTable.add(101, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(101, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(101, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(101, PRIMARY\_EXPR, 72);  
   
 // State 102  
 \_actionTable.addDefault(102, {REDUCE, 91});  
   
 // State 103  
 \_actionTable.addDefault(103, {REDUCE, 92});  
   
 // State 104  
 \_actionTable.add(104, CLOSED\_PAREN, {SHIFT, 139});  
   
 // State 105  
 \_actionTable.addDefault(105, {REDUCE, 96});  
   
 // State 106  
 \_actionTable.addDefault(106, {REDUCE, 89});  
   
 // State 107  
 \_actionTable.addDefault(107, {REDUCE, 90});  
   
 // State 108  
 \_actionTable.addDefault(108, {REDUCE, 95});  
   
 // State 109  
 \_actionTable.addDefault(109, {REDUCE, 93});  
   
 // State 110  
 \_actionTable.addDefault(110, {REDUCE, 94});  
   
 // State 111  
 \_actionTable.add(111, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(111, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(111, STAR, {SHIFT, 53});  
 \_actionTable.add(111, MINUS, {SHIFT, 54});  
 \_actionTable.add(111, BANG, {SHIFT, 55});  
 \_actionTable.add(111, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(111, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(111, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(111, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(111, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(111, RELATIONAL\_EXPR, 140);  
 \_gotoTable.add(111, ADD\_EXPR, 66);  
 \_gotoTable.add(111, MUL\_EXPR, 67);  
 \_gotoTable.add(111, UNARY\_EXPR, 68);  
 \_gotoTable.add(111, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(111, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(111, PRIMARY\_EXPR, 72);  
   
 // State 112  
 \_actionTable.add(112, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(112, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(112, STAR, {SHIFT, 53});  
 \_actionTable.add(112, MINUS, {SHIFT, 54});  
 \_actionTable.add(112, BANG, {SHIFT, 55});  
 \_actionTable.add(112, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(112, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(112, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(112, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(112, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(112, RELATIONAL\_EXPR, 141);  
 \_gotoTable.add(112, ADD\_EXPR, 66);  
 \_gotoTable.add(112, MUL\_EXPR, 67);  
 \_gotoTable.add(112, UNARY\_EXPR, 68);  
 \_gotoTable.add(112, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(112, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(112, PRIMARY\_EXPR, 72);  
   
 // State 113  
 \_actionTable.add(113, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(113, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(113, STAR, {SHIFT, 53});  
 \_actionTable.add(113, MINUS, {SHIFT, 54});  
 \_actionTable.add(113, BANG, {SHIFT, 55});  
 \_actionTable.add(113, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(113, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(113, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(113, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(113, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(113, ADD\_EXPR, 142);  
 \_gotoTable.add(113, MUL\_EXPR, 67);  
 \_gotoTable.add(113, UNARY\_EXPR, 68);  
 \_gotoTable.add(113, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(113, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(113, PRIMARY\_EXPR, 72);  
   
 // State 114  
 \_actionTable.add(114, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(114, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(114, STAR, {SHIFT, 53});  
 \_actionTable.add(114, MINUS, {SHIFT, 54});  
 \_actionTable.add(114, BANG, {SHIFT, 55});  
 \_actionTable.add(114, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(114, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(114, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(114, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(114, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(114, ADD\_EXPR, 143);  
 \_gotoTable.add(114, MUL\_EXPR, 67);  
 \_gotoTable.add(114, UNARY\_EXPR, 68);  
 \_gotoTable.add(114, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(114, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(114, PRIMARY\_EXPR, 72);  
   
 // State 115  
 \_actionTable.add(115, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(115, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(115, STAR, {SHIFT, 53});  
 \_actionTable.add(115, MINUS, {SHIFT, 54});  
 \_actionTable.add(115, BANG, {SHIFT, 55});  
 \_actionTable.add(115, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(115, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(115, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(115, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(115, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(115, ADD\_EXPR, 144);  
 \_gotoTable.add(115, MUL\_EXPR, 67);  
 \_gotoTable.add(115, UNARY\_EXPR, 68);  
 \_gotoTable.add(115, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(115, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(115, PRIMARY\_EXPR, 72);  
   
 // State 116  
 \_actionTable.add(116, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(116, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(116, STAR, {SHIFT, 53});  
 \_actionTable.add(116, MINUS, {SHIFT, 54});  
 \_actionTable.add(116, BANG, {SHIFT, 55});  
 \_actionTable.add(116, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(116, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(116, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(116, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(116, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(116, ADD\_EXPR, 145);  
 \_gotoTable.add(116, MUL\_EXPR, 67);  
 \_gotoTable.add(116, UNARY\_EXPR, 68);  
 \_gotoTable.add(116, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(116, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(116, PRIMARY\_EXPR, 72);  
   
 // State 117  
 \_actionTable.add(117, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(117, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(117, STAR, {SHIFT, 53});  
 \_actionTable.add(117, MINUS, {SHIFT, 54});  
 \_actionTable.add(117, BANG, {SHIFT, 55});  
 \_actionTable.add(117, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(117, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(117, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(117, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(117, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(117, ADD\_EXPR, 146);  
 \_gotoTable.add(117, MUL\_EXPR, 67);  
 \_gotoTable.add(117, UNARY\_EXPR, 68);  
 \_gotoTable.add(117, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(117, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(117, PRIMARY\_EXPR, 72);  
   
 // State 118  
 \_actionTable.add(118, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(118, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(118, STAR, {SHIFT, 53});  
 \_actionTable.add(118, MINUS, {SHIFT, 54});  
 \_actionTable.add(118, BANG, {SHIFT, 55});  
 \_actionTable.add(118, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(118, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(118, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(118, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(118, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(118, ADD\_EXPR, 147);  
 \_gotoTable.add(118, MUL\_EXPR, 67);  
 \_gotoTable.add(118, UNARY\_EXPR, 68);  
 \_gotoTable.add(118, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(118, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(118, PRIMARY\_EXPR, 72);  
   
 // State 119  
 \_actionTable.add(119, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(119, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(119, STAR, {SHIFT, 53});  
 \_actionTable.add(119, MINUS, {SHIFT, 54});  
 \_actionTable.add(119, BANG, {SHIFT, 55});  
 \_actionTable.add(119, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(119, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(119, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(119, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(119, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(119, MUL\_EXPR, 148);  
 \_gotoTable.add(119, UNARY\_EXPR, 68);  
 \_gotoTable.add(119, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(119, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(119, PRIMARY\_EXPR, 72);  
   
 // State 120  
 \_actionTable.add(120, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(120, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(120, STAR, {SHIFT, 53});  
 \_actionTable.add(120, MINUS, {SHIFT, 54});  
 \_actionTable.add(120, BANG, {SHIFT, 55});  
 \_actionTable.add(120, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(120, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(120, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(120, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(120, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(120, MUL\_EXPR, 149);  
 \_gotoTable.add(120, UNARY\_EXPR, 68);  
 \_gotoTable.add(120, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(120, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(120, PRIMARY\_EXPR, 72);  
   
 // State 121  
 \_actionTable.add(121, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(121, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(121, STAR, {SHIFT, 53});  
 \_actionTable.add(121, MINUS, {SHIFT, 54});  
 \_actionTable.add(121, BANG, {SHIFT, 55});  
 \_actionTable.add(121, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(121, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(121, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(121, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(121, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(121, MUL\_EXPR, 150);  
 \_gotoTable.add(121, UNARY\_EXPR, 68);  
 \_gotoTable.add(121, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(121, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(121, PRIMARY\_EXPR, 72);  
   
 // State 122  
 \_actionTable.add(122, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(122, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(122, STAR, {SHIFT, 53});  
 \_actionTable.add(122, MINUS, {SHIFT, 54});  
 \_actionTable.add(122, BANG, {SHIFT, 55});  
 \_actionTable.add(122, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(122, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(122, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(122, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(122, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(122, MUL\_EXPR, 151);  
 \_gotoTable.add(122, UNARY\_EXPR, 68);  
 \_gotoTable.add(122, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(122, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(122, PRIMARY\_EXPR, 72);  
   
 // State 123  
 \_actionTable.add(123, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(123, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(123, STAR, {SHIFT, 53});  
 \_actionTable.add(123, MINUS, {SHIFT, 54});  
 \_actionTable.add(123, BANG, {SHIFT, 55});  
 \_actionTable.add(123, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(123, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(123, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(123, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(123, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(123, UNARY\_EXPR, 152);  
 \_gotoTable.add(123, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(123, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(123, PRIMARY\_EXPR, 72);  
   
 // State 124  
 \_actionTable.add(124, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(124, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(124, STAR, {SHIFT, 53});  
 \_actionTable.add(124, MINUS, {SHIFT, 54});  
 \_actionTable.add(124, BANG, {SHIFT, 55});  
 \_actionTable.add(124, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(124, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(124, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(124, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(124, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(124, UNARY\_EXPR, 153);  
 \_gotoTable.add(124, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(124, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(124, PRIMARY\_EXPR, 72);  
   
 // State 125  
 \_actionTable.add(125, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(125, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(125, STAR, {SHIFT, 53});  
 \_actionTable.add(125, MINUS, {SHIFT, 54});  
 \_actionTable.add(125, BANG, {SHIFT, 55});  
 \_actionTable.add(125, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(125, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(125, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(125, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(125, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(125, UNARY\_EXPR, 154);  
 \_gotoTable.add(125, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(125, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(125, PRIMARY\_EXPR, 72);  
   
 // State 126  
 \_actionTable.add(126, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(126, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(126, OPEN\_CURLY, {SHIFT, 155});  
 \_actionTable.add(126, STAR, {SHIFT, 53});  
 \_actionTable.add(126, MINUS, {SHIFT, 54});  
 \_actionTable.add(126, BANG, {SHIFT, 55});  
 \_actionTable.add(126, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(126, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(126, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(126, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(126, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(126, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(126, ASSIGN\_VALUE, 156);  
 \_gotoTable.add(126, EXPR, 157);  
 \_gotoTable.add(126, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(126, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(126, ADD\_EXPR, 66);  
 \_gotoTable.add(126, MUL\_EXPR, 67);  
 \_gotoTable.add(126, UNARY\_EXPR, 68);  
 \_gotoTable.add(126, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(126, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(126, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(126, PRIMARY\_EXPR, 72);  
   
 // State 127  
 \_actionTable.addDefault(127, {REDUCE, 26});  
   
 // State 128  
 \_actionTable.addDefault(128, {REDUCE, 31});  
   
 // State 129  
 \_actionTable.addDefault(129, {REDUCE, 24});  
   
 // State 130  
 \_actionTable.add(130, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(130, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(130, STAR, {SHIFT, 53});  
 \_actionTable.add(130, MINUS, {SHIFT, 54});  
 \_actionTable.add(130, BANG, {SHIFT, 55});  
 \_actionTable.add(130, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(130, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(130, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(130, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(130, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(130, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(130, EXPR, 158);  
 \_gotoTable.add(130, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(130, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(130, ADD\_EXPR, 66);  
 \_gotoTable.add(130, MUL\_EXPR, 67);  
 \_gotoTable.add(130, UNARY\_EXPR, 68);  
 \_gotoTable.add(130, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(130, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(130, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(130, PRIMARY\_EXPR, 72);  
   
 // State 131  
 \_actionTable.addDefault(131, {REDUCE, 33});  
   
 // State 132  
 \_actionTable.add(132, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(132, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(132, STAR, {SHIFT, 53});  
 \_actionTable.add(132, MINUS, {SHIFT, 54});  
 \_actionTable.add(132, BANG, {SHIFT, 55});  
 \_actionTable.add(132, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(132, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(132, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(132, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(132, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(132, MINUS\_MINUS, {SHIFT, 61});  
 \_gotoTable.add(132, EXPR, 159);  
 \_gotoTable.add(132, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(132, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(132, ADD\_EXPR, 66);  
 \_gotoTable.add(132, MUL\_EXPR, 67);  
 \_gotoTable.add(132, UNARY\_EXPR, 68);  
 \_gotoTable.add(132, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(132, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(132, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(132, PRIMARY\_EXPR, 72);  
   
 // State 133  
 \_actionTable.add(133, EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, PLUS\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, MINUS\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, SLASH\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, STAR\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, AMPERSAND\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, PIPE\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, CARET\_EQUALS, {REDUCE, 34});  
 \_actionTable.add(133, TILDE\_EQUALS, {REDUCE, 34});  
 \_actionTable.addDefault(133, {REDUCE, 96});  
   
 // State 134  
 \_actionTable.add(134, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(134, OPEN\_CURLY, {SHIFT, 160});  
 \_actionTable.add(134, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(134, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(134, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(134, STAR, {SHIFT, 30});  
 \_actionTable.add(134, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(134, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(134, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(134, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(134, TYPE, 35);  
 \_gotoTable.add(134, BASE\_TYPE, 14);  
 \_gotoTable.add(134, SIMPLE\_STMT, 161);  
 \_gotoTable.add(134, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(134, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(134, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(134, IF\_STMT, 162);  
 \_gotoTable.add(134, WHILE\_STMT, 163);  
 \_gotoTable.add(134, FOR\_STMT, 164);  
 \_gotoTable.add(134, BODY, 165);  
   
 // State 135  
 \_actionTable.add(135, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(135, OPEN\_CURLY, {SHIFT, 160});  
 \_actionTable.add(135, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(135, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(135, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(135, STAR, {SHIFT, 30});  
 \_actionTable.add(135, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(135, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(135, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(135, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(135, TYPE, 35);  
 \_gotoTable.add(135, BASE\_TYPE, 14);  
 \_gotoTable.add(135, SIMPLE\_STMT, 161);  
 \_gotoTable.add(135, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(135, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(135, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(135, IF\_STMT, 162);  
 \_gotoTable.add(135, WHILE\_STMT, 163);  
 \_gotoTable.add(135, FOR\_STMT, 164);  
 \_gotoTable.add(135, BODY, 166);  
   
 // State 136  
 \_actionTable.add(136, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(136, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(136, STAR, {SHIFT, 53});  
 \_actionTable.add(136, MINUS, {SHIFT, 54});  
 \_actionTable.add(136, BANG, {SHIFT, 55});  
 \_actionTable.add(136, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(136, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(136, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(136, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(136, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(136, MINUS\_MINUS, {SHIFT, 61});  
 \_actionTable.addDefault(136, {REDUCE, 53});  
 \_gotoTable.add(136, EXPR\_OPT, 167);  
 \_gotoTable.add(136, EXPR, 63);  
 \_gotoTable.add(136, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(136, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(136, ADD\_EXPR, 66);  
 \_gotoTable.add(136, MUL\_EXPR, 67);  
 \_gotoTable.add(136, UNARY\_EXPR, 68);  
 \_gotoTable.add(136, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(136, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(136, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(136, PRIMARY\_EXPR, 72);  
   
 // State 137  
 \_actionTable.add(137, CLOSED\_PAREN, {SHIFT, 168});  
   
 // State 138  
 \_actionTable.add(138, CLOSED\_BRACKET, {SHIFT, 169});  
   
 // State 139  
 \_actionTable.addDefault(139, {REDUCE, 102});  
   
 // State 140  
 \_actionTable.add(140, LESS\_THAN, {SHIFT, 113});  
 \_actionTable.add(140, GREATER\_THAN, {SHIFT, 114});  
 \_actionTable.add(140, LESS\_THAN\_EQUALS, {SHIFT, 115});  
 \_actionTable.add(140, GREATER\_THAN\_EQUALS, {SHIFT, 116});  
 \_actionTable.add(140, EQUALS\_EQUALS, {SHIFT, 117});  
 \_actionTable.add(140, BANG\_EQUALS, {SHIFT, 118});  
 \_actionTable.addDefault(140, {REDUCE, 70});  
   
 // State 141  
 \_actionTable.add(141, LESS\_THAN, {SHIFT, 113});  
 \_actionTable.add(141, GREATER\_THAN, {SHIFT, 114});  
 \_actionTable.add(141, LESS\_THAN\_EQUALS, {SHIFT, 115});  
 \_actionTable.add(141, GREATER\_THAN\_EQUALS, {SHIFT, 116});  
 \_actionTable.add(141, EQUALS\_EQUALS, {SHIFT, 117});  
 \_actionTable.add(141, BANG\_EQUALS, {SHIFT, 118});  
 \_actionTable.addDefault(141, {REDUCE, 71});  
   
 // State 142  
 \_actionTable.add(142, MINUS, {SHIFT, 119});  
 \_actionTable.add(142, PLUS, {SHIFT, 120});  
 \_actionTable.add(142, PIPE, {SHIFT, 121});  
 \_actionTable.add(142, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(142, {REDUCE, 75});  
   
 // State 143  
 \_actionTable.add(143, MINUS, {SHIFT, 119});  
 \_actionTable.add(143, PLUS, {SHIFT, 120});  
 \_actionTable.add(143, PIPE, {SHIFT, 121});  
 \_actionTable.add(143, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(143, {REDUCE, 76});  
   
 // State 144  
 \_actionTable.add(144, MINUS, {SHIFT, 119});  
 \_actionTable.add(144, PLUS, {SHIFT, 120});  
 \_actionTable.add(144, PIPE, {SHIFT, 121});  
 \_actionTable.add(144, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(144, {REDUCE, 77});  
   
 // State 145  
 \_actionTable.add(145, MINUS, {SHIFT, 119});  
 \_actionTable.add(145, PLUS, {SHIFT, 120});  
 \_actionTable.add(145, PIPE, {SHIFT, 121});  
 \_actionTable.add(145, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(145, {REDUCE, 78});  
   
 // State 146  
 \_actionTable.add(146, MINUS, {SHIFT, 119});  
 \_actionTable.add(146, PLUS, {SHIFT, 120});  
 \_actionTable.add(146, PIPE, {SHIFT, 121});  
 \_actionTable.add(146, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(146, {REDUCE, 73});  
   
 // State 147  
 \_actionTable.add(147, MINUS, {SHIFT, 119});  
 \_actionTable.add(147, PLUS, {SHIFT, 120});  
 \_actionTable.add(147, PIPE, {SHIFT, 121});  
 \_actionTable.add(147, CARET, {SHIFT, 122});  
 \_actionTable.addDefault(147, {REDUCE, 74});  
   
 // State 148  
 \_actionTable.add(148, STAR, {SHIFT, 123});  
 \_actionTable.add(148, SLASH, {SHIFT, 124});  
 \_actionTable.add(148, AMPERSAND, {SHIFT, 125});  
 \_actionTable.addDefault(148, {REDUCE, 81});  
   
 // State 149  
 \_actionTable.add(149, STAR, {SHIFT, 123});  
 \_actionTable.add(149, SLASH, {SHIFT, 124});  
 \_actionTable.add(149, AMPERSAND, {SHIFT, 125});  
 \_actionTable.addDefault(149, {REDUCE, 80});  
   
 // State 150  
 \_actionTable.add(150, STAR, {SHIFT, 123});  
 \_actionTable.add(150, SLASH, {SHIFT, 124});  
 \_actionTable.add(150, AMPERSAND, {SHIFT, 125});  
 \_actionTable.addDefault(150, {REDUCE, 82});  
   
 // State 151  
 \_actionTable.add(151, STAR, {SHIFT, 123});  
 \_actionTable.add(151, SLASH, {SHIFT, 124});  
 \_actionTable.add(151, AMPERSAND, {SHIFT, 125});  
 \_actionTable.addDefault(151, {REDUCE, 83});  
   
 // State 152  
 \_actionTable.addDefault(152, {REDUCE, 85});  
   
 // State 153  
 \_actionTable.addDefault(153, {REDUCE, 86});  
   
 // State 154  
 \_actionTable.addDefault(154, {REDUCE, 87});  
   
 // State 155  
 \_actionTable.add(155, IDENTIFIER, {SHIFT, 51});  
 \_actionTable.add(155, OPEN\_PAREN, {SHIFT, 52});  
 \_actionTable.add(155, STAR, {SHIFT, 53});  
 \_actionTable.add(155, MINUS, {SHIFT, 54});  
 \_actionTable.add(155, BANG, {SHIFT, 55});  
 \_actionTable.add(155, AMPERSAND, {SHIFT, 56});  
 \_actionTable.add(155, INTEGER\_LITERAL, {SHIFT, 57});  
 \_actionTable.add(155, FLOAT\_LITERAL, {SHIFT, 58});  
 \_actionTable.add(155, CHAR\_LITERAL, {SHIFT, 59});  
 \_actionTable.add(155, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(155, MINUS\_MINUS, {SHIFT, 61});  
 \_actionTable.addDefault(155, {REDUCE, 63});  
 \_gotoTable.add(155, EXPR\_LIST, 170);  
 \_gotoTable.add(155, EXPR\_LIST\_NON\_EMPTY, 86);  
 \_gotoTable.add(155, EXPR, 87);  
 \_gotoTable.add(155, LOGICAL\_EXPR, 64);  
 \_gotoTable.add(155, RELATIONAL\_EXPR, 65);  
 \_gotoTable.add(155, ADD\_EXPR, 66);  
 \_gotoTable.add(155, MUL\_EXPR, 67);  
 \_gotoTable.add(155, UNARY\_EXPR, 68);  
 \_gotoTable.add(155, INCREMENT\_EXPR, 69);  
 \_gotoTable.add(155, ADDRESS\_EXPR, 70);  
 \_gotoTable.add(155, DEREFERENCE\_EXPR, 71);  
 \_gotoTable.add(155, PRIMARY\_EXPR, 72);  
   
 // State 156  
 \_actionTable.addDefault(156, {REDUCE, 28});  
   
 // State 157  
 \_actionTable.addDefault(157, {REDUCE, 29});  
   
 // State 158  
 \_actionTable.addDefault(158, {REDUCE, 66});  
   
 // State 159  
 \_actionTable.add(159, CLOSED\_BRACKET, {SHIFT, 171});  
   
 // State 160  
 \_actionTable.addDefault(160, {REDUCE, 15});  
 \_gotoTable.add(160, STMT\_LIST, 172);  
   
 // State 161  
 \_actionTable.add(161, SEMICOLON, {SHIFT, 173});  
   
 // State 162  
 \_actionTable.addDefault(162, {REDUCE, 60});  
   
 // State 163  
 \_actionTable.addDefault(163, {REDUCE, 61});  
   
 // State 164  
 \_actionTable.addDefault(164, {REDUCE, 62});  
   
 // State 165  
 \_actionTable.add(165, KEYWORD\_ELSE, {SHIFT, 174});  
 \_actionTable.addDefault(165, {REDUCE, 44});  
   
 // State 166  
 \_actionTable.addDefault(166, {REDUCE, 46});  
   
 // State 167  
 \_actionTable.add(167, SEMICOLON, {SHIFT, 175});  
   
 // State 168  
 \_actionTable.addDefault(168, {REDUCE, 103});  
   
 // State 169  
 \_actionTable.addDefault(169, {REDUCE, 98});  
   
 // State 170  
 \_actionTable.add(170, CLOSED\_CURLY, {SHIFT, 176});  
   
 // State 171  
 \_actionTable.add(171, EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, PLUS\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, MINUS\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, SLASH\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, STAR\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, AMPERSAND\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, PIPE\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, CARET\_EQUALS, {REDUCE, 33});  
 \_actionTable.add(171, TILDE\_EQUALS, {REDUCE, 33});  
 \_actionTable.addDefault(171, {REDUCE, 98});  
   
 // State 172  
 \_actionTable.add(172, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(172, OPEN\_CURLY, {SHIFT, 28});  
 \_actionTable.add(172, CLOSED\_CURLY, {SHIFT, 177});  
 \_actionTable.add(172, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(172, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(172, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(172, STAR, {SHIFT, 30});  
 \_actionTable.add(172, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(172, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(172, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(172, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(172, TYPE, 35);  
 \_gotoTable.add(172, BASE\_TYPE, 14);  
 \_gotoTable.add(172, STMT, 36);  
 \_gotoTable.add(172, SIMPLE\_STMT, 37);  
 \_gotoTable.add(172, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(172, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(172, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(172, IF\_STMT, 41);  
 \_gotoTable.add(172, WHILE\_STMT, 42);  
 \_gotoTable.add(172, FOR\_STMT, 43);  
   
 // State 173  
 \_actionTable.addDefault(173, {REDUCE, 59});  
   
 // State 174  
 \_actionTable.add(174, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(174, OPEN\_CURLY, {SHIFT, 160});  
 \_actionTable.add(174, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(174, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(174, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(174, STAR, {SHIFT, 30});  
 \_actionTable.add(174, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(174, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(174, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(174, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(174, TYPE, 35);  
 \_gotoTable.add(174, BASE\_TYPE, 14);  
 \_gotoTable.add(174, SIMPLE\_STMT, 161);  
 \_gotoTable.add(174, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(174, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(174, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(174, IF\_STMT, 162);  
 \_gotoTable.add(174, WHILE\_STMT, 163);  
 \_gotoTable.add(174, FOR\_STMT, 164);  
 \_gotoTable.add(174, BODY, 178);  
   
 // State 175  
 \_actionTable.add(175, IDENTIFIER, {SHIFT, 179});  
 \_actionTable.add(175, STAR, {SHIFT, 30});  
 \_actionTable.add(175, PLUS\_PLUS, {SHIFT, 60});  
 \_actionTable.add(175, MINUS\_MINUS, {SHIFT, 61});  
 \_actionTable.addDefault(175, {REDUCE, 55});  
 \_gotoTable.add(175, ASSIGN\_EXPR, 180);  
 \_gotoTable.add(175, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(175, FOR\_UPDATE, 181);  
 \_gotoTable.add(175, INCREMENT\_EXPR, 182);  
   
 // State 176  
 \_actionTable.addDefault(176, {REDUCE, 30});  
   
 // State 177  
 \_actionTable.addDefault(177, {REDUCE, 58});  
   
 // State 178  
 \_actionTable.addDefault(178, {REDUCE, 45});  
   
 // State 179  
 \_actionTable.add(179, OPEN\_BRACKET, {SHIFT, 45});  
 \_actionTable.add(179, PLUS\_PLUS, {SHIFT, 102});  
 \_actionTable.add(179, MINUS\_MINUS, {SHIFT, 103});  
 \_actionTable.addDefault(179, {REDUCE, 32});  
   
 // State 180  
 \_actionTable.addDefault(180, {REDUCE, 56});  
   
 // State 181  
 \_actionTable.add(181, CLOSED\_PAREN, {SHIFT, 183});  
   
 // State 182  
 \_actionTable.addDefault(182, {REDUCE, 57});  
   
 // State 183  
 \_actionTable.add(183, IDENTIFIER, {SHIFT, 27});  
 \_actionTable.add(183, OPEN\_CURLY, {SHIFT, 160});  
 \_actionTable.add(183, KEYWORD\_INT, {SHIFT, 7});  
 \_actionTable.add(183, KEYWORD\_FLOAT, {SHIFT, 8});  
 \_actionTable.add(183, KEYWORD\_CHAR, {SHIFT, 9});  
 \_actionTable.add(183, STAR, {SHIFT, 30});  
 \_actionTable.add(183, KEYWORD\_IF, {SHIFT, 31});  
 \_actionTable.add(183, KEYWORD\_WHILE, {SHIFT, 32});  
 \_actionTable.add(183, KEYWORD\_FOR, {SHIFT, 33});  
 \_actionTable.add(183, KEYWORD\_RET, {SHIFT, 34});  
 \_gotoTable.add(183, TYPE, 35);  
 \_gotoTable.add(183, BASE\_TYPE, 14);  
 \_gotoTable.add(183, SIMPLE\_STMT, 161);  
 \_gotoTable.add(183, VAR\_DECL\_EXPR, 38);  
 \_gotoTable.add(183, ASSIGN\_EXPR, 39);  
 \_gotoTable.add(183, ASSIGN\_TARGET, 40);  
 \_gotoTable.add(183, IF\_STMT, 162);  
 \_gotoTable.add(183, WHILE\_STMT, 163);  
 \_gotoTable.add(183, FOR\_STMT, 164);  
 \_gotoTable.add(183, BODY, 184);  
   
 // State 184  
 \_actionTable.addDefault(184, {REDUCE, 49});  
   
 }  
   
   
 void Parser::initFollowSets()  
 {  
 // \*\*FOLLOW Sets for Non-Terminals\*\*  
 \_followSets[NonTerminal::START] = {SyntaxKind::END\_OF\_FILE};  
 \_followSets[NonTerminal::PROGRAM] = {SyntaxKind::END\_OF\_FILE, SyntaxKind::KEYWORD\_FN};  
 \_followSets[NonTerminal::FUNCTION\_DECL] = {SyntaxKind::END\_OF\_FILE, SyntaxKind::KEYWORD\_FN};  
 \_followSets[NonTerminal::PARAM\_LIST] = {SyntaxKind::CLOSED\_PAREN};  
 \_followSets[NonTerminal::TYPE] = {SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_CURLY};  
 \_followSets[NonTerminal::STMT\_LIST] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_WHILE,  
 SyntaxKind::KEYWORD\_FOR};  
 \_followSets[NonTerminal::PARAM\_LIST\_NON\_EMPTY] = {SyntaxKind::CLOSED\_PAREN, SyntaxKind::COMMA};  
 \_followSets[NonTerminal::PARAM] = {SyntaxKind::CLOSED\_PAREN, SyntaxKind::COMMA};  
 \_followSets[NonTerminal::BASE\_TYPE] = {SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_CURLY, SyntaxKind::OPEN\_BRACKET, SyntaxKind::STAR};  
 \_followSets[NonTerminal::STMT] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_WHILE,  
 SyntaxKind::KEYWORD\_FOR};  
 \_followSets[NonTerminal::SIMPLE\_STMT] = {SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::IF\_STMT] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_ELSE,  
 SyntaxKind::KEYWORD\_WHILE, SyntaxKind::KEYWORD\_FOR};  
 \_followSets[NonTerminal::WHILE\_STMT] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_ELSE,  
 SyntaxKind::KEYWORD\_WHILE, SyntaxKind::KEYWORD\_FOR};  
 \_followSets[NonTerminal::FOR\_STMT] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_ELSE,  
 SyntaxKind::KEYWORD\_WHILE, SyntaxKind::KEYWORD\_FOR};  
 \_followSets[NonTerminal::VAR\_DECL\_EXPR] = {SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::ASSIGN\_EXPR] = {SyntaxKind::CLOSED\_PAREN, SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::EXPR\_OPT] = {SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::INIT\_OPT] = {SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::ASSIGN\_VALUE] = {SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::EXPR\_LIST] = {SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY};  
 \_followSets[NonTerminal::ASSIGN\_TARGET] = {  
 SyntaxKind::EQUALS, SyntaxKind::PLUS\_EQUALS, SyntaxKind::MINUS\_EQUALS, SyntaxKind::SLASH\_EQUALS,  
 SyntaxKind::STAR\_EQUALS, SyntaxKind::AMPERSAND\_EQUALS, SyntaxKind::PIPE\_EQUALS, SyntaxKind::CARET\_EQUALS,  
 SyntaxKind::TILDE\_EQUALS};  
 \_followSets[NonTerminal::ASSIGN\_OP] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR};  
 \_followSets[NonTerminal::CONDITION\_OP] = {SyntaxKind::CLOSED\_PAREN};  
 \_followSets[NonTerminal::BODY] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_ELSE,  
 SyntaxKind::KEYWORD\_WHILE, SyntaxKind::KEYWORD\_FOR};  
 \_followSets[NonTerminal::FOR\_INIT] = {SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::FOR\_UPDATE] = {SyntaxKind::CLOSED\_PAREN};  
 \_followSets[NonTerminal::INCREMENT\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::EXPR\_LIST\_NON\_EMPTY] = {SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA};  
 \_followSets[NonTerminal::LOGICAL\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::ADDRESS\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::DEREFERENCE\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::RELATIONAL\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::ADD\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::MUL\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::STAR, SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::UNARY\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::STAR, SyntaxKind::SEMICOLON};  
 \_followSets[NonTerminal::PRIMARY\_EXPR] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::STAR, SyntaxKind::SEMICOLON};  
   
 // \*\*FOLLOW Sets for Terminals\*\*  
 // Common set for tokens that start expressions (used by operators)  
 std::unordered\_set<SyntaxKind> exprStarters = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::INTEGER\_LITERAL, SyntaxKind::FLOAT\_LITERAL,  
 SyntaxKind::CHAR\_LITERAL, SyntaxKind::STRING\_LITERAL, SyntaxKind::STAR, SyntaxKind::AMPERSAND,  
 SyntaxKind::BANG, SyntaxKind::PLUS\_PLUS, SyntaxKind::MINUS\_MINUS, SyntaxKind::MINUS};  
   
 // Binary operators (e.g., +, -, \*, /, ||, &&, ==, etc.)  
 \_followTerminalsSets[SyntaxKind::PIPE\_PIPE] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::AMPERSAND\_AMPERSAND] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::PLUS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::MINUS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::PIPE] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::CARET] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::SLASH] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::STAR] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::EQUALS\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::BANG\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::LESS\_THAN] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::GREATER\_THAN] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::LESS\_THAN\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::GREATER\_THAN\_EQUALS] = exprStarters;  
   
 // Assignment operators (e.g., =, +=, -=, etc.)  
 \_followTerminalsSets[SyntaxKind::EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::PLUS\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::MINUS\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::SLASH\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::STAR\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::AMPERSAND\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::PIPE\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::CARET\_EQUALS] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::TILDE\_EQUALS] = exprStarters;  
   
 // Unary operators  
 \_followTerminalsSets[SyntaxKind::AMPERSAND] = {SyntaxKind::IDENTIFIER}; // Address-of  
 \_followTerminalsSets[SyntaxKind::BANG] = exprStarters; // Logical not  
 \_followTerminalsSets[SyntaxKind::MINUS] = exprStarters; // Unary minus  
   
 // Increment/Decrement operators  
 \_followTerminalsSets[SyntaxKind::PLUS\_PLUS] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA,  
 SyntaxKind::CLOSED\_BRACKET, SyntaxKind::SEMICOLON};  
 \_followTerminalsSets[SyntaxKind::MINUS\_MINUS] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA,  
 SyntaxKind::CLOSED\_BRACKET, SyntaxKind::SEMICOLON};  
   
 // Literals  
 std::unordered\_set<SyntaxKind> literalFollow = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON, SyntaxKind::PLUS, SyntaxKind::MINUS, SyntaxKind::STAR, SyntaxKind::SLASH,  
 SyntaxKind::PIPE, SyntaxKind::AMPERSAND, SyntaxKind::CARET, SyntaxKind::PIPE\_PIPE, SyntaxKind::AMPERSAND\_AMPERSAND,  
 SyntaxKind::EQUALS\_EQUALS, SyntaxKind::BANG\_EQUALS, SyntaxKind::LESS\_THAN, SyntaxKind::GREATER\_THAN,  
 SyntaxKind::LESS\_THAN\_EQUALS, SyntaxKind::GREATER\_THAN\_EQUALS};  
 \_followTerminalsSets[SyntaxKind::INTEGER\_LITERAL] = literalFollow;  
 \_followTerminalsSets[SyntaxKind::FLOAT\_LITERAL] = literalFollow;  
 \_followTerminalsSets[SyntaxKind::CHAR\_LITERAL] = literalFollow;  
 \_followTerminalsSets[SyntaxKind::STRING\_LITERAL] = literalFollow;  
   
 // Keywords  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_FN] = {SyntaxKind::IDENTIFIER};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_INT] = {SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_CURLY, SyntaxKind::OPEN\_BRACKET, SyntaxKind::STAR};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_FLOAT] = {SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_CURLY, SyntaxKind::OPEN\_BRACKET, SyntaxKind::STAR};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_CHAR] = {SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_CURLY, SyntaxKind::OPEN\_BRACKET, SyntaxKind::STAR};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_RET] = exprStarters; // Followed by optional expression  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_IF] = {SyntaxKind::OPEN\_PAREN};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_ELSE] = {  
 SyntaxKind::OPEN\_CURLY, SyntaxKind::IDENTIFIER, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_WHILE,  
 SyntaxKind::KEYWORD\_FOR};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_WHILE] = {SyntaxKind::OPEN\_PAREN};  
 \_followTerminalsSets[SyntaxKind::KEYWORD\_FOR] = {SyntaxKind::OPEN\_PAREN};  
   
 // Punctuation  
 \_followTerminalsSets[SyntaxKind::OPEN\_PAREN] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::CLOSED\_PAREN] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON, SyntaxKind::PLUS, SyntaxKind::MINUS, SyntaxKind::STAR, SyntaxKind::SLASH,  
 SyntaxKind::PIPE, SyntaxKind::AMPERSAND, SyntaxKind::CARET, SyntaxKind::PIPE\_PIPE, SyntaxKind::AMPERSAND\_AMPERSAND,  
 SyntaxKind::EQUALS\_EQUALS, SyntaxKind::BANG\_EQUALS, SyntaxKind::LESS\_THAN, SyntaxKind::GREATER\_THAN,  
 SyntaxKind::LESS\_THAN\_EQUALS, SyntaxKind::GREATER\_THAN\_EQUALS};  
 \_followTerminalsSets[SyntaxKind::OPEN\_CURLY] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_WHILE,  
 SyntaxKind::KEYWORD\_FOR};  
 \_followTerminalsSets[SyntaxKind::CLOSED\_CURLY] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_ELSE,  
 SyntaxKind::KEYWORD\_WHILE, SyntaxKind::KEYWORD\_FOR};  
 \_followTerminalsSets[SyntaxKind::OPEN\_BRACKET] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::CLOSED\_BRACKET] = {  
 SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA, SyntaxKind::CLOSED\_BRACKET,  
 SyntaxKind::SEMICOLON, SyntaxKind::PLUS, SyntaxKind::MINUS, SyntaxKind::STAR, SyntaxKind::SLASH,  
 SyntaxKind::PIPE, SyntaxKind::AMPERSAND, SyntaxKind::CARET, SyntaxKind::PIPE\_PIPE, SyntaxKind::AMPERSAND\_AMPERSAND,  
 SyntaxKind::EQUALS\_EQUALS, SyntaxKind::BANG\_EQUALS, SyntaxKind::LESS\_THAN, SyntaxKind::GREATER\_THAN,  
 SyntaxKind::LESS\_THAN\_EQUALS, SyntaxKind::GREATER\_THAN\_EQUALS};  
 \_followTerminalsSets[SyntaxKind::COMMA] = exprStarters;  
 \_followTerminalsSets[SyntaxKind::SEMICOLON] = {  
 SyntaxKind::IDENTIFIER, SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_CURLY, SyntaxKind::CLOSED\_CURLY,  
 SyntaxKind::INTEGER\_LITERAL, SyntaxKind::STAR, SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT,  
 SyntaxKind::KEYWORD\_CHAR, SyntaxKind::KEYWORD\_RET, SyntaxKind::KEYWORD\_IF, SyntaxKind::KEYWORD\_WHILE,  
 SyntaxKind::KEYWORD\_FOR};  
 \_followTerminalsSets[SyntaxKind::RIGHT\_ARROW] = {SyntaxKind::KEYWORD\_INT, SyntaxKind::KEYWORD\_FLOAT, SyntaxKind::KEYWORD\_CHAR};  
   
 // Identifier  
 \_followTerminalsSets[SyntaxKind::IDENTIFIER] = {  
 SyntaxKind::OPEN\_PAREN, SyntaxKind::OPEN\_BRACKET, SyntaxKind::PLUS\_PLUS, SyntaxKind::MINUS\_MINUS,  
 SyntaxKind::EQUALS, SyntaxKind::PLUS\_EQUALS, SyntaxKind::MINUS\_EQUALS, SyntaxKind::SLASH\_EQUALS,  
 SyntaxKind::STAR\_EQUALS, SyntaxKind::AMPERSAND\_EQUALS, SyntaxKind::PIPE\_EQUALS, SyntaxKind::CARET\_EQUALS,  
 SyntaxKind::TILDE\_EQUALS, SyntaxKind::CLOSED\_PAREN, SyntaxKind::CLOSED\_CURLY, SyntaxKind::COMMA,  
 SyntaxKind::CLOSED\_BRACKET, SyntaxKind::SEMICOLON, SyntaxKind::PLUS, SyntaxKind::MINUS, SyntaxKind::STAR,  
 SyntaxKind::SLASH, SyntaxKind::PIPE, SyntaxKind::AMPERSAND, SyntaxKind::CARET, SyntaxKind::PIPE\_PIPE,  
 SyntaxKind::AMPERSAND\_AMPERSAND, SyntaxKind::EQUALS\_EQUALS, SyntaxKind::BANG\_EQUALS, SyntaxKind::LESS\_THAN,  
 SyntaxKind::GREATER\_THAN, SyntaxKind::LESS\_THAN\_EQUALS, SyntaxKind::GREATER\_THAN\_EQUALS};  
 }

## parser.cpp

#include "parser.hpp"  
 #include "../errors/errors.hpp"  
 #include "../nodes/nodes.hpp"  
 #include "stackItem/stackItem.hpp"  
 #include "../token/token.hpp"  
 #include "grammerSymbol/grammerSymbol.hpp"  
 #include "productionRule/productionRule.hpp"  
 #include "../symbolTable/functionEntry/functionEntry.hpp"  
 #include <vector>  
 #include <iostream>  
   
 // #define PARSER\_DEBUG  
   
 using namespace std;  
   
 Parser::Parser(vector<SyntaxToken \*> tokens, int numOfStates, ErrorHandler \*handler, SemanticAnalyzer \*semanticAnalyser)  
 : \_actionTable(numOfStates), \_gotoTable(numOfStates), \_semanticAnalyzer(semanticAnalyser), \_rules(), \_stack(), \_errorHandler(handler), \_tokens(tokens), \_cursor(0)  
 {  
 \_stack.push(StackItem{0, new NonTerminalNode(NonTerminal::START)});  
 initProductionRules();  
 fillTables();  
 initFollowSets();  
 }  
   
 SyntaxToken \*Parser::getNextToken()  
 {  
 \_cursor++;  
 return peek(0);  
 }  
   
 SyntaxToken \*Parser::peek(int index)  
 {  
 if (\_cursor + index >= \_tokens.size())  
 {  
 SyntaxToken \*resToken = new SyntaxToken();  
 resToken->kind = SyntaxKind::END\_OF\_FILE;  
 return resToken;  
 }  
   
 return \_tokens[\_cursor + index];  
 }  
   
 SyntaxToken \*Parser::getCurrToken()  
 {  
 return peek(0);  
 }  
   
 void Parser::addProductionRule(productionRule rule)  
 {  
 \_rules.push\_back(rule);  
 }  
   
 action Parser::getCurrAction()  
 {  
 return \_actionTable.get(\_stack.top().state, peek(0)->kind);  
 }  
   
 void Parser::shift(action currAction)  
 {  
 int state = currAction.num;  
 SyntaxToken \*token = getCurrToken();  
   
 #ifdef PARSER\_DEBUG  
 cout << "Shifting to state " << currAction.num << " with token " << syntaxTokenToString(\*token) << endl;  
 #endif  
   
 // Create a new terminalNode  
 ASTNode \*node = new TerminalNode(token);  
   
 // assign the node type for semantic analysis  
 \_semanticAnalyzer->assignNodeType(node);  
   
 // Push to stack  
 \_stack.push(StackItem{state, node});  
   
 // Update current state  
 \_currState = state;  
   
 // update the scope  
 \_semanticAnalyzer -> updateScope(getCurrToken());  
   
 // proceed to next token  
 getNextToken();  
 }  
   
 void Parser::reduce(action currAction)  
 {  
 int productionRuleNum = currAction.num;  
   
 #ifdef PARSER\_DEBUG  
 cout << "Reducing with rule " << productionRuleNum << endl;  
 #endif  
   
 productionRule rule = \_rules[productionRuleNum];  
   
 #ifdef PARSER\_DEBUG  
 cout << rule.toString() << endl;  
 #endif  
   
 // Create a new nonTerminalNode  
 NonTerminalNode \*node = new NonTerminalNode(rule.getLeft());  
   
 if ((rule.getType(0) == GrammarSymbolType::TERMINAL) && (rule.getTerminal(0) == EPSILON)) // if rule is empty return without adding children  
 {  
 \_stack.push(StackItem{\_gotoTable.get(\_stack.top().state, rule.getLeft()), node});  
 }  
 else  
 {  
 reduceStatmentToNode(node, rule);  
   
 // Push to stack  
 int currentState = \_stack.top().state;  
 \_stack.push(StackItem{\_gotoTable.get(currentState, rule.getLeft()), node});  
   
 // Update current state  
 \_currState = currentState;  
 }  
   
 \_semanticAnalyzer->assignNodeType(node); // assign the node type for semantic analysis  
 \_semanticAnalyzer-> updateSybolTable(\_stack.top().node); // update the symbol table with the new node  
 }  
   
 void Parser::reduceStatmentToNode(NonTerminalNode \*node, productionRule rule)  
 {  
 // Pop from stack and create the new node  
 for (int i = rule.getNumOfRightSideSymbols() - 1; i >= 0; i--)  
 {  
 node->AddChildToFront(\_stack.top().node);  
 \_stack.pop();  
 }  
 }  
   
   
   
 ASTNode \*Parser::parse()  
 {  
 action currAction;  
 bool panic = false;  
   
 do  
 {  
 #ifdef PARSER\_DEBUG  
 cout << endl;  
 printStack();  
 cout << endl;  
 #endif  
   
 int state = \_stack.top().state;  
   
 currAction = getCurrAction();  
   
 #ifdef PARSER\_DEBUG  
 cout << "state:" << state << " token:" << syntaxTokenToString(\*peek(0)) << " Action: " << actionTypeToString(currAction) << endl;  
 #endif  
   
 if (currAction.type == actionType::REDUCE)  
 {  
 reduce(currAction);  
 panic = false; // reset panic mode  
 }  
 else if (currAction.type == actionType::SHIFT)  
 {  
 shift(currAction);  
 panic = false; // reset panic mode  
 }  
 else if (currAction.type == actionType::DEFAULT) // error  
 {  
 if (!panic)  
 {  
 panic = true; // enter panic mode  
 reportParsingError();  
 }  
   
 getNextToken();  
   
 if (getCurrToken()->kind == SyntaxKind::END\_OF\_FILE)  
 {  
 currAction.type = actionType::ACCEPT; // end of file token  
 }  
 }  
   
 } while (currAction.type != actionType::ACCEPT);  
   
 \_semanticAnalyzer->checkForMainFunction(); // check if the main function is declared  
   
 \_stack.pop(); // pop the end of file token  
   
 #ifdef PARSER\_DEBUG  
 PrintParseTree(\_stack.top().node);  
 #endif  
   
 return \_stack.top().node;  
 }  
   
 void Parser::reportParsingError()  
 {  
 // adding error to the error handler  
 // with the expected token  
 ASTNode \*node = \_stack.top().node;  
 SyntaxKind recoveryKind;  
 if (node->GetType() == GrammarSymbolType::TERMINAL)  
 {  
 recoveryKind = getTerminalFollowSetItem(((TerminalNode \*)node)->getToken()->kind);  
 \_errorHandler->addError(new SyntacticError(peek(0), recoveryKind));  
 }  
 else  
 {  
 recoveryKind = getNonTerminalFollowSetItem(((NonTerminalNode \*)node)->getNonTerminalKind());  
 \_errorHandler->addError(new SyntacticError(peek(0), recoveryKind));  
 }  
 }  
   
 SyntaxKind Parser::getNonTerminalFollowSetItem(NonTerminal nt)  
 {  
 SyntaxKind res = SyntaxKind::UNEXPECTED\_TOKEN;  
   
 auto it = \_followSets.find(nt);  
 if (it != \_followSets.end())  
 {  
 res = \*it->second.begin();  
 }  
   
 return res; // return the first item in the follow set  
 }  
   
 SyntaxKind Parser::getTerminalFollowSetItem(SyntaxKind kind)  
 {  
 SyntaxKind res = SyntaxKind::UNEXPECTED\_TOKEN;  
   
 auto it = \_followTerminalsSets.find(kind);  
 if (it != \_followTerminalsSets.end())  
 {  
 res = \*it->second.begin();  
 }  
   
 return res; // return the first item in the follow set  
 }  
   
 void Parser::printStack()  
 {  
 if (\_stack.empty())  
 {  
 cout << "[Empty Stack]" << endl;  
 return;  
 }  
   
 stack<StackItem> tempStack;  
 StackItem item;  
   
 while (!\_stack.empty())  
 {  
 item = \_stack.top();  
 tempStack.push(item);  
 \_stack.pop();  
 }  
   
 while (!tempStack.empty())  
 {  
 item = tempStack.top();  
 cout << stackItemToString(item) << ", ";  
 tempStack.pop();  
 \_stack.push(item);  
 }  
   
 cout << " ->" << endl;  
 }  
   
 void Parser::printRules()  
 {  
 for (int i = 0; i < \_rules.size(); i++)  
 {  
 cout << "\e[0;31mRule " << i << "\e[0;37m : " << \_rules[i].toString() << endl;  
 }  
 }  
   
 void Parser::printFollowSet()  
 {  
 for (int i = 0; i < \_rules.size(); i++)  
 {  
 NonTerminal nt = \_rules[i].getLeft();  
 cout << "follow(" << nonTerminalToString(nt) << ") = {";  
   
 int j = \_followSets[\_rules[i].getLeft()].size(); // size of follow set  
 for (const SyntaxKind &terminal : \_followSets[nt])  
 {  
 cout << syntaxKindToString(terminal);  
 if (j != 1)  
 cout << ", ";  
   
 j--;  
 }  
   
 cout << "}" << endl;  
 }  
   
 for (int i = 0; i < SYNTAX\_KIND\_COUNT; i++)  
 {  
 SyntaxKind kind = (SyntaxKind)i;  
 cout << "follow(" << syntaxKindToString(kind) << ") = {";  
   
 int j = \_followTerminalsSets[kind].size(); // size of follow set  
 for (const SyntaxKind &terminal : \_followTerminalsSets[kind])  
 {  
 cout << syntaxKindToString(terminal);  
 if (j != 1)  
 cout << ", ";  
   
 j--;  
 }  
   
 cout << "}" << endl;  
 }  
 }

## parser.hpp

#ifndef PARSER  
 #define PARSER  
   
 #include "productionRule/productionRule.hpp"  
 #include "parseTable/gotoTable/gotoTable.hpp"  
 #include "parseTable/actionTable/actionTable.hpp"  
 #include "productionRule/productionRule.hpp"  
 #include "stackItem/stackItem.hpp"  
 #include "../nodes/nodes.hpp"  
 #include "../errorHandler/errorHandler.hpp"  
 #include "../errors/errors.hpp"  
 #include "../token/token.hpp"  
 #include "symbolTable/symbolTable.hpp"  
 #include "../scope/scope.hpp"  
 #include "../semantic/semantic.hpp"  
 #include <stack>  
 #include <string>  
 #include <vector>  
 #include <sstream>  
 #include <algorithm>  
 #include <unordered\_map>  
 #include <unordered\_set>  
 #include <iostream>  
   
 extern string nonTerminalKindToString[];  
   
 class Parser  
 {  
 private:  
 // parse table  
 ActionTable \_actionTable;  
 GotoTable \_gotoTable;  
 stack<StackItem> \_stack;  
 int \_currState;  
   
 action getCurrAction();  
 void shift(action currAction);  
 void reduce(action currAction);  
 void reduceStatmentToNode(NonTerminalNode \*node, productionRule rule);  
   
 // init functions  
 void initProductionRules();  
 void fillTables();  
   
 // Follow sets  
 unordered\_map<NonTerminal, unordered\_set<SyntaxKind>> \_followSets;  
 unordered\_map<SyntaxKind, unordered\_set<SyntaxKind>> \_followTerminalsSets;  
 SyntaxKind getNonTerminalFollowSetItem(NonTerminal nt);  
 SyntaxKind getTerminalFollowSetItem(SyntaxKind kind);  
   
 void initFollowSets();  
   
 // Navigation and helpers  
 vector<SyntaxToken \*> \_tokens;  
 vector<productionRule> \_rules;  
 int \_cursor;  
   
 SyntaxToken \*getNextToken();  
 SyntaxToken \*getCurrToken();  
 SyntaxToken \*peek(int index);  
 void addProductionRule(productionRule rule);  
   
 // error handling  
 ErrorHandler \*\_errorHandler;  
   
 void reportParsingError();  
   
 // semantic analysis  
 SemanticAnalyzer \*\_semanticAnalyzer;  
 void assignNodeType(ASTNode \*node);  
   
 public:  
 Parser(vector<SyntaxToken \*> tokens, int numOfStates, ErrorHandler \*handler, SemanticAnalyzer \*semanticAnalyzer);  
   
 ASTNode \*parse();  
   
 // Debugging  
 void printStack();  
 void printRules();  
 void printFollowSet();  
 };  
   
 #endif

## action.cpp

#include "action.hpp"  
   
 string actionTypeToString(action type)  
 {  
 stringstream res;  
 switch (type.type)  
 {  
 case actionType::SHIFT:  
 res << "s" << type.num;  
 break;  
 case actionType::REDUCE:  
 res << "r" << type.num;  
 break;  
 case actionType::ACCEPT:  
 res << "acc";  
 break;  
 case actionType::DEFAULT:  
 res << "DEFAULT";  
 break;  
 default:  
 res << "UNKNOWN";  
 break;  
 }  
   
 return res.str();  
 }

## action.hpp

#ifndef \_\_ACTION  
 #define \_\_ACTION  
 #include <string>  
 #include <sstream>  
   
 using namespace std;  
   
 enum actionType  
 {  
 SHIFT,  
 REDUCE,  
 ACCEPT,  
 DEFAULT,  
 };  
   
 struct action  
 {  
 actionType type;  
 int num;  
 };  
   
 string actionTypeToString(action type);  
 #endif

## actionTable.cpp

#include "actionTable.hpp"  
 #include <map>  
 #include <vector>  
 #include "../action/action.hpp"  
 #include "../../../nodes/nodes.hpp"  
 #include "../../../token/token.hpp"  
 #include "../../grammerSymbol/grammerSymbol.hpp"  
   
 using namespace std;  
   
 ActionTable ::ActionTable(int numOfStates) : \_numOfStates(numOfStates)  
 {  
 \_table = new action \*[\_numOfStates];  
 \_defaultActions = new action[\_numOfStates];  
 for (int i = 0; i < \_numOfStates; i++)  
 {  
 \_defaultActions[i] = {actionType::DEFAULT, -1};  
 \_table[i] = new action[SyntaxKind::SYNTAX\_KIND\_COUNT];  
 for (int j = 0; j < SyntaxKind::SYNTAX\_KIND\_COUNT; j++)  
 {  
 \_table[i][j].type = actionType::DEFAULT;  
 \_table[i][j].num = -1;  
 }  
 }  
   
   
 }  
   
 ActionTable ::~ActionTable()  
 {  
 for (int i = 0; i < \_numOfStates; i++)  
 {  
 delete[] \_table[i];  
 }  
   
 delete[] \_defaultActions;  
 delete[] \_table;  
 }  
   
 void ActionTable ::add(int state, SyntaxKind terminal, action act)  
 {  
 if (state >= \_numOfStates || terminal >= SyntaxKind::SYNTAX\_KIND\_COUNT)  
 {  
 runtime\_error("Invalid state or symbol");  
 }  
   
 else if (\_table[state][terminal].type == actionType::DEFAULT)  
 {  
 \_table[state][terminal] = act;  
 }  
   
 else  
 {  
 runtime\_error("Conflict in action table");  
 }  
 }  
   
 void ActionTable::addDefault(int state, action act)  
 {  
 if (state >= \_numOfStates)  
 {  
 runtime\_error("Invalid state");  
 }  
   
 else if (\_defaultActions[state].num == -1)  
 {  
 \_defaultActions[state] = act;  
 }  
   
 else  
 {  
 runtime\_error("Conflict in action table");  
 }  
 }  
   
 action ActionTable ::get(int state, SyntaxKind terminal)  
 {  
 if (state >= \_numOfStates || terminal >= SyntaxKind::SYNTAX\_KIND\_COUNT)  
 {  
 runtime\_error("Invalid state or terminal");  
 }  
   
 if(\_table[state][terminal].type == DEFAULT)  
 {  
 return \_defaultActions[state];  
 }  
   
 return \_table[state][terminal];  
 }

## actionTable.hpp

#ifndef \_\_ACTION\_TABLE  
 #define \_\_ACTION\_TABLE  
   
 #include "../action/action.hpp"  
 #include "../../../token/token.hpp"  
 #include "../../grammerSymbol/grammerSymbol.hpp"  
   
 class ActionTable  
 {  
 private:  
 action \*\*\_table;  
 action \*\_defaultActions;  
 int \_numOfStates;  
   
 public:  
 ActionTable(int numOfStates);  
 ~ActionTable();  
 void add(int state, SyntaxKind terminal, action act);  
 void addDefault(int state, action act);  
 action get(int state, SyntaxKind terminal);  
 };  
   
 #endif

## gotoTable.cpp

#include "gotoTable.hpp"  
 #include "../../../token/token.hpp"  
 #include "../../grammerSymbol/grammerSymbol.hpp"  
 #include <map>  
 #include <vector>  
 #include <iostream>  
   
 using namespace std;  
   
 GotoTable::GotoTable(int numOfStates) : \_numOfStates(numOfStates)  
 {  
 \_table = new int \*[\_numOfStates];  
 for (int i = 0; i < \_numOfStates; i++)  
 {  
 \_table[i] = new int[NonTerminal::NON\_TERMINAL\_COUNT];  
 for (int j = 0; j < NonTerminal::NON\_TERMINAL\_COUNT; j++)  
 {  
 \_table[i][j] = -1;  
 }  
 }  
 }  
   
 GotoTable::~GotoTable()  
 {  
 }  
   
 // adds a new entry to the table  
 void GotoTable::add(int startState, NonTerminal nt, int endState)  
 {  
 if (nt >= NonTerminal::NON\_TERMINAL\_COUNT || startState >= \_numOfStates || endState >= \_numOfStates)  
 {  
 cerr << "Invalid input on gotoTable::add() function" << endl;  
 exit(1);  
 }  
 else if (\_table[startState][nt] == -1)  
 {  
 \_table[startState][nt] = endState;  
 }  
 else  
 {  
 cerr << "Conflict in goto table" << endl;  
 exit(1);  
 }  
 }  
   
 // gets the end state of a non terminal in a given state  
 int GotoTable::get(int state, NonTerminal nt)  
 {  
 if (nt >= NonTerminal::NON\_TERMINAL\_COUNT || state >= \_numOfStates)  
 {  
 runtime\_error("Invalid non terminal or state");  
 }  
   
 return \_table[state][nt];  
 }

## gotoTable.hpp

#ifndef \_\_GOTO\_TABLE  
 #define \_\_GOTO\_TABLE  
   
 #include "../../../token/token.hpp"  
 #include "../../grammerSymbol/grammerSymbol.hpp"  
   
 #include <map>  
 #include <vector>  
   
 class GotoTable  
 {  
 private:  
 int \*\*\_table;  
   
 public:  
 int \_numOfStates;  
   
 GotoTable(int numOfStates);  
 ~GotoTable();  
 void add(int startState, NonTerminal nt, int endState);  
 int get(int state, NonTerminal nt);  
 };  
   
 #endif

## productionRule.cpp

#include "productionRule.hpp"  
 #include "../GrammerSymbol/grammerSymbol.hpp"  
 #include "../../token/token.hpp"  
 #include <iostream>  
 #include <sstream>  
 #include <vector>  
 #include <map>  
   
 using namespace std;  
   
 productionRule::productionRule(NonTerminal left) : \_left(left), \_numOfRightSideSymbols(0) {}  
   
 productionRule::productionRule() :productionRule(NonTerminal::PROGRAM) {}  
   
 productionRule::~productionRule() {}  
   
 productionRule& productionRule::addSymbol(SyntaxKind terminal)  
 {  
 \_rightSideSymbols.push\_back(terminal);  
 \_rightSideTypes.push\_back(GrammarSymbolType::TERMINAL);  
 \_numOfRightSideSymbols++;  
   
 return \*this;  
 }  
   
 productionRule& productionRule::addSymbol(NonTerminal nonTerminal)  
 {  
 \_rightSideSymbols.push\_back(nonTerminal);  
 \_rightSideTypes.push\_back(GrammarSymbolType::NON\_TERMINAL);  
 \_numOfRightSideSymbols++;  
   
 return \*this;  
 }  
   
 productionRule& productionRule::addSymbol()  
 {  
 return addSymbol(SyntaxKind::EPSILON);  
 }  
   
   
   
 SyntaxKind productionRule::getTerminal(int index)  
 {  
 if (index >= \_numOfRightSideSymbols)  
 {  
 cerr << ("Index out of range");  
 exit(1);  
 }  
   
 if (\_rightSideTypes[index] != GrammarSymbolType::TERMINAL)  
 {  
 cerr << ("Symbol at index is not a terminal");  
 exit(1);  
 }  
   
 return (SyntaxKind)\_rightSideSymbols[index];  
 }  
   
 NonTerminal productionRule::getNonTerminal(int index)  
 {  
 if (index >= \_numOfRightSideSymbols)  
 {  
 cerr << ("Index out of range");  
 exit(1);  
 }  
   
 if (\_rightSideTypes[index] != GrammarSymbolType::NON\_TERMINAL)  
 {  
 cerr << ("Symbol at index is not a non terminal");  
 exit(1);  
 }  
   
 return (NonTerminal)\_rightSideSymbols[index];  
 }  
   
 GrammarSymbolType productionRule::getType(int index)  
 {  
 if (index >= \_numOfRightSideSymbols)  
 {  
 runtime\_error("Index out of range");  
 }  
   
 return \_rightSideTypes[index];  
 }  
   
 int productionRule::getNumOfRightSideSymbols()  
 {  
 return \_numOfRightSideSymbols;  
 }  
   
 NonTerminal productionRule::getLeft()  
 {  
 return \_left;  
 }  
   
 void productionRule::setLeft(NonTerminal left)  
 {  
 \_left = left;  
 }  
   
 void productionRule::reset()  
 {  
 \_numOfRightSideSymbols = 0;  
 \_rightSideSymbols.clear();  
 \_rightSideTypes.clear();  
 }  
   
 string productionRule::toString()  
 {  
 stringstream str;  
 str << nonTerminalToString(\_left)  
 << " \e[0;33m=>\033[0m";  
   
 for (int i = 0; i < \_numOfRightSideSymbols; i++)  
 {  
 if (\_rightSideTypes[i] == GrammarSymbolType::TERMINAL)  
 {  
 str << " " << syntaxKindToString((SyntaxKind)(\_rightSideSymbols[i]));  
 }  
 else  
 {  
 str << " " << nonTerminalToString((NonTerminal)\_rightSideSymbols[i]);  
 }  
 }  
   
 return str.str();  
 }

## productionRule.hpp

#ifndef \_\_PRODUCTION\_RULE  
 #define \_\_PRODUCTION\_RULE  
   
 #include "../../token/token.hpp"  
 #include "../grammerSymbol/grammerSymbol.hpp"  
 #include <vector>  
   
 using namespace std;  
   
 class productionRule  
 {  
 private:  
 // left side of the production rule  
 NonTerminal \_left;  
   
 // right side of the production rule  
 int \_numOfRightSideSymbols;  
 vector<int> \_rightSideSymbols;  
 vector<GrammarSymbolType> \_rightSideTypes;  
   
 public:  
 productionRule(NonTerminal left);  
 productionRule();  
 ~productionRule();  
 productionRule& addSymbol(SyntaxKind terminal);  
 productionRule& addSymbol(NonTerminal nonTerminal);  
 productionRule& addSymbol();  
 NonTerminal getNonTerminal(int index);  
 SyntaxKind getTerminal(int index);  
 GrammarSymbolType getType(int index);  
 int getNumOfRightSideSymbols();  
 NonTerminal getLeft();  
 void setLeft(NonTerminal left);  
 void reset();  
 string toString();  
 };  
   
 #endif

## stackItem.cpp

#include "stackItem.hpp"  
 #include "../../nodes/nodes.hpp"  
 #include <string>  
 #include <sstream>  
 #include <iostream>  
   
 using namespace std;  
   
 string stackItemToString(StackItem item)  
 {  
 stringstream res;  
 res << "(" << ""<<AstNodeToString(item.node)<<"" << " | " << item.state << ")";  
 return res.str();  
 }  
   
 void printStackItem(StackItem item)  
 {  
 cout << stackItemToString(item) << endl;  
 }

## stackItem.hpp

#ifndef \_\_STACK\_ITEM  
 #define \_\_STACK\_ITEM  
 #include <string>  
   
 #include "../../nodes/nodes.hpp"  
   
 using namespace std;  
   
 struct StackItem  
 {  
 int state;  
 ASTNode \*node;  
 };  
   
 string stackItemToString(StackItem item);  
 void printStackItem(StackItem item);  
 #endif

## semantic.cpp

#include "semantic.hpp"  
 #include "../symbolTable/symbolTable.hpp"  
 #include "../token/token.hpp"  
 #include <vector>  
   
 using namespace std;  
   
 std::map<SyntaxKind, baseType> assignTerminal = {  
 {KEYWORD\_INT, INT},  
 {KEYWORD\_CHAR, CHAR},  
 {KEYWORD\_FLOAT, FLOAT},  
 {INTEGER\_LITERAL, INT},  
 {CHAR\_LITERAL, CHAR},  
 {FLOAT\_LITERAL, FLOAT}};  
   
 SemanticAnalyzer::SemanticAnalyzer(ErrorHandler \*errorHandler, SymbolTable \*symbolTable) : \_errorHandler(errorHandler), \_symbolTable(symbolTable)  
 {  
 initAssignActions();  
 }  
   
 void SemanticAnalyzer::addFunctionNodeToSymbolTable(NonTerminalNode \*funcDeclNode)  
 {  
 /// if we encounter a function declaration we need to procces it correctly in order to update the symbol table  
   
 // get the second child value (name)  
 TerminalNode \*nameNode = (TerminalNode \*)(funcDeclNode->GetChildren()[1]);  
 string name = ((TerminalNode \*)(funcDeclNode->GetChildren()[1]))->getToken()->val;  
   
 // check if the function already exists in the symbol table  
 functionEntry \*funcEntry = \_symbolTable->getFunction(name);  
 if (funcEntry != nullptr)  
 {  
 \_errorHandler->addError(new semanticError("function already declared", nameNode->getToken()));  
 }  
 else  
 {  
 // get the seventh child value (the type)  
 valType funcVarType = createVarDeclExprType((NonTerminalNode \*)(funcDeclNode->GetChildren()[6]));  
   
 // get the forth child value (the types of the function parameters)  
 vector<valType> paramTypes = createFunctionParamTypes((NonTerminalNode \*)(funcDeclNode->GetChildren()[3]));  
   
 // create the entry  
 functionEntry \*funcEntry = new functionEntry(name, funcVarType, paramTypes);  
   
 // check return statements  
 checkReturnStatements(funcEntry);  
   
 // add the current root scope as their scope  
 funcEntry->setInnerScope(\_currRootScope);  
   
 // and finally, add the function to the table  
 \_symbolTable->addFunction(funcEntry);  
 }  
 }  
   
 void SemanticAnalyzer::addVariableNodeToSymbolTable(NonTerminalNode \*varDeclNode)  
 {  
 // get the current scope  
 scope \*currScope = \_scopeStack.top();  
   
 // check if the variable already exists in the symbol table (in current or parent scope)  
 TerminalNode \*nameNode = (TerminalNode \*)(varDeclNode->GetChildren()[1]);  
 string name = nameNode->getToken()->val;  
 tableEntry entry = currScope->getEntry(name);  
   
 if (entry.name != "\_undeclared")  
 {  
 \_errorHandler->addError(new semanticError("variable already declared", nameNode->getToken()));  
 }  
 else  
 {  
 // add variable to the symbol table  
 currScope->addTableEntry(createTableEntery\_varDec(varDeclNode));  
 }  
 }  
   
 void SemanticAnalyzer::addParamNodeToSymbolTable(NonTerminalNode \*paramNode)  
 {  
 // get the current scope  
 scope \*currScope = \_scopeStack.top();  
   
 // check if the variable already exists in the symbol table (in current or parent scope)  
 TerminalNode \*nameNode = (TerminalNode \*)(paramNode->GetChildren()[1]);  
 string name = nameNode->getToken()->val;  
 tableEntry entry = currScope->getEntry(name);  
   
 if (entry.name != "\_undeclared")  
 {  
 \_errorHandler->addError(new semanticError("variable already declared", nameNode->getToken()));  
 }  
 else  
 {  
 // add variable to the symbol table  
 currScope->addTableEntry(createTableEntery\_param(paramNode));  
 }  
 }  
   
 void SemanticAnalyzer::addParamListToSymbolTable(vector<NonTerminalNode \*> paramNodes)  
 {  
 // get the current scope  
 scope \*currScope = \_scopeStack.top();  
 currScope->printScope();  
   
 for (int i = 0; i < paramNodes.size(); i++)  
 {  
 addParamNodeToSymbolTable(paramNodes[i]);  
 }  
 }  
   
 void SemanticAnalyzer::updateScope(SyntaxToken \*currToken)  
 {  
 if (currToken->kind == SyntaxKind::OPEN\_CURLY)  
 {  
 // entering scope  
 scope \*newScope = new scope();  
   
 bool isFunctionScope = true;  
   
 // if the stack is empty it means its a new function  
 // so just connect it to the stack knowing it dosent have a parent  
 // later wre connect the function to the scope in the updateSymbolTable() func  
   
 if (!\_scopeStack.empty())  
 {  
 scope \*oldScope = \_scopeStack.top();  
 oldScope->addInnerScope(newScope);  
 newScope->setParentScope(oldScope);  
 isFunctionScope = false;  
 }  
   
 \_scopeStack.push(newScope);  
   
 if (isFunctionScope)  
 {  
 addParamListToSymbolTable(\_currFunctionParamNodes);  
 \_currFunctionParamNodes.clear(); // clear the param nodes after adding them to the symbol table  
 }  
 }  
 else if (currToken->kind == SyntaxKind::CLOSED\_CURLY)  
 {  
 // exiting scope  
 scope \*currScope = \_scopeStack.top();  
 \_scopeStack.pop();  
   
 if (\_scopeStack.empty())  
 \_currRootScope = currScope; // if the stack is empty it means we are at the end of the function  
 }  
 }  
   
 void SemanticAnalyzer::updateSybolTable(ASTNode \*node)  
 {  
 if (node->GetType() == NON\_TERMINAL)  
 {  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)node;  
 if (ntNode->getNonTerminalKind() == FUNCTION\_DECL)  
 {  
 addFunctionNodeToSymbolTable(ntNode);  
 }  
   
 else if (ntNode->getNonTerminalKind() == VAR\_DECL\_EXPR)  
 {  
 addVariableNodeToSymbolTable(ntNode);  
 }  
   
 else if (ntNode->getNonTerminalKind() == PARAM\_LIST)  
 {  
 \_currFunctionParamNodes = getFunctionParamNodes(ntNode);  
 }  
   
 else if (isReturnStatement(node))  
 {  
 \_currFunctionReturnNodes.push\_back((NonTerminalNode \*)node);  
 }  
 }  
 }  
   
 void SemanticAnalyzer::assignNodeType(ASTNode \*node)  
 {  
 if (node->GetType() == NON\_TERMINAL)  
 {  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)node;  
 NonTerminal ntKind = ntNode->getNonTerminalKind();  
   
 if (\_nonTerminalAssignActions.find(ntKind) != \_nonTerminalAssignActions.end())  
 {  
 (this->\*\_nonTerminalAssignActions[ntKind])(node);  
 }  
 }  
 else if (node->GetType() == TERMINAL)  
 {  
 TerminalNode \*tNode = (TerminalNode \*)node;  
 SyntaxKind kind = tNode->getToken()->kind;  
   
 if (assignTerminal.find(kind) != assignTerminal.end())  
 {  
 node->SetValType(valType{assignTerminal[kind], 0, false, false});  
 }  
 }  
 }  
   
 bool areTypesEqual(valType a, valType b)  
 {  
 return a.type == b.type &&  
 a.isPointer == b.isPointer &&  
 a.isArray == b.isArray;  
 }  
   
 valType checkArithmeticCompatibility(valType left, valType right, SyntaxToken \*opToken, ErrorHandler \*\_errorHandler)  
 {  
 valType res = {UNDIFINED, 0, false, false};  
 bool errorOccurred = false;  
   
 if (left.isPointer || right.isPointer)  
 {  
 if ((opToken->kind == PLUS || opToken->kind == MINUS) &&  
 ((left.isPointer && right.isPointer)))  
 {  
 res = left.isPointer ? left : right;  
 }  
 else  
 {  
 \_errorHandler->addError(new semanticError("pointer arithmetic not allowed", opToken));  
 errorOccurred = true;  
 }  
 }  
 else if (left.isArray || right.isArray)  
 {  
 \_errorHandler->addError(new semanticError("array arithmetic not allowed", opToken));  
 errorOccurred = true;  
 }  
 else if (left.type == FLOAT || right.type == FLOAT)  
 {  
 res = {FLOAT, 0, false, false};  
 }  
 else if (left.type == INT || right.type == INT)  
 {  
 res = {INT, 0, false, false};  
 }  
 else if (left.type == CHAR || right.type == CHAR)  
 {  
 res = {INT, 0, false, false};  
 }  
 else  
 {  
 \_errorHandler->addError(new semanticError("incompatible types for arithmetic", opToken));  
 errorOccurred = true;  
 }  
   
 if (errorOccurred)  
 res = {UNDIFINED, 0, false, false};  
   
 return res;  
 }  
   
 valType checkComparisonCompatibility(valType left, valType right, SyntaxToken \*opToken, ErrorHandler \*\_errorHandler)  
 {  
 valType res = {INT, 0, false, false};  
 if (!areTypesEqual(left, right))  
 {  
 \_errorHandler->addError(new semanticError("incompatible types for comparison", opToken));  
 res = {UNDIFINED, 0, false, false};  
 }  
 return res;  
 }  
   
 valType checkLogicalCompatibility(valType left, valType right, SyntaxToken \*opToken, ErrorHandler \*\_errorHandler)  
 {  
 valType res = {INT, 0, false, false};  
 if (left.type != INT || right.type != INT)  
 {  
 \_errorHandler->addError(new semanticError("logical operators require int operands", opToken));  
 res = {UNDIFINED, 0, false, false};  
 }  
 return res;  
 }  
   
 valType checkBitwiseCompatibility(valType left, valType right, SyntaxToken \*opToken, ErrorHandler \*\_errorHandler)  
 {  
 valType res = {INT, 0, false, false};  
 if ((left.type != INT && left.type != CHAR) || (right.type != INT && right.type != CHAR))  
 {  
 \_errorHandler->addError(new semanticError("bitwise operators require int or char operands", opToken));  
 res = {UNDIFINED, 0, false, false};  
 }  
 return res;  
 }  
   
 valType SemanticAnalyzer::checkCompatibilityBinaryOp(valType leftOp, valType rightOp, SyntaxToken \*opToken)  
 {  
 valType res = {UNDIFINED, 0, false, false};  
   
 if (opToken->kind == PLUS || opToken->kind == MINUS || opToken->kind == STAR || opToken->kind == SLASH || opToken->kind == COMMA)  
 {  
 res = checkArithmeticCompatibility(leftOp, rightOp, opToken, \_errorHandler);  
 }  
 else if (opToken->kind == EQUALS\_EQUALS || opToken->kind == BANG\_EQUALS ||  
 opToken->kind == LESS\_THAN || opToken->kind == GREATER\_THAN ||  
 opToken->kind == LESS\_THAN\_EQUALS || opToken->kind == GREATER\_THAN\_EQUALS)  
 {  
 res = checkComparisonCompatibility(leftOp, rightOp, opToken, \_errorHandler);  
 }  
 else if (opToken->kind == PIPE\_PIPE || opToken->kind == AMPERSAND\_AMPERSAND)  
 {  
 res = checkLogicalCompatibility(leftOp, rightOp, opToken, \_errorHandler);  
 }  
 else if (opToken->kind == PIPE || opToken->kind == AMPERSAND || opToken->kind == CARET)  
 {  
 res = checkBitwiseCompatibility(leftOp, rightOp, opToken, \_errorHandler);  
 }  
   
 return res;  
 }  
   
 bool isWideningCompatible(baseType target, baseType source)  
 {  
 if (target == FLOAT)  
 return source == FLOAT || source == INT || source == CHAR;  
 if (target == INT)  
 return source == INT || source == CHAR;  
 if (target == CHAR)  
 return source == CHAR;  
 return false;  
 }  
   
 valType SemanticAnalyzer::checkCompatibilityAssignExp(valType leftOp, valType rightOp, SyntaxToken \*opToken)  
 {  
 bool errorOccurred = false;  
 valType res = leftOp;  
   
 if (isPointerToPointerAssignInvalid(leftOp, rightOp, opToken))  
 {  
 errorOccurred = true;  
 }  
 else if (isArrayToPointerAssignAllowed(leftOp, rightOp))  
 {  
 // valid ??? do nothing, res is already correct  
 }  
 else if (isArrayAssignmentInvalid(leftOp, rightOp, opToken))  
 {  
 errorOccurred = true;  
 }  
 else if (isPointerMismatch(leftOp, rightOp, opToken))  
 {  
 errorOccurred = true;  
 }  
 else if (!isWideningCompatible(leftOp.type, rightOp.type))  
 {  
 \_errorHandler->addError(new semanticError("incompatible types for assignment", opToken));  
 errorOccurred = true;  
 }  
   
 if (errorOccurred)  
 {  
 res = {UNDIFINED, 0, false, false};  
 }  
   
 return res;  
 }  
   
 bool SemanticAnalyzer::isPointerToPointerAssignInvalid(valType left, valType right, SyntaxToken \*token)  
 {  
 if (left.isPointer && right.isPointer && left.type != right.type)  
 {  
 \_errorHandler->addError(new semanticError("incompatible pointer types for assignment", token));  
 return true;  
 }  
 return false;  
 }  
   
 bool SemanticAnalyzer::isArrayToPointerAssignAllowed(valType left, valType right)  
 {  
 return left.isPointer && right.isArray && left.type == right.type;  
 }  
   
 bool SemanticAnalyzer::isArrayAssignmentInvalid(valType left, valType right, SyntaxToken \*token)  
 {  
 if (left.isArray && right.isArray)  
 {  
 if (left.size != right.size)  
 {  
 \_errorHandler->addError(new semanticError("array size mismatch", token));  
 return true;  
 }  
 if (!isWideningCompatible(left.type, right.type))  
 {  
 \_errorHandler->addError(new semanticError("incompatible array types for assignment", token));  
 return true;  
 }  
 return false; // valid array-to-array  
 }  
 if (left.isArray && !right.isArray)  
 {  
 \_errorHandler->addError(new semanticError("cannot assign non-array value to array", token));  
 return true;  
 }  
 if (!left.isArray && right.isArray)  
 {  
 \_errorHandler->addError(new semanticError("cannot assign array to non-array", token));  
 return true;  
 }  
 return false;  
 }  
   
 bool SemanticAnalyzer::isPointerMismatch(valType left, valType right, SyntaxToken \*token)  
 {  
 if (!left.isPointer && right.isPointer)  
 {  
 \_errorHandler->addError(new semanticError("cannot assign pointer to non-pointer", token));  
 return true;  
 }  
 if (left.isPointer && !right.isPointer)  
 {  
 \_errorHandler->addError(new semanticError("cannot assign non-pointer to pointer", token));  
 return true;  
 }  
 return false;  
 }  
   
 void SemanticAnalyzer::checkReturnStatements(functionEntry \*funcEntry)  
 {  
 // check if the function has a return statement  
 vector<NonTerminalNode \*> returnNodes = \_currFunctionReturnNodes;  
 valType returnType = funcEntry->getReturnType();  
   
 for (int i = 0; i < returnNodes.size(); i++)  
 {  
 NonTerminalNode \*returnNode = returnNodes[i];  
   
 valType nodeType = returnNode->GetValType();  
   
 if (nodeType.type != returnType.type || nodeType.isPointer != returnType.isPointer || nodeType.isArray != returnType.isArray)  
 {  
 \_errorHandler->addError(new semanticError("return type does not match function declaration"));  
 }  
 }  
   
 \_currFunctionReturnNodes.clear();  
 }  
   
 valType SemanticAnalyzer::getVarType(SyntaxToken \*IDToken)  
 {  
 // get the current scope  
 scope \*currScope = \_scopeStack.top();  
   
 // check if the variable already exists in the symbol table (in current or parent scope)  
 string name = IDToken->val;  
 tableEntry entry = currScope->getEntry(name);  
   
 valType res = {UNDIFINED, 0, false, false};  
   
 if (entry.name == "\_undeclared")  
 {  
 \_errorHandler->addError(new semanticError("variable not declared", IDToken));  
 }  
 else  
 {  
 res = entry.type;  
 }  
   
 return res;  
 }  
   
 // returns the functions return type and checks if the function call parameters match the function declaration parameters  
 // if the function is not declared it will add an error to the error handler  
 // if the function call parameters do not match the function declaration parameters it will add an error to the error handler  
 valType SemanticAnalyzer::getFunctionCallValTypeAndCheck(NonTerminalNode \*funcCallNode)  
 {  
 valType res = {UNDIFINED, 1, false, false};  
 // get the function name  
 TerminalNode \*nameNode = (TerminalNode \*)(funcCallNode->GetChildren()[0]);  
 string name = nameNode->getToken()->val;  
   
 // check if the function call parameters match the function declaration parameters  
 vector<ASTNode \*> children = funcCallNode->GetChildren();  
   
 NonTerminalNode \*exprList = (NonTerminalNode \*)(children[2]);  
 vector<ASTNode \*> paramListChildren = exprList->GetChildren();  
   
 vector<NonTerminalNode \*> paramNodes = getFunctionCallArgsNodes(exprList);  
 functionEntry \*entry = \_symbolTable->getFunction(name);  
   
 bool match = true;  
   
 if (entry == nullptr)  
 {  
 \_errorHandler->addError(new semanticError("function not declared", nameNode->getToken()));  
 }  
 else  
 {  
 res = entry->getReturnType();  
 vector<valType> paramTypes = entry->getParamTypes();  
 if (paramNodes.size() != paramTypes.size())  
 {  
 match = false;  
 }  
 else  
 {  
 for (int i = 0; i < paramNodes.size(); i++)  
 {  
 valType paramType = paramNodes[i]->GetValType();  
 valType expectedType = paramTypes[i];  
   
 if (paramType.type != expectedType.type || paramType.isPointer != expectedType.isPointer || paramType.isArray != expectedType.isArray)  
 {  
 match = false;  
 }  
 }  
 }  
 }  
   
 if (!match)  
 {  
 \_errorHandler->addError(new semanticError("function call parameters do not match", nameNode->getToken()));  
 }  
   
 return res;  
 }  
   
 void SemanticAnalyzer::initAssignActions()  
 {  
 \_nonTerminalAssignActions[PARAM] = &SemanticAnalyzer::assignParamNodeType;  
 \_nonTerminalAssignActions[TYPE] = &SemanticAnalyzer::assignTypeNodeType;  
 \_nonTerminalAssignActions[BASE\_TYPE] = &SemanticAnalyzer::assignBaseTypeNodeType;  
 \_nonTerminalAssignActions[VAR\_DECL\_EXPR] = &SemanticAnalyzer::assignVarDeclExprNodeType;  
 \_nonTerminalAssignActions[ASSIGN\_VALUE] = &SemanticAnalyzer::assignAssignValueNodeType;  
 \_nonTerminalAssignActions[ASSIGN\_EXPR] = &SemanticAnalyzer::assignAssignExprNodeType;  
 \_nonTerminalAssignActions[ASSIGN\_TARGET] = &SemanticAnalyzer::assignAssignTargetNodeType;  
 \_nonTerminalAssignActions[CONDITION\_OP] = &SemanticAnalyzer::assignConditionOpNodeType;  
 \_nonTerminalAssignActions[SIMPLE\_STMT] = &SemanticAnalyzer::assignSimpleStmtNodeType;  
 \_nonTerminalAssignActions[EXPR\_LIST] = &SemanticAnalyzer::assignExprListNodeType;  
 \_nonTerminalAssignActions[EXPR\_LIST\_NON\_EMPTY] = &SemanticAnalyzer::assignExprListNonEmptyNodeType;  
 \_nonTerminalAssignActions[EXPR] = &SemanticAnalyzer::assignExprNodeType;  
 \_nonTerminalAssignActions[LOGICAL\_EXPR] = &SemanticAnalyzer::assignLogicalExprNodeType;  
 \_nonTerminalAssignActions[RELATIONAL\_EXPR] = &SemanticAnalyzer::assignRelationalExprNodeType;  
 \_nonTerminalAssignActions[ADD\_EXPR] = &SemanticAnalyzer::assignAddExprNodeType;  
 \_nonTerminalAssignActions[MUL\_EXPR] = &SemanticAnalyzer::assignMulExprNodeType;  
 \_nonTerminalAssignActions[UNARY\_EXPR] = &SemanticAnalyzer::assignUnaryExprNodeType;  
 \_nonTerminalAssignActions[INCREMENT\_EXPR] = &SemanticAnalyzer::assignIncrementExprNodeType;  
 \_nonTerminalAssignActions[ADDRESS\_EXPR] = &SemanticAnalyzer::assignAddressExprNodeType;  
 \_nonTerminalAssignActions[DEREFERENCE\_EXPR] = &SemanticAnalyzer::assignDereferenceExprNodeType;  
 \_nonTerminalAssignActions[PRIMARY\_EXPR] = &SemanticAnalyzer::assignPrimaryExprNodeType;  
 }  
   
 void SemanticAnalyzer::assignParamNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignTypeNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 valType baseType = children[0]->GetValType();  
 if (children.size() == 1)  
 {  
 resType = baseType;  
 }  
 else if (children.size() == 2)  
 {  
 resType = {baseType.type, baseType.size, true, false};  
 }  
 else if (children.size() == 4)  
 {  
 TerminalNode \*sizeNode = ((TerminalNode \*)(children[2]));  
 int size = atoi(sizeNode->getToken()->val.c\_str());  
   
 resType = {baseType.type, size, false, true};  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignBaseTypeNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignVarDeclExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 TerminalNode \*varNode = (TerminalNode \*)(ntNode->GetChildren()[1]);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 valType leftOpVarType = children[0]->GetValType();  
   
 NonTerminalNode \*initOptNode = (NonTerminalNode \*)(children[2]);  
 vector<ASTNode \*> initOptChildren = initOptNode->GetChildren();  
   
 if (initOptChildren.size() > 0)  
 {  
 TerminalNode \*assignOpNode = (TerminalNode \*)(initOptChildren[0]);  
 SyntaxToken \*opToken = assignOpNode->getToken();  
   
 NonTerminalNode \*assignValueNode = (NonTerminalNode \*)(initOptChildren[1]);  
 valType rightOpVarType = assignValueNode->GetValType();  
   
 resType = checkCompatibilityAssignExp(leftOpVarType, rightOpVarType, assignOpNode->getToken());  
 node->SetValType(resType);  
 }  
 }  
   
 void SemanticAnalyzer::assignAssignValueNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
   
 else if (children.size() == 3)  
 {  
 valType type = children[1]->GetValType();  
 resType = {type.type, type.size, type.isPointer, true};  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignAssignExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() >= 3)  
 {  
 valType leftOp = children[0]->GetValType();  
 valType rightOp = children[2]->GetValType();  
   
 NonTerminalNode \*assignOpNode = (NonTerminalNode \*)(children[1]);  
 SyntaxToken \*opToken = ((TerminalNode \*)(assignOpNode->GetChildren()[0]))->getToken();  
   
 resType = checkCompatibilityAssignExp(leftOp, rightOp, opToken);  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignAssignTargetNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 TerminalNode \*varNode = (TerminalNode \*)(children[0]);  
 resType = getVarType(varNode->getToken());  
 }  
 else if (children.size() == 4)  
 {  
 valType arrType = children[0]->GetValType();  
 if (!arrType.isArray)  
 {  
 \_errorHandler->addError(new semanticError("indexing non-array type", ((TerminalNode \*)(children[1]))->getToken()));  
 }  
 else  
 {  
 resType = {arrType.type, arrType.size, arrType.isPointer, false};  
 }  
 }  
 else if (children.size() == 2)  
 {  
 valType ptrType = \_scopeStack.top()->getEntry(((TerminalNode \*)(children[1]))->getToken()->val).type;  
 if (!ptrType.isPointer)  
 {  
 \_errorHandler->addError(new semanticError("dereferencing non-pointer type", ((TerminalNode \*)(children[0]))->getToken()));  
 }  
 else  
 {  
 resType = {ptrType.type, ptrType.size, false, ptrType.isArray};  
 }  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignConditionOpNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 valType condType = children[0]->GetValType();  
 if (condType.type != INT && !condType.isPointer)  
 {  
 \_errorHandler->addError(new semanticError("condition must be int or pointer", ((TerminalNode \*)(children[0]))->getToken()));  
 }  
 else  
 {  
 resType = condType;  
 }  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignSimpleStmtNodeType(ASTNode \*node)  
 {  
 if (isReturnStatement(node))  
 {  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)node;  
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 NonTerminalNode \*ExptOpt = (NonTerminalNode \*)(children[1]);  
 vector<ASTNode \*> ExptOptChildren = ExptOpt->GetChildren();  
 if (ExptOptChildren.size() > 0)  
 {  
 valType returnType = ExptOptChildren[0]->GetValType();  
 node->SetValType(returnType);  
 }  
 }  
   
 if(isFuncCall(node))  
 {  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)node;  
   
 valType funcCallType = getFunctionCallValTypeAndCheck(ntNode);  
 node->SetValType(funcCallType);  
 }  
 }  
   
 void SemanticAnalyzer::assignExprListNodeType(ASTNode \*node)  
 {  
 vector<ASTNode \*> children = ((NonTerminalNode \*)(node))->GetChildren();  
 valType resType = {UNDIFINED, 0, false, false};  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignExprListNonEmptyNodeType(ASTNode \*node)  
 {  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 valType resType = children[0]->GetValType();  
 resType.isArray = false;  
 resType.isPointer = false;  
   
 if (children.size() == 3)  
 {  
 int prevSize = resType.size;  
   
 valType rightOp = children[2]->GetValType();  
 rightOp.isArray = false;  
 rightOp.isPointer = false;  
   
 SyntaxToken \*opToken = ((TerminalNode \*)(children[1]))->getToken();  
 resType = checkCompatibilityBinaryOp(resType, rightOp, opToken);  
   
 resType = valType{resType.type, prevSize + 1, resType.isPointer, resType.isArray};  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignLogicalExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
 else if (children.size() == 3)  
 {  
 valType leftOp = children[0]->GetValType();  
 valType rightOp = children[2]->GetValType();  
 SyntaxToken \*opToken = ((TerminalNode \*)(children[1]))->getToken();  
 resType = checkCompatibilityBinaryOp(leftOp, rightOp, opToken);  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignRelationalExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
 else if (children.size() == 3)  
 {  
 valType leftOp = children[0]->GetValType();  
 valType rightOp = children[2]->GetValType();  
 SyntaxToken \*opToken = ((TerminalNode \*)(children[1]))->getToken();  
 resType = checkCompatibilityBinaryOp(leftOp, rightOp, opToken);  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignAddExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
 else if (children.size() == 3)  
 {  
 valType leftOp = children[0]->GetValType();  
 valType rightOp = children[2]->GetValType();  
 SyntaxToken \*opToken = ((TerminalNode \*)(children[1]))->getToken();  
 resType = checkCompatibilityBinaryOp(leftOp, rightOp, opToken);  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignMulExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
 else if (children.size() == 3)  
 {  
 valType leftOp = children[0]->GetValType();  
 valType rightOp = children[2]->GetValType();  
 SyntaxToken \*opToken = ((TerminalNode \*)(children[1]))->getToken();  
 resType = checkCompatibilityBinaryOp(leftOp, rightOp, opToken);  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignUnaryExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 1)  
 {  
 resType = children[0]->GetValType();  
 }  
 else if (children.size() == 2)  
 {  
 valType operandType = children[1]->GetValType();  
 SyntaxToken \*opToken = ((TerminalNode \*)(children[0]))->getToken();  
   
 if (operandType.isPointer || operandType.isArray)  
 {  
 \_errorHandler->addError(new semanticError("unary operator not allowed on pointer/array", opToken));  
 }  
 else  
 {  
 resType = operandType;  
 }  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignIncrementExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
   
 TerminalNode \*firstChild = (TerminalNode \*)(children[0]);  
 TerminalNode \*secondChild = (TerminalNode \*)(children[1]);  
   
 TerminalNode \*varNode = firstChild->getToken()->kind == IDENTIFIER ? firstChild : secondChild;  
 valType operandType = getVarType(varNode->getToken());  
   
 if (operandType.type != INT && !operandType.isPointer)  
 {  
 \_errorHandler->addError(new semanticError("increment/decrement requires int or pointer", varNode->getToken()));  
 }  
 else  
 {  
 resType = operandType;  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignAddressExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 2)  
 {  
 valType varType = getVarType(((TerminalNode \*)(children[1]))->getToken());  
   
 if (varType.isPointer || varType.isArray)  
 {  
 \_errorHandler->addError(new semanticError("double pointer abstraction is not allowed!", ((TerminalNode \*)(children[0]))->getToken()));  
 }  
   
 resType = {varType.type, varType.size, true, varType.isArray};  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignDereferenceExprNodeType(ASTNode \*node)  
 {  
 valType resType = {UNDIFINED, 0, false, false};  
 NonTerminalNode \*ntNode = (NonTerminalNode \*)(node);  
   
 vector<ASTNode \*> children = ntNode->GetChildren();  
 if (children.size() == 2)  
 {  
 valType varType = \_scopeStack.top()->getEntry(((TerminalNode \*)(children[1]))->getToken()->val).type;  
 if (!varType.isPointer)  
 {  
 \_errorHandler->addError(new semanticError("dereferencing non-pointer type", ((TerminalNode \*)(children[0]))->getToken()));  
 }  
 else  
 {  
 resType = {varType.type, varType.size, false, varType.isArray};  
 }  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::assignPrimaryExprNodeType(ASTNode \*node)  
 {  
 valType resType{UNDIFINED, 0, false, false};  
 auto \*ntNode = static\_cast<NonTerminalNode \*>(node);  
 auto children = ntNode->GetChildren();  
   
 if (children.size() == 1)  
 {  
 ASTNode \*child = children[0];  
 if (child->GetType() == NON\_TERMINAL)  
 {  
 // Propagate result of DereferenceExpr or IncrementExpr  
 resType = child->GetValType();  
 }  
 else  
 {  
 TerminalNode \*tNode = (TerminalNode \*)(child);  
 if (tNode->getToken()->kind == IDENTIFIER)  
 resType = getVarType(tNode->getToken());  
 else  
 resType = {assignTerminal[tNode->getToken()->kind], 0, false, false};  
 }  
 }  
 else if (children.size() == 3)  
 {  
 // Parenthesized: '(' Expr ')'  
 resType = children[1]->GetValType();  
 }  
 else if (isArrDeref(node))  
 {  
 // Array indexing  
 string varName = static\_cast<TerminalNode \*>(children[0])->getToken()->val;  
 valType type = \_scopeStack.top()->getEntry(varName).type;  
 if (!type.isArray && !type.isPointer)  
 \_errorHandler->addError(new semanticError("indexing non-array type", static\_cast<TerminalNode \*>(children[1])->getToken()));  
 else  
 resType = {type.type, type.size, false, false};  
 }  
 else if (isFuncCall(node))  
 {  
 // Function call  
 resType = getFunctionCallValTypeAndCheck(ntNode);  
 }  
   
 node->SetValType(resType);  
 }  
   
 void SemanticAnalyzer::checkForMainFunction()  
 {  
 // check if the main function is declared  
 functionEntry \*mainFunc = \_symbolTable->getFunction("main");  
 if (mainFunc == nullptr)  
 {  
 \_errorHandler->addError(new semanticError("main function not declared"));  
 }  
 else if (mainFunc->getReturnType().type != INT)  
 {  
 cout << "main function return type: " << mainFunc->getReturnType().type << endl;  
 // check if the main function has a return type of int  
 \_errorHandler->addError(new semanticError("main function must return int"));  
 }  
 }

## semantic.hpp

#ifndef \_\_SEMANTIC\_ANALYZER  
 #define \_\_SEMANTIC\_ANALYZER  
   
 #include "../nodes/nodes.hpp"  
 #include "../nodes/nodeAnalyzer/nodeAnalyzer.hpp"  
 #include "../symbolTable/scope/scope.hpp"  
 #include "../symbolTable/tableEntry/tableEntry.hpp"  
 #include "../symbolTable/symbolTable.hpp"  
 #include "../errorHandler/errorHandler.hpp"  
 #include "../errors/errors.hpp"  
 #include <stack>  
   
 extern map<SyntaxKind, baseType> assignTerminal;  
   
 class SemanticAnalyzer  
 {  
 private:  
 ErrorHandler \*\_errorHandler;  
 SymbolTable \*\_symbolTable;  
 std::stack<scope \*> \_scopeStack;  
 scope \*\_currRootScope;  
 std::vector<NonTerminalNode \*> \_currFunctionParamNodes;  
 std::vector<NonTerminalNode \*> \_currFunctionReturnNodes;  
 std::map<NonTerminal, void (SemanticAnalyzer::\*)(ASTNode \*)> \_nonTerminalAssignActions;  
   
 // Refactored helper functions  
 bool isPointerToPointerAssignInvalid(valType left, valType right, SyntaxToken \*token);  
 bool isArrayToPointerAssignAllowed(valType left, valType right);  
 bool isArrayAssignmentInvalid(valType left, valType right, SyntaxToken \*token);  
 bool isPointerMismatch(valType left, valType right, SyntaxToken \*token);  
   
 // Assignment type inference helpers  
 void initAssignActions();  
 valType getVarType(SyntaxToken \*IDToken);  
 valType getFunctionCallValTypeAndCheck(NonTerminalNode \*funcCallNode);  
   
   
 // Type assignment  
 void assignParamNodeType(ASTNode \*node);  
 void assignTypeNodeType(ASTNode \*node);  
 void assignBaseTypeNodeType(ASTNode \*node);  
 void assignVarDeclExprNodeType(ASTNode \*node);  
 void assignAssignValueNodeType(ASTNode \*node);  
 void assignAssignExprNodeType(ASTNode \*node);  
 void assignAssignTargetNodeType(ASTNode \*node);  
 void assignConditionOpNodeType(ASTNode \*node);  
 void assignSimpleStmtNodeType(ASTNode \*node);  
 void assignExprListNodeType(ASTNode \*node);  
 void assignExprListNonEmptyNodeType(ASTNode \*node);  
 void assignExprNodeType(ASTNode \*node);  
 void assignLogicalExprNodeType(ASTNode \*node);  
 void assignRelationalExprNodeType(ASTNode \*node);  
 void assignAddExprNodeType(ASTNode \*node);  
 void assignMulExprNodeType(ASTNode \*node);  
 void assignUnaryExprNodeType(ASTNode \*node);  
 void assignIncrementExprNodeType(ASTNode \*node);  
 void assignAddressExprNodeType(ASTNode \*node);  
 void assignDereferenceExprNodeType(ASTNode \*node);  
 void assignPrimaryExprNodeType(ASTNode \*node);  
   
 public:  
 SemanticAnalyzer(ErrorHandler \*errorHandler, SymbolTable \*symbolTable);  
   
 void updateSybolTable(ASTNode \*node);  
 void updateScope(SyntaxToken \*currToken);  
 void addFunctionNodeToSymbolTable(NonTerminalNode \*funcDeclNode);  
 void addVariableNodeToSymbolTable(NonTerminalNode \*varDeclNode);  
 void addParamNodeToSymbolTable(NonTerminalNode \*paramNode);  
 void addParamListToSymbolTable(std::vector<NonTerminalNode \*> paramNodes);  
 void assignNodeType(ASTNode \*node);  
   
 valType checkCompatibilityBinaryOp(valType leftOp, valType rightOp, SyntaxToken \*opToken);  
 valType checkCompatibilityAssignExp(valType leftOp, valType rightOp, SyntaxToken \*opToken);  
   
 void checkReturnStatements(functionEntry \*funcEntry);  
 void checkForMainFunction();  
 };  
   
 #endif // SEMANTIC\_HPP

## functionEntry.cpp

#include "functionEntry.hpp"  
 #include <iostream>  
 #include <string>  
 #include <vector>  
   
 void functionEntry::print() const  
 {  
 cout << "Function Name: " << \_name << endl;  
 cout << "Return Type: " << valTypeToString(\_returnType) << endl;  
 cout << "Parameter Types: " << endl;  
 cout << "num of parameters: " << \_paramTypes.size() << endl;  
 for (const auto &paramType : \_paramTypes)  
 {  
 cout << valTypeToString(paramType) << " ";  
 }  
   
 if(\_scopeRoot) \_scopeRoot->printScope();  
   
 cout << "End of Function" << endl;  
 cout << "----------------------------------------" << endl;  
 cout << endl;  
 }

## functionEntry.hpp

#ifndef \_\_FUNCTION\_ENTRY  
 #define \_\_FUNCTION\_ENTRY  
   
 #include "../../token/token.hpp"  
 #include "../scope/scope.hpp"  
 #include <String>  
 #include <vector>  
   
 class functionEntry  
 {  
 private:  
 string \_name;  
 valType \_returnType;  
 vector<valType> \_paramTypes;  
   
 scope \*\_scopeRoot;  
   
 public:  
 functionEntry(string name, valType returnType, vector<valType> paramTypes) : \_name(name), \_returnType(returnType), \_paramTypes(paramTypes)   
 {\_scopeRoot = nullptr;}  
   
 string getName() const { return \_name; }  
 valType getReturnType() const { return \_returnType; }  
 vector<valType> getParamTypes() const { return \_paramTypes; }  
   
 scope \*getInnerScope() const { return \_scopeRoot; }  
   
 void setInnerScope(scope \*innerScope) { \_scopeRoot = innerScope; }  
 void print() const;  
 };  
   
 #endif

## scope.cpp

## scope.hpp

#ifndef SCOPE  
 #define SCOPE  
   
 #include "..\tableEntry\tableEntry.hpp"  
   
 #include <string>  
 #include <vector>  
 #include <iostream>  
   
 using namespace std;  
   
 class scope  
 {  
 private:  
 vector<tableEntry> \_tableEntries;  
 vector<scope \*> \_innerScopes;  
 scope \*\_parentScope = nullptr;  
   
 public:  
 scope() {}  
   
 void addTableEntry(tableEntry entry)  
 {  
 \_tableEntries.push\_back(entry);  
 }  
   
 void addInnerScope(scope \*innerScope) { \_innerScopes.push\_back(innerScope); }  
 void setParentScope(scope \*parentScope) { \_parentScope = parentScope; }  
 scope \*getParentScope() const { return \_parentScope; }  
   
 vector<tableEntry> &getEntries() { return \_tableEntries; }  
   
 // returns the entery with the requested name  
 // if the name is not found in this scope, it will search in the inner scopes  
 tableEntry getEntry(string name)  
 {  
 tableEntry res = {"\_undeclared", {}};  
   
 for (tableEntry entry : \_tableEntries)  
 {  
 if (entry.name == name)  
 {  
 res = entry;  
 }  
 }  
   
 if (res.name == "\_undeclared" && \_parentScope != nullptr)  
 {  
 res = \_parentScope->getEntry(name);  
 }  
   
 return res;  
 }  
 vector<scope \*> getInnerScopes() const { return \_innerScopes; }  
   
 // returns the entery with the requested name  
 tableEntry getInnerScopeEntrie(string name)  
 {  
 tableEntry res = {"\_undeclared", {}};  
 for (tableEntry entry : \_tableEntries)  
 {  
 if (entry.name == name)  
 {  
 res = entry;  
 }  
 }  
 return res;  
 }  
   
 void printScope()  
 {  
   
 for (auto &entry : \_tableEntries)  
 {  
 // Make sure printTableEntery handles nullptrs or malformed data  
 printTableEntery(&entry);  
 }  
   
 for (auto \*innerScope : \_innerScopes)  
 {  
 if (innerScope != nullptr) // <-- this is the fix  
 {  
 innerScope->printScope();  
 }  
 else  
 {  
 cout << "Warning: nullptr inner scope found!" << endl;  
 }  
 }  
   
 }  
 };  
   
 #endif

## symbolTable.cpp

#include "actionTable.hpp"  
 #include "tableEntry/tableEntry.hpp"  
 #include "functionEntry/functionEntry.hpp"  
 #include "symbolTable.hpp"  
 #include "../parser/grammerSymbol/grammerSymbol.hpp"  
 #include "../semantic/semantic.hpp"  
 #include "../token/token.hpp"  
 #include "../nodes/nodes.hpp"  
 #include <map>  
 #include <vector>  
 #include <String>  
 #include <algorithm>  
   
 // returns the entry with the requested name  
 // if function not found returns nullptr  
 functionEntry \*SymbolTable::getFunction(string name)  
 {  
 functionEntry \*res = nullptr;  
 vector<functionEntry \*> enteries = getFunctions();  
   
 for (functionEntry \*entry : enteries)  
 {  
 if (entry->getName() == name)  
 {  
 res = entry;  
 }  
 }  
   
 return res;  
 }  
   
 valType createVarDeclExprType(NonTerminalNode \*typeNode)  
 {  
 vector<ASTNode \*> children = typeNode->GetChildren();  
   
 NonTerminalNode \*baseTypeNode = (NonTerminalNode \*)children[0];  
 SyntaxKind type = ((TerminalNode \*)baseTypeNode->GetChildren()[0])->getToken()->kind;  
   
 int size = 1;  
 bool isPointer = false;  
 bool isArray = false;  
   
 // pointer  
 if (children.size() == 2)  
 {  
 isPointer = true;  
 }  
   
 // array  
 if (children.size() == 4)  
 {  
 isArray = true;  
   
 TerminalNode \*sizeNode = (TerminalNode \*)children[2];  
 size = stoi(sizeNode->getToken()->val);  
 }  
   
 valType res = {assignTerminal[type], size, isPointer, isArray};  
 return res;  
 }  
   
 tableEntry createTableEntery\_varDec(NonTerminalNode \*varDecNode)  
 {  
 tableEntry res = tableEntry{"", valType{UNDIFINED, 1, false, false}, false};  
 vector<ASTNode \*> children = varDecNode->GetChildren();  
   
 // set name  
 res.name = ((TerminalNode \*)children[1])->getToken()->val;  
   
 // set type  
 res.type = createVarDeclExprType((NonTerminalNode \*)children[0]);  
   
 return res;  
 }  
   
 tableEntry createTableEntery\_param(NonTerminalNode \*paramNode)  
 {  
 tableEntry res = tableEntry{"", valType{UNDIFINED, 1, false, false}, false};  
 vector<ASTNode \*> children = paramNode->GetChildren();  
   
 // set name  
 res.name = ((TerminalNode \*)children[1])->getToken()->val;  
   
 // set type  
 res.type = createVarDeclExprType((NonTerminalNode \*)children[0]);  
   
 return res;  
 }  
   
 void createFunctionParamTypesHelper(NonTerminalNode \*paramListNonEmptyNode, vector<valType> \*paramTypes)  
 {  
 vector<ASTNode \*> childern = paramListNonEmptyNode->GetChildren();  
 int numOfChildren = childern.size();  
 NonTerminalNode \*typeNode;  
 if (numOfChildren == 1)  
 {  
 typeNode = ((NonTerminalNode \*)(childern[0]->GetChildren()[0]));  
   
 paramTypes->push\_back(createVarDeclExprType(typeNode));  
 }  
   
 if (numOfChildren == 3)  
 {  
 typeNode = ((NonTerminalNode \*)(childern[2]->GetChildren()[0]));  
   
 paramTypes->push\_back(createVarDeclExprType(typeNode));  
   
 createFunctionParamTypesHelper((NonTerminalNode \*)childern[0], paramTypes);  
 }  
 }  
   
 vector<valType> createFunctionParamTypes(NonTerminalNode \*paramListNode)  
 {  
 vector<ASTNode \*> childern = paramListNode->GetChildren();  
 vector<valType> res;  
   
 if (childern.size() > 0)  
 {  
 createFunctionParamTypesHelper((NonTerminalNode \*)childern[0], &res);  
 }  
   
 std::reverse(res.begin(), res.end());  
 return res;  
 }  
   
 void createFunctionParamEnteriesHelper(NonTerminalNode \*paramListNonEmptyNode, vector<tableEntry> \*enteries)  
 {  
 vector<ASTNode \*> childern = paramListNonEmptyNode->GetChildren();  
 int numOfChildren = childern.size();  
 NonTerminalNode \*paramNode;  
 if (numOfChildren == 1)  
 {  
 paramNode = ((NonTerminalNode \*)(childern[0]));  
 enteries->push\_back(createTableEntery\_param(paramNode));  
 }  
   
 if (numOfChildren == 3)  
 {  
 paramNode = ((NonTerminalNode \*)(childern[2]));  
 enteries->push\_back(createTableEntery\_param(paramNode));  
 createFunctionParamEnteriesHelper((NonTerminalNode \*)childern[0], enteries);  
 }  
 }  
   
 vector<tableEntry> createFunctionParamEnteries(NonTerminalNode \*paramListNode)  
 {  
 vector<ASTNode \*> childern = paramListNode->GetChildren();  
 vector<tableEntry> res;  
   
 if (childern.size() > 0)  
 {  
 createFunctionParamEnteriesHelper((NonTerminalNode \*)childern[0], &res);  
 }  
   
 std::reverse(res.begin(), res.end());  
 return res;  
 }

## symbolTable.hpp

#ifndef \_\_SYMBOL\_TABLE  
 #define \_\_SYMBOL\_TABLE  
   
 #include "tableEntry/tableEntry.hpp"  
 #include "functionEntry/functionEntry.hpp"  
 #include "../parser/grammerSymbol/grammerSymbol.hpp"  
 #include "../token/token.hpp"  
 #include "../nodes/nodes.hpp"  
 #include <map>  
 #include <vector>  
 #include <String>  
   
 using namespace std;  
   
 class SymbolTable  
 {  
 private:  
 vector<functionEntry \*> \_functions;  
   
 public:  
 SymbolTable() {   
 addFunction(new functionEntry("printInt", {INT}, {valType{INT}}));   
 addFunction(new functionEntry("printFloat", {INT}, {valType{FLOAT}}));  
 }  
   
 void addFunction(functionEntry \*function) { \_functions.push\_back(function); }  
 functionEntry \*getFunction(string name);  
 const std::vector<functionEntry \*> &getFunctions() const { return \_functions; }  
   
 void print() const  
 {  
 cout << "Symbol Table:" << endl;  
 cout << "num of functions: " << \_functions.size() << endl;  
 for (const auto &function : \_functions)  
 {  
 function->print();  
 }  
 cout << "End of Symbol Table" << endl;  
 }  
 };  
   
 valType createVarDeclExprType(NonTerminalNode \*varNode);  
 tableEntry createTableEntery\_varDec(NonTerminalNode \*varDecNode);  
 tableEntry createTableEntery\_param(NonTerminalNode \*varDecNode);  
 vector<valType> createFunctionParamTypes(NonTerminalNode \*paramListNode);  
 void createFunctionParamTypesHelper(NonTerminalNode \*paramListNode, vector<valType> \*paramTypes);  
 vector<tableEntry> createFunctionParamEnteries(NonTerminalNode \*paramListNode);  
 void createFunctionParamEnteriesHelper(NonTerminalNode \*paramListNode, vector<tableEntry> \*paramTypes);  
   
 #endif

## tableEntry.cpp

#include "tableEntry.hpp"  
 #include <iostream>  
 #include <string>  
 #include <sstream>  
   
 string valTypeToString(valType vType)  
 {  
 stringstream res;  
   
 res << "\e[0;35m"  
 << baseTypeToString(vType.type)  
 << "\033[0m";  
 if (vType.isPointer)  
 {  
 res << "\*";  
 }  
 if (vType.isArray)  
 {  
 res << "["  
 << vType.size  
 << "]";  
 }  
   
 return res.str();  
 }  
   
 string baseTypeToString(baseType bType)  
 {  
 switch (bType)  
 {  
 case INT:  
 return "int";  
 case FLOAT:  
 return "float";  
 case CHAR:  
 return "char";  
 case UNDIFINED:  
 return "undefined";  
 default:  
 return "unknown type";  
 }  
 }  
   
 void printValType(valType vtype)  
 {  
 cout << valTypeToString(vtype) << endl;  
 }  
   
 void printTableEntery(tableEntry \*entry)  
 {  
 cout << "---------------------------------" << endl;  
 cout << "Variable Name: " << entry->name << endl;  
 cout << "Variable Type: " << valTypeToString(entry->type) << endl;  
 cout << "---------------------------------" << endl;  
 }

## tableEntry.hpp

#ifndef \_\_TABLE\_ENTERY  
 #define \_\_TABLE\_ENTERY  
   
 #include "../../token/token.hpp"  
   
 using namespace std;  
   
 enum baseType  
 {  
 INT,  
 FLOAT,  
 CHAR,  
 UNDIFINED,  
 };  
   
 struct valType  
 {  
 baseType type = UNDIFINED;  
 int size = 1;  
 bool isPointer = false;  
 bool isArray = false;  
 };  
   
 struct tableEntry  
 {  
 string name;  
 valType type;  
 bool isInitialized = false;  
 int offset = 0; // offset in the stack  
 string addr = "";  
 };  
   
 string valTypeToString(valType vType);  
 string baseTypeToString(baseType bType);  
 void printValType(valType vtype);  
 void printTableEntery(tableEntry \*entry);  
   
 #endif

## token.cpp

#include "token.hpp"  
 #include <sstream>  
 #include <iostream>  
   
 using namespace std;  
   
 string syntaxKindEnumToString[] = {  
 "INTEGER\_LITERAL", "FLOAT\_LITERAL", "CHAR\_LITERAL", "STRING\_LITERAL", "IDENTIFIER", "KEYWORD\_IF", "KEYWORD\_ELSE", "KEYWORD\_WHILE", "KEYWORD\_FOR",  
 "KEYWORD\_FN", "KEYWORD\_RET", "KEYWORD\_INT", "KEYWORD\_FLOAT", "KEYWORD\_CHAR", "EQUALS", "COMMA", "PLUS",   
 "MINUS", "SLASH", "STAR", "AMPERSAND","PIPE", "CARET", "TILDE",   
 "BANG", "SEMICOLON", "LESS\_THAN", "GREATER\_THAN", "OPEN\_PAREN", "CLOSED\_PAREN", "OPEN\_CURLY", "CLOSED\_CURLY", "OPEN\_BRACKET",  
 "CLOSED\_BRACKET", "RIGHT\_ARROW", "PLUS\_PLUS", "MINUS\_MINUS", "PLUS\_EQUALS", "MINUS\_EQUALS", "SLASH\_EQUALS", "STAR\_EQUALS",  
 "AMPERSAND\_EQUALS","PIPE\_EQUALS", "CARET\_EQUALS", "TILDE\_EQUALS", "EQUALS\_EQUALS",  
 "LESS\_THAN\_EQUALS", "GREATER\_THAN\_EQUALS", "AMPERSAND\_AMPERSAND", "PIPE\_PIPE", "BANG\_EQUALS", "END\_OF\_FILE", "UNEXPECTED\_TOKEN"  
 };  
   
   
 string syntaxTokenToString(SyntaxToken token)  
 {  
 stringstream res;  
 string tokenValue = token.val;  
 res << syntaxKindToString(token.kind)  
 << " ";  
   
 if (!tokenValue.empty())  
 {  
 res << "| " << tokenValue;  
 }  
   
 return res.str();  
 }  
   
 string syntaxKindToString(SyntaxKind kind)  
 {  
 stringstream res;  
 res << "\e[0;36m"  
 << syntaxKindEnumToString[kind]  
 << "\e[0m";  
   
 return res.str();  
 }  
   
 void printSyntaxToken(SyntaxToken token)  
 {  
 cout << syntaxTokenToString(token) << endl;  
 }

## token.hpp

#ifndef \_\_TOKEN  
 #define \_\_TOKEN  
   
 #include <string>  
   
 using namespace std;  
   
 enum SyntaxKind  
 {  
 INTEGER\_LITERAL,  
 FLOAT\_LITERAL,  
 CHAR\_LITERAL,  
 STRING\_LITERAL,  
 IDENTIFIER,  
 KEYWORD\_IF,  
 KEYWORD\_ELSE,  
 KEYWORD\_WHILE,  
 KEYWORD\_FOR,  
 KEYWORD\_FN,  
 KEYWORD\_RET,  
 KEYWORD\_INT,  
 KEYWORD\_FLOAT,  
 KEYWORD\_CHAR,  
 EQUALS,  
 COMMA,  
 PLUS,  
 MINUS,  
 SLASH,  
 STAR,  
 AMPERSAND,  
 PIPE,  
 CARET,  
 TILDE,  
 BANG,  
 SEMICOLON,  
 LESS\_THAN,  
 GREATER\_THAN,  
 OPEN\_PAREN,  
 CLOSED\_PAREN,  
 OPEN\_CURLY,  
 CLOSED\_CURLY,  
 OPEN\_BRACKET,  
 CLOSED\_BRACKET,  
 RIGHT\_ARROW,  
 PLUS\_PLUS,  
 MINUS\_MINUS,  
 PLUS\_EQUALS,  
 MINUS\_EQUALS,  
 SLASH\_EQUALS,  
 STAR\_EQUALS,  
 AMPERSAND\_EQUALS,  
 PIPE\_EQUALS,  
 CARET\_EQUALS,  
 TILDE\_EQUALS,  
 EQUALS\_EQUALS,  
 LESS\_THAN\_EQUALS,  
 GREATER\_THAN\_EQUALS,  
 AMPERSAND\_AMPERSAND,  
 PIPE\_PIPE,  
 BANG\_EQUALS,  
 END\_OF\_FILE,  
 UNEXPECTED\_TOKEN,  
 EPSILON,  
 SYNTAX\_KIND\_COUNT  
 };  
   
 struct SyntaxToken  
 {  
 SyntaxKind kind;  
 string val;  
 int line;  
 int column;  
 };  
   
   
 string syntaxTokenToString(SyntaxToken token);  
 string syntaxKindToString(SyntaxKind kind);  
 void printSyntaxToken(SyntaxToken token);  
   
 #endif