עבודת גמר

לקבלת תואר טכנאי תוכנה

הנושא: **קומפיילר**

מגיש : איתן רפאל צ'רטוף

ת.ז. המגיש: 215310715

מנחה : מיכאל צ'רנובילסקי

אפריל 2024 תשפ"ד

מבוא

השראה

לפני שלמדתי לתכנת אני חשבתי על תכנות כמשהו מיסטי. לא האמתי שקיימת מכונה מכנית המקבלת הוראות ומבצעת אותם! זה הרגיש לי כמו מכונת קסם. כמובן שזה היה מלפני שלמדתי לתכנת (ושהייתי בגיל 9) ועכשיו שאני יודע איך לתכנת, הנושא הזה לא מרגיש כלכך מיסטי. למרות זאת, אני עדיין לומד כל יום משהו חדש בתכנות ומקווה להמשיך עוד.

הבחירה להכין קומפיילר לפרויקט נובעת מהמשימה שלי להבין איך המכונה המכנית הזאת שנקראת מחשב עובדת. בנוסף, תמיד אהבתי לאתגר את עצמי כשזה מגיע לפרויקטים. משל הסיבות האלה החלטתי להכין קומפיילר.

"There are two kinds of programmers — those who have written compilers and those who haven't."
-Terry A. Davis

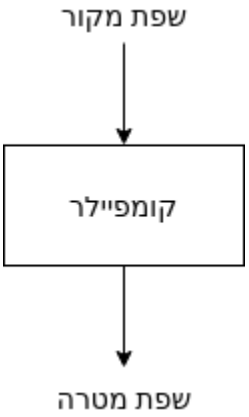
מטרה

במבט אישי המטרה שלי היא להבין איך שפת תכנות מתורגמת לשפת מכונה, לפיכך, איך קומפיילרים עובדים.

במבט אקדמי מטרת הפרויקט היא להכין קומפיילר המתרגם שפה שאני אכין, לשפת סף. שפת הסף תהיה - nasm 64 ביט.

תקציר

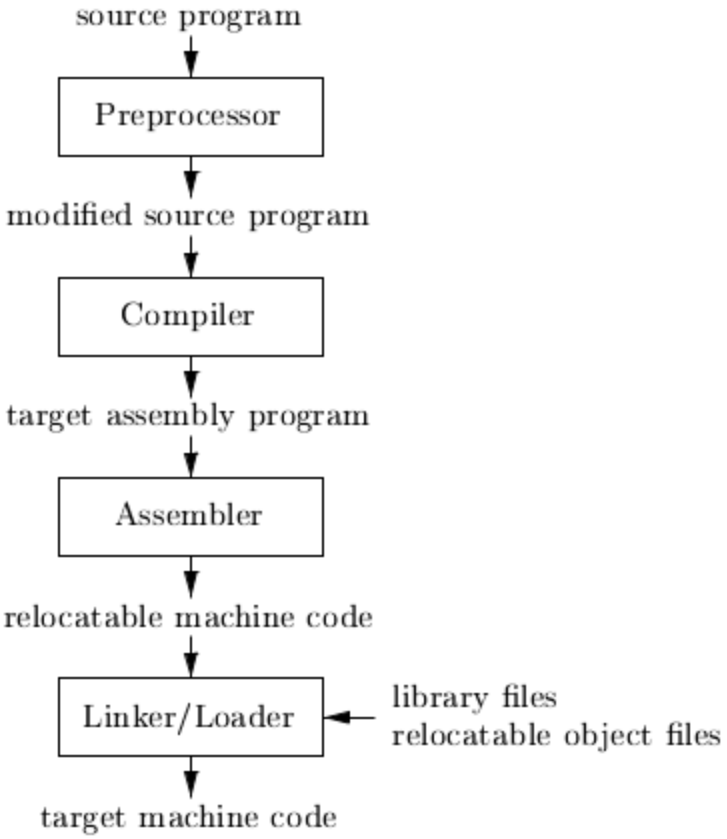
שפת תכנות היא קבוצה של סימונים המשמשת לכתיבת תוכניות מחשב. העולם היום תלוי על שפות תכנות, מכיוון שכל התוכניות הרצות בעולם נכתבו באחת מן כל שפות התכנות. אך, בשביל להריץ תוכנית הכתובה משפת תכנות, צריך לתרגם אותה לצורה שהמחשב יכול להריץ.
המערכות שיכולות לעשות תרגום שכזה נקראות *קומפיילרים* (או בעברית תקינה, *מהדרים*. במהלך כל התיק אני אשתמש במילה קומפיילר). במילים פשוטות, קומפיילר הינו תוכנית המקבלת תוכנית הכתובה בשפת תכנות - *שפת המקור* - ומתרגם את התוכנית הזאת לתוכנית מקבילה לתוכנית המקורית, רק בשפה אחרת - *שפת המטרה*.



עיבוד שפה

כאשר מעבדים שפת תכנות לשפת מכונה, הקומפיילר הוא רק חלק מהתהליך. עוד חלקים מהתהליך הם:

- מעבד מקדים - תכנית הקולטת נתונים מקדימים בשביל שהפלט שלה ישמש בתכנית אחרת. סוג תכנית זו תקרא תמיד לפני תכנית אחרת שתשתמש בפלו תכנית זאת, לכן השם עיבוד מקדים.
- מעבד שפת שף - מתרגם שפת סף לשפת מכונה
- מקשר - תכנית המחברת תוכניות מחשב שעברו הידור לשפת מכונה לתוכנית אחת.



מטרת הקומפיילר בתהליך היא לקחת את הפלט של המעבד המקדים (שלא אמור להיות שונה בצורה גדולה מקוד המקור), ולתרגם אותו לשפת סף.

מושגים

קומפיילר/מהדר

תוכנית המקבלת תוכנית הכתובה בשפת תכנות - *שפת המקור* - ומתרגם את התוכנית הזאת לתוכנית מקבילה לתוכנית המקורית, רק בשפה אחרת - *שפת המטרה*.

מתורגמן/Interpreter

תכנית המבצעת ישירות הוראות שנכתבו בשפות תכנות מבלי לדרוש שהן תורגמו לשפת מכונה. בדרך כלל האינטרפרטר כולל קבוצה של הוראות שאפשר לבצע ורשימה של הוראות אלו לפי הסדר שהתכניתן רצה שההוראות יפעלו.

האינטרפרטר מתרגם ומבצע את התכנית הרצויה שורה אחרי שורה, לכן בדרך כלל האינטרפרטרים יהיו איטיים מקומפיילרים, המתרגמים את כל התכנית.

מושגי תכנות

שפות תכנות

שפת תכנות היא קבוצה של סימונים המשמשת לכתיבת תוכניות מחשב. שפות תכנות נוצרו לראשונה בשביל להקל על בני אדם ליצור תכניות מחשב. אך, בשביל להשתמש בשפות האלה, צריכים תכנית התתרגם את התכנית לשפת מכונה.

כאשר מדברים על שפות תכנות, נהוג לחלק אותם לשני קטגוריות:

- **שפות תכנות עיליות (high level) -** שפת תכנות המיועדת לשימוש ע"י מתכנתים אנושיים. שפות תכנות עיליות משתמשות במבנים תחביריים האלולים להזכיר שפות טבעיות, לכן הן קלות לכתיבה וקריאה ע"י בני אדם. שפות אלה משתמשות בכמות הפשטה גדולה, לא רק בתחביר והסמנטיקה של התכנית, אלה גם במה שקורה ברקע, לדוגמה, שפות מודרניות מנהלות לבד את זיכרון התכנית. שפות תכנות עיליות מודרניות הופכות את תהליך הפיתוח לפשוט ומובן יותר. רמת ההפשטה של השפה מגדירה כמה "עילית" השפה.
- **שפות תכנות נמוכות (low level) -** שפת תכנות המספקת הפשטה מעטה, לכן תהיה משומשת ע"י מכונות ולא ע"י תוכניתנים. שמדברים על מושג ההפשטה בתכנות בקשר לשפות תכנות, בדר"כ מדברים על ההפשטה בין ארכיטקטורת סט ההוראות של המחשב (ISA) לבין השפה. כלומר, מכיוון ששפות סף הם "קרובות" לסט ההוראות של המחשב (מבחינה תחבירית), ההפשטה שלהם היא מעטה ולכן הם low-level, אולם שפה כמו python היא high-level בזכות כמות ההפשטה שהיא מספקת לתכניתן.

משתנים

מקום אחסון בעל שם וסוג ערך שמור. אפשר להתייחס למשתנים בעזרת שמם או כתובת הזיכרון שלהם. בזמן ריצת תכנית מחשב אפשר להכין משתנים, להגדיר להם ערך, לשנות ערך זה, למחוק את המשתנים ועוד.
דוגמאות להגדרת משתנים שישמשו כמידע על בן אדם:

```
// גיל המוגדר כמספר שלם
int age;
// שם המוגדר כמחרוזת של אותיות
string name;
// מספר אהוב המוגדר כשבר
float fav_number;
```

תנאים

הוראה הבודקת תנאי מסוים. תנאים הם דרך לבדוק תנאי מסוים בזמן ריצת התכנית, ואפשר להשתמש במשתנים בתנאים. תנאים בדר"כ כתובים בהוראות if - else, הכוונה היא אם קורה משהו, תעשה משהו, ואם לא תעשה משהו אחר:

```
if (condition):
  statement
else:
  statement
```

לולאות

לולאות משומשות בשביל להריץ חלק של הקוד שוב ושוב עד שתנאי מסוים מתקיים. יש שני סוגים עיקריים של לולאות, ה - for loop וה - while loop. בדר"כ משתמשים בfor שאנחנו יודעים את מספר האיטרציות, ובwhile שאנחנו לא. דוגמא לשני תוכניות המדפיסות 10 כוכביות, אחת לאחר השנייה:

```
int i;
for(i = 0; i < 10; ++i) {
  printf("*");
}

int i = 0;
while(i < 10) {
  printf("*");
  ++i;
}
```

ביטויים

Expression

יחידה תחבירית הניתן לעריכה על מנת לקבוע את ערכה, ומחזירה סוג ערך פרמיטי, כמו שלם או בוליאני.
דוגמאות ל-Expressionים:

- 1
- 2 + 2
- 2 > 4
- (4) && 49 * 8 == 5

Statement

יחידה תחבירית המתארת פעולה. שונה מ-expression כך שאין לו ערך כביכול, אלה תיאור של פעולה. דוגמאות:

- תנאים
- לולאות
- הצהרה על משתנים

תיאור הנושא

מכיוון שמטרת הקומפיילר הינה לתרגם שפת תכנות לשפת מכונה, הקומפיילר צריך לדעת איזה שפה הוא מתרגם, בשביל שיוכל לעבוד אליו. השפה שאנחנו נתרגם היא שפת *Quest*, שפה חדשה שנוצרה במיוחד לפרויקט הזה. השפה היא Turing Complete, כלומר השפה בעלת יכולת לדמות מכונת טיורינג, ובעל משתנים, לולאות, תנאים ועוד...

תכולת השפה

Tokens

להלן כל האסימונים של השפה והסוג שלהם:

```
//-----------------
// Debug Tokens (4)
//-----------------

DEBUG(null)
DEBUG(UNKNOWN)
DEBUG(eof)
DEBUG(COMMENT)
```

```
//------------------
// Base (4)
//------------------

BASETOK(IDENTIFIER)
BASETOK(NUMBER_CONSTANT)
BASETOK(CHAR_CONSTANT)
BASETOK(STRING_LITERAL)

//------------------
// Keywords (24)
//------------------

KEYWORD(BOOL,     "bool")
KEYWORD(BREAK,    "break")
KEYWORD(CASE,     "case")
KEYWORD(CHAR,     "char")
KEYWORD(CONST,    "const")
KEYWORD(CONTINUE, "continue")
KEYWORD(DO,       "do")
KEYWORD(DOUBLE,   "double")
KEYWORD(ELSE,     "else")
KEYWORD(FALSE,    "false")
KEYWORD(FLOAT,    "float")
KEYWORD(FOR,      "for")
KEYWORD(IF,       "if")
KEYWORD(INT,      "int")
KEYWORD(PRINT,    "print")
KEYWORD(RET,      "ret")
KEYWORD(SHORT,    "short")
KEYWORD(SIGNED,   "signed")
KEYWORD(SWITCH,   "switch")
KEYWORD(TRUE,     "true")
KEYWORD(TYPEDEF,  "typedef")
KEYWORD(UNSIGNED, "unsigned")
KEYWORD(VOID,     "void")
KEYWORD(WHILE,    "while")

//------------------
// Operators (38)
//------------------

// one character symbols have theire ASCII value used as theire token's
```

```
value
OPERAVAL(DOT,       ".", '.')
OPERAVAL(COMMA,     ",", ',')
OPERAVAL(TILDE,     "~", '~')
OPERAVAL(NOT,       "!", '!')
OPERAVAL(BITAND,    "&", '&')
OPERAVAL(BITOR,     "|", '|')
OPERAVAL(BITXOR,    "^", '^')
OPERAVAL(BITNOT,    "!", '!')
OPERAVAL(STAR,      "*", '*')
OPERAVAL(SLASH,     "/", '/')
OPERAVAL(PLUS,      "+", '+')
OPERAVAL(MINUS,     "-", '-')
OPERAVAL(PRECENT,   "%", '%')
OPERAVAL(GREATER,   ">", '>')
OPERAVAL(LESSER,    "<", '<')
OPERAVAL(EQUEL,     "=", '=')

OPERATOR(ELLIPSES,        "...")
OPERATOR(ARROW,           "->")
OPERATOR(AND,             "&&")
OPERATOR(OR,              "||")
OPERATOR(XOR,             "^^")
OPERATOR(SHIFTLEFT,       "<<")
OPERATOR(SHIFTRIGHT,      ">>")
OPERATOR(EQUELEQUEL,      "==")
OPERATOR(GREATEREQUEL,    ">=")
OPERATOR(LESSEREQUEL,     "<=")
OPERATOR(PLUSEQUEL,       "+=")
OPERATOR(MINUSEQUEL,      "-=")
OPERATOR(STAREQUEL,       "*=")
OPERATOR(SLASHEQUEL,      "/=")
OPERATOR(PRECENTEQUEL,    "%=")
OPERATOR(ANDEQUEL,        "&=")
OPERATOR(OREQUEL,         "|=")
OPERATOR(XOREQUEL,        "^=")
OPERATOR(NOTEQUEL,        "!=")
OPERATOR(TILDEEQUEL,      "~=")
OPERATOR(SHIFTLEFTEQUEL,  "<<=")
OPERATOR(SHIFTRIGHTEQUEL, ">>=")

//------------------
// Punctuators (10)
```

```
//------------------

// one character symbols have theire ASCII value used as theire token's
value
PUNCTUAVAL(LPAREN,    "(", '(')
PUNCTUAVAL(RPAREN,    ")", ')')
PUNCTUAVAL(LBRACK,    "[", '[')
PUNCTUAVAL(RBRACK,    "]", ']')
PUNCTUAVAL(LBRACE,    "{", '{')
PUNCTUAVAL(RBRACE,    "}", '}')
PUNCTUAVAL(LCHEVRON,  "<", '<')
PUNCTUAVAL(RCHEVRON,  ">", '>')
PUNCTUAVAL(COLON,     ":", ':')
PUNCTUAVAL(SEMICOLON, ";", ';')
```

BNF

להלן ה-bnf של השפה:

```
program ::= statement-list

statement-list ::= statement
                 | statement statement-list

statement ::= expression-statement
            | compound-statement
            | selection-statement
            | iteration-statement
           | print-statement
            | declaration

declaration ::= type-specifier identifier '=' constant-expression ';'

type-specifier ::= 'CHAR'
                 | 'INT'

expression-statement ::= constant-expression ';'

constant-expression ::= assignment-expression

assignment-expression ::= logical-or-expression
                        | assignment-expression assignment-operator
```

```
logical-or-expression

logical-or-expression ::= logical-and-expression
                        | logical-or-expression '||' logical-and-expression

logical-and-expression ::= inclusive-or-expression
                        | logical-and-expression '&&'
inclusive-or-expression

inclusive-or-expression ::= exclusive-or-expression
                            | inclusive-or-expression '|'
exclusive-or-expression

exclusive-or-expression ::= and-expression
                            | exclusive-or-expression '^' and-expression

and-expression ::= equality-expression
                  | and-expression '&' equality-expression

equality-expression ::= relational-expression
                        | equality-expression '==' relational-expression
                        | equality-expression '!=' relational-expression

relational-expression ::= shift-expression
                        | relational-expression '<' shift-expression
                        | relational-expression '>' shift-expression
                        | relational-expression '<=' shift-expression
                        | relational-expression '>=' shift-expression

shift-expression ::= additive-expression
                  | shift-expression '<<' additive-expression
                  | shift-expression '>>' additive-expression

additive-expression ::= multiplicative-expression
                        | additive-expression '+' multiplicative-expression
                        | additive-expression '-' multiplicative-expression

multiplicative-expression ::= primary-expression
                            | multiplicative-expression '*'
primary-expression
                            | multiplicative-expression '/'
primary-expression
                            | multiplicative-expression '%'
```

```
primary-expression

primary-expression ::= identifier
                       | constant
                       | string
                       | '(' expression ')'

expression ::= constant-expression
            | expression ',' constant-expression

assignment-operator ::= '='

unary-operator ::=   '+'
                  | '-'
                  | '~'
                  | '!'

compound-statement ::= '{' statement-list '}'

selection-statement ::= 'IF' '(' constant-expression ')' compound_statement
                              | 'IF' '(' constant-expression ')'
compound_statement 'ELSE' compound_statement

iteration-statement ::= 'WHILE' '(' constant-expression ')'
compound_statement

print-statement ::= 'PRINT' '(' identifier ')' ';'
                  | 'PRINT' '(' constant ')' ';'
```

תכנית לדוגמה

התכנית הבאה מחשבת את ה-greatest common divider בין 2 משתנים, a ו-b, ומדפיסה את אותו מספר:

```
int a = 36;
int b = 81;
char tmp = 0;

if(b > a) {
      tmp = a;
      a = b;
      b = tmp;
}
```

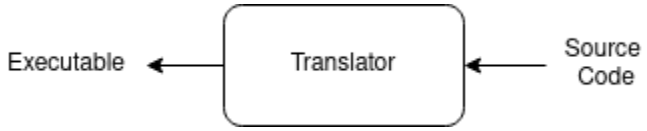
```
while (b > 0) {
      if(a > b) {
          a = a - b;
      } else {
          b = b - a;
      }
}

a = a + 48;
print(a);
```

רקע תיאורטי

תוכניות תרגום

המחשב לא יכולה לקרוא שפה מדוברת ישירות, אפילו לא שפות תכנות, המחשב מבין בינארית. תוכניות תרגום, או מתרגמים, הם תוכניות המתרגמות שפה אחת לשפה אחרת. בדר"כ משתמשים בתוכניות אלה בשביל לתרגם שפה מובנת לבני אדם, כלומר שפה עילית, לשפה מופשטת יותר, כמו שפת סף או בינארית, כלומר שפה תחתונה.

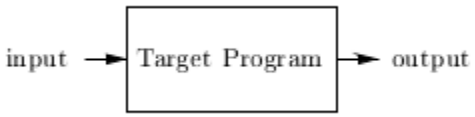
מתרגמים יכולים לתרגם תוכנית הכתובה בשפה עילית לתוכניות שאפשר להריץ על מכונה, ואותה תוכנית יכולה לקבל קלט, לעבד אותו, ולהוציא פלט אחר.



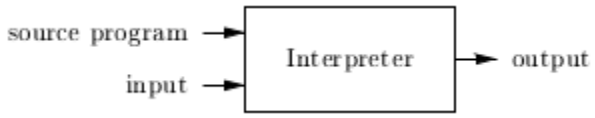
יש סוגים שונים של מתרגמים, כמו *קומפיילרים*, *אינטרפרטים* ו*אסמבלרים*, אך מתרגם יכול להיות כל תוכנית העומדת בתנאים, לא רק שלושת סוגי המתרגמים האלה.

הקומפיילר הינה תוכנית המתרגמת תוכנית משפה אחת לשפה אחרת, ובנוסף למצוא ולהתמודד עם שגיאות כאשר נמצאו. מה שמבדיל את הקומפיילר מסוגים אחרים של מתרגמים היא העובדה שהוא מעבד את כל התוכנית פעם אחת, ובדר"כ מוציא קובץ המתורגם לשפת המטרה.

עבודת הקומפיילר לתרגם תוכנית שנכתבה בשפת המקור ולתרגם אותה לשפת המטרה, ובנוסף למצוא ולהתמודד עם שגיאות מתי שאפשר. נשמע קל נכון? תגלו בהמשך את התשובה לבד...



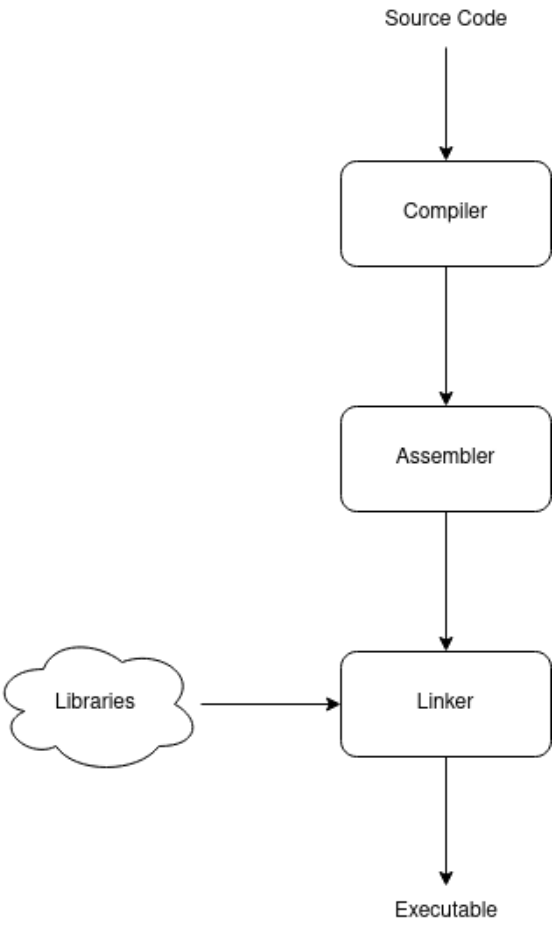
האינטרפרטר הוא עוד סוג של מתרגם. במקום להפיק תכנית כסוג של תרגום, האינטרפרטר נראה כיאלו הוא מריץ כל הוראה בקוד המקור אחד אחרי השני, בהתייחס לפלט. כלומר הוא מריץ את הפקודות אחד אחרי השני בלי קומפילציה של התוכנית.



ההבדל בניהם בא לידי ביטוי בתהליך התרגום. הקומפיילר עובד לפי העקרון "הכל או כלום", כלומר הוא מקמפל את התכנית, וכאשר רואה שגיאה שאי אפשר להתמודד איתה, הוא נעצר ומתאר את השגיאה. אולם האינטרפרט מריץ את התוכנית בצורה המתאימה עד שמבחין בשגיאה, ולאחר מכן מתאר את השגיאה שהגיע אליה. בנוסף אפשר להתייחס לתכנית עצמה. אם שפת המטרה הייתה שפת מכונה, בדר"כ התכנית שתרגם הקומפיילר תהיה יותר מהירה מהתכנית של האינטרפרטר. למרות זאת, אינטרפרטים טובים יותר במציאה וטיפול בשגיאות, מכיוון שהם מריצים את התוכנית הוראה אחת אחרי השנייה.

איך שפה מתורגמת לשפת מכונה

שאנחנו מתרגמים שפה לשפת מכונה בעזרת קומפיילר, הקומפיילר הוא לא השלב היחיד בתהליך התרגום. בדר"כ, קומפיילר מתרגמים את שפת המקור לשפת סף, ומשם ממשיך התהליך. התהליך השלם מתואר בתרשים הבא:



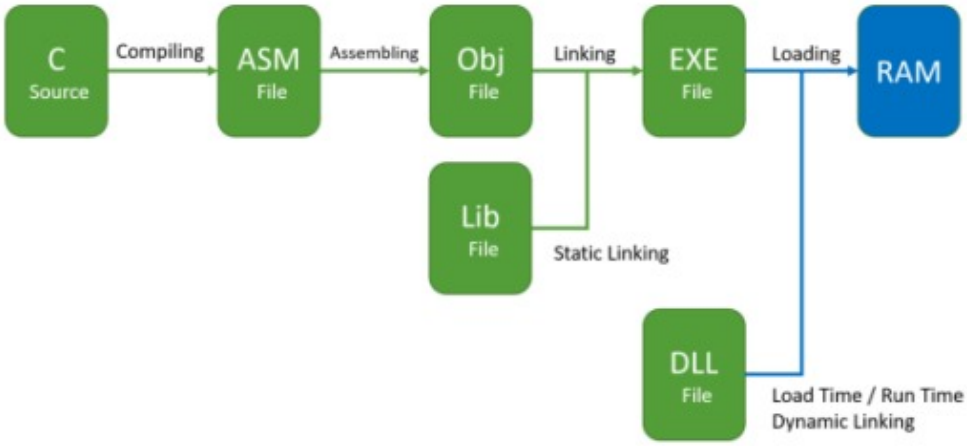
- הערה: בקומפיילרים מודרניים העיבוד המקדים הוא חלק מהקופיילר, לכן התהליך לא נראה כאן.

- מעבד שפת שף (assembler) - מתרגם שפת סף לשפת מכונה
- מקשר (linker) - תכנית המחברת תוכניות מחשב שעברו הידור לשפת מכונה לתוכנית אחת.
- ספריות (libraries) - חלק תוכנה read only שמוסף לתוכנית

התהליך הולך כך:

1. הקומפיילר מתרגם את השפה לשפת סף
2. שפת הסף מתורגמת על ידי האסמבלר
3. המקשר מקשר בין כל הספריות והתוכניות
4. מתקבל קובץ הרצה

שפת מכונה היא תלויה בהרבה דברים, כמו מערכת הפעלה ומשאבי המחשב. למשל, אם היינו רוצים להריץ תוכנית במערכת ההפעלה windows, היינו צריכים להפיק קובץ הרצה בפורמט exe. הנה תרשים המתאר איך תכנית בשפת c יהיה מתורגם לתכנית בwindows:



מבנה של קומפיילר

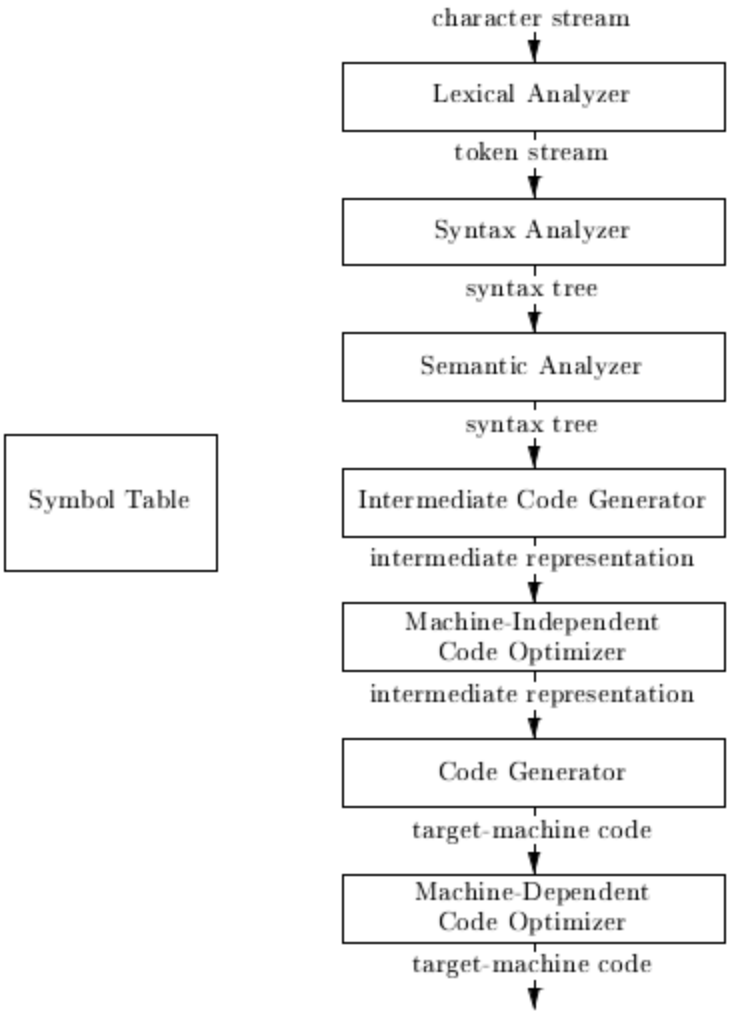
עד כו התייחסנו אל הקומפיילר כקופסה שחורה הממירה תכנית מקור לתוכנית מטרה השקולה לה. אם נפתח קצת את הקופסא הזאת נראה שישנם שני חלקים עיקריים: *אנליזה (analysis)* ו*סינתזה (synthesis)*.

בחלק האנליזה התכנית מפרקת את תכנית המקור למבנים מתאימים וכופה אליהם חוקים מילוניים, תחביריים וסמנטים. בנוסף חלק האנליזה הופך את תוכנית המקור למבנה ביניים המתאר את התכנית ברמה קרובה יותר לתכנית היעד. אם התכנית שמה לב בשגיאות מילוניות, תחביריות וסמנטיות, אליה להגיב ולהתמודד אם אותם שגיאות, ובמידת הצורך גם לתאר אותם למשתמש. בנוסף, התכנית אוספת מידע על תכנית המקור ושמה אותו במבנה שנקרא ו*טבלת הסימנים*. לאחר מכאן, טבלת הסימנים ומבנה הביניים נעברים על חלק הסינתזה.

בחלק הסינתזה התכנית יוצרת את תכנית המטרה בעזרת מבנה הביניים וטבלת הסימנים.

בדר"כ, חלק האנליזה נקרא ה-front end, וחלק הסינתזה נקרא ה-back end.

כאשר נפתח את הקופסה השחורה עוד יותר ונראה את הפרטים הקטנים, נבחין שהתכנית עובדת כסדרה של שלבים, כאשר כל אחד משנה את מבנה התכנית למבנה אחר. התרשים הבא מתאר את כל אחד מהשלבים האלה, את הפלט שלהם ואת הקלט שלהם:



- הערה: אופטימיזציה היא תהליך אופציונלי, לכן תרשימים שונים יכולים להראות רק את אחד משני תהליכי האופטימיזציה, או אף אחד מהם.

ניתוח מילוני (lexical analysis)

השלב הראשון של התכנית נקרא *המנתח המילוני* או *הלקסר (lexer)*. הלקסר קורא את זרם התווים המהווים את תוכנית המקור, ומקבץ תווים למחרוזות בעל ערך בשם *lexemes*. בשביל כל אחד מה-lexemes הלקסר יוצר *אסימון (token)* במבנה הבא:

$$< name, attribute/value >$$

הלקסר יוצר זרם של אותם אסימונים ומעביר אותם לשלב הבא.
בתוך האסימון יש שני ערכים, שם האסימון והערך שלו. שם האסימון הוא סמל מופשט המתאר את סוג האסימון, וערך האסימון הוא ערך שימושי המאוחסן בתוך האסימון. לדוגמה, יכול להיות לנו את מחרוזת התווים הבאה:

Position = initial + rate * 60

תווים יכולים להיות מתאימים לאסימונים באופן הבא:

- Position יהיה ממופה לאסימון <id, 1> כאשר id הוא סמל מופשט המתאר שם של משתנה, ו-Position הוא שם הערך
- = יהיה ממופה לאסימון <=>
- Initial יהיה ממופה לאסימון <id, 2>
- + יהיה ממופה לאסימון <+>
- Rate יהיה ממופה לאסימון <id, 3>
- * יהיה ממופה לאסימון <*>
- 60 יהיה ממופה לאסימון <60>

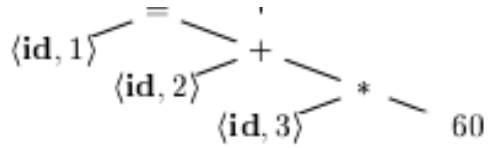
ולכן, זרם האסימון יהיה:

$$\langle \mathbf{id}, 1\rangle \ \langle = \rangle \ \langle \mathbf{id}, 2\rangle \ \langle + \rangle \ \langle \mathbf{id}, 3\rangle \ \langle * \rangle \ \langle 60 \rangle$$

ניתוח תחביר (syntax analysis/parsing)

השלב השני של התכנית נקרא *המנתח המילוני* או *הפרסר (parser)*. הפרסר משתמש בזרם האסימונים מהשלב הראשון בשביל ליצור עץ תחבירי המתאר את המבנה התחבירי של זרם האסימונים. בעץ תחבירי כל צומת מתארת תהליך מסוים, והילדים של כל צומת מתארים את סימני התהליך.

בהתאמה לדוגמה מהשלב הראשון, העץ התחבירי שיופק מזרם האסימונים יכול להיות:



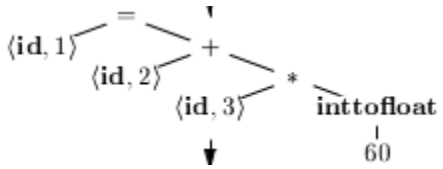
כאשר שורש העץ (=) מתאר את תהליך אחסון הערך של הבן הימני שלו (העץ ששורשו +) לתוך הבן השמאלי שלו, + מתאר את תהליך החיבור בין הבן השמאלי לבן הימני, ו-* מתאר את תהליך הכפל בין הבן השמאלי והבן הימני.

חוקי המנתח התחבירי מוגדרים בתכנית בקוד, אך יש מסמכים כמו מסמכי BNF המתארים את תחביר השפה. קוד הקומפיילר אעקוב אחרי מסמך הBNF בשביל מקור חוקי התחביר שלו.

ניתוח סמנטי (semantic analysis)

השלב השלישי של התכנית נקרא *המנתח הסמנטי*. הוא משתמש בעץ התחביר ההופק בשלב הקודם בשביל לבדוק עקביות סמנטית בתכנית המקור בעזרת חוקים סמנטים המוגדרים בקוד. בנוסף, הוא אוסף נתונים על התכנית בעזרת העץ התחבירי ושומר אותם או בטבלת הסימנים או בעץ סמנטי.

לדוגמה, לפי העץ התחבירי מהשלב הקודם, העץ הסמנטי יכול להיות:



כאשר השינוי היחיד הוא הגדרת שינוי סוג המספר 60, שהיה מסוג int אך שונה לfloat, מכיוון שהצומת משמאלו היא מסוג float. בנוסף, טבלת הסימנים הייתה נראת כך:

1	position	...
2	initial	...
3	rate	...

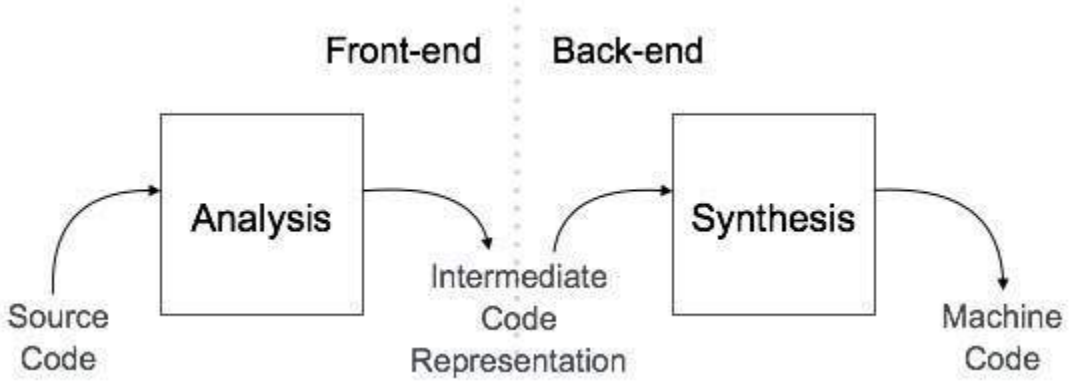
כאשר כל אחד מהתאים בטבלה מתאים למשתנה בתכנית.

חלק חשוב בשלב הוא בדיקת סוגי משתנים, כאשר הקומפיילר בודק שלכל operator יש operands מתאימים. לדוגמה, להרבה שפות יש חוק המגדיר ש-index של מערך חייב להיות מסוג שלם, ואם הוא לא מסוג שלם, הקומפיילר מדווח על זאת. לפעמים הקומפיילר יכול להתמודד סוגי משתנים לא מתאימים, כפי שנראה בדוגמה הקודמת.

יצירת קוד ביניים (intermediate code generation)

השלב הרביעי של התכנית הוא יצירת קוד הביניים, בוא הקומפיילר אבנה אחד או יותר מבני קוד ביניים, שיכולים להיות מבנים שונים המגשימים מטרות שונות. דוגמה למבנה אחד של קוד ביניים הוא העץ הסמנטי.

קוד הביניים הינו מבנה המתאר את את תכנית המקור ברמה נמוכה, כמו תכנית מופשטת של התכנית המקורית. המטרה העיקרית של קוד הביניים להיות קל ליצור ולתרגם, ולהשאיר את התכנית המקורית באותה מבנה שהייתה. לכן, מבנים שונים מתאימים לקומפיילרים שונים.



אופטימיזציה (code optimization)

בשלב החמישי של התכנית, אופטימיזציה, התכנית מנסה לשפר את קוד הביניים כך שיהיה קוד מטרה טוב יותר מתרגום ישיר של קוד המקור, אך עם השארת הלאגוריתם כפי שהיא. בדר"כ כאשר מדברים על אופטימיזציה מתכוונים לשיפור התכנית ביחס למהירות, אך גם משתנים אחרים יכולים להכנס כמו שימוש באחסון או שימוש במשאבים.

למרות זאת, שלב האופטימיזציה הוא שלב אופציונלי לקומפיילר.

יצירת קוד (code generation)

בשלב השישי והאחרון של הקומפיילר התכנית צריכה ליצור את קוד המטרה בעזרת קוד הביניים. יצירת הקוד יכולה להיות תרגום ישיר של קוד הביניים, או להשתנות בהתאם ליחסים שונים של התכנית.

אם שפת המטרה היא שפת מכונה, רגיסטרים וזיכרון צריכים להיות מתאימים להוראות בשפה. ואז קוד הביניים מתורגם למחרוזת הוראות המתארות את אותו אלגוריתם שהיה מתואר בקוד המקור.
לדוגמה, אם שפת המטרה היא שפת סף, קוד המטרה יכול להיראות כך:

```
LDF  R2,  id3
MULF R2,  R2, #60.0
LDF  R1,  id2
ADDF R1,  R1, R2
STF  id1, R1
```

טבלת סימנים (symbol table)

טבלת סימונים (Symbol Table) משמשת ככלי חיוני לניהול מידע אודות המשתנים, הפונקציות, הנהלים והטיפוסים המוגדרים בקוד המקור.

טבלת הסימונים היא מבנה נתונים דינמי המאחסן מידע מפורט על כל אובייקט סימנטי בקוד המקור, כולל:

- **שם האובייקט:** שם ייחודי המזהה את האובייקט בקוד המקור.
- **סוג האובייקט:** משתנה, פונקציה, נתונים, טיפוס מוגדר על ידי המשתמש, ועוד.
- **ערך האובייקט:** ערך התחלתי (במידה וקיים) או טווח ערכים אפשריים.
- **מיקום האובייקט:** מיקום האובייקט בזיכרון (במידה וקיים) או מידע אחר הקשור למיקומו.
- **מידע נוסף:** מידע רלוונטי נוסף, כגון סוגי נתונים של פרמטרים של פונקציה, תכונות של משתנים, ועוד.

טבלת הסימונים משמשת לאורך כל תהליך הקומפילציה, החל מניתוח תחבירי ועד יצירת קוד מכונה. השימושים העיקריים בטבלת הסימונים כוללים:

- **זיהוי ופתרון שגיאות:** הקומפילטור משתמש בטבלת הסימונים כדי לזהות שגיאות תחביריות וסמנטיות בקוד המקור, כגון שימוש כפול באותו שם משתנה, הצהרה על משתנה שאינו מוגדר, או קריאת פונקציה שאינה קיימת.
- **בניית קוד ביניים:** טבלת הסימונים מספקת מידע חיוני עבור בניית קוד ביניים יעיל, כגון הקצאת זיכרון למשתנים, יצירת קישורים בין פונקציות, וביצוע אופטימיזציות שונות.
- **יצירת קוד מכונה:** טבלת הסימונים משמשת ליצירת קוד מכונה המתאים לקוד המקור, תוך התחשבות בארכיטקטורת המחשב ובמערכת ההפעלה.

טבלת הסימונים היא מרכיב קריטי בקומפילטור, והיא תורמת רבות ליעילות ודיוק תהליך הקומפילציה.

חשוב לציין שקיימות גישות שונות למימוש טבלאות סימונים בקומפילטורים שונים. גישות אלו נבדלות זו מזו במבנה הנתונים, באלגוריתמי הניהול, ובמידת המידע המאוחסן בטבלה.

ניתוח מילוני

הניתוח המילוני הוא השלב הראשון בתהליך בעל שלושת שלבים שהקומפיילר משתמש בשביל להבין את תכנית המקור. המנתח במילוני, או לקסר, מקבל את זרם התווים שמרכיבים את התכנית, מעבד אותם ומוציא זרם של אסימונים המרכיבים את התכנית.

תפקיד המנתח המילוני

למה אנחנו צריכים להפוך את מחרוזת התווים המרכיבה את התכנית לזרם של אסימונים? אותו זרם אסימונים שהלקסר מעבד עוזר לנו להפשיט ולנתח את השפה בשלבי הניתוח הבאים. זרם האסימונים נשלח על המנתח התחבירי, שם הוא משתמש באסימונים המופשטים בשביל ליצור עץ תחבירי, שבדר"כ חלקו בנוי מאותם אסימונים. קומפיילרים מסוימים אוספים מידע על משתנים בניתוח המילוני ושומרים את אותו מידע בטבלת הסימנים.

אסימון, תבנית ולקסמה

- אסימון (token) הוא זוג המורכב משם אסימון וערך מאפיין אופציונלי. שם האסימון הוא סמל מופשט המייצג סוג של יחידה לקסיקלית, לדוגמה מילת מפתח מסוימת, או רצף תווים בקלט המציין מזהה. שמות האסימונים הם סימני הקלט שהמנתח התחבירי מעבד. המאפיין הוא ערך המגדיר את האסימון כאשר לקסמה מתאים ליותר מתבנית אחת, לדוגמה, אם יש לנו אסימון עם השם **number** (כלומר אסימון המתאים למספר) אותו אסימון יכול להתאים לכמה מספרים. לכן, נתאים לו מאפיין שיהיה ערך המספר, וכך נדע מה הערך המספרי של **number**.

- תבנית היא תיאור של הצורה בה עשויים להופיע הלקסמות של אסימון. במקרה של מילת מפתח כאסימון, התבנית היא רק רצף התווים המרכיבים את מילת המפתח. עבור מזהים ואסימונים אחרים, התבנית היא מבנה מורכב יותר שמוצא התאמה על ידי מחרוזות רבות.

- לקסמה היא רצף תווים בתוכנית המקור שתואם לתבנית עבור אסימון ומזוהה על ידי המנתח הלקסיקלי כמקרה ספציפי של אותו אסימון.

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

התרשים מלמעלה מתאר דוגמאות לאסימונים.

תהליך הניתוח המילוני

מכיוון שלקסר הוא החלק בקומפיילר שקורא את תכנית המקור, יש לו עוד תת-מטרות שהוא צריך לעשות בשביל להכין זרם נכון של אסימונים. חלק אחד מן תת המטרות האלה יהיה להתעלם מהערות שיכולות להימצא בקוד ורווחים (space, blank, tab, newline וכו..), חלק אחר יהיה לשמור מונה של השורות של התכנית. יש עוד הרבה תת-מטרות כאלה, אך הם תלויים בהגדרת השפה.

הגדרת השפה

להלן הגדרות לשפה פורמלית:

- *א"ב* מסוים הוא כל סט סופי של סימנים. לדוגמה, הסט {0, 1} מגדיר את הא"ב הבינארית. נהוג לסמן את הא"ב בסימן Σ.
- *מחרוזת* מעל א"ב מסוים הינה רצף של סימנים הלקוחים מא"ב מסוים. נהוג לסמן אורך של מחרוזת (מספר הסימנים במחרוזת) s כ-$|s|$. המחרוזת הריקה, ε, היא המחרוזת בעל האורך 0.
- *שפה* היא כל סט של מחרוזות מעל א"ב מסוים. נהוג לסמן שפה באות L, ונהוג להגדיר שפה כך:

$$L = \{s \in \Sigma^* \mid Condition\}$$

  - כאשר s הינו מחרוזת
  - *Σ הינו אוסף כל המילים מעל הא"ב Σ
  - Condition הינו תנאי מסוים שהמילים בשפה מוגבלים אליו

כמה תהליכים חשובים על שפה מוגדרים בטבלה הבאה:

Operation	Definition and Notation
Union of L and M	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
Kleene closure of L	$L^* = \cup_{i=0}^{\infty} L^i$
Positive closure of L	$L^+ = \cup_{i=1}^{\infty} L^i$

- Regular expression הם אנוטציה המגדירה תבניתים של lexemes. בעזרתם נוכל להגדיר את התבניות של אסימונים בשפה. לדוגמה, אם נרצה להגדיר תבנית המתארת את כל האנוטציות האפשריות בשביל שם משתנה ב-C, נגדיר אותם כך (בעזרת Regular Expression):

*letter_(letter_ | digit)**

תהליכים חשובים בשביל regular expression מוגדרים בטבלה הבאה:

LAW	DESCRIPTION
$r\|s = s\|r$	$\|$ is commutative
$r\|(s\|t) = (r\|s)\|t$	$\|$ is associative
$r(st) = (rs)t$	Concatenation is associative
$r(s\|t) = rs\|rt;\ (s\|t)r = sr\|tr$	Concatenation distributes over $\|$
$\epsilon r = r\epsilon = r$	ϵ is the identity for concatenation
$r^* = (r\|\epsilon)^*$	ϵ is guaranteed in a closure
$r^{**} = r^*$	$*$ is idempotent

- Regular definition הינה אנוטציה הנותנת לתת שמות לRegular expressionים מסוים (אקרה Regular expression מהלך גם בשם Regex ול-Regular definition כ-Regdef). ההגדרה הפורמלית של Regdef הינה:

כאשר Σ הינה א"ב של סימנים, אז Regdef הינה רצף של הגדרות מהמבנה:

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\dots$$
$$d_n \rightarrow r_n$$

כאשר:

- d_i הינו סימן חדש, לא ב-Σ ולא דומה לכל d אחר
- r_i הינו Regex מעל $\sum \cup \{d_1, d_2, d_3, \dots, d_{i-1}\}$

עכשיו בעזרת Regdef, אנחנו יכולים להגדיר את שמות המשתנים בצורה יותר פורמלית כך:

$$\begin{aligned} letter\_ &\rightarrow \texttt{A} \mid \texttt{B} \mid \cdots \mid \texttt{Z} \mid \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \mid \texttt{\_} \\ digit &\rightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\ id &\rightarrow letter\_\ (\ letter\_ \mid digit\ )^* \end{aligned}$$

להלן הגדרות על Regdef שנשתמש בעתיד:

EXPRESSION	MATCHES	EXAMPLE
c	the one non-operator character c	a
\c	character c literally	*
"s"	string s literally	"**"
.	any character but newline	a.*b
^	beginning of a line	^abc
$	end of a line	abc$
[s]	any one of the characters in string s	[abc]
[^s]	any one character not in string s	[^abc]
r*	zero or more strings matching r	a*
r+	one or more strings matching r	a+
r?	zero or one r	a?
$r\{m,n\}$	between m and n occurrences of r	a{1,5}
$r_1 r_2$	an r_1 followed by an r_2	ab
$r_1 \mid r_2$	an r_1 or an r_2	a|b
(r)	same as r	(a|b)
r_1/r_2	r_1 when followed by r_2	abc/123

בעזרת כלים אלו אנחנו יכולים להגדיר מבנים ללקסימות והאסימנים המתאימים להם. לדוגמה, הנה הגדרה פורמלית לאסימונים בשפה, שיהיו מנותחים ע"י מנתח מילוני:

$$
\begin{array}{rcl}
digit & \rightarrow & \texttt{[0-9]} \\
digits & \rightarrow & digit^{+} \\
number & \rightarrow & digits\ (\texttt{.}\ digits)?\ (\ \texttt{E}\ \texttt{[+-]}?\ digits\ )? \\
letter & \rightarrow & \texttt{[A-Za-z]} \\
id & \rightarrow & letter\ (\ letter \mid digit\ )^{*} \\
if & \rightarrow & \texttt{if} \\
then & \rightarrow & \texttt{then} \\
else & \rightarrow & \texttt{else} \\
relop & \rightarrow & \texttt{<} \mid \texttt{>} \mid \texttt{<=} \mid \texttt{>=} \mid \texttt{=} \mid \texttt{<>}
\end{array}
$$

בנוסף נרצה להגדיר עוד אסימון בשביל למחוק רווחים:

$$ws \rightarrow (\ \textbf{blank} \mid \textbf{tab} \mid \textbf{newline}\ )^{+}$$

להלן טבלה המגדירה lexemes ואת האסימונים שלהם:

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
if	**if**	–
then	**then**	–
else	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**number**	Pointer to table entry
<	**relop**	LT
<=	**relop**	LE
=	**relop**	EQ
<>	**relop**	NE
>	**relop**	GT
>=	**relop**	GE

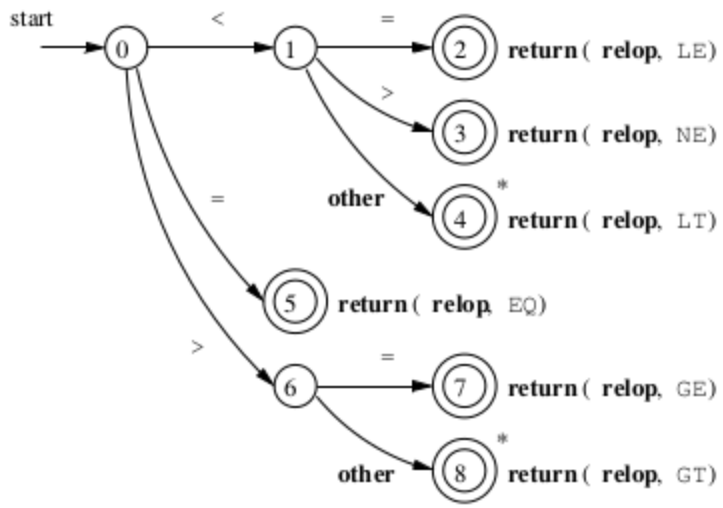
מכונת מצבים

מכונת מצבים (Finite State Machine - FSM) היא מודל חישובי מופשט המשמש לתיאור מערכות שמתנהלות דרך סדרת מצבים.

רכיבים עיקריים של מכונת מצבים:

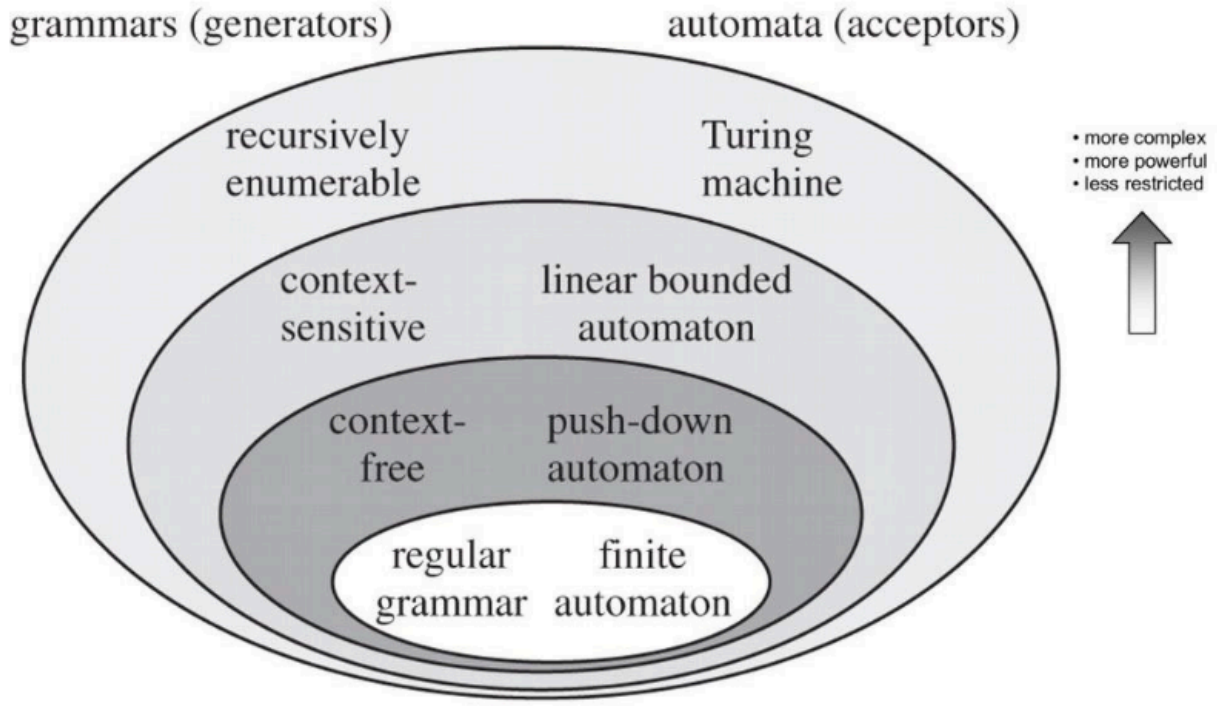
- **מצבים:** קבוצה סופית של מצבים, המייצגים את כל התצורות האפשריות של המערכת. מתואר ע"י עיגול בדר"כ בדיאגרמה, או ע"י האותיות q ו-Q.
- **מצב התחלתי:** מצב מיוחד שבו המערכת מתחילה. מתואר ע"י חץ בלי מצב מצב מתחיל, או ע"י האות q_0
.
- **מצבים מקבלים:** קבוצת מצבים מיוחדים שמסמנים שהמערכת ביצעה את הפעולה בהצלחה. מתוארים בדר"כ ע"י עיגול כפול בדיאגרמה, או ע"י האות F.
- **פונקציות המעברים:** כללים המגדירים כיצד המערכת עוברת ממצב אחד למצב אחר בתגובה לקלט. מתוארים בדר"כ ע"י חץ בעל מצב מתחיל ומקבל בדיאגרמה, או ע"י האות δ.
- **פונקציית פלט:** פונקציה שמספקת פלט עבור כל מצב.

לדוגמה, הנה מכונת מצבים המגדירה את מכונת המצבים לאסימון **relop**, שהוגדר לפני:



סוגי אוטומטים

ישנם כמה סוגים של אוטומטים, כאלה שיכולים לקבל קבוצות שונות של שפות. סוגי האוטומטים מוגדרים ע"פ ה-Chomsky Hierarchy, היררכיה של אוטומטים, כאשר כל אוטומט רחב יותר עוטף את שאר האוטומטים בהיררכיה. היא מגדירה איזה אוטומט "חזק יותר" משאר האוטומטים ואיזה סוגי שפות כל אחד מהאוטמטים האלה יכול להגדיר. ההיררכיה הוגדרה לראשונה ע"י Noam Chomsky, וחברו Marcel-Paul Schützenberger עזר לו להגדיר את ההיררכיה. המאמר בוא הגדירו את ההיררכיה מופיע בביליוגרפיה.



אפשר לראות בדיאגרמה שהאוטומט החלש ביותר, אוטומט סופי, יכול להגדיר רק שפה המוגדרת לפי RegRx, ושהאוטומט החזק ביותר, turing machine, שהינו המודל המרכזי בתיאור אופן פעולת המחשב המודרני, יכול

להגדיר שפות רקורסיביות. למעשה יהיה ניתן לראות שבמהלך תהליך הניתוח, ננתח את השפה בעזרת אוטומטים יותר ויותר מסובכים, עד שנגיע לשפה שהיא turing-complete.

אוטומט סופי דטרמיניסטי/לא דטרמיניסטי

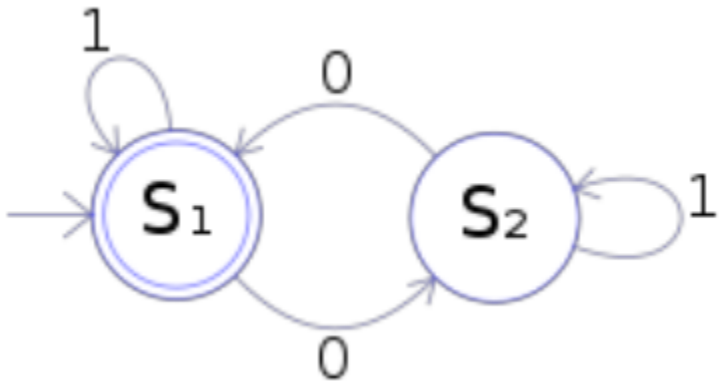
אוטומט סופי דטרמיניסטי ואוטומט סופי לא דטרמיניסטי (בהתאמה DFA ו NFA) הם סוגי אוטומטים סופיים.
שניהם מורכבים מהרכיבים (שהוגדרו לפני זאת):

- מצבים - q
- מצב התחלתי - q_0
- מצבים מקבלים - F
- פונקציות מעבר - δ

האוטומט הסופי יכול להיות מוגדר ע"י RegEx.

האוטומט הסופי הדטרמיניסטי הוא אוטומט שלכל מצב יש פונקצית מעבר עבור כל סימן בא"ב שלו.
האוטומט הסופי הלא דטרמיניסטי הוא אוטומט שלא לכל מצב יש פונקצית מעבר עבור כל סימן בא"ב שלו. כלומר, יש לפחות מצב אחד שאין לו פונקצית מעבר עבור סימן בא"ב.
ההבדל בין האוטומט הסופי דטרמיניסטי והאוטומט הסופי הלא דטרמיניסטי הוא בהגדרתם, כאשר יש לפחות מצב אחד שאין לו פונקציית מעבר עבור סימן מסוים, הוא לא טדרמיניסטי, אך כאשר לכל המצבים יש פונקציות מעבר עבור כל הסימנים, האוטמט הוא דטרמיניסטי.

להלן דוגמה לאוטומסט סופי דטרמיניסטי המגדיר שפה מעל הא"ב {0, 1} המקבל רק מחרוזות עם מספר זוגי של אפסים:



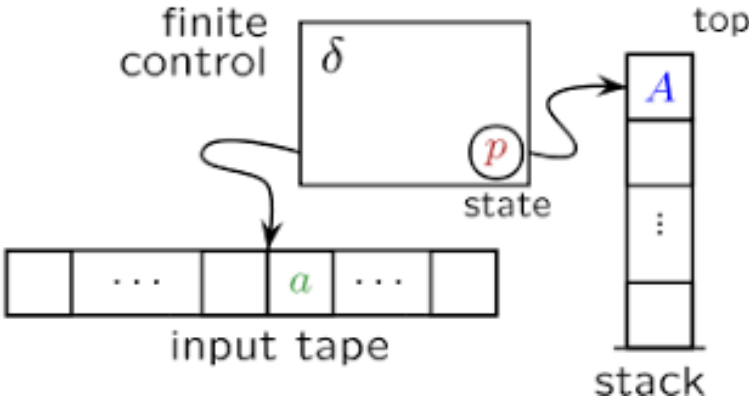
אוטומט מחסנית

אוטומט מחסנית (Push Down Automaton) הוא אוטומט סופי אשר נעזרת במחסנית לביצוע מעברים. פונקצית המעבר לא מוגדרת אך ורק על גבי הסימן המתקבל, אלא גם על ידי מה שנמצא בראש המחסנית. הפעולות היחידות האפשריות על המחסנית הם Push ו-Pop של איברים הנמצאים בא"ב של המחסנית. הרכיבים המגדירים PDA הם:
Q - אוסף סופי של מצבים
Σ - הא"ב של הקלט
Γ - הא"ב של המחסנית

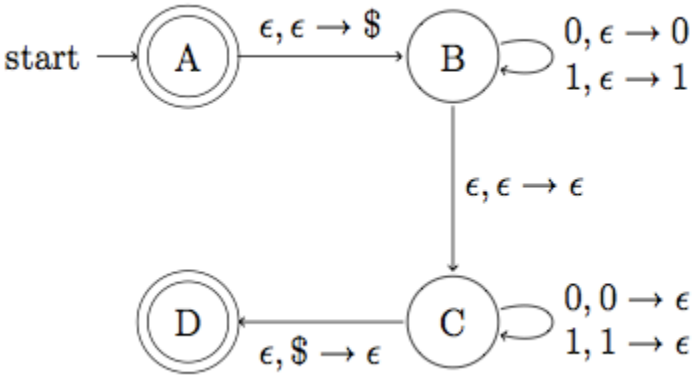
δ - פונקציות המעבר המוגדרות ע"י הקלט והמחסנית
q_0 - המצב ההתחלתי של האוטומט
Z - הסמל ההתחלתי על המחסנית
F - אוסף המצבים המקבלים

אוטומט מחסנית יכול להגדיר שפות חופשיות הקשר.

להלן דיאגרמה המתארת בצורה מופשטת מה זה אוטומט מחסנית:



להלן דיאגרמה המתארת אוטומט מחסנית המקבל פלינדרומים מהא"ב {0, 1}:

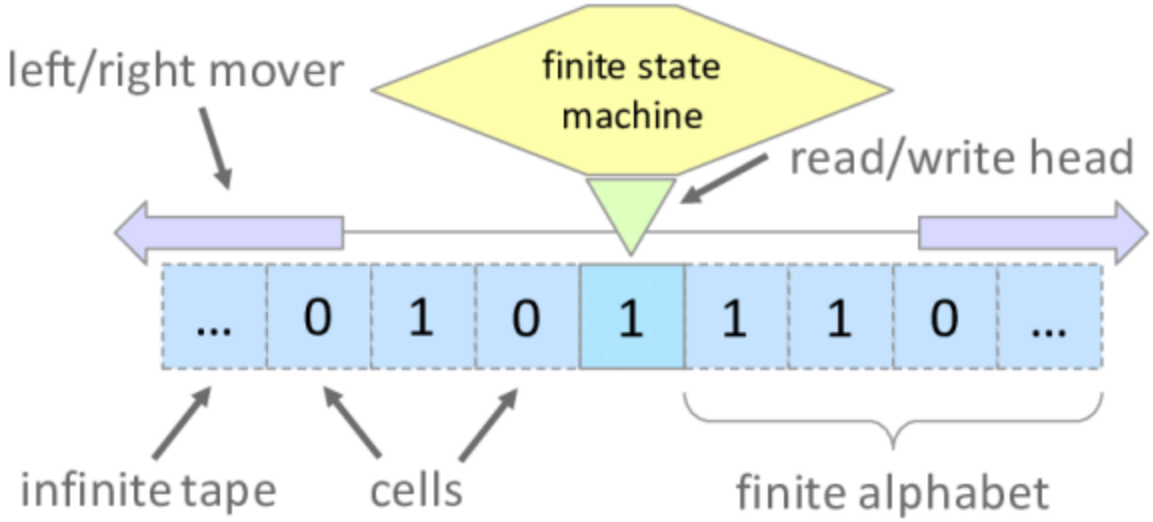


מכונת טיורינג

מכונת טיורינג היא מודל מתמטי המתאר מכונה מופשטת המשתמשת בסימנים על גבי סרט וסט של חוקים בשביל אלגוריתם. למרות פשטות המכונה, אפשר להגדיר העזרתה כל אלגוריתם חישובי שניתן להגדיר במחשבים מודרנים. המודל הומצא ע"י אלן טיורינג בשנת 1936.

מכונת טיורינג נחשבת לאוטומט החזק ביותר, אך היא לא יכולה להגדיר כל שפה שקיימת. הוכחה לזה קיימת בשני מאמרים שיצאו בשנת 1936, אחד ע" אלן טיורינג והשני ע"י אלאנזו צ'רצ', העונים על השאלה החישובית, ה-Entscheidungsproblem.

להלן דיאגרמה אבסטרקטית המתארת מכונת טיורינג:



ניתוח תחבירי

Context-Free Grammars

Grammars מתארים את התחביר של מבני שפות תכנות כמו expressions ו-statements. לדוגמה, אנחנו יכולים להשתמש במשתנים בשם stmt ו-expr בשביל לתאר statements ו-expressions בחוק תחבירי כך:

$$stmt \rightarrow if\ (expr)\ stmt\ else\ stmt$$

מה שמתאר statemente של if-else.

ההגדרה של שפות חסרות הקשר:

- טרמינלים - הסימנים הבסיסיים מהן מחרוזות נבנות. שמות האסימונים הם בעצם הטרמינלים, כאשר האסימונים הם זרם האסימונים היוצא מהמנתח המילוניי
- לא-טרמינלים (non-terminals) - משתנים תחבירים המתארים סטים של מחרוזות. למשל, בדוגמה הקודמת stmt ו-expr הם לא-טרמינלים. לא-טרמינלים יוצרים את המבנה ההירכי של העץ התחבירי
- סימן התחלה - בכל חוק יש סימן התחלה, ולאחר מכן סט של סימנים.
- חוק - כל חוק מגדיר את הדרך בא טרמינלים ולא-טרמינלים מתחברים בשביל ליצור מחרוזת, ואילו מחרוזות מוגדרות ע"י לא-טרמינל מסוים. כל חוק בנוי מ:
  - הצד השמאלי או הראש של החוק; מגדיר לאיזה לא-טרמינל הצד הימני נגזר
  - סימן כמו → אן =:: בשביל לתאר את הקשר בין הראש והגוף
  - הצד הימני או גוף החוק; מגדיר את המחרוזת של טרמינלים ולא-טרמינלים הנגזרים לצד הימני

לדוגמה, שני דרכים לרשום תחביר הבנוי מהטרמינלים הבאים:

id + - * / ( )

$$
\begin{array}{rcl}
expression & \rightarrow & expression + term \\
expression & \rightarrow & expression - term \\
expression & \rightarrow & term \\
term & \rightarrow & term * factor \\
term & \rightarrow & term \ / \ factor \\
term & \rightarrow & factor \\
factor & \rightarrow & (\ expression\ ) \\
factor & \rightarrow & \mathbf{id}
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid E - T \mid T \\
T & \rightarrow & T * F \mid T \ / \ F \mid F \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

Derivations

בניית העץ התחבירי יכולה להיות מובנת במהותה ע"י נגזרות וחוקים תחביריים. אפשר לחשוב על נגזרת כהתהליך של להחליף את ראש החוק התחבירי עם הגוף שלו, וכך כל צעד של נגזרת בעץ התחבירי מתאים לחוק.
לדוגמה, אם ניקח את החוק התחבירי הבא:

$$E \rightarrow E + E \mid E * E \mid - E \mid (\ E\ ) \mid \mathbf{id}$$

נוכל לתאר את הנגזרת שלו כך:

$$E \Rightarrow - E$$

המתאר "אם E מתאר expression, אזי E- גם מתאר expression".
הסימן ⇒ מתאר נגזרת.

יש שני סוגים של נגזרות:

- Leftmost Derivation - קובע שה non-terminal השמאלי ביותר יוחלף
- Rightmost Derivation - קובע שה non-terminal הימני ביותר יוחלף

לדוגמה, אם ניקח את המחרוזת (**id** + **id**)- נוכל לגזור אותה בשני דרכים:

1. left-most:

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

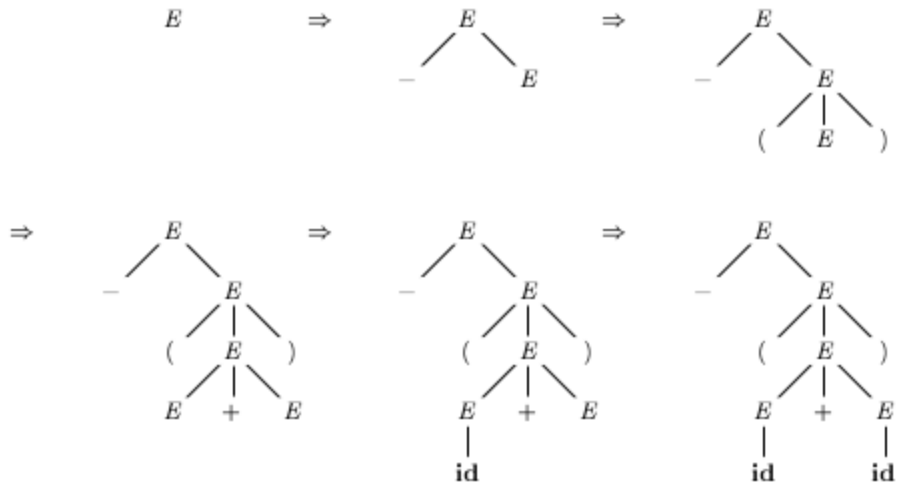
2. right-most:

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id})$$

עץ תחביר

עץ תחביר הוא דרך ויזואלית להגדיר את שלבי הנגזרת שעוברת מחרוזת של סימנים. כל צומת פנימי בעץ מגדיר את ה non-terminal הנגזר, ובניו מגדירים את גוף החוק ממנו נגזר.

לדוגמה, דרכי בניית העץ התחבירי המתאים לדוגמה בשביל left-most derivation:



Language definitions

לא מעט תראו את הביטויים LL(0) או LR(k) וכו. מה זה אומר?

- האות הראשונה מייצגת את הצורה בא הקלט נקרא
- האות השנייה מייצגת את סוג ה-derivation
- הקבוע בתוכו מציג את כמות סימני ה-lookahead

לדוגמה, כאשר תראו את LL(k):

- ה-L הראשונה מייצגת שהקלט נקרא משמאל לימין
- ה-L השנייה מייצגת שהשפה מסוג left most derivation
- ה-k מייצג את כמות סימני ה-lookahead

First & Follow

נגדיר את First(a), כאשר a הינו כל מחרוזת של סימנים, להיות הסט של טרמינלים שמתחילים מחרוזות הנגזרות מ-a.

נגדיר את Follow(A), כאשר A הוא non-terminal, להיות הסט של כל הטרמינלים a שיכולים להופיע מידע אחרי הימין של A.

אוטומט LR

Closure of item sets

אם I הוא טס של פריטים בתחביר G, אזי CLOSURE(I) הוא סט של הפריטים הנבנים בעזרת שני חוקים:

1. תוסיף כל פריט ב-I ל-CLOSURE
2. אם $A \to \alpha \cdot B\beta$ נמצא ב CLOSURE, וץ $B \to \gamma$ הוא חוק תחבירי, אזי תוסיף את הפריט $B \to \cdot\ \gamma$ לCLOSURE. תמשיך את החוק הזה עד שאין יותר פריטים להוסיף.

פסאודו קוד מתאים:

```
SetOfItems CLOSURE(I) {
    J = I;
    repeat
        for ( each item A → α·Bβ in J )
            for ( each production B → γ of G )
                if ( B → ·γ is not in J )
                    add B → ·γ to J;
    until no more items are added to J on one round;
    return J;
}
```

Goto

נגדיר את GOTO(I, X), כאשר I הוא סט של פריטים ו-X הוא סימן תחבירי להיות ה-CLOSURE של הסט של כל הפריטים $[A \to \alpha X \cdot \beta]$ כך ש $[A \to \alpha \cdot X\beta]$ נמצא בתוך I. כלומר, זה מגדיר את פונקציית המעבר באוטומת ה-LR(0).

בניית האוסף הקנוני של סטים של פריטים של LR(0)

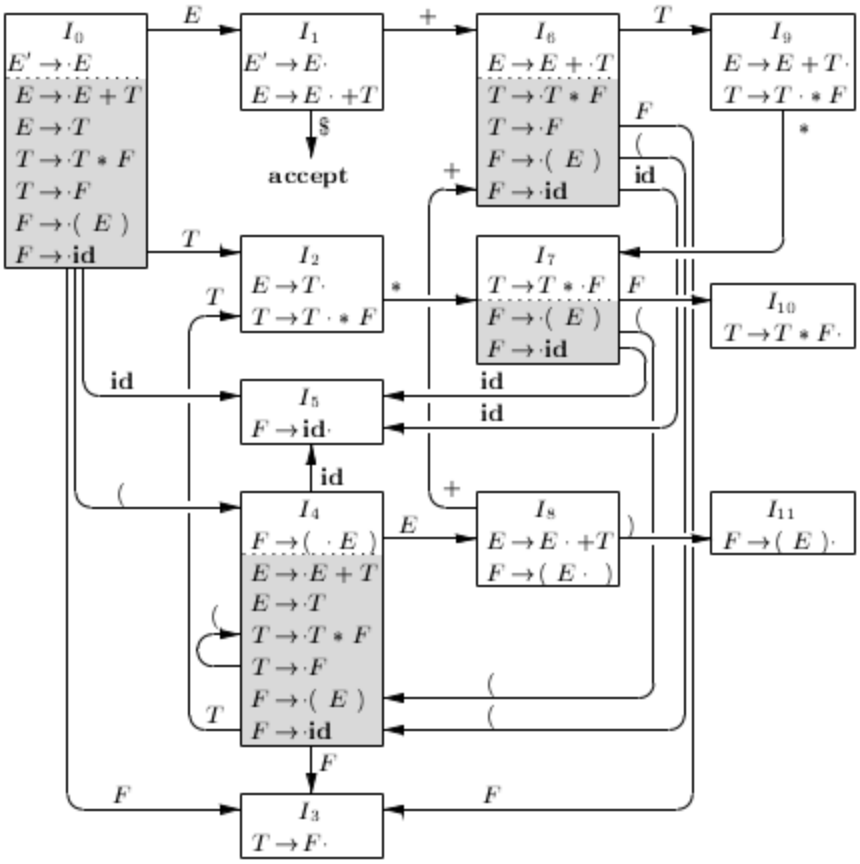
להלן פסאודו קוד המתאר איך לבנות את האוסף הקנוני של סטים של פריטים של LR(0):

1. C = {CLOSURE($[S' \to \cdot S]$)}
2. תחזור על עצמך
    - 2.1. בשביל כל סט פריטים I בתוך C
        - 2.1.1. בשביל כל סימן תחבירי X
            - 2.1.1.1. אם GOTO(X, I) לא ריק וגם לא בתוך C
                - 2.1.1.1.1. תוסיף אותו ל-C
3. עד שאין סט של פריטים שנאסף ל-C באיטרציה

כאשר נבצע את האלגוריתם הבא על התחביר הבא:

$$
\begin{array}{lcl}
E & \to & E + T \mid T \\
T & \to & T * F \mid F \\
F & \to & (E) \mid \mathbf{id}
\end{array}
$$

נקבל את האוטומט הבא:

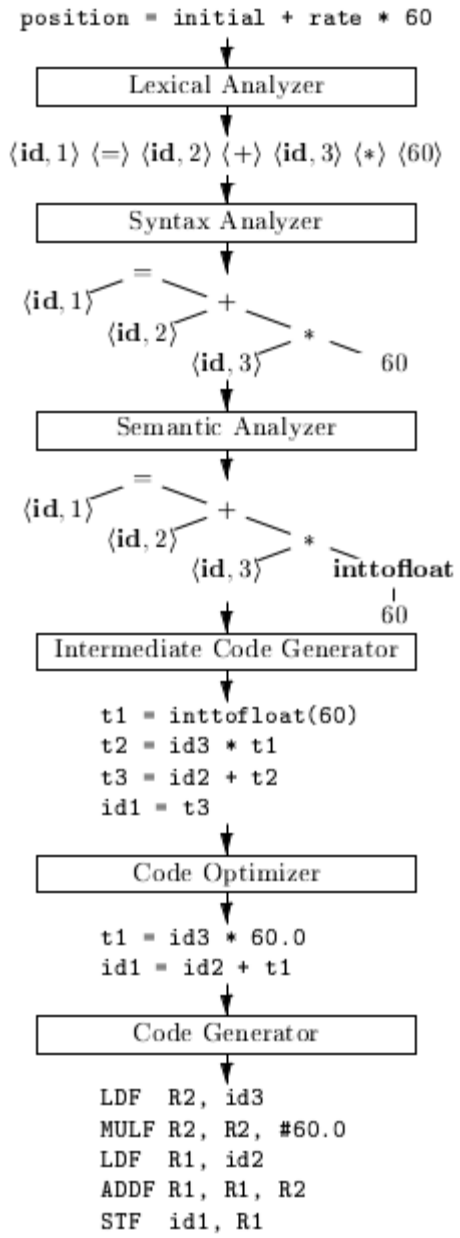


תיאור הבעיה האלגוריתמית

מטרת הפרויקט היא ליצור קומפיילר המתרגם שפת מקור לשפת מטרה, לכן הבעיה האלגוריתמית היא לא תרגום שפת מקור לשפת מטרה, אלה הבעיות האלגוריתמיות העולות בבניית קומפיילר, כלומר, הבעיה בכל שלב בקומפילציה.

לשם כך נפריד בין שלבי הקומפילציה ונתאר את הבעיה העולה בכל שלב. בנוסף נתאר את הבעיות העולות בבנייה מבנים חשובים לקומפיילר, כמו הבעיות העולות בבניית ה-symbol table. בכל שלב יהיה מתואר מה הקלט בכל שלב, מה הפלט בכל שלב, וסקירה של הבעיה העולה בבניית אלגוריתם כזה. לא יתוארו כל הפרטים הקטנים במהלך הקומפילציה, אלה רק התהליכים המרכזיים המופשטים.

התרשים הבא מתאר את שלבי הקומפילציה, מקבלת שפת המקור למנתח המילולי עד הוצאת שפת המטרה. חשוב לדעת שאין דרך אחת נכונה לבצע כל שלב ושלב, אך בתיאור הבעיה אתייחס למבנים ולאלגוריתמים שאני בחרתי בכל שלב ושלב הקומפילציה.



ניתוח מילוני

- **קלט:** מחרוזת של אותיות שהינם התוכנית הכתובה בשפת המקור.
- **פלט:** זרם של אסימונים המתאימים לתוכנית.

הבעיה היא, איך ניצור אלגוריתם שיכול לתרגם רצף של תווים המתאימים לשפת המקור לאסימונים? כלומר, כאשר נראה את התו '+' או את המספר 425, איך נוכל להתאים אותם לאסימון? איך נוכל להבדיל בין רצף התווים "+=" לרצף התווים "=="? ובנוסף, איך נדע איזה ערך אנחנו צריכים לייחס לאסימון?

ניתוח תחבירי

- **קלט:** זרם של אסימונים

- **פלט:** עץ המתאר את התוכנית בצורה הגיונית, מסודרת, ונכונה בצורה תחבירית.

שאלת השאלות היא איך ניצור את העץ התחבירי הזה? בשביל לבנות עץ כזה צריך להגדיר איך אסימונים מתחברים ויוצרים תחביר, כלומר את תחביר השפה. לכן האלגוריתם עוקב אחרי חוקי התחביר של השפה ויוצר מזרם האסימונים עץ תחבירי מתאים. לצורך הגדרת תחביר השפה ניצור מסמך בפורמט BNF המתאר את תחביר השפה.

בעיה גדולה בבניית העץ היא לדעת איך להתמודד עם אסימונים במצבים מסוימים. לדוגמה, איך האלגוריתם יודע מה לעשות כאשר האסימונים האחרון שקיבל הוא identifier (אסימון המתאר שם משתנה) (האסימון האחרון מתאר את "המצב" של המנתח) והאסימון הבא הינו plus (אסימון המתאר את הסימן החשבוני פלוס)? איך האלגוריתם ידע שהמצב הנוכחי שלו הוא מקובל תחבירית, ושאין שגיאה תחבירית בתוכנית?

ניתוח סמנטי

- **קלט:** עץ תחבירי
- **פלט:** עץ סמנטי מופשט

שלב הניתוח הסמנטי בקומפילציה הוא תהליך מורכב שמטרתו לוודא שהקוד המקור תקין מבחינה תחבירית וסמנטית. הבעיה האלגוריתמית העיקרית בשלב הניתוח הסמנטי היא זיהוי ופתרון טעויות סמנטיות.

טעויות סמנטיות הן טעויות שאינן קשורות לתחביר הקוד, אלא למשמעותו. לדוגמה, ייתכן שקיים בקוד משפט המנסה לבצע פעולה על סוג נתונים לא חוקי, או שייתכן שקיים בקוד משפט המשתמש במשתנה שאינו מוגדר. חשוב לזהות ולתקן טעויות סמנטיות בשלב הניתוח הסמנטי, מכיוון שהן עלולות להוביל לתוצאות בלתי צפויות ואף לקריסת התוכנה.

בשביל לפענח את הטעויות בקוד נהוג לסרוק את העץ התחבירי וליצור עץ מופשט יותר, המתאר את התכנית בצורה מופשטת אך שם דגש על משמעות התכנית. כלומר האלגוריתם יכול לעשות דברים כמו להסיר את הצמתים בעץ המתארים סוגריים, מכיוון שהם לא תורמים סמנטית לתכנית ולתרגום שלה.

יצירת קוד ביניים

- **קלט:** עץ סמנטי מופשט
- **פלט:** ייצוג של קוד המקור ברמה נמוכה יותר ע"י מבנה מופשט

שלב בניית קוד הביניים בקומפילציה הוא תהליך שבו הקוד המקור של התוכנה מומר לקוד ביניים. קוד ביניים הוא ייצוג של קוד המקור ברמה נמוכה יותר, אך עדיין בר-הבנה על ידי מכונה.

קוד הביניים יכול להיות מיוצג ע"י הרבה מבנים, וקומפיילרים מודרנים בדר"כ ישתמשו ביותר ממבנה אחד בתהליך הקומפילציה. מטרת קוד הביניים היא להיות נאמן לקוד המקור ולתאר כל דבר שימושי לקוד המקור, אך להיות יעיל לעבודה. בחירה של קוד ביניים מתאים דורש הבנה של שפת המקור ושפת המטרה, לכן כאשר נתרגם את השפה לשפה גבוהה, נרצה להשתמש במבנה במרמה גבוהה יותר ממתי שנתרגם את השפה לשפה נמוכה יותר.

אופטימיזציה

- **קלט:** קוד ביניים
- **פלט:** קוד הביניים רק יעיל יותר

שלב האופטימיזציה בקומפילציה הוא תהליך מורכב שמטרתו לשפר את ביצועי קוד התוכנה. הוא עושה זאת על ידי ביצוע שינויים בקוד הביניים, כגון הסרת קוד מיותר, שינוי סדר הפקודות, או שינוי סוגי נתונים.

מצד אחד, קיימות טכניקות אופטימיזציה רבות שיכולות לשפר משמעותית את ביצועי קוד התוכנה. עם זאת, יישום טכניקות אלו עלול להוביל לשינויים משמעותיים בקוד, מה שעלול לגרום לבעיות בתפקוד התוכנה. בנוסף, אופן האופטימיזציה והסוגים השונים של אופטימיזציה תלויים בייצוג קוד הביניים, לכן קומפיילרים לפעמים משתמשים ביותר מייצוג קוד ביניים אחד.

יצירת קוד

- **קלט:** קוד ביניים
- **פלט:** קוד המטרה

שלב יצירת הקוד בקומפילציה הוא תהליך שבו קוד הביניים מומר לקוד המטרה. השלב הזה מתרגם את קוד הביניים לקוד המטרה, ויהיה מתאים למבנים שונים בקוד הביניים.

אם שפת המטרה היא שפת מכונה, רגיסטרים וזיכרון צריכים להיות מתאימים להוראות בשפה. ואז קוד הביניים מתורגם למחרוזת הוראות המתארות את אותו אלגוריתם שהיה מתואר בקוד המקור.

טבלת סימנים

טבלת הנתונים שומרת מסמכים של שמות נתונים ואוספת תכונות על אותם משתנים. לכן הבעיה היא איך המידע והנתונים נאספים ונשמרים. טבלת הסימנים אמורה להיות בנויה כך שהיא מתאימה לנתונים הנאספים ושהמידע יהיה נאסף בצורה מהירה.

התכונות של המשתנים יכולים להיות:

- סוג המשתנה
- איפה נשמר בזיכרון
- ה-scope של משתנים

לכן חלק מהבעיה היא איך נאסף המידע הזה.

התמודדות עם שגיאות

בכל תהליך הקומפילציה צריך להתמודד עם שגיאות, לכן קיים מתמודד השגיאות (error handler). מתמודד השגיאות הוא רכיב בקומפילטור שאחראי לטיפול בשגיאות במהלך הקומפילציה. הבעיה היא להתמודד עם שגיאות בצורה יעילה.

סקירת אלגוריתמים בתחום הבעיה

כל אלגוריתם מתעסק בשלב ספציפי בתהליך הקומפילציה, לכן אחלק את הנושא הזה לכל שלב והאלגוריתמים השונים שלו.

ניתוח מילוני

מכיוון שהמטרה העיקרית של המנתח המילוני הינה ליצור זרם אסימונים המתאים לתכנית, אנחנו צריכים טכניקה לבחון זרם תווים.

conditionals

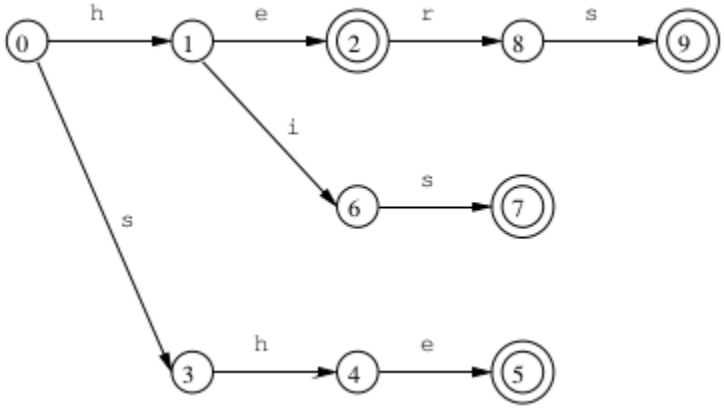
האלגוריתם הברור מאליו הוא switch גדול, הבוחן כל מחרוזת עם כל אפשרות אחת אחרי השנייה עד הוא מגיע לתשובה:

1. אם מחרוזת הינה "="
2. אם מחרוזת הינה "=="
3. אם מחרוזת הינה ">"
4. ...

זמן הריצה יהיה O(n*m), כאשר n שווה לגודל המחרוזת, ו-m שווה למספר האפשרויות. אנחנו יכולים לעשות טוב יותר.

automata

אלגוריתם המשתמש באוטומט בשביל למצוא סוג אסימון מתאים למילה. אלגוריתם כזה לא מתעסק בלבדוק כל סוג וסוג, אלה לאיזה מצב כל תו במחרוזת מעבירה אותנו.
להלן אוטומט המקבל את המרוזות *he, she, his, hers*:



זמן הריצה של אלגוריתם כזה הוא O(n), כאשר n הינה גודל המחרוזת. אנחנו יכולים להניח שגודל המחרוזת הוא קבוע, לכן זמן הריצה יהיה קבוע.

ניתוח תחבירי

כדי ליצור Parse Tree קיימים כמה אלגוריתמים הנקראים Parsing Algorithms.
אלגוריתמים אלה מתחלקים בעיקר לשני סוגים:

- Top Down Parsing
- Bottom Up Parsing

שפות תכנות הן בדר"כ חסרות הקשר, שהן שפות שאפשר לנתח באמצעות אוטומט מחסנית. לכן, נשתמש באוטומט מחסנית בשביל לנתח ולעבד את השפה.

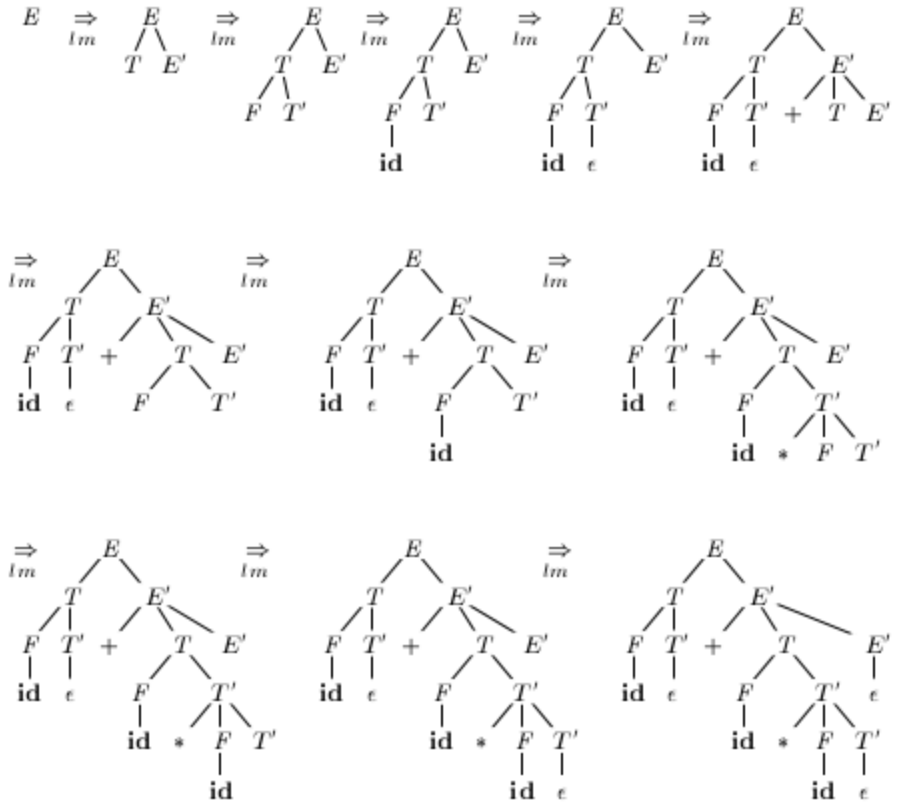
Top Down Parsing

פרסור למעלה-למעטה היא הטכניקה בשביל לבנות עץ תחבירי מזרם של סימנים, המתחיל משורש העץ לצמתים האחרים, כאשר מגיעים לצמתים בצורה preorder. או במילים אחרות, למצוא את ה-left modt derivation לזרם הסימנים.

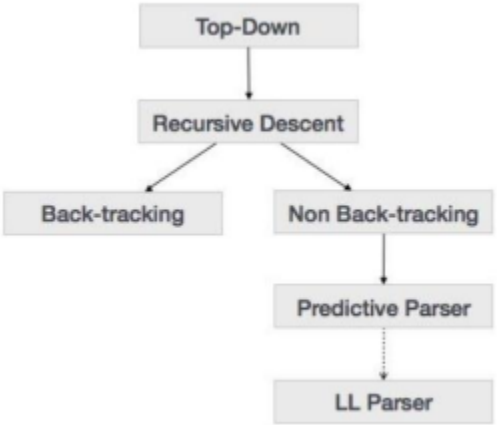
בכל מצב בסוג פרסור, הבעיה העיקרית היא לבחור איזה חוק תחבירי מתאים ל non-terminal. כאשר החוק נבחר, כל שאר התהליך הוא להתאים את הטרמינלים לגוף חוק.

להלן דוגמה ל-Top Down Parsing בשביל המחרוזת **id+id*id** והתחביר:

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$



כל תת-סוגי הפרסור:



Recursive-Descent Parsing

טכניקת פרסור הבנויה מסט של פרוצדורות, אחת בשביל כל טרמינל.

התכנית מתחילה עם הרצה של פרוצדורה מתאימה לסימן ההתחלה, ומסתיימת כאשר כל זרם הסימנים נקרא בהצלחה. להלן פסאודו קוד לסוג הפרסור:

```
void A() {
        Choose an A-production, A → X1 X2 ··· Xk;
        for ( i = 1 to k ) {
                if ( Xi is a nonterminal )
                        call procedure Xi();
                else if ( Xi equals the current input symbol a )
                        advance the input to the next symbol;
                else /* an error has occurred */;
        }
}
```

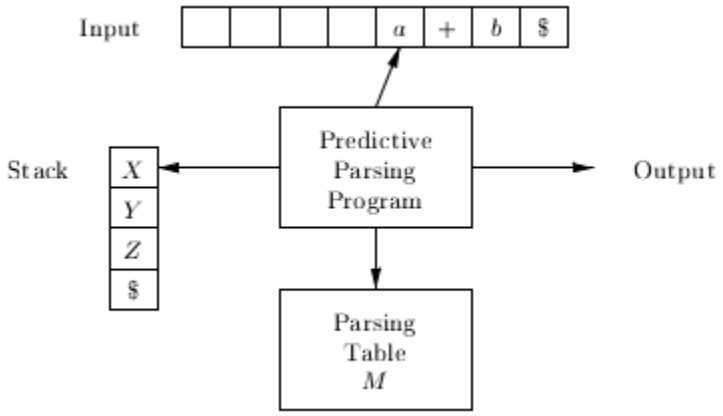
לפעמים, backtracking מוצרך בשביל recursive-descent. בגלל זאת, זמן הריצה של התכנית הינו אקספוננציאלי, O(2^n).

Predictive Parsers

פרסרים מסוג recursive descent שיכולים לחזות איזה חוק תחבירי צריך להשתמש בו בשביל להחליף את הקלט, לכן backtracking הוא לא בעיה יותר.

בשביל לעשות זאת, הפרסר "מציץ" לסמלים הבאים בקלט. הפעולה הזאת מגיעה עם בעיה, התחבירים המתאימים לפרסור כזה הם כל התחבירים בקבוצה LL(k > 0) ומעלה. בשביל לממש את סוג הפרסור הזה, התכנית משתמשת במחסנית וטבלת פרסור מתאימה בשביל לקבל החלטות.

להלן מבנה התכנית:



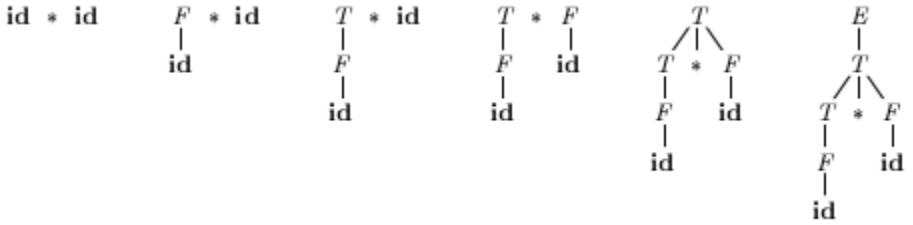
הרבה סוגים שונים של פרסרים משתמשים במבנה דומה בשביל לממש את טכניקת הפרסור שלהם.

להלן פסאודו קוד:

```
let a be the first symbol of w;
let X be the top stack symbol;
while ( X ≠ $ ) { /* stack is not empty */
        if ( X = a ) pop the stack and let a be the next symbol of w;
        else if ( X is a terminal ) error();
        else if ( M[X, a] is an error entry ) error();
        else if ( M[X, a] = X → Y1 Y2 ··· Yk ) {
                output the production X → Y1 Y2 ··· Yk;
                pop the stack;
                push Yk, Yk−1, ... , Y1 onto the stack, with Y1 on top;
        }
        let X be the top stack symbol;
}
```

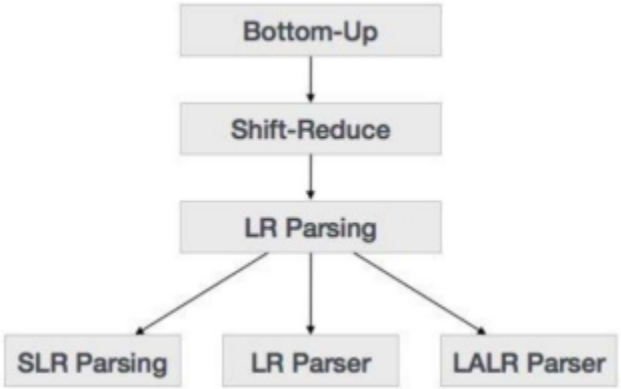
Bottom Up Parsing

פרסור למעלה-למטה היא הטכניקה בשביל לבנות עץ תחבירי מזרם של סימנים, המתחיל מעלי העץ לשורש העץ,. או במילים אחרות, למצוא את ה-rightmost derivation לזרם הסימנים. לדוגמה, בשביל המחרוזת * **id** **id**, נבנה העץ התחבירי הבא:



התהליך הזה משתמש ברדוקציה, שזה פשוט התהליך ההפוך של נגזרת.

כל תת-סוגי הפרסור:



כל זמני הריצה של אלגוריתמי הפרסור מסוג זה הם O(n)

Shift-Reduce Parsing

סוג פרסור שבו מחסנית שומרת את סימני התחביר. בסוג הפרסור הזה יש ארבעה סוגים של פעולות עיקריות:

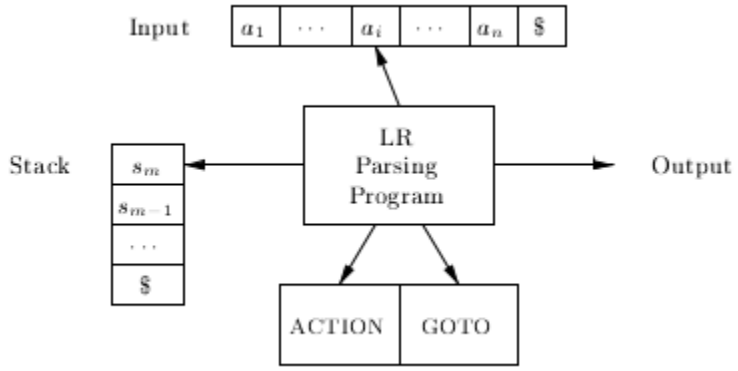
shift - דחוף את הסימן הבא בזרם למחסנית
reduce - תוציא את המחרוזת המתאימה בסטאק ותחליף אותה ב non-terminal המתאים.
accept - תגדיר שהפרסור הסתיים בהצלחה
error - תמצא שגיאה תחבירית מתאימה ותחזיר תגובה לשגיאה

להלן הפעולות המתאימות לדוגמא מלמעלה

STACK	INPUT	ACTION
$	$\mathbf{id}_1 * \mathbf{id}_2$ $	shift
$ $\mathbf{id}_1$	$* \mathbf{id}_2$ $	reduce by $F \to \mathbf{id}$
$ F	$* \mathbf{id}_2$ $	reduce by $T \to F$
$ T	$* \mathbf{id}_2$ $	shift
$ $T *$	$\mathbf{id}_2$ $	shift
$ $T * \mathbf{id}_2$	$	reduce by $F \to \mathbf{id}$
$ $T * F$	$	reduce by $T \to T * F$
$ T	$	reduce by $E \to T$
$ E	$	accept

הערה: מאתחלים את המחסנית עם הסימן '$', המגדיר EOF.

מבנה התכנית של פרסר LR:



SLR

שיטת פרסור LR המשתמשת במבנה פשוט של טכניקת הפרסור. להלן ההוראות לבנות את טבלאות ה-action ו-goto:

1. The ACTION function takes as arguments a state i and a terminal a (or \$, the input endmarker). The value of ACTION$[i, a]$ can have one of four forms:

   (a) Shift j, where j is a state. The action taken by the parser effectively shifts input a to the stack, but uses state j to represent a.

   (b) Reduce $A \rightarrow \beta$. The action of the parser effectively reduces β on the top of the stack to head A.

   (c) Accept. The parser accepts the input and finishes parsing.

   (d) Error. The parser discovers an error in its input and takes some corrective action. We shall have more to say about how such error-recovery routines work in Sections 4.8.3 and 4.9.4.

2. We extend the GOTO function, defined on sets of items, to states: if GOTO$[I_i, A] = I_j$, then GOTO also maps a state i and a nonterminal A to state j.

LR

שיטת פרסור יותר מתוחכמת המתשתמש בסימני lookahead בשביל להרחיב את כמות השפות שהיא יכולה לפרסר.

LALR

טכניקה המפחיתה את כמות המצבים בטבלת הפרסור.

האלגוריתם הנבחר לפתרון

במהלך הפרק אציין לגבי כל שלב בקומפילציה באיזה מבני נתונים ובאיזה אלגוריתמים השתמשתי בשביל לתרגם את קוד המקור לקוד המטרה. בשביל כל בחירה אציין בנוסף למה בחרתי בא, עם התייחסות למשתנים כמו זמן ריצה, יעילות, פשטות, שימוש חוזר ועוד...

חשוב לדעת שאני לא אציין שני שלבים, יצירת קוד הביניים ואופטימיזציה. לא מוסבר על קוד ביניים מכיוון שבחרתי להשתמש בAST כמבנה קוד הביניים שלי, שמובנה כבר בניתוח הסמנטי. לא הסברתי על אופטימיזציה מכיוון שאין אופטימיזציה לקוד. למרות זאת, קוד המקור מתורגם והתרגום הוא בזמן ריצה O(n).

ניתוח מילוני (lexical analysis)

בשלב זה, המנתח מקבל את התכנית כמחרוזת, והוא צריך להוציא זרם של אסימונים המתאימים למילון השפה.

מכונת המצבים

בשלב הניתוח המילוני החלטתי להשתמש באוטומט סופי דטרמיניסטי (DFA) בשביל לתאם lexemes לאסימונים מתאימים. השכבה הזאת מנתחת את שפה רגולרית, לכן אוטומט סופי מתאים לשלב זה.

כפי שהוסבר בחלק מבני הנתונים (כאן), התאמה של מילה למצב מקבל וכך לסוג אסימון לוקחת זמן ריצה O(1).

להלן כל מילה שיכולה להיות קיימת בשפה, בפורמט הבא:
(<ערך סוג האסימון>) <מספר מצב>:(<מילה/סוג>)

```
(null): 0 (1)
(TOK_IDENTIFIER): 1 (4)
(TOK_NUMBER_CONSTANT): 2 (5)
(CHAR): 3 (11)
(CHAR_DENY): 4 (1)
(STRING): 5 (7)
(TOK_UNKNOWN): 6 (1)
b: 7 (4)
bo: 8 (4)
boo: 9 (4)
bool: 10 (8)
br: 11 (4)
bre: 12 (4)
brea: 13 (4)
break: 14 (9)
c: 15 (4)
ca: 16 (4)
cas: 17 (4)
case: 18 (10)
ch: 19 (4)
cha: 20 (4)
char: 21 (11)
co: 22 (4)
con: 23 (4)
cons: 24 (4)
const: 25 (12)
cont: 26 (4)
conti: 27 (4)
contin: 28 (4)
continu: 29 (4)
continue: 30 (13)
d: 31 (4)
do: 32 (14)
```

```
dou: 33 (4)
doub: 34 (4)
doubl: 35 (4)
double: 36 (15)
e: 37 (4)
el: 38 (4)
els: 39 (4)
else: 40 (16)
f: 41 (4)
fa: 42 (4)
fal: 43 (4)
fals: 44 (4)
false: 45 (17)
fl: 46 (4)
flo: 47 (4)
floa: 48 (4)
float: 49 (18)
fo: 50 (4)
for: 51 (19)
i: 52 (4)
if: 53 (20)
in: 54 (4)
int: 55 (21)
r: 56 (4)
re: 57 (4)
ret: 58 (22)
s: 59 (4)
sh: 60 (4)
sho: 61 (4)
shor: 62 (4)
short: 63 (23)
si: 64 (4)
sig: 65 (4)
sign: 66 (4)
signe: 67 (4)
signed: 68 (24)
sw: 69 (4)
swi: 70 (4)
swit: 71 (4)
switc: 72 (4)
switch: 73 (25)
t: 74 (4)
tr: 75 (4)
```

```
tru: 76 (4)
true: 77 (26)
ty: 78 (4)
typ: 79 (4)
type: 80 (4)
typed: 81 (4)
typede: 82 (4)
typedef: 83 (27)
u: 84 (4)
un: 85 (4)
uns: 86 (4)
unsi: 87 (4)
unsig: 88 (4)
unsign: 89 (4)
unsigne: 90 (4)
unsigned: 91 (28)
v: 92 (4)
vo: 93 (4)
voi: 94 (4)
void: 95 (29)
w: 96 (4)
wh: 97 (4)
whi: 98 (4)
whil: 99 (4)
while: 100 (30)
p: 101 (4)
pr: 102 (4)
pri: 103 (4)
prin: 104 (4)
print: 105 (31)
.: 106 (46)
,: 107 (44)
~: 108 (126)
!: 109 (33)
&: 110 (38)
|: 111 (124)
^: 112 (94)
*: 113 (42)
/: 114 (47)
+: 115 (43)
-: 116 (45)
%: 117 (37)
>: 118 (62)
```

```
<: 119 (60)
=: 120 (61)
..: 121 (1)
...: 122 (62)
->: 123 (63)
&&: 124 (64)
||: 125 (65)
^^: 126 (66)
<<: 127 (67)
>>: 128 (68)
==: 129 (69)
>=: 130 (70)
<=: 131 (71)
+=: 132 (72)
-=: 133 (73)
*=: 134 (74)
/=: 135 (75)
%=: 136 (76)
&=: 137 (77)
|=: 138 (78)
^=: 139 (79)
!=: 140 (80)
~=: 141 (81)
<<=: 142 (82)
>>=: 143 (83)
(: 144 (40)
): 145 (41)
[: 146 (91)
]: 147 (93)
{: 148 (123)
}: 149 (125)
:: 150 (58)
;: 151 (59)
```

פסאודו קוד המתאר איך lexeme מותאם לאסימון:

1. תתחיל ב-q0
2. כל עוד לא הגעתה למצב מקבל ומצב 1-
   2.1. תסתקל על הערך במצב הנוכחי ובעמודה המתאימה לסימן
   2.2. תעבור למצב המתאים לערך
3. החזר אסימון מתאים למצב

קבלת כל אסימון והעברתו למנתח התחבירי

זרם האסימונים שמקבל המנתח התחבירי יהיה במבנה הנתונים תור. בחרתי להשתמש בתור מכמה סיבות:

- הכנסה לתור היא O(1)
- תור אשמור על סדר האסימונים

פסאודו קוד המתאר את יצירת זרם האסימונים

1. אתחל אסימון בעל ערך NULL
2. כל עוד האסימון לא מסוג EOF
    2.1. תקרא לפונקציית קבלת האסימון הבא ותכניס את הערך המוחזר לאסימון
    2.2. אם האסימון לא ידוע
        2.2.1. תזרוק שגיאה ותפסיק את התוכנית
    2.3. תשמור את האסימון בתור

ניתוח תחביר (syntax analysis/parsing)

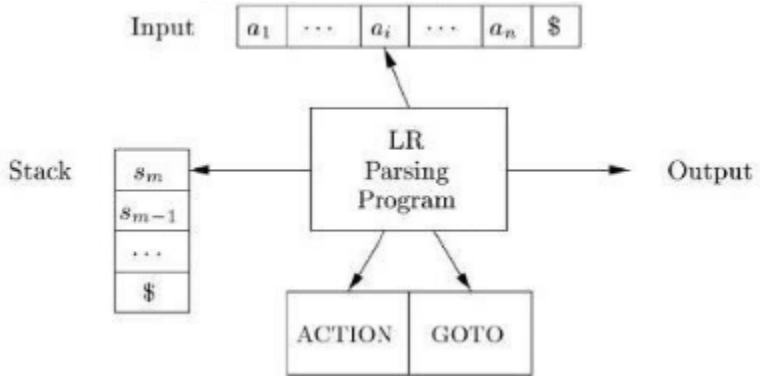
שלב זה מקבל זרם של אסימונים והמנתח המילוני ומוציא עץ תחבירי.

מכונת המצבים

בשלב הניתוח התחבירי החלטתי להשתמש באוטומט מחסנית בשביל להתאים אסימונים לnon-terminal בעזרת חוקים תחביריים. השכבה הזאת מנתחת את שפה חסרת הקשר (CFG), לכן אוטומט מחסנית מתאים לשלב זה. אלגוריתם הפרסור הוא אלגוריתם LR.

אלגוריתם הניתוח התחבירי מורכב מכמה חלקים:

- טבלת הפרסור, המחולקת לשני טבלאות
    - טבלת אקשן (action table)
    - טבלת גוטו (goto table)
- מחסנית
- זרם אסימונים



Action table

טבלת האקשן מתארת פונקציות המעבר שהאוטומט עושה בפורמט הבא:

<action><modifier (if any)>

כל שורה מייצג מצב וכל עמודה מייצג טרמינל מתאים. יש ארבעה סוגים שונים של אקשנים:

- Shift
- Reduce
- Accept
- Error

Goto table

מציין למנתח לאיזה מצב הוא צריך לעבור לאחר ביצוע Reduce.

פסאודו קוד לפרסור LR

1. תן ל-a להיות הטרמינל הראשון, ול-s להיות ראש הסטאק תמיד
2. כל עוד action_table[s, a] לא accept
    - 2.1. אם action_table[s, a] הינו shift t
        - 2.1.1. תדחוף t לסטאק
        - 2.1.2. תן ל-a להיות הטרמינל הבא
    - 2.2. אם action_table[s, a] הינו reduce A -> b
        - 2.2.1. תוציא $|b|$ מהמחסנית
        - 2.2.2. תן ל-t להיות ראש המחסנית
    - 2.3. אם האקשן הינו accept
        - 2.3.1. כלום
    - 2.4. תזרוק שגיאה ותעצור את התכנית

Parser stack

מחסנית הפרסר תשמור בתוכה שלמים המייצגים את המצב הנוכחי. המצב הראשון יהיה '$' המתאר EOF.
אפשר לראות שהאקשן accept מופיע אך ורק פעם אחת בעמודה המתאימה לטרמינל $.

עץ תחבירי

בנוסף לבדיקה של תקינות התוכנית, נרצה לבנות עץ ניתוח שייצג את פעולתה.

בנייתה עץ תעשה תוך כדי תהליך הניתוח, כך שכל פעם שנבצע Reduce נוסיף לעץ את הNon-terminal מצד שמאל של החוק ונקבע שהבנים שלו יהיו כל הTerminalים והnon-terminalים שמימין של חוק הדקדוק, כך לאט-לאט נבנה העץ התחבירי.

תחביר השפה

להלן ה-BNF של השפה:

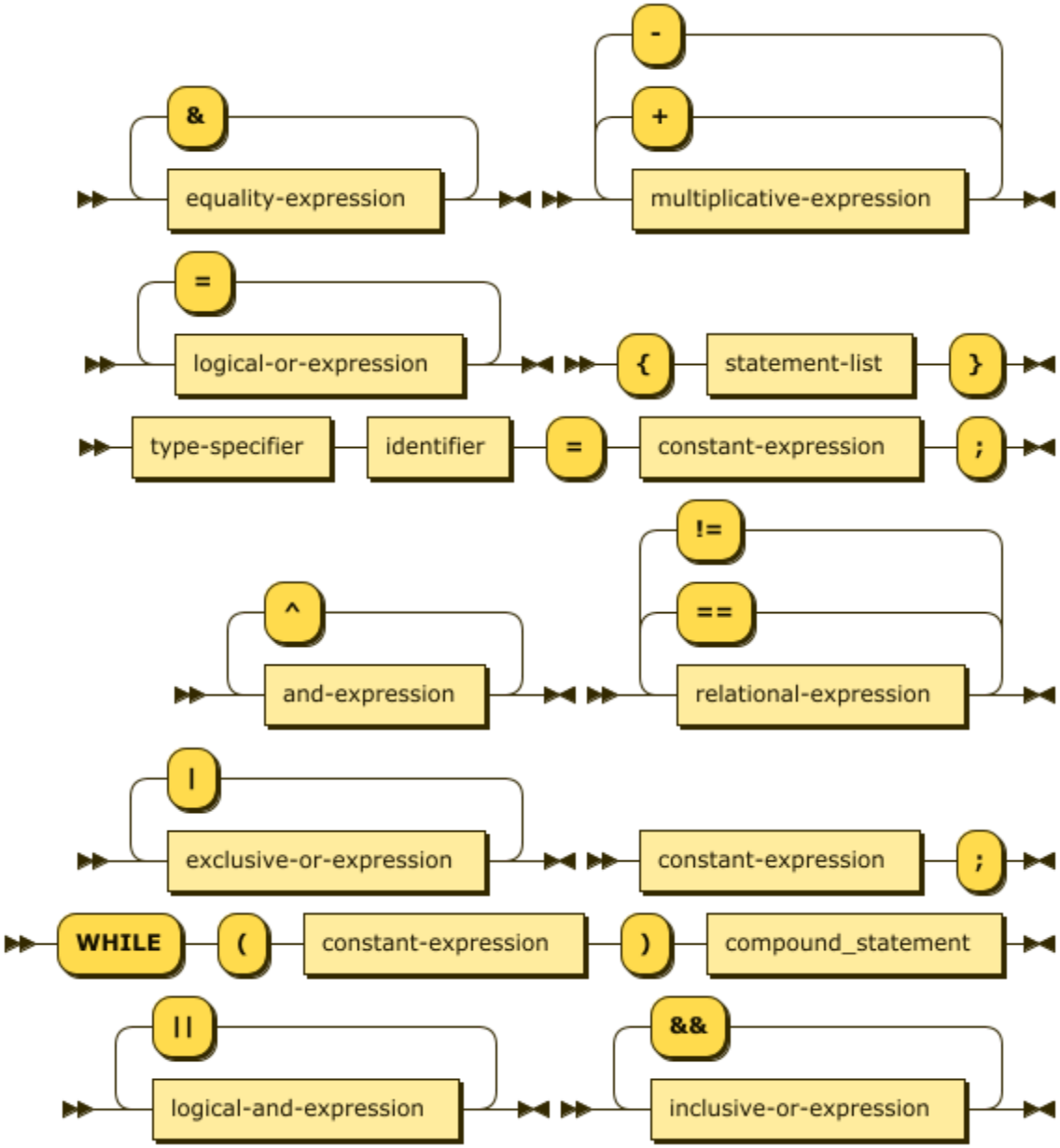
```
program ::= statement-list
```

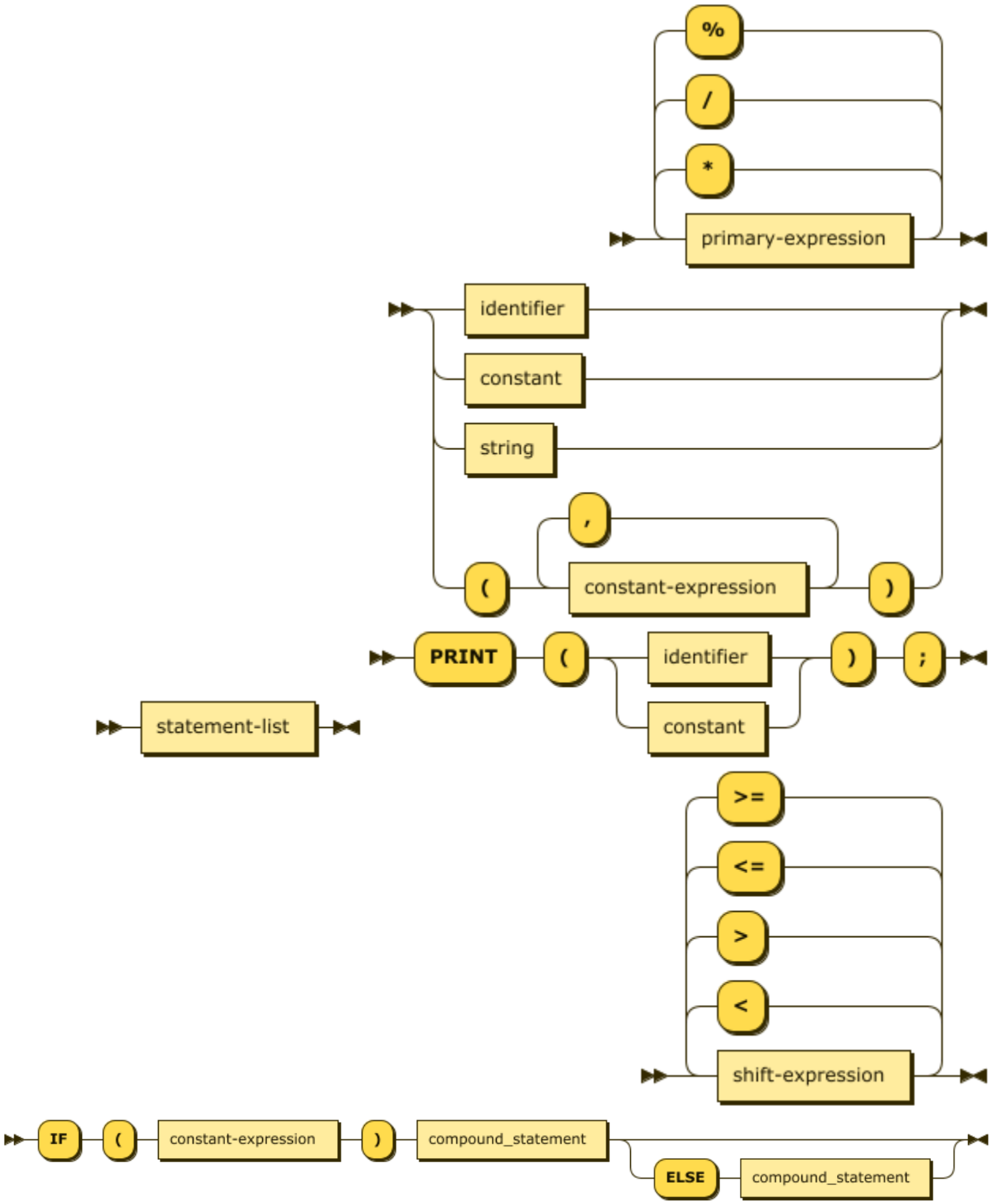
```
statement-list ::= statement
                  | statement statement-list

statement ::= expression-statement
             | compound-statement
             | selection-statement
             | iteration-statement
            | print-statement
             | declaration

declaration ::= type-specifier identifier '=' constant-expression ';'

type-specifier ::= 'CHAR'
                    | 'INT'

expression-statement ::= constant-expression ';'

constant-expression ::= assignment-expression

assignment-expression ::= logical-or-expression
                        | assignment-expression assignment-operator
logical-or-expression

logical-or-expression ::= logical-and-expression
                        | logical-or-expression '||' logical-and-expression

logical-and-expression ::= inclusive-or-expression
                        | logical-and-expression '&&'
inclusive-or-expression

inclusive-or-expression ::= exclusive-or-expression
                              | inclusive-or-expression '|'
exclusive-or-expression

exclusive-or-expression ::= and-expression
                              | exclusive-or-expression '^' and-expression

and-expression ::= equality-expression
                  | and-expression '&' equality-expression

equality-expression ::= relational-expression
                        | equality-expression '==' relational-expression
```

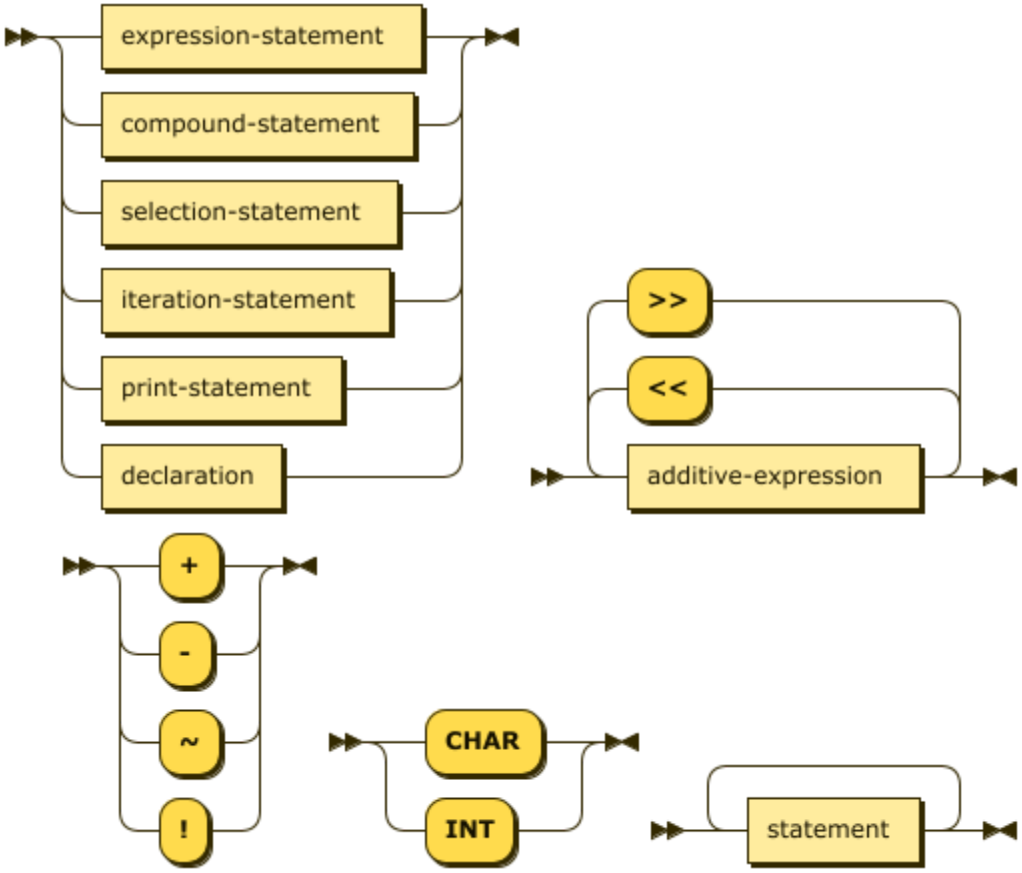
```
                        | equality-expression '!=' relational-expression

relational-expression ::= shift-expression
                        | relational-expression '<' shift-expression
                        | relational-expression '>' shift-expression
                        | relational-expression '<=' shift-expression
                        | relational-expression '>=' shift-expression

shift-expression ::= additive-expression
                   | shift-expression '<<' additive-expression
                   | shift-expression '>>' additive-expression

additive-expression ::= multiplicative-expression
                        | additive-expression '+' multiplicative-expression
                        | additive-expression '-' multiplicative-expression

multiplicative-expression ::= primary-expression
                              | multiplicative-expression '*'
primary-expression
                              | multiplicative-expression '/'
primary-expression
                              | multiplicative-expression '%'
primary-expression

primary-expression ::= identifier
                        | constant
                        | string
                        | '(' expression ')'

expression ::= constant-expression
            | expression ',' constant-expression

assignment-operator ::= '='

unary-operator ::=   '+'
                   | '-'
                   | '~'
                   | '!'

compound-statement ::= '{' statement-list '}'

selection-statement ::= 'IF' '(' constant-expression ')' compound_statement
                              | 'IF' '(' constant-expression ')'
```

```
compound_statement 'ELSE' compound_statement

iteration-statement ::= 'WHILE' '(' constant-expression ')'
compound_statement

print-statement ::= 'PRINT' '(' identifier ')' ';'
                  | 'PRINT' '(' constant ')' ';'
```

בנוסף railroad diagram של התחביר:




%
/
*
primary-expression
identifier
constant
string
,
(
constant-expression
)
PRINT
(
identifier
constant
)
;
statement-list
>=
<=
>
<
shift-expression
IF
(
constant-expression
)
compound_statement
ELSE
compound_statement



כל הטבלאות התוארו יהיו בנספח

ניתוח סמנטי (semantic analysis)

שלב זה מקבל עץ עץ תחבירי ומתרגם אותו לעץ סמנטי מתאים. החלטתי לעשות זאת ע"י sdt המחבר טכניקת תרגום לכל חוק תחבירי, וכך אפשר לעבור על העץ ולתרגם אותו לעיל. בנוסף, החלטתי לממש נתונים לטבלת הסימנים בשלב זה.

טכניקת תרגום

טכניקת התרגום אחראית על לתרגם את העץ התחבירי לעץ הסמנטי. זה נעשה ע"י תיאום של חוק תחבירי לפונקציה סמנטית. כמובן שכל פעולה סמנטית תהיה פונקציה בקומפיילר.

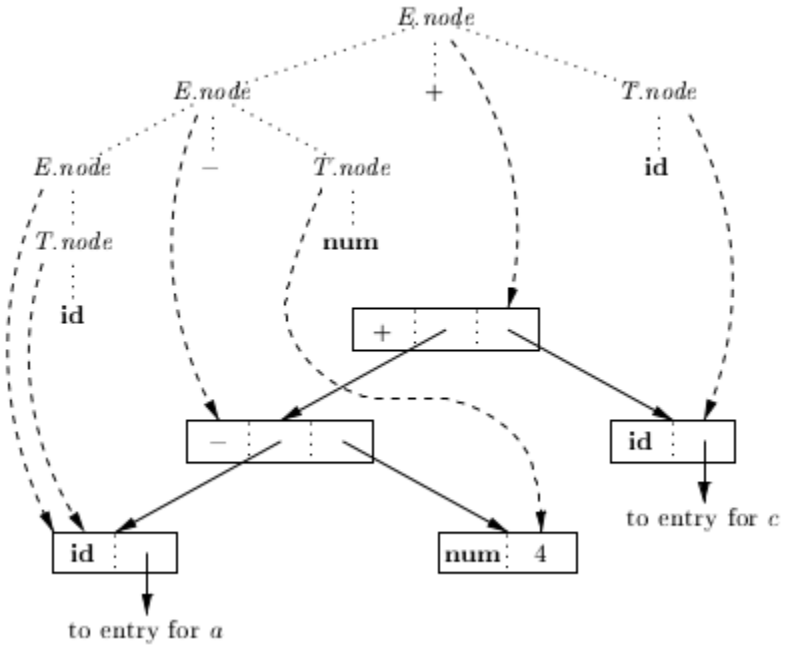
בשביל לתרגם צומת בזמן הריצה היעיל ביותר, החלטתי להשתמש במערך פונקציות בשביל לתרגם כך צומת. במהלך הניתוח התחבירי, כל צומת קיבלה מפתח חד-חד ערכי המתאר את החוק התחבירי המיוחד שממנו נוצרה. כך, אפשר להתאים כל צומת לפונקציית התרגום המתאימה שלה, בזמן O(1).

כל פונקציית תרגום משתמשת בשני רכיבים עיקריים בשביל תרגום:

- צומת בעץ התחבירי
- מחסנית בא נשמרים סימני העץ התחבירי

להלן פסאודו קוד המתאר את טכניקת התרגום, המקבלת עץ התחבירי ומחסנית:

תגדיר תרגום-עץ-תחבירי(עץ תחבירי, מחסנית):
1. אם צומת העץ ריקה
   1.1. תחזור (return)
2. אם צומת העץ טרמינל
   2.1. תדחוף למחסנית את הצומת
   2.2. תחזור
3. בשביל כל ילד i בצומת העץ
   3.1. תרגום-עץ-תחבירי(ילד במקום i של צומת העץ, מחסנית)
4. אם קיים פונקציית תרגום לצומת העץ
   4.1. תקיים את פונקציית התרגום המתאימה



יצירת קוד (code generation)

שלב זה מקבל עץ סמנטי ומתרגם אותו לשפת סף. ספציפית, ל-nasm x64. בשלב זה הקומפיילר צריך לדעת איך לתרגום נכון את העץ ואיך להתאים רגיסטרים להוראות.

תהליך התרגום דומה לתרגום הניתוח הסמנטי, כאשר אנחנו עוברים על העץ ומתאימים צמתים לפונקציות תרגום. בחרתי בטכניקה הזאת עוד פעם מכיוון שהיא פשוטה ויעילה, עם זמן כולל של כל התרגום של O(n).

אלוקציית רגיסטרים

אלוקציית הרגיסטרים נעשת בעזרת שני מבנים:
- רגיסטר - מגדיר רגיסטר וכל תכונתו

- בריכת רגיסטרים - מבנה השומר את המידע על כל רגיסטר בתכנית.

בזמן התרגום, פונקציות התרגום משתמשות בבריכת הרגיסטרים בשביל לראות איזה רגיסטרים פנויים, מה הערך בכל רגיסטר וכו...

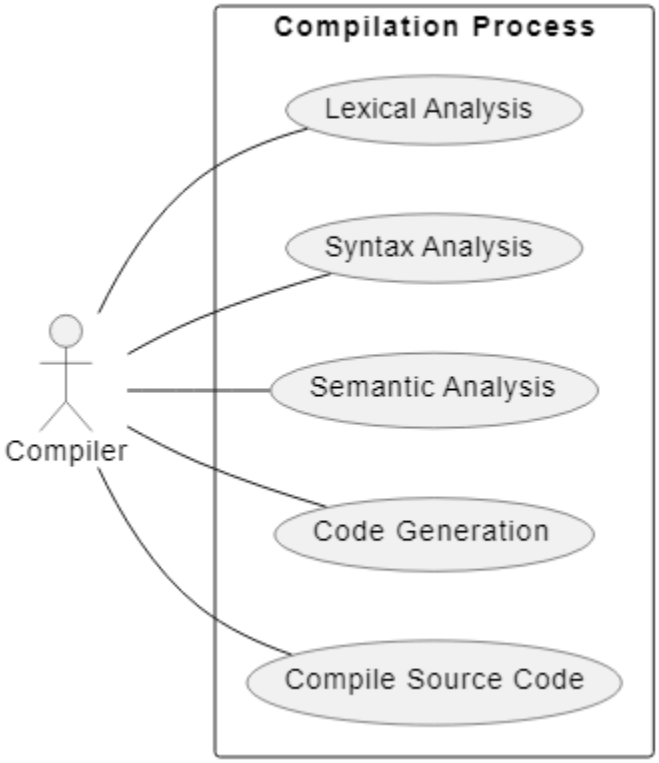
לקבלת רגיסטרים יש כמה פונציות עזר:

- Get_reg - מחזיר רגיסטר פנוי
- Reg_free - משחרר רגיסטר

טכניקת תרגום

לכל צומת מותאם פונקציית תרגום מתאימה, שהינה מחזירה מחרוזת מתאימה לקוד המטרה. המתרגם יעבור על העץ, ואבדוק בשביל כל צומת עם יש לה פונקציית תרגום מתאימה.

תרשים מקרי שימוש



מבנה נתונים

set

מבנה הנתונים המופשט *סט* הינו אוסף השומר נתונים שונים אחד מהשני, בדיוק כמו סט במתמטיקה. תכונות המפתח של סט הינו:

- **ייחודיות** - כל האלמנטים בסט הם ייחודיים בלי שום הכפלות.
- **אי-סידוריות** - לאיברים בקבוצה אין סדר ספציפי.

המימוש של הסט בקוד נוצרה בעזרת שני מבנים:

- Set - הסט עצמו
  - מצביע לצומת ראש
  - מצביע לפונקציית ההשוואה
  - גודל הסט
- Set node - צומת המגדיר איבר בסט, יוצר רשימה מקושרת
  - ערך גנרי
  - מצביע לצומת הבאה ברשימה

הקוד:

```
typedef struct GENERIC_SET_NODE_STRUCT {
   void *data;                            // Pointer to the element data
(can hold any type)
   struct GENERIC_SET_NODE_STRUCT *next;  // Pointer to the next node in
the set
} set_node_T;

// Define the set structure
typedef struct GENERIC_SET_STRUCT {
   set_node_T *head;                          // Pointer to the head node
of the set
   int (*compare)(const void*, const void*);   // Pointer to a comparison
function
   size_t size;                               // Size of the set (number
of elements)
} set_T;
```

hashset

אימפלמנטציה של סט בעזרת hash table (טבלת גיבוב).

טבלת גיבוב, הידועה גם כטבלת ערבול, היא מבנה נתונים מיוחד במדעי המחשב שמאפשר אחזור וניהול של נתונים בצורה יעילה. טבלת הגיבוב מורכבת משני רכיבים עיקריים:

- **אחסון**: רכיב המכיל את הנתונים.
- **פונקציית גיבוב**: פונקציה זו מקבלת מפתח (לדוגמה, שם) ומחזירה אינדקס במערך. האינדקס הזה מצביע על התא שבו מאוחסן ערך המפתח (לדוגמה, מספר טלפון).

טבלת הגיבוב מאפשרת גישה מהירה לנתונים. הסיבה לכך היא שפונקציית הגיבוב מחשבת ישירות את מיקום הנתונים במערך, ללא צורך בחיפוש סדרתי.

המימוש המיוחד של סט בעזרת טבלת גיבוב מחייבת בנוסף גם פונקציית השוואה בין איברים.

הקוד:

```
typedef struct HASHSET_NODE_STRUCT {
   void *data;                          /* The data stored in the node */
   struct HASHSET_NODE_STRUCT *next;  /* Pointer to the next node in the
bucket */
} hashset_node_T;
typedef struct HASHSET_STRUCT {
   int size;                /* Number of elements in the set */
   int capacity;            /* Maximum number of elements in the set */
   float load_factor;       /* Maximum ratio of number of elements to number
of buckets */
   hashset_node_T **buckets; /* Array of bucket pointers */
   unsigned int (*hash_func)(void *); /* Pointer to a hash function to hash
data */
   int (*compare_func)(void *, void *); /* Pointer to a comparison function
to compare data */
} hashset_T;
```

queue

תור (Queue) הוא מבנה נתונים מופשט שמתפקד כמו תור המתנה. בדומה לתור פיזי של אנשים, בתור נתונים האיבר הראשון שנכנס לתור הוא גם הראשון שיוצא ממנו. עיקרון זה ידוע בשם "נכנס ראשון יוצא ראשון" (FIFO).

תכונות מפתח:
First in first out (FIFO): האיבר שנכנס ראשון לתור יצא ראשון מהתור, האיבר שנכנס שני יצא שני...

פעולות עיקריות:

- **הכנסה (enqueue)**: הוספת איבר חדש לסוף התור.
- **הוצאה (dequeue)**: הוצאת האיבר הראשון מהתור.
- **בדיקה אם ריק (isEmpty)**: בדיקה האם התור ריק מאיברים.

- **בדיקת ערך בראש התור (peek)**: הצגת ערך האיבר הראשון בתור מבלי להוציא אותו.

ישנן דרכים שונות ליישם תור, ביניהן:

- **רשימה מקושרת**: רשימה מקושרת חד כיוונית, בה האיברים מאוחסנים בצמתים והקשרים ביניהם מצביעים על סדר הכניסה לתור.
- **מערך**: מערך דינמי שבו איברים חדשים מתווספים בסוף ומאוחסנים לפי סדר הכניסה.
- **תור מעגלי**: מערך שבו האינדקסים "מתעטפים" כך שהאיברים תמיד מאוחסנים באותו גודל מערך, ללא בזבוז מקום.

אני בחרתי לממש את התור ברשימה מקושרת, כך נשמר הפשטות של התור בלי ויתור על יעילות. מימוש התור:

```
/**
* Generic Queue Node Structure
*
* Contains data and a pointer to the next node in the queue.
*/
typedef struct GENERIC_QUEUE_NODE_STRUCT {
   void* data;                  /* Data stored in the node */
   struct GENERIC_QUEUE_NODE_STRUCT *next;  /* Pointer to the next node in
the queue */
} queue_node_T;

/**
* Generic Queue Structure
*
* Contains a pointer to the head and tail nodes of the queue,
* as well as the current size of the queue.
*/
typedef struct GENERIC_QUEUE_STRUCT {
   queue_node_T *head;   /* Pointer to the head node of the queue */
   queue_node_T *tail;   /* Pointer to the tail node of the queue */
   int size;            /* Current size of the queue */
} queue_T;
```

stack

מחסנית (Stack) היא מבנה נתונים מופשט הפועל בצורה דומה למחסנית רובה: האיבר שנכנס ראשון למחסנית יוצא ממנה אחרון. עיקרון זה ידוע בשם "נכנס אחרון יוצא ראשון" (LIFO).

תכונות מפתח:

- **Last In First Out (LIFO)**: האיבר הראשון שיכנס אצא אחרון.

פעולות עיקריות:

- **דחיפה (push)**: הוספת איבר חדש לראש המחסנית.
- **שליפה (pop)**: הוצאת האיבר העליון מהמחסנית.
- **בדיקה אם ריקה (isEmpty)**: בדיקה האם המחסנית ריקה מאיברים.
- **הצצה (peek)**: הצגת ערך האיבר העליון במחסנית מבלי להוציא אותו.

ישנן דרכים שונות ליישם מחסנית, ביניהן:

- **רשימה מקושרת**: רשימה מקושרת חד כיוונית, בה האיברים מאוחסנים בצמתים והקשרים ביניהם מצביעים על סדר הכניסה למחסנית.
- **מערך**: מערך דינמי שבו איברים חדשים מתווספים לראש ומאוחסנים לפי סדר הכניסה.

אני בחרתי לממש את המחסנית ברשימה מקושרת, כך נשמר הפשטות של המחסנית בלי ויתור על יעילות. מימוש המחסנית:

```
/**
* Structure of a stack node
*/
typedef struct STACK_NODE_STRUCT {
 void *data;                        // The data stored in the stack node
 struct STACK_NODE_STRUCT *next;  // Pointer to the next stack node
} stack_node_T;

/**
* Structure of a generic stack
*/
typedef struct GENERIC_STACK_STRUCT {
 stack_node_T *top;                // Pointer to the top of the stack
 size_t size;                      // Number of items in the stack
} stack_T;
```

ניתוח מילוני (lexical analysis)

אסימון (Token)

כפי שצוין בחלק התיאורטי, אסימון הינו יחידה הבנויה משני ערכים:

- שם אסימון - סימן מופשט המגדיר את סוג האסימון.
- ערך - ערך המבדיל אסימון מאסימונים בעל אותו הסוג. למשל, כאשר שם האסימון הוא מתאר מזהה של משתנה, הערך הוא מה שמבדיל את המזהים אחד מהשני.

אסימונים מגדירים את השפה כיחידות שאפשר לעבד ולעבוד אליהם. לדוגמה, ניקח את המשפט הבא הרשום ב-C:

printf("Total = %d\n", score);

יהיה אפשר להגדיר בהתאמה את המשפט לאסימונים:

- <**id**, “printf”>
- <**l_paren**, “(”>
- <**string_literal**, “"Total = %d\n"”>
- <**comma**, “,”>
- <**id**, “score”>
- <**r_paren**, “)”>
- <**semicolon**, “;”>

אני בחרתי לממש אסימון כ-struct הבנוי משם אסימון ומחרוזת של תווים המסמנת את מילת האסימון:

```
typedef struct TOKEN_STRUCT {
      char* value;
      token_type_E type;
} token_T;
```

אוטומט סופי דטרמיניסטי המתאים למנתח המילוני

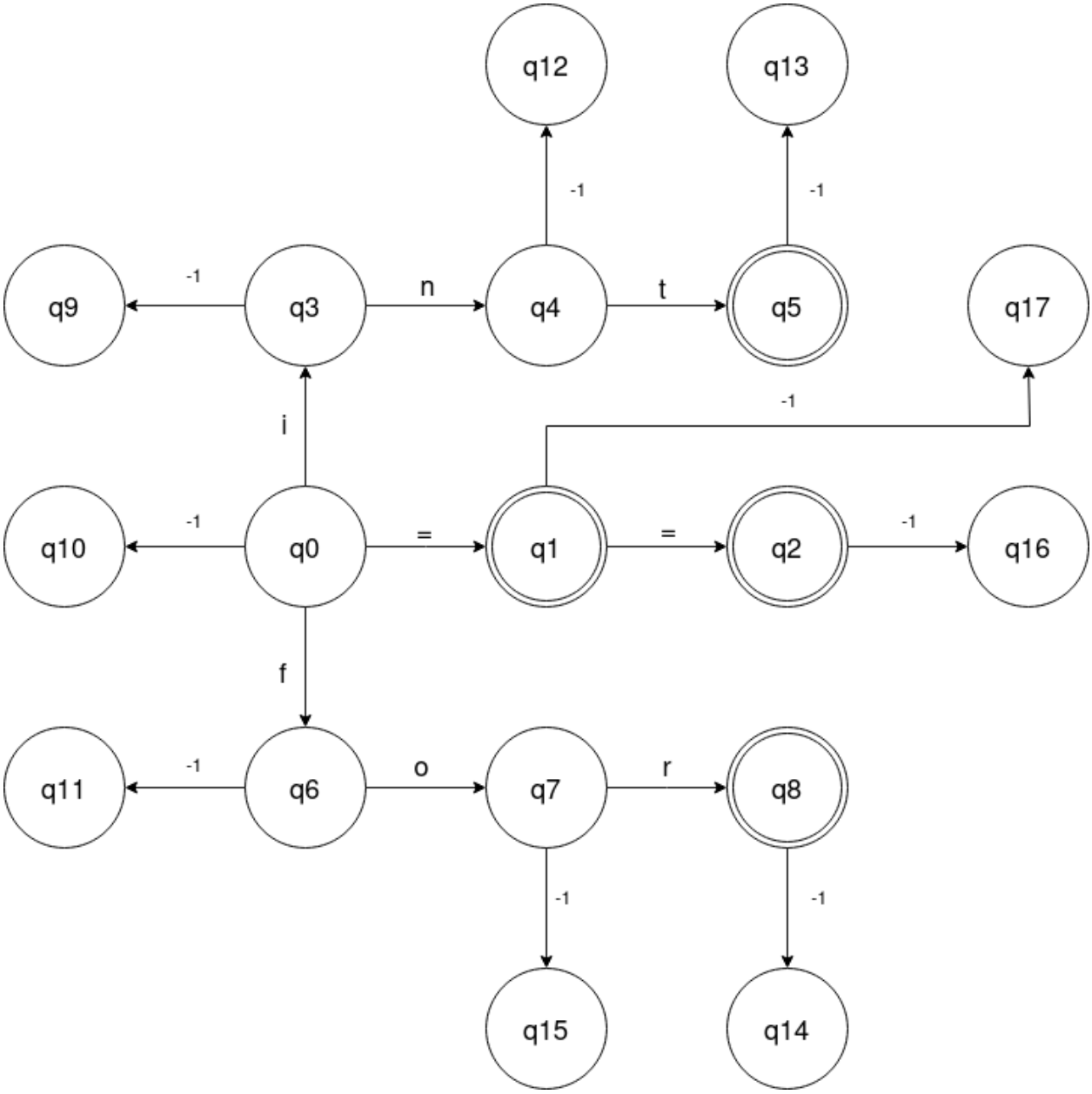
בשביל זיהוי lexemeים בצורה אופטימלית במהלך הניתוח המילוני החלטתי להשתמש באוטומט סופי דטרמיניסטי, שנותן לי לזהות מילה ממילון הטבלה בזמן O(1).

הקלט יהיה אוסף של אותיות (מילה) מתוך הא"ב של השפה, שהיא בנויה מקבוצה של סימנים, לדוגמה, בשפת Quest, הא"ב יהיה כל תווי הascii. האוטמט עובר על המילה ורואה בשביל כל מצב איזו פונקציית מעבר מתאימה לו, כלומר במקרה הזה. כאשר אין יותר סימנים במילה, והמילה היא מילה במילון השפה, הזיהוי הסתיים בהצלחה. תהליך זה מראה שזיהוי מילה הוא אכן בזמן O(1) (כאשר אורך מילה מקסימלית בשפה הוא קבוע).

האוטומט יהיה ממומש בעזרת גרף, כאשר הגרף יהיה ממומש ע"י מטריצת סמיכויות. נממש מבנה זה בשפת C בעזרת מערך דו-ממדי, כאשר:

- כל 128 תווי ה-ascii יוצגו ע"י עמודות המערך הדו-ממדי, כאשר כל אינדקס מותאם לערך האות (לפי טבלת הascii).
- כל המצבים ייוצגו ע"י שורות המערך הדו-ממדי, המצב האחרון שנהיה בוא יהיה תואם לאסימון.
  - אנחנו נגדיר מצב מתחיל שבו תמיד נתחיל את האנליזה, שהוא יהיה המצב הראשון שנקרא לו q0.
- כל ערך במטריצה מגדיר את פונקציית המעבר.
  - ערך שלם אי-שלילי מגדיר את המצב הבא.
  - הערך 1- מגדיר שהמילה לא במילון השפה ושצריך להפסיק את חיפוש המילה.

לדוגמא, נגדיר DFA שלוקח את המילים: int, for, ==, =



נמיר את זה למערך דו ממדי (נתעלם מזה שאינדקס העמודה מתאים לערך האות):

	=	i	n	t	f	o	r
0	1	3	-1	-1	6	-1	-1
1	-1	2	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1

3	-1	-1	4	-1	-1	-1	-1
4	-1	-1	-1	5	-1	-1	-1
5	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	7	-1
7	-1	-1	-1	-1	-1	-1	8
8	-1	-1	-1	-1	-1	-1	-1

ניתוח תחביר (syntax analysis/parsing)

Grammar

תחביר השפה מתואר ע"י Context Free Grammar, שהינה אנוטציה הבנויה מ:

- Terminals/Tokens - יחידות השפה הבסיסיות המוגדרות לפי התחביר ונוצרות בשלב הניתוח המילוני.
- Non-Terminals/Semantic-Variable - יחידות המגדירות מחרוזות של טרמינלים וnon-terminals ההוגדרו לפי התחביר.
- Productions/Rules - סט של חוקים, כאשר כל חוק בנוי מnon-terminal הנקרא ה-left side או ה-head של החוק, חץ, ומחרוזת של terminals ו-non-terminals הנקראים ה-right side של החוק.

לדוגמה, הינה CFG המגדיר רשימה של מספרים מופרדים ע"י סימני פלוס או מינוס:

$$
\begin{aligned}
list &\rightarrow list + digit \\
list &\rightarrow list - digit \\
list &\rightarrow digit \\
digit &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

האנוטציה הזאת שימושית לתיאור השפה ועוזרת לעבד אותה, כפי שהוסבר בחלק התיאורתי.

מימוש התחביר בקוד בנוי משני חלקים:

- סט של כל החוקים של השפה
- סט של כל סימני השפה

```
typedef struct GRAMMAR_STRUCT {
   set_T *rules;
   set_T *symbols;
} grammar_T;
```

lr(0) item

ה-LR parser בוחר עם לעשות shift או reduce ע"י לשמור את מצב הפרסור. מצבים מתוארים ע"י סטים של פריטים (lr(0) item).

פריט של תחביר G הוא חוק של תחביר G עם נקודה בגוף של החוק בין שני סימנים. לדוגמה, החוק $A \rightarrow XYZ$ גורם לארבעה הפריטים הבאים:

$$A \rightarrow \cdot\ XYZ$$
$$A \rightarrow X \cdot YZ$$
$$A \rightarrow XY \cdot Z$$
$$A \rightarrow XYZ\ \cdot$$

אותו פריט ממומש ב-C ע"י המבנה הבא:

```
typedef struct LR_ITEM_STRUCT {
   rule_T *rule;           // grammar rule for example E -> E + T
   size_t dot_index;       // index of the dot in the rule
} lr_item_T;
```

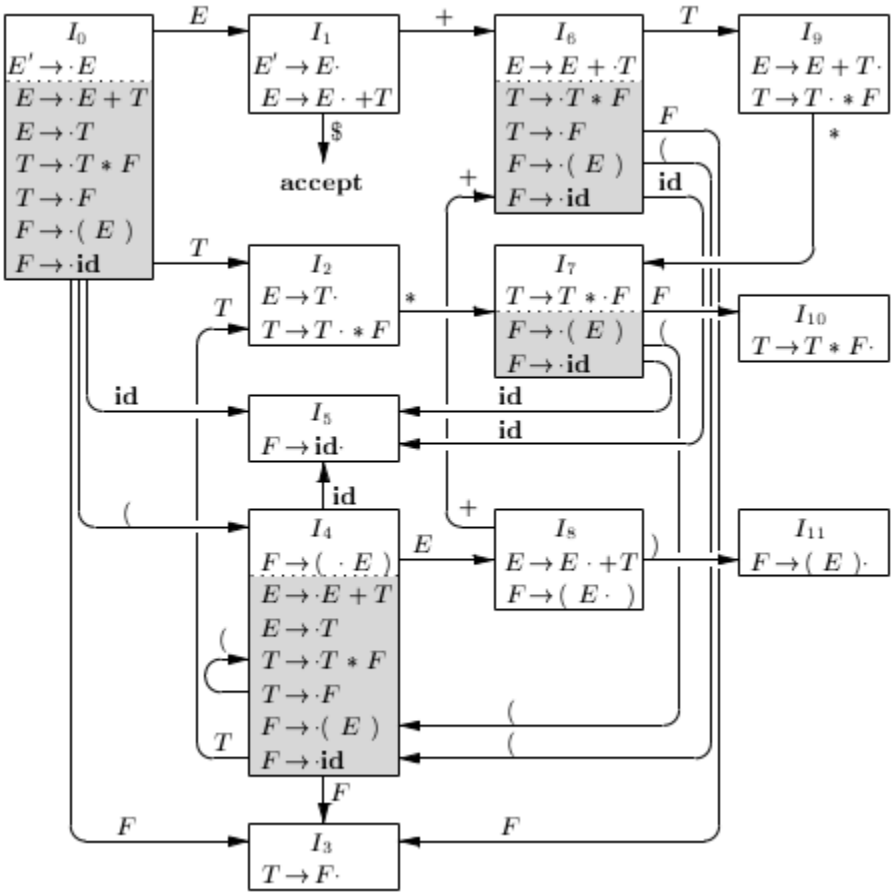
lr(0) automaton

אוטומט ה(lr(0 או ה-(lr(0 canonical הינו אוסף של פריטים שממנו יוצרים אוטומט סופי דטרמיניסטי המשומש כאשר הפרסר צריך לעשות בחירות בתהליך הפרסור. תהליך יצירת האוטומט מתוארת בחלק התיאורטי.

לדוגמה, לתחביר הבא:

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & (\ E\ ) \mid \mathbf{id} \end{array}$$

יש את האוטומט הבא:



האוטומט ממומש בקוד ע"י מערך דינמי של סטים של פריטים.

Action table

טבלת הפרסור בנויה משני חלקים, טבלת action וטבלת goto. טבלת ה-action מורכבת כמטריצה, כאשר השורות הינם המצבים והעמודות הם הטרמינלים. כל ערך בטבלה מוגדר כך:

טבלת האקשן מקבלת את מצב i וטרמינל a כפרמטרים. הערך של action[i, a] יכול להיות אחד מארבעה מצבים:

- Shift j, כאשר j הינו מצב. האקשן הנלקח ע"י הפרסר עושה shift ל-a לתוך מחסנית הפרסר, ומשתמש ב-j בשביל לתאר את a.
- Reduce $A \rightarrow \beta$. עושה reduce ל-β בראש המחסנית ל-A.
- Accept, כלומר מקבל את הפלט ומסיים לפרסר.
- Error, מגלה שגיאה בקלט.

מימוש טבלת ה-action כמבנה בקוד:

```
typedef struct ACTION_TABLE_STRUCT {
   char ***actions;        // a matrix of string describing the actions the
parser should do
   token_T **terminals;    // an array of terminals, a terminals index is
```

```
its index in the action table
   size_t n_terminals;     // number of terminals
   size_t n_states;        // number of states in the action table
} action_tbl_T;
```

Goto table

טבלת ה-goto גם מורכבת כמטריצה, כאשר כל ערך בטבלה מוגדר כך:

טבלת ה-goto מקבלת את המצב I_i ואת ה-non-terminal A כפרמטרים, כאשר: $goto[I_i, A] = I_j$, כלומר, goto ממפה את מצב i ו-non-terminal A למצב j.

להלן דוגמה של טבלת הפרסור המתאימה לדוגמת התחביר של האוטומט ה-lr(0):

STATE	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

מימוש של ה-goto כמבנה בקוד:

```
typedef struct GOTO_TABLE_STRUCT {
   int **gotos;                    // a matrix of integers describing the
goto the parser should do
   non_terminal_T **non_terminals; // an array of terminals, a terminals
index is its index in the action table
   size_t n_non_terminals;         // number of terminals
   size_t n_states;                // number of states in the goto table
} goto_tbl_T;
```

lr stack

משומש ע"י הפרסר בשביל להגדיר את המצב הנוכחי בטבלת ה-parsing. המחסנית הינה ממומשת כמו מחסנית רגילה, עם פונקציות ה-shift ו-reduce המתאימות, שאפשר לממש בקלות בעזרת הפונקציות push ו-pop.

המחסנית תאותחל עם הסימן '$', המציין את סוף התכנית. נראה שבטבלת ה-action'ים, פעולת ה-accept נמצאת אך ורק פעם אחת, בעמודה של $, מכיוון שכאשר נגיע אליו התכנית אמורה להיות בסיומה.

מימוש המחסנית בקוד:

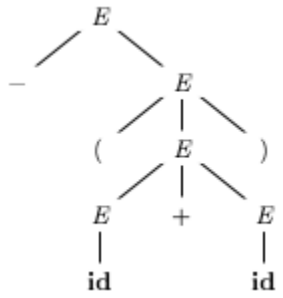
```
typedef struct LR_STACK_STRUCT {
   int *top;           // top of the stack
   size_t size;        // size of stack
   size_t capacity;    // capacity of stack
   size_t alloc_size;  // how many bytes to allocate when stack is full
} lr_stack_T;
```

עץ תחביר (Parse Tree)

העץ התחבירי הינו מבנה הנתונים המועבר למנתח הסמנטי בסוף שלב הניתוח התחבירי, ומתאר את תהליך העיבוד התחבירי של השפה. העץ בנוי מטרמינלים ו-non-terminals כצמתי העץ, כאשר כל צומת בעץ מתארת את הגזרה בא חוקים מתאימים ל-left side בשביל ליצור non-terminal מתאים.

בניית העץ נעשית תוך כדי הניתוח, כאשר כל פעם שנעשה reduce, נוסיף לעץ את ה-non-terminals שבצד שמאל של החוק, ושבניו יהיו כל הסימנים מצידו הימני. תשימו לב שגם לסימנים מצד ימים יכולים להיות ילדים, וכך העץ ממשיך להיבנות.

להלן, בהתייחס לדוגמאות ממבני הנתונים הקודמים בפרק הזה, העץ התחבירי בשביל המשפט -(**id** + **id**) :



יש לשים לב שכל העלים הם טרמינלים ושכל הצמתים הם non-terminals, זאת מכיוון שלטרמינלים אין גזרה.

מימוש העץ התחבירי בקוד, הבנוי משני מבנים:

- צומת בעץ
- מבנה המצביע לראש העץ

```
typedef struct PARSE_TREE_NODE_STRUCT {
```

```
   symbol_T *symbol;                          // symbol in curr node on
parse tree
   ssize_t rule_index;                        // index of rule in rules
arr (or -1 indicating that it has no rule where its in the ledt side)
   struct PARSE_TREE_NODE_STRUCT **children;  // arr of children of parse
tree node
   size_t n_children;                         // number of children the
node has
} parse_tree_node_T;

typedef struct PARSE_TREE_STRUCT {
   parse_tree_node_T *root;    // root node of parse tree
   rule_T **rules;             // array of all rules. in array so getting
of rule is theta(1);
   size_t n_rules;             // number of rules
} parse_tree_T;
```

ניתוח סמנטי (semantic analysis)

SDT

Syntax-directed translation הינה דרך לממש Syntax-directed definitions, שהינה CFG המתואם עם תכונות וחוקים סמנטיים, כאשר:

- תכונה - ערך כלשהו המותאם לסימן
- חוק - פעולה כלשהי המתאימה לחוק התחבירי

להלן דוגמה ל-sdd:

	Production	Semantic Rules
1)	$L \to E\ \mathbf{n}$	$L.val = E.val$
2)	$E \to E_1\ +\ T$	$E.val = E_1.val + T.val$
3)	$E \to T$	$E.val = T.val$
4)	$T \to T_1\ *\ F$	$T.val = T_1.val \times F.val$
5)	$T \to F$	$T.val = F.val$
6)	$F \to (\ E\ )$	$F.val = E.val$
7)	$F \to \mathbf{digit}$	$F.val = \mathbf{digit}.\mathrm{lexval}$

להלן שני מבנים הממשמקים sdt בקוד. הראשון, המתאים חוק סמנטי לחוק תחבירי:

```
typedef struct SEMANTIC_RULE_STRUCT {
   rule_T *rule;
// production rule, for example, E -> E + T
```

```
   void (*definition)(stack_T *astack, parse_tree_node_T *tree, stack_T
*st_s);   // definition of rule, for example. E.val = E1.val + T.val,
implemented by function ptr
} semantic_rule_T;
```

המבנה השני, המתאר את ה-sdt עצמו:

```
typedef struct SDT_STRUCT {
   semantic_rule_T **definitions;
   size_t n_defenitions;
} sdt_T;
```

AST

עץ תחביר מופשט (Abstract Syntax Tree) הוא ייצוג ויזואלי של מבנה התוכנית בצורה היררכית. הוא מתאר את היחסים הסינטקטיים בין רכיבי התוכנית, ללא תלות בסימון הספציפי של שפת התכנות. מכיוון שהעץ הסמנטי מתאר את הסמנטיקה של השפה, סימנים מסוימים מהעץ התחבירי יימחקו, מכיוון שהם לא תורמים לסמנטיקה שלה. למשל, סוגריים המתארים אילו הוראות ייקראו קודם בתוכנית

העץ הסמנטי הוא התוצאה של תרגום העץ התחבירי בעזרת sdt, כאשר לכל צומת יש תרגום ישיר ל-ast.

מימוש ast בקוד:

```
typedef struct ABSTRACT_SYNTAX_TREE_NODE_STRUCT {
   symbol_T *symbol;
   struct ABSTRACT_SYNTAX_TREE_NODE_STRUCT **children;
   size_t n_children;
   symbol_table_entry_T *st_entry;
} ast_node_T;
```

יצירת קוד (code generation)

TTS & translation rule

טכניקת תרגום עץ (Tree Translation scheme) הינה טכניקה המקבלת עץ סמנטי כלשהו ומתאימה מבנה מסוים של העץ לתכנית. הטכניקה מורכבת מתרגומים המתאימים למבנה בעץ ותכנית תרגום.

בעזרת השיטה הזאת אפשר לעבור על העץ ולתרגם אותו לקוד המטרה.

מימוש בקוד:

```
typedef struct TRANSLATION_RULE_STRUCT {
```

```
   symbol_T *symbol;
   char *(*translation)(ast_node_T *ast, stack_T *astack, stack_T
*code_stack, register_pool_T **regs, uint32_t *label_counter);
} translation_rule_T;
```

```
typedef struct TREE_TRANSLATION_SCHEME_STRUCT {
   translation_rule_T **tok_translation;
   size_t n_tok;
   translation_rule_T **non_term_translation;
   size_t n_non_term;
} tts_T;
```

Register & Pool

כאשר קוד המטרה הוא שפת סף, התכנית צריכה לדעת איך להתמודד אלוקציה של רגיסטרים לנתונים. לשם כך נוצר המבנה register וה-register pool. שני המבנים די מובנים מאליו לפי השמות שלהם, register מתאר את הרגיסטר המשומש, וה-register pool מתאר את בריכת הרגיסטרים בתכנית, כלומר כל הרגיסטרים ותכונות אליהם.

מבנה כזה נותן לנו לעקוב אחרי כל הרגיסטרים כרצוננו ולשמור על הרגיסטרים המשומשים בתכנית בזמן התרגום.

מימוש אותם מבנים בקוד:

```
typedef struct REGISTER_STRUCT {
   register_E reg;         // reg type
   register_size_E size;   // reg size
   char *name;             // name
} register_T;

typedef struct REGISTER_POOL_STRUCT {
   uint64_t value;         // reg pool value
   uint64_t in_use;        // is reg or part of reg in use
   register_type_E type;   // type of reg
} register_pool_T;
```

ה-pool נשמר כיחידה של רגיסטר והבייתים שלו משומשים כמתאימים לרגיסטר, וערך ה-in_use מתאר איזה בית בשימוש. כאשר הביט דלוק, הביט המתאים בערך value בשימוש.

טבלת סימנים (symbol table)

Symbol table

אנו משתמשים ב-symbol table לאורך כל תהליך הקומפילציה. היא עוזרת לנו לשמור מידע ולקבל מידע באופן יעיל על כל משתנה בתוכנית, כמו גם היא עוזרת לנו לנהל את ה-scopeים של התכנית. היא למעשה מאגר מידע המכיל את כל המשתנים השונים בתוכנית, כגון שמות של משתנים, פונקציות, מחלקות וכדומה. לכל רשומה בטבלה זו מקושרים פרטים נוספים אודות המרכיב, כגון:

- שם משתנה
- ערך מתאים
- סוג
- עוד...

הסימנים נשמרים בטבלת גיבוב, כך שמירת ואחזור סימנים תהיה ב-O(1).

מימוש בקוד:

```
typedef struct SYMBOL_TABLE_ENTRY_STRUCT {
   char *name;
   int type;
   void *value;
   entry_type_E declaration_type;
   struct SYMBOL_TABLE_ENTRY_STRUCT *next;
} symbol_table_entry_T;

typedef struct SYMBOL_TABLE_STRUCT {
   size_t size;                                    // current entries in
the symbol table
   size_t capacity;                                // number of entries
that can be in the symbol table, is effected by load factor
   float load_factor;                              // load factor of table
   symbol_table_entry_T **buckets;                 // actual members of
hash table. members are arranged in a chained table, meaning each member is
using a linked list if there is any collision
   unsigned int (*hash)(char *, size_t length);    // hash function of the
table
} symbol_table_T;
```

scope tree

ניהול scopeים נעשה ע"י scope tree, כאשר כל צומת בעץ מתארת עוד שלב ב-scope. כלומר, שורש העץ יהיה ה-global scope, כאשר כל הנתונים ב-scope הגלובלי נתונים לכל התכנית, וכל בן מגדיר עוד scope שנוצר באותו ה-scope.

מימוש של ה-scope tree בקוד:

```
typedef struct SYMBOL_TABLE_TREE_NODE_STRUCT {
   symbol_table_T *table;
   struct SYMBOL_TABLE_TREE_NODE_STRUCT **children;
   size_t n_children;
} symbol_table_tree_node_T;

typedef struct SYMBOL_TABLE_TREE_STRUCT {
   symbol_table_tree_node_T *root;
} symbol_table_tree_T;
```

מטפל השגיאות (error handler)

סביבת העבודה ושפת התכנות

סביבת העבודה

OS: Debian GNU/Linux 12 (bookworm) x86_64
Host: VivoBook_ASUSLaptop X415JA_X415JA 1.0
Kernel: 6.1.0-18-amd64
Uptime: 4 days, 4 hours, 48 mins
Packages: 2944 (dpkg), 7 (flatpak), 24 (snap)
Shell: bash 5.2.15
Resolution: 1366x768, 1920x1080
DE: Xfce 4.18
WM: Xfwm4
WM Theme: empty
Theme: Xfce [GTK2], Adwaita [GTK3]
Icons: Tango [GTK2], Adwaita [GTK3]
Terminal: x-terminal-emul
CPU: Intel i5-1035G1 (8) @ 3.600GHz
GPU: Intel Iris Plus Graphics G1
Memory: 6875MiB / 7682MiB

עורכי הקוד

VSCodium v1.88.1

NVIM v0.9.4

שפת תכנות וקומפיילר

כל הקוד נכתב ב-C, עם ה-gcc קומפיילר.

gcc: v12.2.0 (Debian 12.2.0-14)

שפת התכנות

הכל נכתב ב-C

אלגוריתם ראשי

להלן תכנית **מאוד מופשטת** המתארת את האלגוריתם הראשי של הקומפיילר, הבנוי מארבעת שלבי הקומפילציה, ניתוח מילוני, ניתוח תחבירי, ניתוח סמנטי ויצירת קוד:

1. קבל מהמשתמש את שם הקובץ
2. תקרא את התכנית בתוך הקובץ ותהפוך אותה למחרוזת
3. אתחל את כל כלי הקומפילציה, כלומר הלקסר, הפרסר, המנתח הסמנטי, יוצר הקוד, וטבלת הסימנים
4. אתחל את האוטומטים של הלקסר והפרסר
5. עבור על התווים בקובץ הקלט:
    - 5.1. עבור אל המצב ההתחלתי
    - 5.2. כל עוד לא הגעת למצב 1-
        - 5.2.1. עבור למצב הבא על פי המצב הנוכחי והתו הנוכחי מקוד המקור
        - 5.2.2. התקדם תו אחד בקוד המקור
    - 5.3. צור אסימון על פי המצב שהגעת אליו
    - 5.4. הוסף את האסימון לתור האסימונים
6. אתחל את הניתוח הסמטי
7. עבור על כל האסימונים
    - 7.1. תן ל-a להיות הטרמינל הראשון, ול-s להיות ראש הסטאק תמיד
    - 7.2. כל עוד action_table[s, a] לא accept
        - 7.2.1. אם action_table[s, a] הינו shift t
            - 7.2.1.1. תדחוף t למחסנית המצבים
            - 7.2.1.2. תדחוף את a למחסנית העצים
            - 7.2.1.3. תן ל-a להיות הטרמינל הבא
        - 7.2.2. אם action_table[s, a] הינו reduce A -> b
            - 7.2.2.1. תוציא $|b|$ ממחסנית המצבים
            - 7.2.2.2. תוציא $|b|$ ממחסנית העצים ושים אותם במשתנה c
            - 7.2.2.3. תבנה עץ מ-A ובניו c
            - 7.2.2.4. תדחוף את העץ החדש למחסנית העצים
            - 7.2.2.5. תן ל-t להיות ראש מחסנית המצבים
        - 7.2.3. אם האקשן הינו accept
            - 7.2.3.1. החזר את עץ התחביר

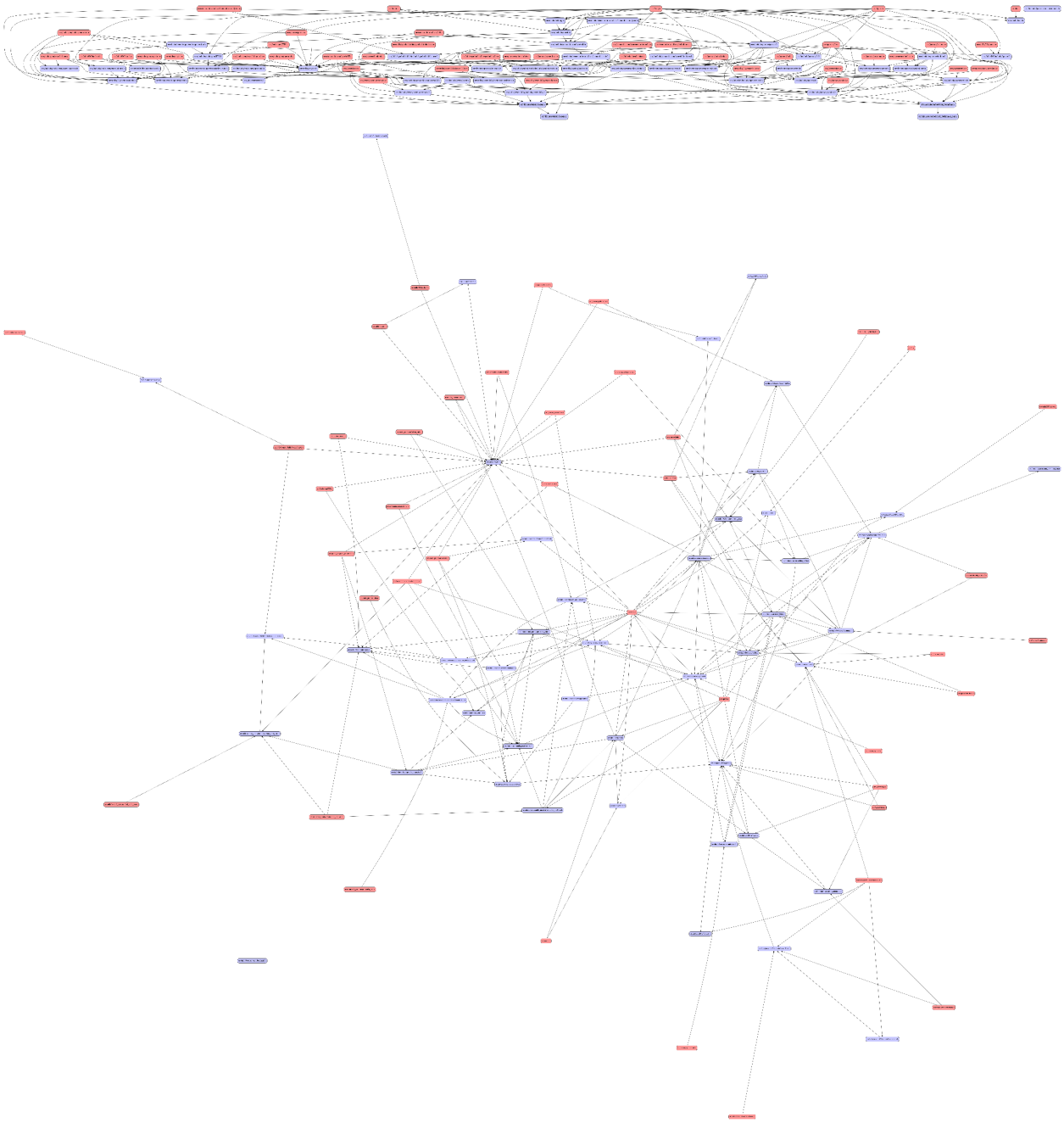
7.2.4. תזרוק שגיאה ותעצור את התכנית
8. עובר על כל הצמתים בגרף
8.1. אם נמצא משתנה חדש
8.1.1. תוסיף אותו ל-symbol table עם הפרמטרים המתאימים
8.2. אם יש פונקציית תרגום מתאימה
8.2.1. תריץ את פונקציית התרגום
8.2.2. תוסיף בהתאמה את חלק העץ שנוצר ל-AST
9. עבור על ה-AST
9.1. אם נמצאה פונקציית תרגום מתאימה לצומת
9.1.1. תרגם את הצומת ותחזיר את התרגום
9.1.2. תוסיף את התרגום לתוכנית מטרה
9.2. תחזיר את תכנית המטרה
10. אם היו שגיאות בכל מהלך התכנית, תדווח אליהם ועל תוציא את תכנית הפלט
11. תכתוב את תכנית המטרה לקובץ פלט

תיאור ממשקים חיצוניים

Make - תוכנה לבניית קוד אוטומטית בעזרת קובץ ייעודי
Cmake - תוכנה לבניית קוד אוטומטי, טסטים ועוד..
Gcc - הקומפיילר שהתשתמשי בו.
C - שפת התכנות שהשתמשתי בא.

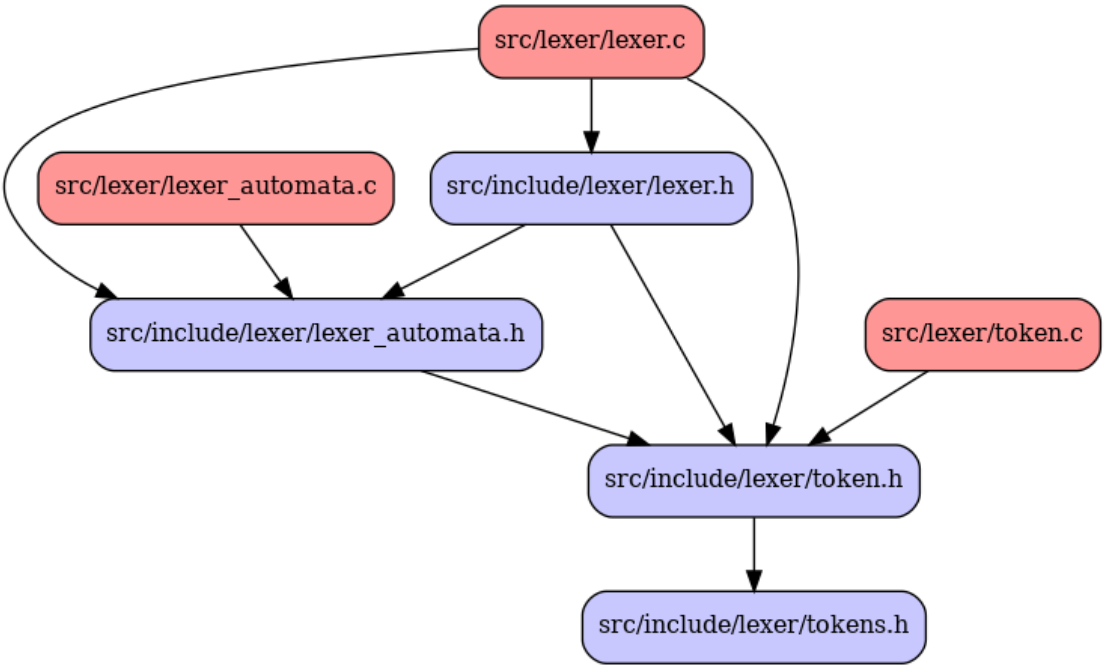
תרשים מחלקות

מכיוון שאין מחלקות ב-C, הכנתי dependency diagram. להלן שני גרסאות, אחת מפוזרת ואחת לא:

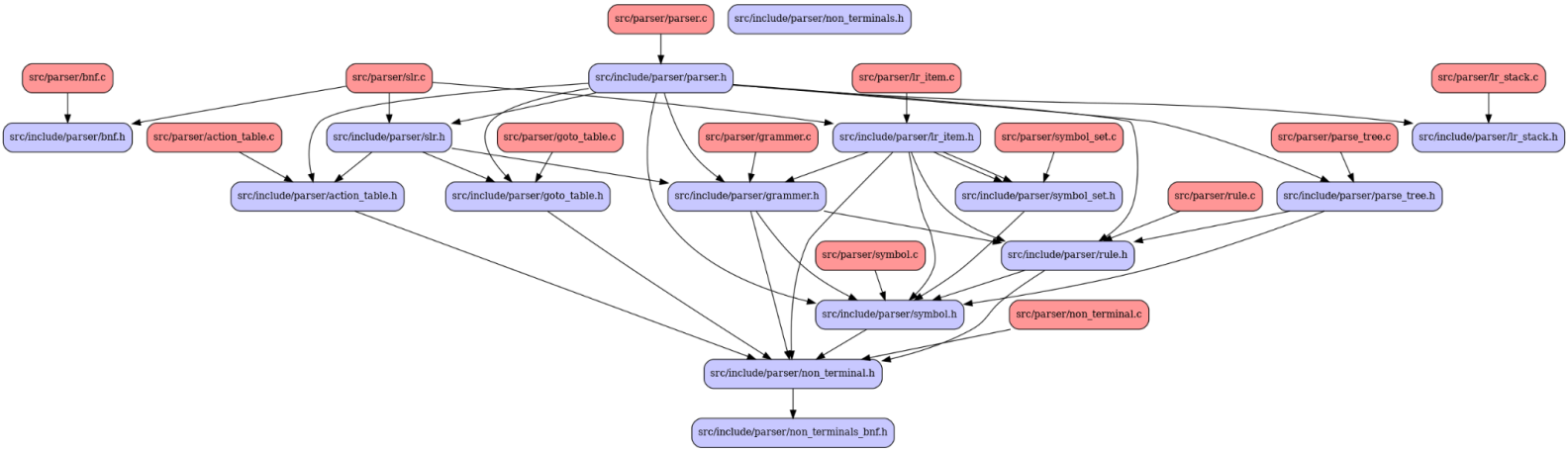


אי אפשר להבין כלכך הרבה משניהם על דף, אז הם יוספו כנספחים. בנוסף, dependency graph בשביל כל שלב.

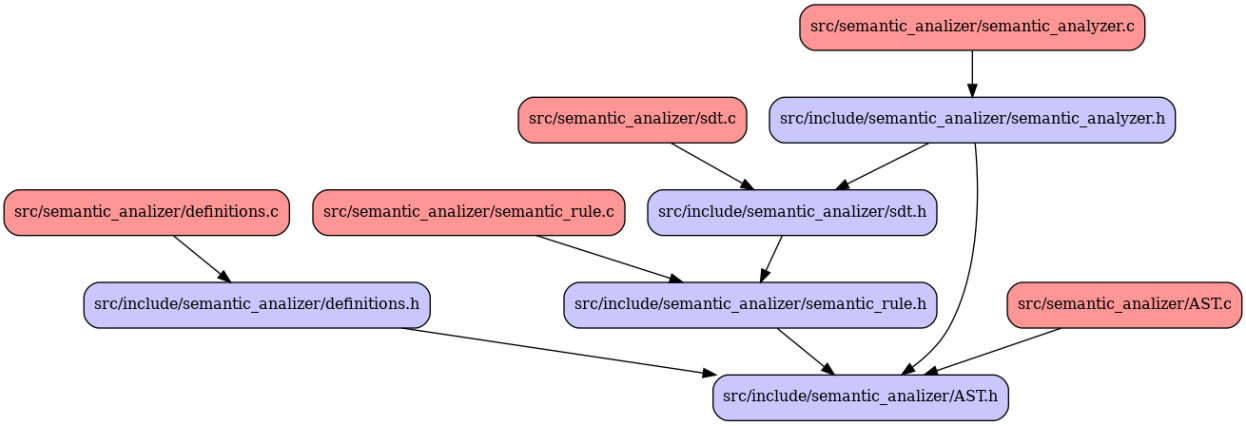
Lexer



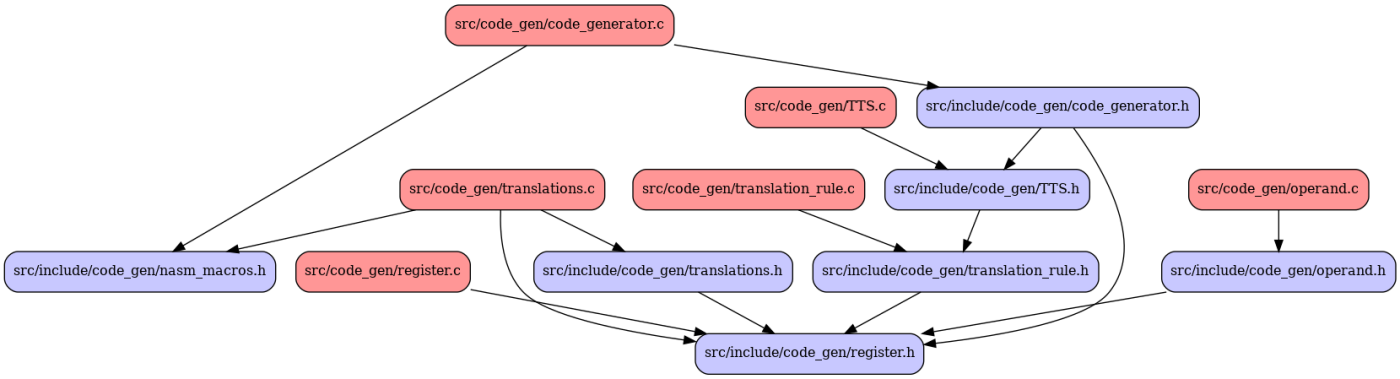
Parser



Semantic Analysis



Code Generation



מודולים ופונקציות ראשיות

מבט על

שימו לב שקיימות תיקיות include, שהם תיקיות השומרות קבצי header. בהמשך אני אתייחס לפעמים לקובץ ה-c וקובץ ה-h כאחד.

- code_gen
  - TTS.c
  - code_generator.c
  - operand.c
  - register.c
  - translation_rule.c
  - translations.c
- include
  - code_gen
    - TTS.h
    - code_generator.h
    - nasm_macros.h
    - operand.h
    - register.h
    - translation_rule.h
    - translations.h
  - io.h
  - lang.h
  - lexer
    - lexer.h
    - lexer_automata.h
    - token.h
    - tokens.h
  - macros.h
  - parser
    - action_table.h
    - bnf.h
    - goto_table.h
    - grammer.h
    - lr_item.h
    - lr_stack.h
    - non_terminal.h
    - non_terminals.h
    - non_terminals_bnf.h
    - parse_tree.h
    - parser.h
    - rule.h
    - slr.h
    - symbol.h
    - symbol_set.h
  - quest.h
  - semantic_analizer
    - AST.h
    - definitions.h

            - sdt.h
            - semantic_analyzer.h
            - semantic_rule.h
- io.c
- lang.c
- lexer
    - lexer.c
    - lexer_automata.c
    - token.c
- main.c
- parser
    - action_table.c
    - bnf.c
    - goto_table.c
    - grammer.c
    - lr_item.c
    - lr_stack.c
    - non_terminal.c
    - parse_tree.c
    - parser.c
    - rule.c
    - slr.c
    - symbol.c
    - symbol_set.c
- quest.c
- semantic_analizer
    - AST.c
    - definitions.c
    - sdt.c
    - semantic_analyzer.c
    - semantic_rule.c
- utils
    - DS
        - generic_set.c
        - hashset.c
        - include
            - generic_set.h
            - hashset.h
            - queue.h
            - stack.h
        - queue.c
        - stack.c
    - err
        - err.c

        - err.h
        - errors.h
    - hashes
        - hashes.c
        - hashes.h
    - lexer_DFA
        - include
            - lexer_DFA.h
            - transitions.h
        - lexer_DFA.c
        - transitions.c
    - symbol_table
        - include
            - symbol_table.h
            - symbol_table_tree.h
    - symbol_table.c
    - symbol_table_tree.c

להלן הסברים עבור כל קובץ חשוב, עם הסברים על מבנים והפונקציות הראשיות שבניהם.
בשביל להסביר על כל struct אשתמש בטבלה הבא:

	שם מבנה	
הסבר	סוג	שם משתנה

ובשביל להסביר על פונקציה אשתמש בטבלה הבא:

שם פונקציה	קלט	פלט	יעילות	תיאור

code_gen

TTS

	TREE_TRANSLATION_SCHEME_STRUCT	
הסבר	סוג	שם משתנה
מערך דינמי השומר מצביעים ל-structים המגדירים תרגומים של אסימונים	translation_rule_T **	tok_translation
מספר האיברים במערך	size_t	n_tok
מערך דינמי השומר מצביעים ל-structים המגדירים תרגומים של non-terminals	translation_rule_T **	non_term_translation
מספר האיברים במערך	size_t	n_non_term

תיאור	יעילות	פלט	קלט	שם פונקציה
יוצר משתנה מסוג tts_T* ומחזיר אותו.	O(1)	tts_T*	translation_rule_T **tok_translation, size_t n_tok, translation_rule_T **non_term_translation, size_t n_non_term	init_tts
יוצר משתנה מסוג tts_T* בנוסף למיקום במערך המתאים לערך האסימון או לערך ה-non terminal.	O(1)	tts_T*	translation_rule_T **tok_translation, size_t n_tok, translation_rule_T **non_term_tra	create_tts

	nslation, size_t n_non_term			

code_generator

	CODE_GENERATOR_STRUCT	
הסבר	**סוג**	**שם משתנה**
מערך דינמי השומר מצביעים ל-structים המגדירים תרגומים של אסימונים	register_pool_T **	registers
מספר האיברים במערך	tts_T *	tts
מערך דינמי השומר מצביעים ל-structים המגדירים תרגומים של non-terminals	symbol_table_tree_T *	sym_tbl
מונה בשביל יצירת labelים	uint32_t	label_counter
מחרוזת תווים המייצגת את הקוד הנוצר	char *	output

תיאור	יעילות	פלט	קלט	שם פונקציה
יוצר משתנה מסוג code_gen* ומחזיר אותו.	O(1)	code_gen_T *	register_pool_T **registers, tts_T *tts, symbol_table_tree_T *sym_tbl	init_code_gen
יוצר את הקוד ומחזיר מחרוזת של תכנית המטרה	O(n)	char *	ast_node_T *ast, code_gen_T *cg	generate_code

operand

	OPERAND_TYPES_ENUM	
שם משתנה	סוג	הסבר
SYMBOL	uint = 0	מגדיר סמל
REGISTER	uint = 1	מגדיר רגיסטר

	OPERAND_UNION	
שם משתנה	סוג	הסבר
sym	symbol_T *	מצביע לסמל
reg	register_T *	מצביע לרגיסטר

	OPERAND_STRUCT	
שם משתנה	סוג	הסבר
type	operand_type_E	סוג המשתנה ב-union
operand	operand_U *	מגדיר אופרנד

שם פונקציה	קלט	פלט	יעילות	תיאור
init_operand_symbol	symbol_T *	operand_T *	O(1)	יוצר מצביע לאופרנד שיש בתוכו סימן

init_operand_register	register_T *	operand_T *	O(1)	יוצר מצביע לאופרנד שיש בתוכו רגיסטר

register

	REGISTER_STRUCT	
שם משתנה	סוג	הסבר
reg	register_E	סוג רגיסטר
size	register_size_E	גודל רגיסטר
name	char *	שם רגיסטר

	REGISTER_POOL_STRUCT	
שם משתנה	סוג	הסבר
value	uint64_t	הערך ברגיסטר
in_use	uint64_t	האם כל בית ברגיסטר בשימוש או לא
type	register_type_E	סוג הערך של הרגיסטר

שם פונקציה	קלט	פלט	יעילות	תיאור
init_register	uint8_t type, uint8_t size, char *name	register_T *	O(1)	איתחול הרגיסטר עם הקלט

אתחול בריכת הרגיסטרים המשומשים בקוד	O(1)	register_pool_T *	uint8_t type	init_register_pool
מוצא רגיסטר חופשי בבריכת הרגיסטרים המתאים לקלט	O(n)	register_T *	register_pool_T **regs, uint8_t type, uint8_t size	get_register
מחזיר את שם הרגיסטר לפי מטריצת שמות הרגיסטרים	O(1)	char *	uint8_t type, uint8_t size	get_register_name
מחזיר את שם הרגיסטר לפי מטריצת שמות הרגיסטרים מסום DATA ובגודל BYTE	O(1)	char *	uint8_t type, uint8_t bits	get_byte_data_reg_name
משחרר את השימוש ברגיסטר בבריכת הרגיסטרים	O(1)	void	register_pool_T **regs, register_T *reg	reg_free

Translation_rule

	TRANSLATION_RULE_STRUCT	
הסבר	**סוג**	**שם משתנה**
הסמל המתאים לתרגום	symbol_T *	symbol
מצביע לפונקציה שמתרגמת את הסימן בהתאם לפרמטרים של הפונקציה	char *(*translation)(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T **regs)	translation

שם פונקציה	קלט	פלט	יעילות	תיאור
init_translation_rule	l_T *symbol, char *(*translation)(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T **regs)	translation_rule_T *	O(1)	מאתחל מצביע לחוק תרגום

translations

שם פונקציה	קלט	פלט	יעילות	תיאור
translation_… (כל פונקציית תרגום)	ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T **regs	char *	O(1)	מתרגם סימן בהתאמה לפרמטרים

lexer

lexer

	LEXER_STRUCT	
שם משתנה	סוג	הסבר
src	char *	קוד המקור
src_size	size_t	גודל קוד המקור

c	char	התו שהלקסר עליו
i	unsigned int	האינדקס של התו ממצביע קוד המקור
automata	lexer_automata_T *	האוטומט של הלקסר

שם פונקציה	קלט	פלט	יעילות	תיאור
init_lexer	char *src	lexer_T *	O(1)	מאתחל משתנה מסוג מצביע ללקסר
lexer_advance	lexer_T* lex	void	O(1)	עובר לתו הבא במחרוזת התכנית
lexer_skip_whitespace	lexer_T *lex	void	O(1)	עובר על רווחים
lexer_skip_bullshit	lexer_T *lex	void	O(1)	עובר על תווים לא מוגדרים בסימיני השפה
lexer_next_token	lexer_T *lex	token_T *	O(1)	מחזיר את האסימון הבא בתכנית

lexer_automata

LEXER_AUTOMATA_STRUCT		
שם משתנה	סוג	הסבר
automata	short **	מטריצה המגדירה את האוטומט
state_type	token_type_E *	מערך המגדיר את סוגי המצבים

n_state	unsigned int	מספר המצבים
n_symbols	unsigned int	מספר הסימנים מתחילת טבלת הASCII

שם פונקציה	קלט	פלט	יעילות	תיאור
init_lexer_automata	const char *auto_src, const char *states_src	lexer_automata_T *	O(1)	מאתחל את האוטומט של הלקסר

token

TOKEN_STRUCT		
שם משתנה	סוג	הסבר
value	char *	ערך האסימון
type	token_type_E	סוג האסימון

שם פונקציה	קלט	פלט	יעילות	תיאור
init_token	char *value, int type	token_T *	O(1)	מאתחל אסימון
token_cmp	const token_T *tok1, const token_T *tok2	int	O(1)	משווה בין שני אסימונים

token_cmp_generic	const void *tok1, const void *tok2	int	O(1)	משווה בין שני אסימונים עם קלט גנרי

parser

action_table

ACTION_TABLE_STRUCT		
שם משתנה	סוג	הסבר
actions	char ***	מטריצה של מחרוזות המתארת את ה-actionים שהפרסר לוקח
terminals	token_T **	מערך של טרמינלים המתארים את הטרמינלים במטריצה
n_terminals	size_t	מספר הטרמינלים
n_states	size_t	מספר המצבים

שם פונקציה	קלט	פלט	יעילות	תיאור
init_action_tbl	token_T **terminals, size_t n_terminals, size_t n_states	action_tbl_T *	O(1)	מאתחל את טבלת האקשנים
action_tbl_find_terminal	action_tbl_T *tbl, token_T *term	int	O(n)	מחזיר את האינדקס של הטרמינל במערך

				ובהתאמה במטריצה
action_tbl_print_to_file	action_tbl_T *tbl, char *dest	void	O(n)	מדפיס את טבלת האקשנים לקובץ
action_tbl_pretty_print_to_file	action_tbl_T *tbl, char *dest	void	O(n)	מדפיס את טבלת האקשנים לקובץ רק יפה

bnf

שם פונקציה	קלט	פלט	יעילות	תיאור
bnf_make_non_terminals	char *src, char *dest	void	O(n)	יוצר non-terminals בקוד מקובץ bnf

goto_table

GOTO_TABLE_STRUCT		
שם משתנה	סוג	הסבר
gotos	int **	מטריצה של שלמים המתארת את ה-gotoים שהפרסר לוקח

non_terminals	non_terminal_T **	מערך של non-terminals המתארים את ה-non-terminals במטריצה
n_non_terminals	size_t	מספר ה-non-terminals
n_states	size_t	מספר המצבים

שם פונקציה	קלט	פלט	יעילות	תיאור
init_goto_tbl	non_terminal_T **non_terminals, size_t n_non_terminals, size_t n_states	goto_tbl_T *	O(1)	מאתחל את טבלת ה-gotoים
goto_tbl_find_non_terminal	goto_tbl_T *tbl, non_terminal_T *nterm	size_t	O(n)	מחזיר את האינדקס של הnon-terminal במערך ובהתאמה במטריצה
goto_tbl_print_to_file	goto_tbl_T *tbl, char *dest	void	O(n)	מדפיס את טבלת הgotoים לקובץ
goto_tbl_pretty_print_to_file	goto_tbl_T *tbl, char *dest	void	O(n)	מדפיס את טבלת הgotoים לקובץ רק יפה

grammer

	GRAMMER_STRUCT	
שם משתנה	סוג	הסבר

rules	set_T *	סט של חוקים
symbols	set_T *	סט של סימנים

שם פונקציה	קלט	פלט	יעילות	תיאור
init_grammer	set_T *rules, set_T *symbols	grammer_T *	O(1)	מאתחל את הדקדוק
terminals_in_symbol_set	set_T *symbols	token_T **	O(n)	מחזיר את כל הטרמינלים מסט של סימנים
terminals_in_symbol_set_and_dollar	set_T *symbols	token_T **	O(n)	מחזיר את כל הטרמינלים מסט של סימנים פלוס סימן של דולר המסמן התחלה
n_terminals_in_symbol_set	set_T *symbols	size_t	O(n)	מחזיר את מספר הטרמינלים בסט של סימנים
non_terminals_in_symbol_set	set_T *symbols	non_terminal_T **	O(n)	מחזיר את כל הnon-terminals מסט של סימנים
n_non_terminals_in_symbol_set	set_T *symbols	size_t	O(n)	מחזיר את מספר הnon-terminals בסט של סימנים
find_right_grammer_index	const grammer_T *gram, symbol_T **right, size_t right_size	size_t	O(n)	מחזיר את האינדקס של חוק בדקדוק שיש לו את אותו הצד השמאלי של החוק

lr_item

	LR_ITEM_STRUCT	
הסבר	סוג	שם משתנה
מצביע לחוק של הפריט	rule_T *	rule
האינדקס של הנקודה של הפריט	size_t	dot_index
ה-lookaheadים של הפריט	set_T *	lookaheads

תיאור	יעילות	פלט	קלט	שם פונקציה
מאתחל את הפריט	O(1)	int	rule_T *rule, size_t dot_index, set_T *lookaheads	init_lr_item
משווה את הפריט	O(1)	int	const lr_item_T *item1, const lr_item_T *item2	lr_item_cmp
משווה את הפריט הגנרי	O(1)	int	const void *item1, const void *item2	lr_item_cmp_generic
משווה סט פריטים	O(1)	int	const set_T *set1, const set_T *set2	lr_item_set_cmp
משווה סט פריטים גנרי	O(1)	set_T *	const void *item1, const void *item2	lr_item_set_cmp_generic

מוצא את ה-first set של סימן מסוים	O(n)	set_T *	const grammer_T *gram, const symbol_T *sym	first
מוצא את ה-follow set של non-terminal	O(n)	set_T *	const grammer_T *gram, const non_terminal_T *nt	follow
מוצא את ה-clouser של סט של פריטים	O(n)	set_T *	grammer_T *grammer, set_T *items	closure
מוצא את ה-goto של סט של פריטים וסימן מסוים	O(n)	set_T *	grammer_T *grammer, set_T *items, symbol_T *symbol	go_to
מוצא את ה-clouser של סט של פריטים עם lookahead	O(n)	set_T *	grammer_T *grammer, set_T *items	closure_lookahead
מוצא את ה-goto של סט של פריטים עם lookahead וסימן מסוים	O(n)	set_T *	grammer_T *grammer, set_T *items, symbol_T *symbol	go_to_lookahead
מוצא את הסט של הסטים של פריטים של תחביר lr(0)	O(n)	set_T *	grammer_T *grammer, lr_item_T *starting_item	lr0_items
מוצא את הסט של הסטים של פריטים של תחביר lr(1)	O(n)	set_T *	grammer_T *grammer, lr_item_T *starting_item	lr1_items

lr_stack

	LR_STACK_STRUCT	

שם משתנה	סוג	הסבר
top	int *	הגג של הסטאק
size	size_t	גודל הסטאק
capacity	size_t	הגודל המקסימלי של הסטאק
alloc_size	size_t	כמות הביתים להקצות כאשר הסטאק מלא

שם פונקציה	קלט	פלט	יעילות	תיאור
init_lr_stack	size_t alloc_size	lr_stack_T *	O(1)	מאתחל את סטאק הפריטים
lr_stack_push	lr_stack_T *s, int data	void	O(1)	דוחף איבר מגג הסטאק
lr_stack_pop	lr_stack_T *s	int	O(1)	מוציא איבר מגג הסטאק
lr_stack_peek	lr_stack_T *s	int	O(1)	מסתקל על האיבר בגג הסטאק
lr_stack_peek_inside	lr_stack_T *s, int n	int	O(1)	מסתקל על האיבר בגג הסטאק במרחק מסוים
lr_stack_full	lr_stack_T *s	int	O(1)	בודק האם הסטאק מלא
lr_stack_clear	lr_stack_T *s	void	O(n)	מוחק את כל הסטאק

non_terminal

NON_TERMINAL_STRUCT

שם משתנה	סוג	הסבר
type	non_terminal_E	סוג הnon-terminal
value	char *	ערך הnon-terminal

שם פונקציה	קלט	פלט	יעילות	תיאור
init_non_terminal	char *value, non_terminal_E type	non_terminal_T *	O(1)	מאתחל את הnon-terminal
non_terminal_cmp	const non_terminal_T *nt1, const non_terminal_T *nt2	int	O(1)	משווה בין שניnon-terminal

parse_tree

	PARSER_TREE_NODE_STRUCT	
שם משתנה	סוג	הסבר
symbol	symbol_T *	סימן הצומת
rule_index	ssize_t	האינדקס של החוק במערך החוקים
children	struct PARSE_TREE_NODE_STRUCT **	ילדי הצומת

n_children	size_t	מספר הילדים

PARSE_TREE_STRUCT		
שם משתנה	סוג	הסבר
root	parse_tree_node_T *	שורש העץ התחבירי
rules	rule_T **	מערך החוקים
n_rules	size_t	מספר החוקים

שם פונקציה	קלט	פלט	יעילות	תיאור
init_parse_tree	parse_tree_node_T *root, rule_T **rules, size_t n_rules	parse_tree_T *	O(1)	מאתחל את העץ התחבירי
parse_tree_free	parse_tree_node_T *tree	void	O(1)	מוחק את העץ התחבירי
init_parse_tree_node	symbol_T *sym, ssize_t rule_index, parse_tree_node_T **children, size_t n_children	parse_tree_T *	O(1)	מאתחל צומת עץ תחבירי
init_parse_tree_leaf	symbol_T *sym	parse_tree_T *	O(1)	מאתחל עלה עץ תחבירי

parse_tree_traverse_preorder	parse_tree_node_T *tree, int layer	void	O(n)	עובר על העץ התחבירי בסדר preorder
parse_tree_traverse_postorder	parse_tree_node_T *tree, int layer	void	O(n)	עובר על העץ התחבירי בסדר postorder

parser

	PARSER_STRUCT	
שם משתנה	סוג	הסבר
action	action_tbl_T *	ה-action table
go_to	goto_tbl_T *	ה-goto table
rules	rule_T **	מערך דינמי של חוקים
n_rules	size_t	מספר החוקים
stack	lr_stack_T *	סטאק הפרסר

שם פונקציה	קלט	פלט	יעילות	תיאור
init_parser	slr_T *slr	parser_T *	O(1)	מאתחל את המנתח התחבירי

מפרסר	O(n)	parse_tree_T *	parser_T *prs, queue_T *queue_tok	parse
עושה shift לסטאק	O(1)	void	parser_T *prs, int data	parser_shift
עושה reduce לסטאק	O(1)	symbol_T *	parser_T *prs, rule_T *rule	parser_reduce

rule

	RULE_STRUCT	
הסבר	סוג	שם משתנה
הצד השמאלי של החוק התחבירי	non_terminal_T *	left
הצד הימני של החוק התחבירי	symbol_T **	right
מספר הסימנים בצד הימני	size_t	right_size

תיאור	יעילות	פלט	קלט	שם פונקציה
מאתחל חוק התחבירי	O(1)	rule_T *	non_terminal_T *left, symbol_T **right, size_t right_size	init_rule

rule_cmp	const rule_T *rule1, const rule_T *rule2	int	O(1)	משווה בין שני חוקים
rule_cmp_generic	const void *rule1, const void *rule2	int	O(1)	משווה בין שני חוקים גנרים
find_first_nt	const rule_T *rule, int offset	int	O(n)	מוצא את הnon-terminal הראשון בצד ימין בחוק

slr

	SLR_STRUCT	
שם משתנה	סוג	הסבר
lr0_cc	set_T **	הסט הקאנוני של של כל הפריטים של lr(0)
lr0_cc_size	size_t	גודל הסט הקאנוני
action	action_tbl_T *	ה-action table
go_to	goto_tbl_T *	ה-goto table
grammer	grammer_T *	תחביר השפה

שם פונקציה	קלט	פלט	יעילות	תיאור

מאתחל את ה-slr	O(1)	slr_T *	set_T *lr0, grammer_T *gram	init_slr
כותב את ה-slr לקובץ בינארי	O(n)	void	slr_T *slr, char *dest	slr_write_to_bin
קורא את ה-slr לקובץ בינארי	O(n)	slr_T *	char *src	slr_read_from_bin
מאתחל את ה-slr הדיפולטי	O(1)	slr_T *	void	init_default_slr

symbol

	SYMBOL_TYPES_ENUM	
הסבר	**סוג**	**שם משתנה**
מסמן טרמינל	uint = 0	TERMINAL
מסמן non-terminal	uint = 1	NON_TERMINAL

	SYMBOL_UNION	
הסבר	**סוג**	**שם משתנה**
אסימון	token_T *	terminal
non-termial	non_terminal_T *	non_terminal

	SYMBOL_STRUCT	

שם משתנה	סוג	הסבר
sym_type	symbol_type_E	סוג הסימן
symbol	symbol_U *	ה-union המגדיר את הסימן

שם פונקציה	קלט	פלט	יעילות	תיאור
init_symbol	symbol_U *symbol, symbol_type_E type	symbol_T *	O(1)	מאתחל סימן
init_symbol_terminal	token_T *tok	symbol_T *	O(1)	מאתחל סימן עם טרמינל
init_symbol_non_terminal	non_terminal_T *nt	symbol_T *	O(1)	מאתחל סימן עם non-terminal
symbol_equals	const symbol_T *sym1, const symbol_T *sym2	int	O(1)	בודק אם הסימן שווה
symbol_cmp	const symbol_T *sym1, const symbol_T *sym2	int	O(1)	משווה בין שני סימנים
symbol_cmp_generic	const void *sym1, const void *sym2	int	O(1)	משווה בין שני סימנים

symbol_set

	SYMBOL_SET_STRUCT	
שם משתנה	סוג	הסבר

set	symbol_T **	סט של סימנים
size	size_t	גודל הסט של סימנים

שם פונקציה	קלט	פלט	יעילות	תיאור
init_symbol_set	void	symbol_set_T *	O(1)	מאתחל את סט הסימנים
init_symbol_set _with_symbols	symbol_T **syms, const size_t size	symbol_set_T *	O(n)	מאתחל את סט הסימנים עם סימנים קיימים
add_symbol	symbol_set_T *set, symbol_T *item	int	O(n)	מוסיף סימן אם הוא לא קיים בסט
remove_symbo l	symbol_set_T *set, symbol_T *item	int	O(n)	מוחק סימן בסט

semantic_analyzer

AST

ABSRACT_SYNTAX_TREE _NODE_STRUCT		
שם משתנה	סוג	הסבר
symbol	symbol_T *	סימן הצומת

children	struct ABSTRACT_SYNTAX_TREE_NODE_STRUCT **	הילדים של הצומת בעץ הסמנטי המופשט
n_children	size_t	מספר הילדים
st_entry	symbol_table_entry_T *	איבר ה-symbol table המתאים לצומת

שם פונקציה	קלט	פלט	יעילות	תיאור
init_ast_node	symbol_T *symbol, ast_node_T **children, size_t n_children	ast_node_T *	O(1)	מאתחל צומת בעץ הסמנטי
init_ast_leaf	symbol_T *symbol	ast_node_T *	O(1)	מאתחל צומת בעלה הסמנטי
ast_add_to_node	ast_node_T *ast, ast_node_T *child	void	O(1)	מוסיף ילד לצומת
traverse_ast	ast_node_T *ast, int layer	void	O(n)	מדפיס את העץ

definitions

שם פונקציה	קלט	פלט	יעילות	תיאור
defenition_…	stack_T *astack, parse_tree_node_T *tree, stack_T *st_s	void	O(1)	מתרגם צמתים בעץ התחבירי לעץ הסמנטי

sdt

	SDT_STRUCT	
שם משתנה	סוג	הסבר
definitions	semantic_rule_T **	מערך דינמי עם חוקים תחביריים
n_defenitions	size_t	גודל המערך

שם פונקציה	קלט	פלט	יעילות	תיאור
init_sdt	semantic_rule_T **definitions, size_t n_defenitions	sdt_T *	O(1)	מאתחל sdt
init_default_sdt	rule_T **rules, size_t n_rules	sdt_T *	O(n)	מאתחל את ה-sdt הדיפולטי

semantic_analyzer

שם פונקציה	קלט	פלט	יעילות	תיאור
build_ast	parse_tree_T *tree, quest_T *q	ast_node_T *	O(n)	בונה את העץ התחבירי

semantic_rule

	SEMANTIC_RULE_STRUCT	
שם משתנה	סוג	הסבר

rule	rule_T *	חוק תחבירי המתאים להגדרה הסמנטית
definition	void (*)(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s)	ההגדרה הסמנטית המתאימה לחוק התחבירי

שם פונקציה	קלט	פלט	יעילות	תיאור
init_sementic_rule	rule_T *rule, void (*)(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s)	semantic_rule_T *	O(1)	מאתחל את החוק הסמנטי

מבני נתונים

generic_set

	GENERIC_SET_NODE_STRUCT	
שם משתנה	סוג	הסבר
data	void *	הערך הגנרי של הסט
next	struct GENERIC_SET_NODE_STRUCT *	הצומת הבאה בסט

	GENERIC_SET_STRUCT	
שם משתנה	סוג	הסבר

head	set_node_T *	צומת הראש של הסט
compare	int (*)(const void*, const void*)	מצביע לפונקציית ההשוואה של הסט
size	size_t	גודל הסט

שם פונקציה	קלט	פלט	יעילות	תיאור
set_init	int (*)(const void*, const void*)	set_T *	O(1)	מאתחל את הסט הגנרי
set_add	set_T* set, void* data	int	O(1)	מוסיף איבר לסט
set_add_all	set_T* set, set_T *more_set	int	O(n)	מוסיף סט לסט
set_add_arr	set_T* set, void **more_set, size_t size	int	O(n)	מוסיף מערך ערכים לסט
set_remove	set_T* set, void* data	int	O(n)	מוחק איבר מהסט
set_contains	set_T* set, void* data	int	O(n)	בודק אם ערך מסוים נמצא בסט
set_flip	set_T* set	void	O(n)	הופך את הסט
set_free	set_T* set	void	O(n)	מוחק את הסט

hashset

	HASHSET_NODE_STRUCT	

שם משתנה	סוג	הסבר
data	void *	ערך הצומת
next	struct HASHSET_NODE_STRUCT *	הצומת הבאה

	HASHSET_NODE_STRUCT	
שם משתנה	סוג	הסבר
size	int	גודל הסט
capacity	int	הגודל המקסימלי של הסט
load_factor	float	ה-load factor של הסט
buckets	hashset_node_T **	הצמתים בסט
hash_func	unsigned int (*)(void *)	פונקציית ה-hash של הסט
compare_func	unsigned int (*)(void *)	פונקציית ההשוואה של הסט

שם פונקציה	קלט	פלט	יעילות	תיאור
init_hash_set	int initial_capacity, float load_factor, unsigned int	hashset_T *	O(1)	מאתחל את ה-hashset

			(*hash_func)(void *), int (*compare_func)(void *, void *)	
מוסיף ערך לסט	O(1)	int	hashset_T *set, void* data	hash_set_add
מוסיף סט לסט אחר	O(n)	int	hashset_T *set, hashset_T *other_set	hash_set_add_all
בודק אם ערך מסוים קיים בסט	O(1)	int	hashset_T *set, void* data	hash_set_contains
משנה את גודל הסט	O(n)	int	hashset_T *set, int new_capacity	hash_set_resize
פונקציית ההשוואה של הסט	O(1)	int	void *a, void *b	hash_set_compare
פונקציית ה-hash של הסט	O(1)	unsigned int	void *data	hash_set_hash

queue

	GENERIC_QUEUE_NODE_STRUCT	
הסבר	**סוג**	**שם משתנה**
ערך הצומת	void *	data
הצומת הבאה	struct GENERIC_QUEUE_NODE_STRUCT *	next

	GENERIC_QUEUE_STRUCT	
שם משתנה	**סוג**	**הסבר**
head	queue_node_T *	ראש הטור
tail	queue_node_T *	זנב הטור
size	int	גודל הטור

שם פונקציה	קלט	פלט	יעילות	תיאור
queue_init	void	queue_T *	O(1)	מאתחל את ה-queue
is_empty	queue_T* q	int	O(1)	בודק עם התור ריק
queue_enqueue	queue_T* q, void* data	void	O(1)	מכניס לתור ערך
queue_dequeue	queue_T* q	void *	O(1)	מוציא מהתור ערך
queue_peek	queue_T* q	void *	O(1)	מחזיר את הערך בראש התור
queue_clear	queue_T* q	void	O(n)	מוחק את כל התור
queue_size	queue_T* q	int	O(1)	מחזיר את גודל התור

stack

	STACK_NODE_STRUCT	
שם משתנה	סוג	הסבר
data	void *	ערך בצומת
next	struct STACK_NODE_STRUCT *	הצומת הבאה

	GENERIC_STACK_STRUCT	
שם משתנה	סוג	הסבר
top	stack_node_T *	ראש הסטאק
size	size_t	גודל הסטאק

שם פונקציה	קלט	פלט	יעילות	תיאור
stack_init	void	stack_T *	O(1)	מאתחל את הסטאק
stack_push	stack_T* s, void* data	void	O(1)	דוחף לראש לסטאק
stack_pop	stack_T* s	void *	O(1)	להוציא מראש הסטאק

stack_peek	stack_T* s	void *	O(1)	מחזיר את ערך בראש הסטאק
stack_clear	stack_T* s	void	O(n)	מוחק את הסטאק
stack_flip	stack_T* s	void	O(n)	הופך את הסטאק
stack_size	stack_T* s	size_t	O(1)	מחזיר את גודל הסטאק

שגיאות

שם פונקציה	קלט	פלט	יעילות	תיאור
thrw	int err	void	O(1)	זורק שגיאה מתאימה לקלט ועוצר את התכנית

hashes

שם פונקציה	קלט	פלט	יעילות	תיאור
hash_JOAAT	char *key, size_t length	uint32_t	O(1)	מימוש של ה-"Jenkins One At A Time" Hash

lexer_DFA

lexer_DFA

	LEXER_DFA_STATE_STRUCT	
שם משתנה	סוג	הסבר
index	unsigned int	האינדקס של מצב ה-DFA
lexeme	char *	ה-lexeme של המצב
type	token_type_E	סוג המצב

	LEXER_DFA_STRUCT	
שם משתנה	סוג	הסבר
DFA	short **	מטריצה המגדירה את הDFA
toks	token_T **	מערך דימני של אסימונים המתאימים ל-DFA
n_toks	unsigned int	גודל מערך האסימונים
states	lexer_dfa_state_T **	מערך דינמי של מצבי ה-DFA

המופע האחרון של כל מצב מכל סוג	lexer_dfa_state_T **	last_states
מספר המצבים	unsigned int	n_states
שם הקובץ בו נשמר ה-DFA	char *	filename
דגלים המגדירים את ההגדרות של ה-DFA	unsigned short	flags
מספר המצבים המוגדרים ע"ג הדגלים	unsigned char	n_flag_states

תיאור	יעילות	פלט	קלט	שם פונקציה
מאתחל את ה-DFA	O(n)	int	token_T **toks, const size_t n_toks, const char *DFA_filename, const char *DFA_states_filename, const char *DFA_states_details_filename, unsigned short flags	init_dfa
מאתחל את ה-DFA הדיפולטי	O(n)	int	void	init_default_dfa

transitions

תיאור	יעילות	פלט	קלט	שם פונקציה
פונקציות המוסיפות סוגים	O(1)	void	lexer_dfa_T *dfa,	add_…_transition

	lexer_dfa_state_T *src_state, int dest_state_index, int c			שונים של פונקציות מעברים ב-DFA

symbol_table

symbol_table.h

	SYMBOL_TABLE_ENTRY_STRUCT	
שם משתנה	סוג	הסבר
name	char *	שם הערך של הרשומה
type	int	סוג הרשומה
value	void *	ערך הרשומה
declaration_type	entry_type_E	סוג הצהרה
next	struct SYMBOL_TABLE_ENTRY_STRUCT *	הרשומה הבאה

	SYMBOL_TABLE_STRUCT	

הסבר	סוג	שם משתנה
גודל הטבלה	size_t	size
הגודל המקסימלי של הטבלה	size_t	capacity
ה-load factor של ה-symbol table	float	load_factor
מערך דינמי השומר את הרשומות	symbol_table_entry_T **	buckets
פונקציית ה-hash של ה-symbol table	unsigned int (*)(char *, size_t length)	hash

symbol_table_tree.h

	SYMBOL_TABLE_TREE_NODE_STRUCT	
הסבר	סוג	שם משתנה
ה-symbol table של הצומת	symbol_table_T *	table
ילדי הצומת	struct SYMBOL_TABLE_TREE_NODE_STRUCT **	children

n_children	size_t	מספר ילדי הצומת

	SYMBOL_TABLE_TREE_STRUCT	
שם משתנה	סוג	הסבר
root	symbol_table_tree_node_T *	שורש עץ ה-symbol table

שם פונקציה	קלט	פלט	יעילות	תיאור
init_symbol_table_tree	symbol_table_tree_node_T *root	symbol_table_tree_T *	O(1)	מאתחל את עץ ה-symbol table
init_symbol_table_tree_node	symbol_table_T *table, symbol_table_tree_node_T **children, size_t n_children	symbol_table_tree_T *	O(1)	מאתחל צומת בעץ ה-symbol table
init_symbol_table_tree_leaf	symbol_table_T *table	symbol_table_tree_T *	O(1)	מאתחל עלה בעץ ה-symbol table
symbol_table_tree_node_add	symbol_table_tree_node_T *sttn, symbol_table_tree_node_T *sttn_child	void	O(1)	מוסיף ילד לצומת בעץ ה-symbol table
symbol_table_tree_print	symbol_table_tree_T *stt	void	O(n)	מדפיס את עץ ה-symbol table

מודולים ראשיים

io

שם פונקציה	קלט	פלט	יעילות	תיאור
read_file	const char *filename	char *	O(n)	קורה את הערך בתוך קובץ
write_file	const char *filename, const char *content	void	O(1)	כותב לקובץ
write_file_append	const char *filename, const char *content	void	O(1)	מוסיף לקובץ
print_to_stdout	char *content	void	O(1)	כותב ל-stdout
print_to_stderr	char *content	void	O(1)	כותב ל-stderr
get_new_filename	const char *filename, const char *new_name	char *	O(1)	מחזיר את שם הקובץ המתאים

lang

שם פונקציה	קלט	פלט	יעילות	תיאור
init_quest	const char *filename	quest_T	O(1)	מאתחל את המבנה המחזיק מבנים חשובים

quest

	QUEST_STRUCT	
שם משתנה	סוג	הסבר
srcfile	char *	שם קובץ תכנית המקור
destfile	char *	שם קובץ תכנית המטרה
src	char *	תכנית המקור
lexer	lexer_T *	הלקסר
parser	parser_T *	הפרסר
sdt	sdt_T *	ה-sdt
code_gen	code_gen_T *	יוצר הקוד

שם פונקציה	קלט	פלט	יעילות	תיאור
compile	quest_T *q	void	O(n)	מקמפל
compile_file	const char *filename	void	O(n)	מקמפל קובץ

התוכנית הראשית

```
1.  function main()
2.      input_file = get_input_file_name()
3.      output_file = get_output_file_name()
4.
5.      try
6.          source_code = read_source_code(input_file)
7.
8.          tokens = tokenize(source_code)  // Break code into tokens
9.          syntax_tree = parse(tokens)  // Analyze grammatical structure
10.
11.         target_code = generate_code(syntax_tree)  // Generate target machine
    code
12.         write_output(target_code, output_file)
13.
14.     catch error as e
15.         report_error(e)
16.
17. end function
```

מה התוכנית הראשית עושה

- בשורות 1 ו-2 התוכנית מקבלת את שמות קבצי הקלט והפלט.
- שורה 6 קוראת את קוד המקור, ומכניסה את הקוד לתוך משתנה כמחרוזת של תווים.
- שורה 8 קוראת לפונקציה tokenize, כלומר למנתח המילוני, ומכניסה לתוך המשתנה tokens את כל האסימונים המעובדים מהניתוח.
- שורה 9 קוראת לפונקציה parse, כלומר למנתח התחבירי והסמנטי, המקבלת תעבורה של אסימונים, ומכניסה לתוך המשתנה syntax_tree עץ תחביר.
- שורה 11 קוראת לפונקציה generate_code, כלומר ליוצר קוד הביניים והמטרה, ויוצרת את הקוד.
- שורה 12 כותבת את קוד המטרה לקובץ הפלט.
- שורה 5 ו14 אחראיות לתפוס כל שגיאה, ולתפל בא כראוי, כלומר היא מתנהגת כמו הerror handler
- בנוסף, כל הפונקציות השתמשו בsymbol table.

מדריך למשתמש

דרישות

- הקומפיילר, שאפשר להוריד מדף הגיטהאב של הפרויקט, או שאתם יכולים לקליד את הפקודה הבא ל-CLI:

```
git clone https://github.com/EthanChartoff/Quest
```

- cmake, make, gcc ו-nasm שאפשר להוריד את כולם בדפים שלהם באינטרנט

בניית הקומפיילר

בטרמינל שלכם, תעברו לתיקיית ה-build של הפרויקט ותכתבו את הפקודה הבאה:

```
cmake .
```

קימפול והרצת התכנית

תוכלו להריץ את הקומפיילר בעזרת קובץ ההרצה qc, הנמצא ב-build. כלומר, בשביל לקמפל, תריצו את הפקודה הבא:

```
./qc <path_to_program>
```

לאחר מכאן, תוכלו לעשות אסמבל לקובץ בעזרת nasm ולעשות link בעזרת ld, כך:

```
nasm -felf64 <path_to_asm>
ld <path_to_object>
```

ותקבלו קובץ הרצה!

רפלקציה

קודם כל אני חייב לאמר שאני לא הבן אדם הכי טוב עם מילים, לכן אני חושב שבשביל באמת להבין איך היה התהליך הזה, עליכם לכתוב קומפיילר בעצמכם. למרות זאת, הנה הניסיון הכי טוב שלי להסביר את התהליך במילים.

כיצד הייתה עבורי העבודה על הפרויקט

שנכנסתי לפרויקט הזה, חשבתי שאני יכול לעשות אותו כמו כל שאר הפרויקטים שלי עד היום, כלומר לפתוח פרויקט חדש ולהתחיל לתכנת סתם ככה עד שאני מוצא אלגוריתם שעובד. אז זה מה שעשיתי באמת. שסיימתי את הפרוטוטייפ ללקסר, שהיה פשוט switch גדול, הגעתי למנחה שלי ואמרתי לו מה עשיתי. שהוא שמע מה עשיתי הוא אמר לי "אתה לא יכול לעשות ככה, אתה צריך להשתמש באוטומט!" ואני לא הבנתי על מה הוא מדבר. כן למדתי על אוטומטים בכיתה יא אז הבנתי מה זה, אבל לא הבנתי איך שני הנושאים קשורים. לכן נכנסתי לספרות בשביל להבין מה קורה, והבנתי שהפרויקט הזה לא יהיה כמו כל שאר הפרויקטים שעשיתי.

במהלך קריאת התאוריה, שזה חודש אחרי תחילת הפרויקט שלי (עדיין לא הבנתי את העומס של מכללה ביחס לתיכון), הבנתי שאני לא יכול סתם לתכנת עד שאני מוצא משהו שעובד, ושאני חייב לעבוד עם מבני נתונים ואלגוריתמים בצורה חכמה.

כשהבנתי את החומר זה הרגיש וניגשתי לעבוד זה הרגיש כמו הפעם הראשונה שאני באמת מתכנת, אבל באמת. כל שאר הפעמים אני למדתי על מבני נתונים, אלגוריתמים וכו, אבל עכשיו אני באמת בונה תוכנה, שמשתמשת בכל המושגים האלה בשביל לבנות מערכת ענקית אך חכמה ויעילה.

העבודה על הפרויקט הייתה קשה יותר מכל דבר תכנותי שעשיתי אי-פעם. הנושאים היו מסובכים, העבודה הייתה קשה וארוכה, אך זה שווה את זה רק בשביל לראות שהכל רץ.

מה קיבלתי מהפרויקט

קיבלתי שלושה דברים עיקריים מהפרויקט:

- אני עכשיו יודע איך קומפיילרים עובדים.
- אני עכשיו יודע איך לעבוד על פרויקט גדול מאוד עם מספר שלבים אלגוריתמים.
- אני עכשיו יודע את המהוט של עבודה עם אלגוריתמים ומבני נתונים.

איך קומפיילרים עובדים

המטרה העיקרית שלי שהתחלתי את הפרויקט הייתה להבין איך קומפיילרים עובדים. עכשיו אני יודע. אם מישהו יאמר "אוקי אני יודע איך תוכנה עובדת, אבל איך תוכנה *באמת* עובדת?" אני יוכל לענות לו (ואחרי שיחה ארוכה מאוד הוא יתחרט שהוא שאל את זה). אבל ברצינות, לדעת איך קומפיילרים עובדים פותח בשבילי את הצעד הבא כמתכנת. עכשיו שאני רושם קוד, אני לא רק שם לב להוראות, אלה אני לוקח בחשבון את התהליכים שקוראים ברקע.

איך לעבוד על פרויקט גדול

אף פעם לא עבדתי על פרויקט כלכך גדול. בפרויקט יש מספר שלבים, עם הרבה אלגוריתמים ומבנים שצריכים דוקומנטציה וסידור. לכן למדתי להסתדר עם זה. איך לעבוד עם הרבה קבצים, איך לסדר אותם, איך לא לשכוח מה פונקציה מסוימת עושה ואיך לרשום דוקומנטציה שימושית שלא מרגישה כמו בזבוז.

מהוט העבודה עם אלגוריתמים ומבני נתונים

עד הפרויקט הזה לא הבנתי איך אלגוריתמים ומבני נתונים מתחברים, ולמה הם חיוניים לקוד. הפרויקט הזה לימד אותי איך להשתמש בשני הכלים העלה יחד בשביל ליצור מכונה יעילה. כן הייתי יכול לעשות לקסר שמשתמש ב-switch גדול, וזה כנראה מה שהייתי עושה לפני הפרויקט, אבל למדתי להשתמש במבנה נתונים ולעשות תוכנה טובה.

הכלים שאני לוקח איתי להמשך

הכלי העיקרי שאני לוקח איתי להמשך הוא למידה עצמית. בשביל פרויקט מסוג חייבים הרבה סבלנות וריכוז בשביל להעמיק בחומר וכנראה בזכות הקושי של האתגר לקחתי את עצמי בידיים והבנתי שאם אני רוצה להצליח אני צריך לשבת ולקרוא עד שאני מבין, וזה מה שעשיתי. מכיוון שעוד מעט אני הולך לצבא זה אולי הכלי הכי חשוב שהייתי יכול לקבל עכשיו. אני מגיע לפרק חדש בחיים שלי שאין בו מורים ומרצים, לכן היכולת להבין חומרים קשים לבד היא דבר הכרחי.

בנוסף, למדתי איך לתכנן בקפדנות קוד ואיך לעבוד על פרויקט גדול. ניהול פרויקט כזה גדול יעזור לי בעתיד כאשר אני יעבוד על קוד מקצועי עם קבוצה.

האתגרים שעמדו בפני

קודם כל, האתגר הכי גדול שעמד בפני זה הזמן. לא רק שאני במכללה לומד מדעי המחשב, אלה גם אני מקבל עבודות מהצבא, מכיוון שאני מתמיין ליחידת סיגיט. מה הייתי יכול לעשות בשביל למנוע את זה? אני חושב שהייתי יכול לסדר את הזמן שלי יותר טוב. למרות שהיו זמנים שעבדתי קשה בשביל לסיים את הפרויקט, הלו"ז שלי לא היה מסודר, ולא נתתי לעצמי לעבוד על הפרויקט בנוסף ללמוד לבחינה או לעשות עבודה. למרות זאת אני לא חושב שעבדתי מעט, להפך, זה היה הפרויקט הגדול והמסובך ביותר שעבדתי אליו והוא קיבל את הזמן והיכולת המתאימים לו.

בנוסף היה את האתגר של כמות החומר והרמה שלו. קראתי שני ספרים מעל 500 עמודים בשביל הפרויקט הזה שהחומר בהם הוא בסדר גודל מעל החומר הלימודי במכללה, לכן לא היה קל להתמודד עם זה. למרות זאת, אני חושב שעשיתי עבודה טובה בנוגע לנושא הזה, אני בקיא בחומר ולמדתי הרבה.

האתגר האחרון הוא גודל הפרויקט. לא חשבתי שהפרויקט יהיה כל כך גדול ועם כמות המבנים והאלגוריתמים שהיו בו. בשביל להתמודד רשמתי הרבה דוקומנטציה והשתמשתי בהרבה separation of concerns, מה שבסופו של יום עבד.

מה הייתי עושה אחרת לו הייתי מתחיל היום

אני מנחש שכאשר אתחיל מחדש אין לי שום מהידע שיש לי היום לגבי קומפיילרים. לכן, קודם כל הייתי קורא את כל הספרות שקראתי בשביל הפרויקט לפני שאני נוגע בפרויקט. בנוסף, הייתי מתכנן בעזרת פסאודו קוד ותרשימים את מבנה הפרויקט ואת האלגוריתמים, ורק אחרי שהכל מוכן הייתי מתחיל לתכנת.

יתר על כן הייתי עושה לו"ז שמראה את כל התהליך.

לאחר מכן שאני רושם את הקוד הייתי מוסיף בהרבה יותר דוקומנטציה. לא שיש לי קצת דוקומנטציה, פשוט עוד דוקומנטציה לא פגעה באף אחד.

מה הייתי עושה לו הייתי יכול להמשיך את הפרויקט

אם הייתי יכול להמשיך את הפרויקט הייתי עובד בעיקר על אופטימיזציה. לקראת הסוף התחלתי לקרוא על אופטימיזציה, עם התקווה שלמרות שזה לא הכרחי הייתי מוסיף קצת, וזה הנושא שהכי מעניין אותי עד כו בקומפיילרים.

בנוסף, הייתי מוסיף מערכים, מה שמוסיף לשפה המון תכניות חדשות.

מה אני ממליץ למי שחושב לעשות קומפיילר

קודם כל, לקרוא. אני יודע שבהתחלה שאני התחלתי לקרוא חשבתי שלא לא הכרחי, אך אחרי שעשיתי קומפיילר אני יכול לאמר שבלי זה אין מצב שהייתי מסיים את הקומפיילר, ואם כן, הקומפיילר היה גרוע.

לאחר מכן, תתכננו מה אתם עומדים לעשות. לא משנה איך אתם עושים את זה, רק שאתם תבינו מה אתם עומדים לעשות. להיכנס לקוד בלי שום רעיון תכנותי זו דרך ישירה לבאגים וסבל.

ולבסוף, אל תתקמצנו על דוקומנטציה. יש סיכוי לא קטן שאתם תצטרכו לחזור לחלק בקומפיילר שלא חשבתם עליו חודשיים ולא תזכרו כלום. לכן דוקומנטציה זה מאוד חשוב.

ביבליוגרפיה

- Aho, Alfred V., Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. **Compilers: Principles, Techniques, and Tools. (**2nd ed. Addison-Wesley, 2006).
- Cooper, Keith D., and Linda Torczon. ***Engineering a Compiler*.** (2nd ed. Morgan Kaufmann, 2011.)
- NASM Documentation Team. **"The Netwide Assembler (NASM) Manual."** retrieved from: https://nasm.us/xdoc/2.16rc12/html/nasmdoc1.html

קוד הפרויקט

```
#include "../include/parser/goto_table.h"
#include "../utils/err/err.h"
#include <stdio.h>
#include <stdlib.h>

goto_tbl_T *init_goto_tbl(non_terminal_T **non_terminals, size_t n_non_terminals, size_t
n_states) {
    int i;
    goto_tbl_T *gotot = malloc(sizeof(goto_tbl_T));
    if(!gotot)
        thrw(ALLOC_ERR);

    gotot->gotos = malloc(sizeof(int *) * n_states);
    if(!gotot->gotos)
        thrw(ALLOC_ERR);

    for(i = 0; i < n_states; ++i) {
        gotot->gotos[i] = malloc(sizeof(int) * n_non_terminals);
        if(!gotot->gotos[i])
            thrw(ALLOC_ERR);
    }
    gotot->non_terminals = non_terminals;
    gotot->n_non_terminals = n_non_terminals;
    gotot->n_states = n_states;

    return gotot;
}

size_t goto_tbl_find_non_terminal(goto_tbl_T *tbl, non_terminal_T *nterm) {
    int i;

    for(i = 0; i < tbl->n_non_terminals; ++i) {
        if(tbl->non_terminals[i]->type == nterm->type) {
            return i;
        }
    }
    return -1;
}

void goto_tbl_print_to_file(goto_tbl_T *tbl, char *dest) {
```

```
;int i, j
;("FILE *fp = fopen(dest, "w

} (for(i = 0; i < tbl->n_states; ++i
} (for(j = 0; j < tbl->n_non_terminals; ++j
;([fprintf(fp, "%3d ", tbl->gotos[i][j
{
;("fprintf(fp, "\n
{

;(fclose(fp
{

} (void goto_tbl_pretty_print_to_file(goto_tbl_T *tbl, char *dest
;int i, j
;("FILE *fp = fopen(dest, "w

print header //
;("|   " ,fprintf(fp
} (for(i = 0; i < tbl->n_non_terminals; ++i
;(fprintf(fp, "%20s ", tbl->non_terminals[i]->value
{
;("fprintf(fp, "\n

;("|   " ,fprintf(fp
} (for(i = 0; i < tbl->n_non_terminals; ++i
;("_____" ,fprintf(fp
{
;("fprintf(fp, "\n

} (for(i = 0; i < tbl->n_states; ++i
;(fprintf(fp, " %2d|", i
} (for(j = 0; j < tbl->n_non_terminals; ++j
;([fprintf(fp, "%20d ", tbl->gotos[i][j
{
;("fprintf(fp, "\n
{

;(fclose(fp
"include "../include/parser/lr_item.h#{
"include "../include/macros.h#
<include <stdio.h#
<include <stdlib.h#
```

```
} (lr_item_T *init_lr_item(rule_T *rule, size_t dot_index, set_T *lookaheads
    ;((lr_item_T *item = malloc(sizeof(lr_item_T

    ;item->rule = rule
    ;item->dot_index = dot_index
    ;item->lookaheads = lookaheads

    ;return item
{

} (int lr_item_cmp(const lr_item_T *item1, const lr_item_T *item2
    ;int delta, i
    ;set_node_T *cur1, *cur2

    ;(delta = rule_cmp(item1->rule, item2->rule
    (IF_SIGN(delta

    ;delta = item1->dot_index - item2->dot_index
    ;(IF_SIGN(delta

    ;return 0
{

} (int lr_item_cmp_generic(const void *item1, const void *item2
    ;(return lr_item_cmp((const lr_item_T *) item1, (const lr_item_T *) item2
{

} (int lr_item_set_cmp(const set_T *set1, const set_T *set2
    ;int i, delta
    ;set_node_T *cur_set1, *cur_set2

    ;delta = set1->size - set2->size
    ;(IF_SIGN(delta

    ;cur_set1 = set1->head
    ;cur_set2 = set2->head
    } (for(i = 0; i < set1->size; ++i
    ;(delta = lr_item_cmp_generic(cur_set1->data, cur_set2->data
    ;(IF_SIGN(delta

    ;cur_set1 = cur_set1->next
    ;cur_set2 = cur_set2->next
```

```
{

;return 0
{

} (int lr_item_set_cmp_generic(const void *item1, const void *item2
;(return lr_item_set_cmp((const set_T *) item1, (const set_T *) item2
{

} (set_T *first(const grammer_T *gram, const symbol_T *sym
;int i
;(set_T *first_set = set_init(token_cmp_generic
;set_node_T *cn
;rule_T *cur_rule

if symbol is term, return it //
} (if(sym->sym_type == TERMINAL
;(set_add(first_set, sym->symbol->terminal
{
sym is non-term //
} else
;cn = gram->rules->head
} (for(i = 0; i < gram->rules->size; ++i
;cur_rule = cn->data

get rules of symbol and see what is there start symbol //
} ((if(!non_terminal_cmp(sym->symbol->non_terminal, cur_rule->left
} (if(sym->symbol->non_terminal != cur_rule->right[0]->symbol->non_terminal
;(([set_add_all(first_set, first(gram, cur_rule->right[0
{
{

;cn = cn->next
{
{

;return first_set
{

} (set_T *follow(const grammer_T *gram, const non_terminal_T *nt
;int i, j
;(set_T *follow_set = set_init(token_cmp_generic
;set_node_T *cn
;rule_T *cur_rule
```

```
(if(nt->type == NON_TERM_start
;((set_add(follow_set, init_token("$", TOK_eof

;cn = gram->rules->head
} (for(i = 0; i < gram->rules->size; ++i
;cur_rule = cn->data

get immidiate terminals, meanining terminals in the grammer that follow the nt //
} (for(j = 0; j < cur_rule->right_size; ++j

if(cur_rule->right[j]->sym_type == NON_TERMINAL
(non_terminal_cmp(nt, cur_rule->right[j]->symbol->non_terminal! &&
} (j + 1 < cur_rule->right_size &&
;(([set_add_all(follow_set, first(gram, cur_rule->right[j + 1
{
{

get non-immidiate terminals, meaning follow of non-terminal ending symbol //
if(cur_rule->right[cur_rule->right_size - 1]->sym_type == NON_TERMINAL
non_terminal_cmp(cur_rule->right[cur_rule->right_size - 1]->symbol->non_terminal,! &&
} ((nt
;(printf("%s\n", nt->value //

;((set_add_all(follow_set, follow(gram, cur_rule->left
{

;cn = cn->next
{

;return follow_set
{

Closure of Item Sets //
//
If I is a set of items for a grammar G, then CLOSURE(I) is the set of items //
:constructed from I by the two rules //
.(Initially, add every item in I to CLOSURE(I .1 //
If A -> a.Bb is in CLOSURE(I) and B ! is a production, then add the .2 //
.item B -> .y to CLOSURE(I), if it is not already there //
} (set_T *closure(grammer_T *grammer, set_T *items
;int changed_c, found, i, j, k
;(set_T *closure_items = set_init(lr_item_cmp_generic
```

```
;set_node_T *cn_closure_item, *cn_gram_rule
;lr_item_T *tmp, *cur_item
;rule_T *cur_gram_rule


;(set_add_all(closure_items, items

} do
;changed_c = 0
;cn_closure_item = closure_items->head
go over all closure items //
} (for(i = 0; i < closure_items->size && cn_closure_item != NULL; ++i
;cur_item = cn_closure_item->data

;cn_gram_rule = grammer->rules->head
} (for(j = 0; j < grammer->rules->size; ++j
;cur_gram_rule = cn_gram_rule->data

)if
cur_item->dot_index < cur_item->rule->right_size
cur_gram_rule->left->type == &&
} (cur_item->rule->right[cur_item->dot_index]->symbol->non_terminal->type

)changed_c += set_add
,closure_items
)init_lr_item
,cur_gram_rule
,0
NULL
(
;(
{
;cn_gram_rule = cn_gram_rule->next
{
;cn_closure_item = cn_closure_item->next
{
;(while(changed_c {

;return closure_items
{

} (set_T *go_to(grammer_T *grammer, set_T *items, symbol_T *symbol
;int i
;(set_T *goto_items = set_init(lr_item_cmp_generic
```

```
;set_node_T *cn_closure_item
;lr_item_T *tmp, *cur_item

;cn_closure_item = items->head
} (for(i = 0; i < items->size && cn_closure_item != NULL; ++i
;cur_item = cn_closure_item->data

)if
cur_item->dot_index < cur_item->rule->right_size
} ((symbol_equals(cur_item->rule->right[cur_item->dot_index], symbol &&

)set_add
,goto_items
(init_lr_item(cur_item->rule, cur_item->dot_index + 1, NULL
;(
{
;cn_closure_item = cn_closure_item->next

{
;(return closure(grammer, goto_items
{


} (set_T *lr0_items(grammer_T *grammer, lr_item_T *starting_item
;(set_T *lr1_items, *goto_set, *tmp = set_init(lr_item_cmp_generic
;(set_T *lr1 = set_init(lr_item_set_cmp_generic
;set_node_T *cn_item, *cn_sym
;symbol_T *cur_sym

;int changed, i, j, count, c

;(set_add(tmp, starting_item
;(lr1_items = closure(grammer, tmp
;(set_add(lr1, lr1_items

} do
;changed = 0

;cn_item = lr1->head
} (for(i = 0; i < lr1->size && cn_item != NULL; ++i
;lr1_items = cn_item->data
;cn_sym = grammer->symbols->head
} (for (j = 0; j < grammer->symbols->size; ++j
;cur_sym = cn_sym->data
```

```
)if
goto_set = go_to(grammer, lr1_items, cur_sym))->size != 0)

count = set_add(lr1, goto_set)) != 0) &&
} (
;changed = 1
{
;cn_sym = cn_sym->next
{
;cn_item = cn_item->next
{
;(while (changed {


;return lr1
{



"include "../include/parser/bnf.h#
"include "../utils/err/err.h#
"include "../utils/DS/include/generic_set.h#
"include "../include/macros.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#

} (static int strcmp_generic(const void *str1, const void *str2
;(int delta = strcmp(str1, str2

;return delta ? delta / abs(delta) : delta
{

} (static void replace_char(char *str, char to_replace, char replace_with
;int i
} (++for (i = 0; str[i] != '\0'; i
} (if (str[i] == to_replace
;str[i] = replace_with
{
{
{

} (static void handle_left(char *left, set_T *left_set, FILE *dest
;('_' ,'-' ,replace_char(left
```

```
print dest //
;(fprintf(dest, "NON_TERM(%s, \"%s\")\n", left, left

;(set_add(left_set, left
{

} (static void handle_right(char *left, char *right, FILE *xlat
;(" " ,char *cur_sym = strtok(right
;char **whole_right

} (while(cur_sym
;(" " ,cur_sym = strtok(NULL
{
{

} (static void print_syms(set_T *s, FILE *xlat
;char *cur_left
;int i
;set_node_T *cn = s->head

fprintf(xlat, "symbol_T *start = init_symbol_non_terminal(init_non_terminal(\"S'\",
;("NON_TERM_start));\n

} (for(i = 0; i < s->size; ++i
;cur_left = cn->data

,fprintf(xlat
symbol_T *%s = init_symbol_non_terminal(init_non_terminal(\"%s\","
,"NON_TERM_%s));\n
,cur_left
,cur_left
;(cur_left

;cn = cn->next
{

;("fprintf(xlat, "\n
;("fprintf(xlat, "set_add(syms, start);\n

;cn = s->head
} (for(i = 0; i < s->size; ++i
;cur_left = cn->data
```

```
,fprintf(xlat
,"set_add(syms, %s);\n"
;(cur_left

;cn = cn->next
{
{

} (void bnf_make_non_terminals(char *src, char *dest
;FILE *fp_src, *fp_dest, *fp_xlat
;char *line = NULL
;size_t len = 0
;ssize_t read = 1
;char* buffer
;(set_T *left_set = set_init(strcmp_generic

initialize values //
;("fp_src = fopen(src, "r
(if(!fp_src
;(thrw(OPEN_FILE_ERR

;("fp_dest = fopen(dest, "w
(if(!fp_dest
;(thrw(OPEN_FILE_ERR

;("fp_xlat = fopen(PARSER_BNF_XLAT, "w
(if(!fp_xlat
;(thrw(OPEN_FILE_ERR

;((buffer = malloc(sizeof(char
(if(!buffer
;(thrw(ALLOC_ERR
;'buffer[0] = '\0

header //
;(fprintf(fp_dest, "%s\n", PARSER_NON_TERMINALS_HEADER
;("fprintf(fp_xlat, "set_T *syms = set_init(symbol_cmp_generic);\n


} ((while (read != -1 && (read = getline(&line, &len, fp_src) != -1
} (if(strlen(line) > 1
;(("=:: " ,buffer = strdup(strtok(line
;(handle_left(buffer, left_set, fp_dest
```

```
                                handle right //
;(handle_right(buffer, strtok(NULL, "") + 4, fp_xlat //
} (while ((read = getline(&line, &len, fp_src) != -1) && strlen(line) > 1
                                ;("|" ,strtok(line //
;(handle_right(buffer, strtok(NULL, "") + 1, fp_xlat //
                                {
                                {
                                {

                                ;(print_syms(left_set, fp_xlat

                                footer //
;(fprintf(fp_dest, "%s", PARSER_NON_TERMINALS_FOOTER


                                ;(fclose(fp_src
                                ;(fclose(fp_dest
                                ;(fclose(fp_xlat
                                (if(line
                                ;(free(line
                                {
"include "../include/parser/non_terminal.h#
"include "../include/macros.h#
<include <stdlib.h#
<include <string.h#

} (non_terminal_T *init_non_terminal(char *value, non_terminal_E type
                                ;((non_terminal_T *nt = malloc(sizeof(non_terminal_T

                                ;nt->type = type
                                ;(nt->value = strdup(value
                                ;return nt
                                {

} (int non_terminal_cmp(const non_terminal_T *nt1, const non_terminal_T *nt2
                                ;int delta

                                ;delta = nt1->type - nt2->type
                                ;(IF_SIGN(delta

                                ;(delta = strcmp(nt1->value, nt2->value
                                ;(IF_SIGN(delta

                                ;return 0
```

```
"include "../include/parser/rule.h#{
"include "../include/macros.h#
<include <stdio.h#
<include <stdlib.h#


initialize rule using tokens //
} (rule_T *init_rule(non_terminal_T *left, symbol_T **right, size_t right_size
;((rule_T *rule = malloc(sizeof(rule_T

} (if(!rule
;("printf("cant allocate memory for rule
;(exit(EXIT_FAILURE
{

;rule->left = left
;rule->right = right
;rule->right_size = right_size

;return rule
{

} (int rule_cmp(const rule_T *rule1, const rule_T *rule2
;int i, delta

;delta = rule1->left->type - rule2->left->type
;((IF_SIGN((delta


;delta = rule1->right_size - rule2->right_size
;(IF_SIGN(delta

} (for(i = 0; i < rule1->right_size; ++i
;[delta = rule1->right[i] - rule2->right[i
;(IF_SIGN(delta
{

;return 0
{

} (int rule_cmp_generic(const void *rule1, const void *rule2
;(return rule_cmp((const rule_T *) rule1, (const rule_T *) rule2
{
```

```
} (int find_first_nt(const rule_T *rule, int offset
;int i

} (for(i = offset; i < rule->right_size; ++i
(if(rule->right[i]->sym_type == NON_TERMINAL
;return i
{
;return -1
"include "../include/parser/symbol.h#{
<include <stdlib.h#

} (symbol_T *init_symbol_tok(symbol_U *symbol, symbol_type_E type
;((symbol_T *sym = malloc(sizeof(symbol_T

;sym->symbol = symbol
;sym->sym_type = type

;return sym
{

} (symbol_T *init_symbol_terminal(token_T *tok
;((symbol_T *sym = malloc(sizeof(symbol_T

;((sym->symbol = malloc(sizeof(symbol_U
;sym->symbol->terminal = tok
;sym->sym_type = TERMINAL

;return sym
{

} (symbol_T *init_symbol_non_terminal(non_terminal_T *nt
;((symbol_T *sym = malloc(sizeof(symbol_T

;((sym->symbol = malloc(sizeof(symbol_U
;sym->symbol->non_terminal = nt
;sym->sym_type = NON_TERMINAL

;return sym
{

} (int symbol_equals(const symbol_T *sym1, const symbol_T *sym2
return sym1->sym_type == sym2->sym_type
sym1->sym_type == TERMINAL ?
sym1->symbol->terminal->type == sym2->symbol->terminal->type ?
```

```
sym1->symbol->non_terminal->type == sym2->symbol->non_terminal->type :
;0 :
{

} (int symbol_cmp(const symbol_T *sym1, const symbol_T *sym2
return sym1->sym_type == sym2->sym_type
sym1->sym_type == TERMINAL ?
(token_cmp(sym1->symbol->terminal, sym2->symbol->terminal ?
(non_terminal_cmp(sym1->symbol->non_terminal, sym2->symbol->non_terminal :
;sym1->sym_type - sym2->sym_type :
{

} (int symbol_cmp_generic(const void *sym1, const void *sym2
;(return symbol_cmp((const symbol_T *) sym1, (const symbol_T *) sym2
===----------= slr.h =--===//{
//
,In this file will be all functions relating to building the slr parser //
.including action and goto tables //
//
===----------===//

"include "../include/parser/slr.h#
"include "../include/parser/lr_item.h#
"include "../include/parser/bnf.h#
"include "../include/macros.h#
"include "../utils/err/err.h#
<include <string.h#
<include <stdlib.h#
<include <stdio.h#
<include <string.h#
<include <threads.h#

.finds items goto index inside cc arr //
static size_t find_goto_index(grammer_T *gram, set_T *items, symbol_T *symbol, set_T **cc,
} (size_t cc_size
;size_t i
;(set_T *gt_set = go_to(gram, items, symbol

} (for(i = 0; i < cc_size; ++i
((if(!lr_item_set_cmp(cc[i], gt_set
;return i
{

;return -1
```

```
{

} (static goto_tbl_T *init_goto_table(slr_T *lr0
;int i, j, gti
;set_T *itemset
;set_node_T *cur_itemset_node, *cn_item
;lr_item_T *item
)goto_tbl_T *gotos = init_goto_tbl
,(non_terminals_in_symbol_set(lr0->grammer->symbols
,(n_non_terminals_in_symbol_set(lr0->grammer->symbols
;(lr0->lr0_cc_size

go over collection //
} (for (i = 0; i < lr0->lr0_cc_size && lr0->lr0_cc[i]; ++i
;[itemset = lr0->lr0_cc[i

} (for(j = 0; j < gotos->n_non_terminals; ++j
)gti = find_goto_index
,lr0->grammer
,itemset
,([init_symbol_non_terminal(gotos->non_terminals[j
,lr0->lr0_cc
;(lr0->lr0_cc_size


;gotos->gotos[i][goto_tbl_find_non_terminal(gotos, gotos->non_terminals[j])] = gti

{
{

;return gotos
{

} (static action_tbl_T *init_action_table(slr_T *lr0
;int i, j, k, tmp
;[char action[5
;action_tbl_T *actions
;set_T *itemset, *followset
;set_node_T *cur_itemset_node, *cn_item, *cn_term
;lr_item_T *item
;token_T *cur_term

)actions = init_action_tbl
```

```
,(terminals_in_symbol_set_and_dollar(lr0->grammer->symbols
,n_terminals_in_symbol_set(lr0->grammer->symbols) + 1
;(lr0->lr0_cc_size

go over collection //
} (for(i = 0; i < lr0->lr0_cc_size; ++i
;[itemset = lr0->lr0_cc[i

go ovet items in set //
;cn_item = itemset->head

} (for(j = 0; j < itemset->size; ++j
;item = cn_item->data

If [A -> a.ab ] is in Ii and GOTO(Ii, a) = Ij , then set ACTION[i, a] to //
.shift j. Here a must be a terminal //
if(item->dot_index < item->rule->right_size
} (item->rule->right[item->dot_index]->sym_type == TERMINAL &&
)tmp = find_goto_index
,lr0->grammer
,itemset
,[item->rule->right[item->dot_index
,lr0->lr0_cc
;(lr0->lr0_cc_size

} (if(tmp != -1
;(sprintf(action, "s%d", tmp
actions->actions[i][action_tbl_find_terminal(actions,
;(item->rule->right[item->dot_index]->symbol->terminal)] = strdup(action
{
{
If [A -> a.] is in Ii , then set ACTION[i, a] to "reduce A -> a" for all //
.'a in FOLLOW(A); here A may not be S //
else if(item->dot_index == item->rule->right_size && item->rule->left->type !=
} (NON_TERM_start

;(followset = follow(lr0->grammer, item->rule->left
tmp = find_right_grammer_index(lr0->grammer, item->rule->right,
;(item->rule->right_size

} (if(tmp != -1
;(sprintf(action, "r%d", tmp

;cn_term = followset->head
```

```
} (for(k = 0; k < followset->size; ++k
;cur_term = cn_term->data

;(actions->actions[i][action_tbl_find_terminal(actions, cur_term)] = strdup(action

;cn_term = cn_term->next
{
{
{
.If [S -> S.] is in Ii , then set ACTION[i, $] to accept //
else if(item->rule->left->type == NON_TERM_start && item->dot_index ==
} (item->rule->right_size
;("sprintf(action, "acc
;((tmp = action_tbl_find_terminal(actions, init_token("$", TOK_eof
;(actions->actions[i][tmp] = strdup(action
{

;cn_item = cn_item->next
{
{

} (for(i = 0; i < actions->n_states; ++i
} (for(j = 0; j < actions->n_terminals; ++j
([if(!actions->actions[i][j
;("actions->actions[i][j] = strdup("e
{
{

;return actions
{

} (static set_T **lr0_set_to_arr(const set_T *lr0
;int i
;set_T **lr0_arr = malloc(sizeof(set_T *) * lr0->size), *curr_item_set
(if(!lr0_arr
;(thrw(ALLOC_ERR
;set_node_T *cn = lr0->head

} (for(i = 0; i < lr0->size; ++i
;curr_item_set = cn->data

;((lr0_arr[lr0->size - i - 1] = malloc(sizeof(set_T
([if(!lr0_arr[lr0->size - i - 1
;(thrw(ALLOC_ERR
```

```
;lr0_arr[lr0->size - i - 1]->size = curr_item_set->size
;lr0_arr[lr0->size - i - 1]->compare = curr_item_set->compare
;lr0_arr[lr0->size - i - 1]->head = curr_item_set->head

;cn = cn->next
{

;return lr0_arr
{

} (slr_T *init_slr(set_T *lr0, grammer_T *gram
;((slr_T *slr = malloc(sizeof(slr_T
(if(!slr
;(thrw(ALLOC_ERR

;slr->grammer = gram
;(slr->lr0_cc = lr0_set_to_arr(lr0

;slr->lr0_cc_size = lr0->size
;(slr->action = init_action_table(slr
;(slr->go_to = init_goto_table(slr

;return slr
{

} (void slr_write_to_bin(slr_T *slr, char *dest
;("FILE *fp = fopen(dest, "wb
(if(!fp
;(thrw(OPEN_FILE_ERR

;(fwrite(slr, sizeof(slr_T), 1, fp
{

} (slr_T *slr_read_from_bin(char *src
;((slr_T *slr = malloc(sizeof(slr_T
(if(!slr
;(thrw(ALLOC_ERR
;("FILE *fp = fopen(src, "rb
(if(!fp
;(thrw(OPEN_FILE_ERR

;(fread(slr, sizeof(slr_T), 1, fp
;return slr
{
```

```
} ()slr_T *init_default_slr
;return NULL
"include "../include/parser/parser.h#{
"include "../utils/err/err.h#
"include "../utils/DS/include/stack.h#
<include <ctype.h#
<include <stddef.h#
<include <stdlib.h#
<include <stdio.h#
<include <stdlib.h#

this function is what the parser should do when encountering a shift action in the action table //
} (static int shift_action(parser_T *prs, token_T *tok, parse_tree_node_T *node, stack_T *sym_s
;(int terminal_col = action_tbl_find_terminal(prs->action, tok
;(int top = lr_stack_peek(prs->stack

;(((stack_push(sym_s, init_parse_tree_leaf(init_symbol_terminal(tok

;((parser_shift(prs, atoi(prs->action->actions[top][terminal_col] + 1
;return SHIFT
{

this function is what the parser should do when encountering a reduce action in the action //
table
static int reduce_action(parser_T *prs, token_T *tok, parse_tree_node_T *node, stack_T
} (*sym_s

;(int terminal_col = action_tbl_find_terminal(prs->action, tok
;(int top = lr_stack_peek(prs->stack
;int i
;[(rule_T *cur_rule = prs->rules[atoi(prs->action->actions[top][terminal_col] + 1
parse_tree_node_T **children = malloc(cur_rule->right_size * sizeof(parse_tree_node_T
;*)), *root = NULL

} (for(i = 0; i < cur_rule->right_size; ++i
;(children[i] = stack_pop(sym_s
{

root = init_parse_tree_node(init_symbol_non_terminal(cur_rule->left),
;(atoi(prs->action->actions[top][terminal_col] + 1), children, cur_rule->right_size
;(stack_push(sym_s, root
;(parser_reduce(prs, cur_rule
;return REDUCE
```

```
{

this function is what the parser should do when encountering an accept action in the action //
table
static int accept_action(parser_T *prs, token_T *tok, parse_tree_node_T *node, stack_T
} (*sym_s
;return ACCEPT
{

this function is what the parser should do when encountering an error action in the action table //
} (static int error_action(parser_T *prs, token_T *tok, parse_tree_node_T *node, stack_T *sym_s
;(thrw(PARSER_ACTION_ERR
;return ERR
{

static int parse_tok(parser_T *prs, token_T *tok, parse_tree_node_T *node, stack_T *sym_s, int
(*action_funcs[LETTERS_SIZE])(parser_T *prs, token_T *tok, parse_tree_node_T *node,
} ((stack_T *sym_s
char action =
;[prs->action->actions[lr_stack_peek(prs->stack)][action_tbl_find_terminal(prs->action, tok)][0

((if(!islower(action
;(thrw(PARSER_ACTION_ERR
;(return action_funcs[action - 'a'](prs, tok, node, sym_s
{

} (parser_T *init_parser(slr_T *slr
;((parser_T *prs = malloc(sizeof(parser_T
;set_node_T *cn = slr->grammer->rules->head
;rule_T *rule
;int i

;prs->action = slr->action
;prs->go_to = slr->go_to
;prs->n_rules = slr->grammer->rules->size
;(prs->rules = malloc(sizeof(rule_T *) * prs->n_rules
} (for(i = 0; i < prs->n_rules; ++i
;rule = cn->data

;((prs->rules[i] = malloc(sizeof(rule_T
;prs->rules[i]->left = rule->left
;prs->rules[i]->right = rule->right
;prs->rules[i]->right_size = rule->right_size
```

```
;cn = cn->next
{
;(prs->stack = init_lr_stack(10
;(lr_stack_push(prs->stack, 0

;return prs
{

} (parse_tree_T *parse(parser_T *prs, queue_T *queue_tok
;int parse_status = SHIFT
;int n_children = 0, children_capacity = 0, i
;token_T *tok
;parse_tree_node_T **children = NULL, **tmp = NULL, *root
;()stack_T *sym_s = stack_init
int (*action_funcs[LETTERS_SIZE])(parser_T *prs, token_T *tok, parse_tree_node_T
;(*node, stack_T *sym_s

;action_funcs['s' - 'a'] = &shift_action
;action_funcs['r' - 'a'] = &reduce_action
;action_funcs['a' - 'a'] = &accept_action

} (for(i = 0; i < LETTERS_SIZE; ++i
} ('if((i + 'a') != 's' && (i + 'a') != 'r' && (i + 'a') != 'a
;action_funcs[i] = &error_action
{
{

} do
(if(parse_status == SHIFT
;(tok = queue_dequeue(queue_tok

;(parse_status = parse_tok(prs, tok, root, sym_s, action_funcs
;(while(parse_status != ACCEPT {

;(return init_parse_tree(stack_pop(sym_s), prs->rules, prs->n_rules
{

} (void parser_shift(parser_T *prs, int data
;(lr_stack_push(prs->stack, data
{

} (symbol_T *parser_reduce(parser_T *prs, rule_T *rule
;int i, state
```

```
} (for(i = 0; i < rule->right_size; ++i
;(lr_stack_pop(prs->stack
{

;(state = lr_stack_peek(prs->stack

;[(i = prs->go_to->gotos[state][goto_tbl_find_non_terminal(prs->go_to, rule->left
;(lr_stack_push(prs->stack,  i
;(return init_symbol_non_terminal(rule->left
"include "../include/parser/action_table.h#{
"include "../utils/err/err.h#
<include <stdio.h#
<include <stdlib.h#

} (action_tbl_T *init_action_tbl(token_T **terminals, size_t n_terminals, size_t n_states
;int i
;((action_tbl_T *action = malloc(sizeof(action_tbl_T
(if(!action
;(thrw(ALLOC_ERR

;(action->actions = malloc(sizeof(char **) * n_states
(if(!action->actions
;(thrw(ALLOC_ERR
} (for(i = 0; i < n_states; ++i
;(action->actions[i] = malloc(sizeof(char *) * n_terminals
([if(!action->actions[i
;(thrw(ALLOC_ERR
{
;action->terminals = terminals
;action->n_terminals = n_terminals
;action->n_states = n_states

;return action
{

} (int action_tbl_find_terminal(action_tbl_T *tbl, token_T *term
;int i

} (for(i = 0; i < tbl->n_terminals; ++i
(if(tbl->terminals[i]->type == term->type
;return i
{
;return -1
{
```

```
} (void action_tbl_print_to_file(action_tbl_T *tbl, char *dest
;int i, j
;("FILE *fp = fopen(dest, "w

} (for(i = 0; i < tbl->n_states; ++i
} (for(j = 0; j < tbl->n_terminals; ++j
([if(!tbl->actions[i][j
;("       " ,fprintf(fp
else
;([fprintf(fp, "%3s ", tbl->actions[i][j
{
;([fprintf(fp, "\n", tbl->actions[i][j
{

;(fclose(fp
{

} (void action_tbl_pretty_print_to_file(action_tbl_T *tbl, char *dest
;int i, j
;("FILE *fp = fopen(dest, "w

print header //
;("|   " ,fprintf(fp
} (for(i = 0; i < tbl->n_terminals; ++i
;(fprintf(fp, "%10s ", tbl->terminals[i]->value
{
;("fprintf(fp, "\n

;("|   " ,fprintf(fp
} (for(i = 0; i < tbl->n_terminals; ++i
;("____________" ,fprintf(fp
{
;("fprintf(fp, "\n


} (for(i = 0; i < tbl->n_states; ++i
;(fprintf(fp, " %2d|", i
} (for(j = 0; j < tbl->n_terminals; ++j
([if(!tbl->actions[i][j
;("       " ,fprintf(fp
else
;([fprintf(fp, "%10s ", tbl->actions[i][j
{
```

```
;("fprintf(fp, "\n
{

;(fclose(fp
"include "../include/parser/symbol_set.h#{
<include <stdlib.h#

} ()symbol_set_T *init_symbol_set
;((symbol_set_T *set = (symbol_set_T*) malloc(sizeof(symbol_set_T

;set->set = NULL
;set->size = 0
;return set
{

} (symbol_set_T *init_symbol_set_with_symbols(symbol_T **syms, const size_t size
;((symbol_set_T *set = (symbol_set_T*) malloc(sizeof(symbol_set_T
;int i

;(set->set = malloc(sizeof(symbol_T *) * size
;set->size = size

} (for(i = 0; i < set->size; ++i
;((set->set[i] = malloc(sizeof(symbol_T

;set->set[i]->symbol = syms[i]->symbol
;set->set[i]->sym_type = syms[i]->sym_type
{

;return set
{

} (int add_symbol(symbol_set_T *set, symbol_T *item
;size_t i

} (++for (i = 0; i < set->size; i
} ((if (symbol_equals(set->set[i], item
;return 0
{
{

;((set->set = (symbol_T**) realloc(set->set, sizeof(symbol_T*) * (set->size + 1
;set->set[set->size] = item
;++set->size
```

```
;return 1
{

} (int remove_symbol(symbol_set_T *set, symbol_T *item
;int i, j

} (for(i = 0; i < set->size; ++i
} ((if(symbol_equals(set->set[i], item
} (for(j = i; j < set->size - 1; ++j
;[set->set[j] = set->set[j+1
{
;--set->size
;break
{
{
;(set->set = realloc(set->set, sizeof(symbol_T*) * set->size

;return 1
{

"include "../include/parser/lr_stack.h#
"include "../utils/err/err.h#
<include <stdlib.h#

} (lr_stack_T *init_lr_stack(size_t alloc_size
;((lr_stack_T *s = malloc(sizeof(lr_stack_T
(if(!s
;(thrw(ALLOC_ERR

;s->top = NULL
;s->size = 0
;s->capacity = 0
;s->alloc_size = alloc_size

;return s
{

} (void lr_stack_push(lr_stack_T *s, int data
((if(lr_stack_full(s
;(s->top = realloc(s->top, sizeof(int) * s->capacity + s->alloc_size

;s->top[s->size] = data
;++s->size
```

```
;s->capacity += s->alloc_size
{

} (int lr_stack_pop(lr_stack_T *s
((if(LR_IS_EMPTY(s
;return -1

;[int data = s->top[--s->size
;(s->top = realloc(s->top, sizeof(int) * s->capacity

;return data
{

} (int lr_stack_peek(lr_stack_T *s
((if(LR_IS_EMPTY(s
;return -1

;[return s->top[s->size - 1
{

} (int lr_stack_peek_inside(lr_stack_T *s, int n
;int i, tmp
;(int *items = malloc(sizeof(int) * n
(if(!items
;(thrw(ALLOC_ERR

} (for(i = 0; i < n; ++i
;(items[i] = lr_stack_pop(s
{
;(tmp = lr_stack_peek(s
} (for(i = 0; i < n; ++i
;([lr_stack_push(s, items[n - i - 1
{
;(free(items
;return tmp
{

} (int lr_stack_full(lr_stack_T *s
;return s->capacity <= s->size
{

} (void lr_stack_clear(lr_stack_T *s
((while (!LR_IS_EMPTY(s
;(lr_stack_pop(s
```

```
{
"include "../include/parser/grammer.h#
<include <stdlib.h#

**/
brief Create a grammar@ *
param rules set of rules of the grammar@ *
param symbols set of symbols of the grammar@ *
return grammar@ *
/*
} (grammer_T *init_grammer(set_T *rules, set_T *symbols
;((grammer_T *gram = malloc(sizeof(grammer_T

;gram->rules = rules
;gram->symbols = symbols

;return gram
{

**/
brief Get all terminal symbols from a set of symbols of a grammar@ *
param symbols set of symbols@ *
return array of pointers to terminal symbols@ *
/*
} (token_T **terminals_in_symbol_set(set_T *symbols
;token_T **terms
;symbol_T *cur_sym
;set_node_T *cn = symbols->head
;int i, termc = 0

/* count number of terminal symbols */
} (for(i = 0; i < symbols->size; ++i
;cur_sym = cn->data
(if(cur_sym->sym_type == TERMINAL
;++termc
;cn = cn->next
{

;(terms = malloc(sizeof(token_T *) * termc
;termc = 0

/* add terminal symbols to array */
;cn = symbols->head
} (for(i = 0; i < symbols->size; ++i
```

```
;cur_sym = cn->data

} (if(cur_sym->sym_type == TERMINAL
;terms[termc] = cur_sym->symbol->terminal
;++termc
{
;cn = cn->next
{

;return terms
{

**/
brief Get all terminal symbols from a set of symbols of a grammar and the terminal symbol@ *
"$"
param symbols set of symbols@ *
return array of pointers to terminal symbols@ *
/*
} (token_T **terminals_in_symbol_set_and_dollar(set_T *symbols
;(token_T **terms = terminals_in_symbol_set(symbols

;((* terms = realloc(terms, (n_terminals_in_symbol_set(symbols) + 1) * sizeof(token_T
;(terms[n_terminals_in_symbol_set(symbols)] = init_token("$", TOK_eof

;return terms
{

**/
brief Get number of terminal symbols in a set of symbols of a grammar@ *
param symbols set of symbols@ *
return number of terminal symbols@ *
/*
} (size_t n_terminals_in_symbol_set(set_T *symbols
;symbol_T *cur_sym
;set_node_T *cn = symbols->head
;int i, termc = 0

/* count number of terminal symbols */
} (for(i = 0; i < symbols->size; ++i
;cur_sym = cn->data
(if(cur_sym->sym_type == TERMINAL
;++termc
;cn = cn->next
{
```

```
;return termc
}

**/
brief Get all non-terminal symbols from a set of symbols of a grammar@ *
param symbols set of symbols@ *
return array of pointers to non-terminal symbols@ *
/*
} (non_terminal_T **non_terminals_in_symbol_set(set_T *symbols
;non_terminal_T **nterms
;symbol_T *cur_sym
;set_node_T *cn = symbols->head
;int i, termc = 0

/* count number of non-terminal symbols */
} (for(i = 0; i < symbols->size; ++i
;cur_sym = cn->data
(if(cur_sym->sym_type == NON_TERMINAL
;++termc
;cn = cn->next
{

;(nterms = malloc(sizeof(non_terminal_T *) * termc
;termc = 0

/* add non-terminal symbols to array */
;cn = symbols->head
} (for(i = 0; i < symbols->size; ++i
;cur_sym = cn->data
} (if(cur_sym->sym_type == NON_TERMINAL
;nterms[termc] = cur_sym->symbol->non_terminal
;++termc
{
;cn = cn->next
{

;return nterms
{

**/
brief Get number of non-terminal symbols in a set of symbols of a grammar@ *
param symbols set of symbols@ *
return number of non-terminal symbols@ *
```

```
/*
} (size_t n_non_terminals_in_symbol_set(set_T *symbols
;symbol_T *cur_sym
;set_node_T *cn = symbols->head
;int i, termc = 0

/* count number of non-terminal symbols */
} (for(i = 0; i < symbols->size; ++i
;cur_sym = cn->data
(if(cur_sym->sym_type == NON_TERMINAL
;++termc
;cn = cn->next
{

;return termc
{

**/
brief Find the index of a rule in the grammar, given the right side of the rule@ *
param gram grammar@ *
param right right side of rule@ *
param right_size number of symbols in right side of rule@ *
return index of rule in grammar or -1 if not found@ *
/*
} (size_t find_right_grammer_index(const grammer_T *gram, symbol_T **right, size_t right_size
;int i ,j, equ = 0
;set_node_T *cn = gram->rules->head
;rule_T *rule

/* find rule in grammar */
} (for(i = 0; i < gram->rules->size; ++i
;rule = cn->data
} (if(rule->right_size == right_size
;equ = 1
} (for(j = 0; j < right_size; ++j
} (([if(symbol_cmp(rule->right[j], right[j
;equ = 0
;break
{
{
(if(equ
;return i
{
;cn = cn->next
```

```
{

;return -1
{
"include "../include/parser/parse_tree.h#
"include "../utils/err/err.h#
<include <stdio.h#
<include <stdlib.h#

} (parse_tree_T *init_parse_tree(parse_tree_node_T *root, rule_T **rules, size_t n_rules
;((parse_tree_T *t = malloc(sizeof(parse_tree_T

;t->root = root
;t->rules = rules
;t->n_rules = n_rules
{

parse_tree_node_T *init_parse_tree_node(symbol_T *sym, ssize_t rule_index,
} (parse_tree_node_T **children, size_t n_children
;((parse_tree_node_T *pt = malloc(sizeof(parse_tree_node_T
(if(!pt
;(thrw(ALLOC_ERR

;pt->symbol = sym
;pt->rule_index = rule_index
;pt->children = children
;pt->n_children = n_children

;return pt
{

} (void parse_tree_free(parse_tree_node_T *tree
(if(tree == NULL
;return

} (if(tree->children
;size_t i
} (for(i = 0; i < tree->n_children; ++i
;([parse_tree_free(tree->children[i
{

;(free(tree->children
{
```

```
;(free(tree
{

} (parse_tree_node_T *init_parse_tree_leaf(symbol_T *sym
;((parse_tree_node_T *pt = malloc(sizeof(parse_tree_node_T
(if(!pt
;(thrw(ALLOC_ERR

;pt->symbol = sym
;pt->rule_index = -1
;pt->children = NULL
;pt->n_children = 0

;return pt
{


} (void parse_tree_traverse_preorder(parse_tree_node_T *tree, int layer
(if(tree == NULL
;return

;int i
char *val = tree->symbol->sym_type == NON_TERMINAL
tree->symbol->symbol->non_terminal->value ?
;tree->symbol->symbol->terminal->value :

int type = tree->symbol->sym_type == NON_TERMINAL
tree->symbol->symbol->non_terminal->type ?
;tree->symbol->symbol->terminal->type :

} (for(i = 0; i < layer; ++i
;("  ")printf
{

;(printf("%s, %zd\n", val, tree->rule_index

} (for(i = 0; i < tree->n_children; ++i
;(parse_tree_traverse_preorder(tree->children[i], layer + 1
{
{

} (void parse_tree_traverse_postorder(parse_tree_node_T *tree, int layer
(if(tree == NULL
;return
```

```
;int i
char *val = tree->symbol->sym_type == NON_TERMINAL
tree->symbol->symbol->non_terminal->value ?
;tree->symbol->symbol->terminal->value :

int type = tree->symbol->sym_type == NON_TERMINAL
tree->symbol->symbol->non_terminal->type ?
;tree->symbol->symbol->terminal->type :

} (for(i = 0; i < tree->n_children; ++i
;(parse_tree_traverse_postorder(tree->children[i], layer + 1
{

} (for(i = 0; i < layer; ++i
;("  ")printf
{

;(printf("%s\n", val
{
/$} ([]Mmain    main.c    /^int main(int argc, char* argv
/$} (compile    quest.c    /^void compile(quest_T *q
/$} (compile_file    quest.c    /^void compile_file(const char *filename
/get_new_filename    io.c    /^char *get_new_filename(const char *filename, const
/$   } (init_default_lang    lang.c    /^static slr_T *init_default_lang(quest_T *q
/$} (init_quest    lang.c    /^quest_T *init_quest(const char *filename
/$} (print_to_stderr    io.c    /^void print_to_stderr(char *content
/$} (print_to_stdout    io.c    /^void print_to_stdout(char *content
/$} (read_file    io.c    /^char* read_file(const char *filename
/* write_file    io.c    /^void write_file(const char *filename, const char
/ write_file_append    io.c    /^void write_file_append(const char *filename, char
ifndef ERR#
(define ERR(name, msg#
endif#

("ERR(UNKNOWN, "Unknown error happened\n
("ERR(OPEN_FILE, "Error while opening file, terminating\n
("ERR(WRT_FILE, "Error while writing file, terminating\n
("ERR(ALLOC, "Error while allocating memory, terminating\n
("ERR(PARSER_ACTION, "Error parsing, specifically in action table\n
("ERR(ARG, "Invalid argument to function\n
("ERR(UNIMPL, "Unimplemented feature\n
("ERR(REG_NOT_FOUND, "Register not found\n
```

```
undef ERR#ifndef QUEST_ERR#
define QUEST_ERR#

"define PROG "qc#

;(define ERR_MSG() printf("%s: error: in file: %s line: %d\n", prog, status.fname, status.linec#
(define PERROR_PROG(err) fprintf(stderr, "%s: %s", PROG, err#

} typedef enum ERROR_TYPES_ENUM
,define ERR(name, msg) name##_ERR#
"include "errors.h#
undef ERR#
N_ERRS
;err_E {

;(void thrw(int err

===----------= endif//===--= err.c#
//
.this file deals with error handling //
//
===----------===//

"include "err.h#
<include <stdio.h#
<include <stdlib.h#


throw an error corresponding to the exit code and immediately stops the program (treating //
(assembler errors only
} (void thrw(int code
} = []char *errs
,define ERR(name, msg) msg#
"include "errors.h#
undef ERR#
;{

;([PERROR_PROG(errs[code
;(exit(code
===----------= symbol_table.h =--===//{
//
a symbol table //
//
===----------===//
```

```
#ifndef QUEST_SYMBOL_TABLE_H
#define QUEST_SYMBOL_TABLE_H

#define DEFAULT_INITIAL_CAPACITY 8
#define DEFAULT_LOAD_FACTOR 0.75
#define DEFAULT_RESIZE_CONSTANT 2

#include <stddef.h>

typedef enum ENTRY_TYPES_ENUM {
    GLOBAL,
    LOCAL
} entry_type_E;

typedef struct SYMBOL_TABLE_ENTRY_STRUCT {
    char *name;
    int type;
    void *value;
    entry_type_E declaration_type;
    struct SYMBOL_TABLE_ENTRY_STRUCT *next;
} symbol_table_entry_T;

typedef struct SYMBOL_TABLE_STRUCT {
    size_t size;                    // current entries in the symbol table
    size_t capacity;                // number of entries that can be in the symbol table, is effected by load factor
    float load_factor;              // load factor of table
    symbol_table_entry_T **buckets; // actual members of hash table. members are arranged in a chained table, meaning each member is using a linked list if there is any collision
    unsigned int (*hash)(char *, size_t length); // hash function of the table
} symbol_table_T;

symbol_table_T *init_symbol_table(unsigned int capacity, float load_factor, unsigned int (*hash)(char *, size_t length));
symbol_table_T *init_symbol_table_default();

symbol_table_entry_T *init_symbol_table_entry(char *name, int type, void *value, entry_type_E declaration_type);

unsigned int symbol_table_insert(symbol_table_T *st, symbol_table_entry_T *ste);
symbol_table_entry_T *symbol_table_find(symbol_table_T *st, char *name);
unsigned int symbol_table_resize(symbol_table_T *st);
```

```
;(void symbol_table_print(symbol_table_T *st

endif /* QUEST_SYMBOL_TABLE_H *////===--= symbol_table_tree.h#
===----------------------------------------=
//
a symbol table tree is used to traverse scopes. each symbol table //
represents a scope, and children of a node in the tree represents a //
,nested scope. this means that the root of the tree is the global scope //
...the main program is a child of the global scope, and so on //
//
===----------------------------------------------------------------===//

ifndef QUEST_SYMBOL_TABLE_TREE_H#
define QUEST_SYMBOL_TABLE_TREE_H#

"include "symbol_table.h#
<include <stddef.h#

} typedef struct SYMBOL_TABLE_TREE_NODE_STRUCT
;symbol_table_T *table
;struct SYMBOL_TABLE_TREE_NODE_STRUCT **children
;size_t n_children
;symbol_table_tree_node_T {

} typedef struct SYMBOL_TABLE_TREE_STRUCT
;symbol_table_tree_node_T *root
;symbol_table_tree_T {

;(symbol_table_tree_T *init_symbol_table_tree(symbol_table_tree_node_T *root
symbol_table_tree_node_T *init_symbol_table_tree_node(symbol_table_T *table,
;(symbol_table_tree_node_T **children, size_t n_children
;(symbol_table_tree_node_T *init_symbol_table_tree_leaf(symbol_table_T *table

void symbol_table_tree_node_add(symbol_table_tree_node_T *sttn,
;(symbol_table_tree_node_T *sttn_child

;(void symbol_table_tree_print(symbol_table_tree_T *stt



"endif /* QUEST_SYMBOL_TABLE_TREE_H */#include "include/symbol_table.h#
"include "../err/err.h#
"include "../hashes/hashes.h#
```

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>

/**
 * @param capacity The initial capacity of the symbol table.
 * @param load_factor The load factor of the symbol table.
 * @param hash A function that hashes strings.
 * @return A new symbol table.
 */
symbol_table_T *init_symbol_table(unsigned int capacity, float load_factor, unsigned int
    (*hash)(char *, size_t length) {
    symbol_table_T *st = malloc(sizeof(symbol_table_T));
    if(!st)
        thrw(ALLOC_ERR);

    st->size = 0;
    st->capacity = capacity ? capacity : DEFAULT_INITIAL_CAPACITY;
    st->load_factor = load_factor ? load_factor : DEFAULT_LOAD_FACTOR;
    st->buckets = (symbol_table_entry_T **)malloc(st->capacity *
        sizeof(symbol_table_entry_T *));
    if(!st->buckets)
        thrw(ALLOC_ERR);
    st->hash = hash ? hash : hash_JOAAT;

    return st;
}

/**
 * Creates a new symbol table with default capacity and load factor.
 * @return A new symbol table.
 */
symbol_table_T *init_symbol_table_default() {
    symbol_table_T *st = malloc(sizeof(symbol_table_T));
    if(!st)
        thrw(ALLOC_ERR);

    st->size = 0;
```

```
;st->capacity = DEFAULT_INITIAL_CAPACITY
;st->load_factor = DEFAULT_LOAD_FACTOR
st->buckets = (symbol_table_entry_T **)calloc(st->capacity,
;((* sizeof(symbol_table_entry_T
(if(!st->buckets
;(thrw(ALLOC_ERR
;st->hash = hash_JOAAT

;return st
{

**/
.Creates a new entry for the symbol table *
.param name The name of the variable@ *
.param type The type of the variable@ *
.param value The value of the variable@ *
.param declaration_type The declaration type of the variable@ *
.return The new entry@ *
/*
symbol_table_entry_T *init_symbol_table_entry(char *name, int type, void *value, entry_type_E
} (declaration_type
;((symbol_table_entry_T *entry = malloc(sizeof(symbol_table_entry_T

;entry->name = name
;entry->type = type
;entry->value = value
;entry->declaration_type = declaration_type
;entry->next = NULL

;return entry
{

**/
.Inserts an entry into the symbol table *
.param st The symbol table@ *
.param ste The entry to insert@ *
.return 1 on success, 0 on failure@ *
/*
} (unsigned int symbol_table_insert(symbol_table_T *st, symbol_table_entry_T *ste
} (if(!ste
;(thrw(ARG_ERR
;return 0
{
```

```
;uint32_t i = st->hash(ste->name, strlen(ste->name)) % st->capacity

([if(!st->buckets[i
;st->buckets[i] = ste
} else
;[ste->next = st->buckets[i
;st->buckets[i] = ste
{

check if need to resize //
((if(((float)st->size / st->capacity >= st->load_factor
;(return symbol_table_resize(st

;++st->size

;return 1
{


**/
.Finds an entry in the symbol table *
.param st The symbol table@ *
.param name The name of the entry to find@ *
.return The found entry or NULL if not found@ *
/*
} (symbol_table_entry_T *symbol_table_find(symbol_table_T *st, char *name
;((uint32_t i = st->hash(name, strlen(name
;[symbol_table_entry_T *tmp = st->buckets[i

} (while(tmp
((if(strcmp(tmp->name, name
;[return st->buckets[i
;tmp = tmp->next
{

;return NULL
{

**/
.Resizes the symbol table *
.param st The symbol table@ *
.return 1 on success, 0 on failure@ *
/*
} (unsigned int symbol_table_resize(symbol_table_T *st
```

```
check if need to resize //
((if(((float)st->size / st->capacity >= st->load_factor
;return 0

;symbol_table_entry_T **new_buckets, *cur, *next
;int i, new_index

;st->capacity *= DEFAULT_RESIZE_CONSTANT
new_buckets = (symbol_table_entry_T **)calloc(st->capacity,
;((* sizeof(symbol_table_entry_T
(if(!new_buckets
;(thrw(ALLOC_ERR

rehash all elements into new table //
} (for(i = 0; i < st->capacity; ++i
;[cur = st->buckets[i

} (while(cur
;new_index = st->hash(cur->name, strlen(cur->name)) % st->capacity
;next = cur->next

;[cur->next = new_buckets[new_index
;new_buckets[new_index] = cur
;cur = next
{
{

free prev table //
;(free(st->buckets
;st->buckets = new_buckets

;return 0
{

**/
* Prints the symbol table.
* param st The symbol table@.
/*
} (void symbol_table_print(symbol_table_T *st
} (for(int i = 0; i < st->capacity; ++i
;[symbol_table_entry_T *cur = st->buckets[i

} (while(cur
;(printf("name: %s, type: %d, value: %p\n", cur->name, cur->type, cur->value
```

```
;cur = cur->next
{
{
{


//===----------------------------------------= symbol_table_tree.c =--===
//
//
//
//===----------------------------------------------------------------===

#include "include/symbol_table_tree.h"
#include <stdlib.h>

symbol_table_tree_T *init_symbol_table_tree(symbol_table_tree_node_T *root) {
;((symbol_table_tree_T *stt = malloc(sizeof(symbol_table_tree_T

;stt->root = root

;return stt
}

symbol_table_tree_node_T *init_symbol_table_tree_node(symbol_table_T *table,
symbol_table_tree_node_T **children, size_t n_children) {
;((symbol_table_tree_node_T *sttn = malloc(sizeof(symbol_table_tree_node_T

;sttn->table = table
;sttn->children = children
;sttn->n_children = n_children

;return sttn
}

symbol_table_tree_node_T *init_symbol_table_tree_leaf(symbol_table_T *table) {
;((symbol_table_tree_node_T *sttn = malloc(sizeof(symbol_table_tree_node_T

;sttn->table = table
;sttn->children = NULL
;sttn->n_children = 0

;return sttn
}
```

```
void symbol_table_tree_node_add(symbol_table_tree_node_T *sttn,
} (symbol_table_tree_node_T *sttn_child
sttn->children = realloc(sttn->children, sizeof(symbol_table_tree_node_T *) *
;(((sttn->n_children + 1

;sttn->children[sttn->n_children] = sttn_child
;++sttn->n_children
{

;(void symbol_table_tree_print(symbol_table_tree_T *stt


"include "hashes.h#

} (uint32_t hash_JOAAT(char *key, size_t length
;size_t i = 0
;uint32_t hash = 0
} (while (i != length
;[++hash += key[i
;hash += hash << 10
;hash ^= hash >> 6
{
;hash += hash << 3
;hash ^= hash >> 11
;hash += hash << 15
;return hash
===----------= hashes.h =--===//{
//
,hashes for veriuos purposes. special hashes for different purposes //
.like JOAAT hash for the symbol table //
//
===----------===//

ifndef QUEST_HASHES_H#
define QUEST_HASHES_H#

<include <stddef.h#
<include <stdint.h#

;(uint32_t hash_JOAAT(char *key, size_t length

"endif#include "include/lexer_DFA.h#
<include <ctype.h#
<include <stdio.h#
```

```
add transition for keyword states //
} (void add_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int dest_state_index, int c
)if
&& (is_id(c
&& (dfa->flags & ADD_IDENTIFIER_STATE_FLAG)
&& ([dfa->last_states[TOK_IDENTIFIER)
src_state->type == TOK_IDENTIFIER
} (
;dfa->DFA[src_state->index][c] = dfa->last_states[TOK_IDENTIFIER]->index
{
} else
;dfa->DFA[src_state->index][c] = -1
{
{

add transition for id state according to the languages rules //
also add transition from first state to id state //
void add_id_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int dest_state_index, int
} (c
} ((if(is_id(c
;dfa->DFA[src_state->index][c] = dest_state_index
;dfa->DFA[0][c] = dest_state_index
} else {
;dfa->DFA[src_state->index][c] = -1
{
{

add transition for num state according to the languages rules //
also add transition from first state to num state //
void add_num_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int dest_state_index,
} (int c
} ((if(isdigit(c
;dfa->DFA[src_state->index][c] = dest_state_index
;dfa->DFA[0][c] = dest_state_index
} else {
;dfa->DFA[src_state->index][c] = -1
{
{

add transition for first char state according to the languages rules //
also add transition from first state to first char state //
void add_first_char_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
} (dest_state_index, int c
```

```
if input is not single quotes //
} ((if(is_single_quotes(c
;dfa->DFA[src_state->index][c] = dest_state_index
;dfa->DFA[0][c] = src_state->index
} else {
;dfa->DFA[src_state->index][c] = src_state->index
{
{

add transition for second char state //
,note: the state isn't suppose to have a transition to another state //
its the end of the token //
void add_second_char_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
} (dest_state_index, int c
;dfa->DFA[src_state->index][c] = -1
{

add transition for first string state according to the languages rules //
also add transition from first state to first string state //
void add_first_string_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
} (dest_state_index, int c
if input is quotes //
} ((if(is_quotes(c
;dfa->DFA[src_state->index][c] = dest_state_index
;dfa->DFA[0][c] = src_state->index
} else {
;dfa->DFA[src_state->index][c] = src_state->index
{
{

add transition for second string state //
,note: the state isn't suppose to have a transition to another state //
its the end of the token //
void add_second_string_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
} (dest_state_index, int c
;dfa->DFA[src_state->index][c] = -1
"include "lexer_DFA.h#{

;(void add_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int dest_state_index, int c
void add_id_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int dest_state_index, int
;(c
void add_num_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int dest_state_index,
;(int c
```

```
void add_first_char_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
;(dest_state_index, int c
void add_second_char_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
;(dest_state_index, int c
void add_first_string_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
;(dest_state_index, int c
void add_second_string_transition(lexer_dfa_T *dfa, lexer_dfa_state_T *src_state, int
===---------- dest_state_index, int c);//===--- The Quest Language: lexer_DFA.h
//
,The DFA header. defines the main DFA's struct //
.the states struct and its type, and macros relating to the dfa //
//
//
WORK IN PROGRESS //
//
===----------------------------------------------===//

ifndef QUEST_LEXER_DFA_H#
define QUEST_LEXER_DFA_H#

"include "../../../include/lexer/token.h#
<include <stddef.h#

define ASCII_SIZE 128#

.flags for the dfa, mask to check if needed //
define ADD_IDENTIFIER_STATE_FLAG 1#
define ADD_NUMBER_STATE_FLAG 2#
define ADD_CHAR_STATES_FLAG 4#
define ADD_STRING_STATES_FLAG 8#

('_' == (define is_id(c) (isalpha(c) || (c#
(define is_single_quotes(c) ((c) == 39#
(define is_quotes(c) ((c) == 34#

} typedef struct LEXER_DFA_STATE_STRUCT
unsigned int index;   // index of state
// lexeme of state, if the state does not accept this will be NULL  char *lexeme;
// states type  token_type_E type;
;lexer_dfa_state_T {

} typedef struct LEXER_DFA_STRUCT
// matrix defining the DFA  short **DFA;
// tokens in the DFA  token_T **toks;
```

```
unsigned int n_toks;          // number of tokens
lexer_dfa_state_T **states;   // states in the DFA
lexer_dfa_state_T **last_states;   // saved last instance of each type of state
unsigned int n_states;        // number of states
char *filename;               // where to contain the DFA
unsigned short flags;         // flags, represented as bits
unsigned char n_flag_states;  // number of states defined by the flags
};lexer_dfa_T

int init_dfa(token_T **toks, const size_t n_toks, const char *DFA_filename, const char *DFA_states_filename, const char *DFA_states_details_filename , unsigned short flags);
int init_default_dfa();

#include "include/lexer_DFA.h"endif
#include "include/transitions.h"
#include "../../include/lexer/tokens.h"
#include "../../include/macros.h"
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// modify lexeme because its first and last chars are ""
// TODO: fix, this is not a good method for doing this. doesnt need to exist because lexeme is a string.
// TODO: make a free function for the structcha
static char * remove_first_last_char(char *str) {
    char* cpy = malloc(strlen(str));
    if(cpy == NULL) {
        perror("Error allocating memory for the DFA");
        exit(EXIT_FAILURE);
    }

    strcpy(cpy, str);
    cpy[strlen(cpy)-1] = '\0';
    ++cpy;
    return cpy;
}



// add a state to the dfa
// if dest_state is NULL, dest_state will be equal to the source
// return state
```

```
static lexer_dfa_state_T * add_state_to_DFA(lexer_dfa_T *dfa, int type, char *lexeme, void
} ((*add_transition)(lexer_dfa_T *, lexer_dfa_state_T *, int, int), int dest_state
;int i, j

add state //
dfa->states = (lexer_dfa_state_T**)realloc(dfa->states, ++dfa->n_states *
;((*sizeof(lexer_dfa_state_T
} (if(dfa->states == NULL
;("perror("Error allocating memory for the DFA
;(exit(EXIT_FAILURE
{

;((dfa->states[dfa->n_states - 1] = (lexer_dfa_state_T *)malloc(sizeof(lexer_dfa_state_T
;dfa->states[dfa->n_states - 1]->index = dfa->n_states - 1
;dfa->states[dfa->n_states - 1]->lexeme = lexeme
;dfa->states[dfa->n_states - 1]->type = type


change last state type to this state unless state is spacial state //

} (if((type == TOK_UNKNOWN) || dfa->last_states[type] == NULL
;[dfa->last_states[type] = dfa->states[dfa->n_states - 1
{

add row //
;((*dfa->DFA = (short**)realloc(dfa->DFA, dfa->n_states * sizeof(short
} (if(dfa->DFA == NULL
;("perror("Error allocating memory for the DFA
;(exit(EXIT_FAILURE
{
;((dfa->DFA[(dfa->n_states) - 1] = (short*)malloc(ASCII_SIZE * sizeof(short

} (if(dfa->DFA[(dfa->n_states) - 1] == NULL
;("perror("Error allocating memory for the DFA
;(exit(EXIT_FAILURE
{

if dest_state is 0, dest_state will be equal to the source //
} (if(dest_state == 0
;dest_state = dfa->states[dfa->n_states - 1]->index
{
} (for(i = 0; i < ASCII_SIZE; ++i
;(add_transition)(dfa, dfa->states[dfa->n_states - 1], dest_state, i*)
{
```

```
;[return dfa->states[dfa->n_states - 1
{

add a token to the lexer's DFA //
} (static int add_tok_to_DFA(lexer_dfa_T *dfa, token_T *tok
unsigned int i, curr_state = 0, str_tok_len = strlen(tok->value), state_type = dfa->flags &
;ADD_IDENTIFIER_STATE_FLAG ? TOK_IDENTIFIER : TOK_UNKNOWN
;char *lexeme = malloc(sizeof(char)), *tmp_lex

check if tok has lexeme //
(if(str_tok_len == 0
;return 0

} (for(i = 0; i < str_tok_len; ++i
add to state lexeme //
;((lexeme = (char *)realloc(lexeme, (i + 2) * sizeof(char
;'lexeme[i + 1] = '\0
;[lexeme[i] = tok->value[i


check if state type is valid to be an identifier //
(if(!(is_id(tok->value[i])) && state_type == TOK_IDENTIFIER
;state_type = TOK_UNKNOWN

check if char at state has a response //
)if
dfa->DFA[curr_state][tok->value[i]] == -1
dfa->DFA[curr_state][tok->value[i]] > 0 && dfa->DFA[curr_state][tok->value[i]] <) ||
(dfa->n_flag_states
} (
;(add_state_to_DFA(dfa, state_type, strdup(lexeme), add_transition, 0
;dfa->DFA[curr_state][tok->value[i]] = dfa->n_states - 1
{
;[[curr_state = dfa->DFA[curr_state][tok->value[i
{


state settings //
(TODO: change state types (e.g. ACCEPT) to tok types (e.g. TOK_CHAR //
;dfa->states[curr_state]->type = tok->type

;((dfa->states[curr_state]->lexeme = (char *) malloc(str_tok_len * sizeof(char
;(strcpy(dfa->states[curr_state]->lexeme, tok->value
```

```
;return 0
{

} (static void init_special_states(lexer_dfa_T *dfa
} (if(dfa->flags & ADD_IDENTIFIER_STATE_FLAG
;++dfa->n_flag_states
;(add_state_to_DFA(dfa, TOK_IDENTIFIER, "(TOK_IDENTIFIER)", add_id_transition, 0
{
} (if(dfa->flags & ADD_NUMBER_STATE_FLAG
;++dfa->n_flag_states
add_state_to_DFA(dfa, TOK_NUMBER_CONSTANT, "(TOK_NUMBER_CONSTANT)",
;(add_num_transition, 0
{
} (if(dfa->flags & ADD_CHAR_STATES_FLAG
;dfa->n_flag_states += 2
add_state_to_DFA(dfa, TOK_UNKNOWN, "(CHAR_DENY)", add_first_char_transition,
;(add_state_to_DFA(dfa, TOK_CHAR, "(CHAR)", add_second_char_transition, 0)->index
{
} (if(dfa->flags & ADD_STRING_STATES_FLAG
;dfa->n_flag_states += 2
add_state_to_DFA(dfa, TOK_UNKNOWN, "(TOK_UNKNOWN)",
add_first_string_transition, add_state_to_DFA(dfa, TOK_STRING_LITERAL, "(STRING)",
;(add_second_string_transition, 0)->index
{

{

initialize the lexer's Deterministic Finite Automata //
.according to a set of tokens that set the rules //
int init_dfa(token_T **toks, const size_t n_toks, const char *DFA_filename, const char
} (*DFA_states_filename, const char *DFA_states_details_filename , unsigned short flags
;((lexer_dfa_T *dfa = malloc(sizeof(lexer_dfa_T
;("FILE *DFA_fp = fopen(DFA_filename, "w
;("FILE *DFA_states_fp = fopen(DFA_states_filename, "w
;("FILE *DFA_states_details_fp = fopen(DFA_states_details_filename, "w
;int i, j

handle file not opening //
} (if(DFA_fp == NULL || DFA_states_fp == NULL
;("perror("Error in opening DFA file
;(exit(EXIT_FAILURE
{
```

```
// -- init dfa variables --

// init DFA with one row
((*dfa->DFA = (short**)malloc(sizeof(short));
if(dfa->DFA == NULL) {
perror("Error allocating memory for the DFA");
exit(EXIT_FAILURE);
}

((dfa->DFA[0] = (short*)malloc(ASCII_SIZE * sizeof(short));
if(dfa->DFA[0] == NULL) {
perror("Error allocating memory for the DFA");
exit(EXIT_FAILURE);
}
for(i = 0; i < ASCII_SIZE; ++i)
dfa->DFA[0][i] = -1;

// init tokens
dfa->n_toks = n_toks;
((*dfa->toks = (token_T**)malloc(dfa->n_toks * sizeof(token_T));
if(dfa->toks == NULL) {
perror("Error allocating memory for the DFA");
exit(EXIT_FAILURE);
}

// init states
dfa->n_states = 1;
((*dfa->states = (lexer_dfa_state_T**)malloc(dfa->n_states * sizeof(lexer_dfa_state_T));
if(dfa->states == NULL) {
perror("Error allocating memory for the DFA");
exit(EXIT_FAILURE);
}

((dfa->states[0] = (lexer_dfa_state_T*)malloc(sizeof(lexer_dfa_state_T));
dfa->states[0]->index = 0;
dfa->states[0]->type = TOK_UNKNOWN;

((*dfa->last_states = (lexer_dfa_state_T**)malloc(NUM_TOK * sizeof(lexer_dfa_state_T));
if(dfa->last_states == NULL) {
perror("Error allocating memory for the DFA");
exit(EXIT_FAILURE);
}
for(i = 0; i < NUM_TOK; ++i) {
((dfa->last_states[i] = (lexer_dfa_state_T*)malloc(sizeof(lexer_dfa_state_T));
```

```
;dfa->last_states[i] = NULL
{
;[dfa->last_states[TOK_UNKNOWN] = dfa->states[0

init filename //
;((dfa->filename = (char*)malloc(sizeof(char) * (strlen(DFA_filename) + 1

init flags //
;dfa->flags = flags
;dfa->n_flag_states = 0

init special states of dfa //
;(init_special_states(dfa

add all of the tokens to the DFA //
(for(i = 0; i < dfa->n_toks; ++i
;([add_tok_to_DFA(dfa, toks[i


print DFA into file to save it and print state lexme //
;(fprintf(DFA_fp, "%d\n%d\n", dfa->n_states, ASCII_SIZE

} (for(i = 0; i < dfa->n_states; ++i
,"fprintf(DFA_states_fp, "%d %d\n
,dfa->states[i]->index
dfa->states[i]->type
;(

,"fprintf(DFA_states_details_fp, "%s: %d (%d)\n
,dfa->states[i]->lexeme
,dfa->states[i]->index
dfa->states[i]->type
;(

} (for(j = 0; j < ASCII_SIZE; ++j
;([fprintf(DFA_fp, "%d ", dfa->DFA[i][j
{
;("fprintf(DFA_fp, "\n
{
;(fclose(DFA_fp

;return 0
{
```

```
default configuration for the DFA //
} ()int init_default_dfa
;[token_T *toks[NUM_TOK
;int i = 0

define TOK(name, lexeme, val, is_kw) toks[i++] =#
;(init_token(remove_first_last_char(#lexeme), TOK_##name
include TOKS_PATH#
undef TOK#

,init_dfa(toks
,NUM_TOK
,LEXER_DFA_PATH
,LEXER_DFA_STATES_PATH
,LEXER_DFA_STATES_DETAILS_PATH
ADD_IDENTIFIER_STATE_FLAG + ADD_NUMBER_STATE_FLAG +
;(ADD_CHAR_STATES_FLAG + ADD_STRING_STATES_FLAG

(for(i = 0; i < NUM_TOK; ++i
;([free(toks[i

;return EXIT_SUCCESS
"include "include/queue.h#{


} ()queue_T* queue_init
;((queue_T* q = (queue_T*)malloc(sizeof(queue_T
;q->head = NULL
;q->tail = NULL
;q->size = 0
;return q
{

} (int is_empty(queue_T* q
;(return (q->head == NULL
{

} (void queue_enqueue(queue_T* q, void* data
;((queue_node_T* new_node = (queue_node_T*)malloc(sizeof(queue_node_T
;new_node->data = data
;new_node->next = NULL
} ((if (is_empty(q
;q->head = new_node
} else {
```

```
;q->tail->next = new_node
{
;q->tail = new_node
;++q->size
{

} (void* queue_dequeue(queue_T* q
} ((if (is_empty(q
;return NULL
{
;queue_node_T* temp = q->head
;void* data = temp->data
;q->head = q->head->next
} ((if (is_empty(q
;q->tail = NULL
{
;(free(temp
;--q->size
;return data
{

} (void* queue_peek(queue_T* q
} ((if (is_empty(q
;return NULL
{
;return q->head->data
{

} (void queue_clear(queue_T* q
} ((while (!is_empty(q
;(queue_dequeue(q
{
{

} (int queue_size(queue_T* q
;return q->size
{
ifndef QUEST_STACK_H#
define QUEST_STACK_H#

<include <stdlib.h#

(define IS_EMPTY(s) ((s)->top == NULL#
```

```
**/
Structure of a stack node *
/*
} typedef struct STACK_NODE_STRUCT
// The data stored in the stack node void *data;
struct STACK_NODE_STRUCT *next;  // Pointer to the next stack node
;stack_node_T {

**/
Structure of a generic stack *
/*
} typedef struct GENERIC_STACK_STRUCT
// Pointer to the top of the stack stack_node_T *top;
// Number of items in the stack size_t size;
;stack_T {

;()stack_T *stack_init
;(void stack_push(stack_T* s, void* data
;(void *stack_pop(stack_T* s
;(void *stack_peek(stack_T* s
;(void stack_clear(stack_T* s
;(void stack_flip(stack_T* s
;(size_t stack_size(stack_T* s

/* endif /* QUEST_STACK_H#
ifndef QUEST_QUEUE_H#
define QUEST_QUEUE_H#

<include <stdlib.h#


**/
Generic Queue Node Structure *
*
.Contains data and a pointer to the next node in the queue *
/*
} typedef struct GENERIC_QUEUE_NODE_STRUCT
/* /* Data stored in the node void* data;
struct GENERIC_QUEUE_NODE_STRUCT *next;  /* Pointer to the next node in the
/* queue
;queue_node_T {

**/
Generic Queue Structure *
```

```
                                                                                *
                                                  ,Contains a pointer to the head and tail nodes of the queue *
                                                                  .as well as the current size of the queue *
                                                                                                          /*
                                                                  } typedef struct GENERIC_QUEUE_STRUCT
                                        /* queue_node_T *head;   /* Pointer to the head node of the queue
                                             /* queue_node_T *tail;   /* Pointer to the tail node of the queue
                                                        /* /* Current size of the queue          int size;
                                                                                          ;queue_T {

                                                                                  ;()queue_T *queue_init
                                                                              ;(int is_empty(queue_T* q
                                                                   ;(void queue_enqueue(queue_T* q, void* data
                                                                          ;(void *queue_dequeue(queue_T* q
                                                                              ;(void *queue_peek(queue_T* q
                                                                               ;(void queue_clear(queue_T* q
                                                                                  ;(int queue_size(queue_T* q

                                                                                 /* endif /* QUEST_QUEUE_H#
                                                  ===----------------------------------------------------= hashset.h =--===//
                                                                                                          //
                                                                                 .a generic hash set. you know whats a hash set //
                                                                                                          //
                                                  ===-----------------------------------------------------------------===//

                                                                  } typedef struct HASHSET_NODE_STRUCT
                                        /*  /* The data stored in the node                          void *data;
                                   /* struct HASHSET_NODE_STRUCT *next;  /* Pointer to the next node in the bucket
                                                                                          ;hashset_node_T {

                                                                  } typedef struct HASHSET_STRUCT
                                                        /* /* Number of elements in the set          int size;
                                                  /*  /* Number of buckets in the set                int capacity;
                                /* /* Maximum ratio of number of elements to number of buckets        float load_factor;
                                                  /* hashset_node_T **buckets; /* Array of bucket pointers
                                        /* unsigned int (*hash_func)(void *); /* Pointer to a hash function to hash data
                             /* int (*compare_func)(void *, void *); /* Pointer to a comparison function to compare data
                                                                                          ;hashset_T {


hashset_T *init_hash_set(int initial_capacity, float load_factor, unsigned int (*hash_func)(void *),
                                                                   ;((* int (*compare_func)(void *, void

                                                                   ;(int hash_set_add(hashset_T *set, void* data
```

```
;(int hash_set_add_all(hashset_T *set, hashset_T *other_set
;(int hash_set_contains(hashset_T *set, void* data
;(int hash_set_resize(hashset_T *set, int new_capacity

;(int hash_set_compare(void *a, void *b
;(unsigned int hash_set_hash(void *data
ifndef QUEST_GENERIC_SET#
define QUEST_GENERIC_SET#

<include <stdlib.h#

Define a node structure for the set elements //
} typedef struct GENERIC_SET_NODE_STRUCT
void *data;  // Pointer to the element data (can hold any type
struct GENERIC_SET_NODE_STRUCT *next;  // Pointer to the next node in the set
;set_node_T {

Define the set structure //
} typedef struct GENERIC_SET_STRUCT
set_node_T *head;  // Pointer to the head
node of the set
int (*compare)(const void*, const void*);  // Pointer to a comparison function
size_t size;  // Size of the set (number of
(elements
;set_T {

;((*set_T *set_init(int (*compare)(const void*, const void
;(int set_add(set_T* set, void* data
;(int set_add_all(set_T* set, set_T *more_set
;(int set_add_arr(set_T* set, void **more_set, size_t size
;(int set_remove(set_T* set, void* data
;(int set_contains(set_T* set, void* data
;(void set_flip(set_T* set
;(void set_free(set_T* set

endif#

<include <stdlib.h#
"include "include/generic_set.h#

Function to initialize a new set //
} ((*set_T* set_init(int (*compare)(const void*, const void
;((set_T* set = (set_T*)malloc(sizeof(set_T
;set->head = NULL
```

```
;set->compare = compare
;set->size = 0
;return set
{

Function to add an element to the set //
} (int set_add(set_T* set, void* data
(Check for duplicates (using the comparison function //
;set_node_T* current = set->head
} (while (current != NULL
} (if (set->compare(current->data, data) == 0
return 0;  // Element already exists
{
;current = current->next
{

Create a new node for the data //
;((set_node_T* new_node = (set_node_T*)malloc(sizeof(set_node_T
;new_node->data = data
;new_node->next = NULL

Insert the new node at the head of the list //
;new_node->next = set->head
;set->head = new_node
;++set->size
return 1;  // Element added successfully
{

} (int set_add_all(set_T* set, set_T *more_set
;int i, c
;set_node_T *cur

;cur = more_set->head
} (for(i = 0; i < more_set->size; ++i
;(c += set_add(set, cur->data
;cur = cur->next
{
;return c
{

} (int set_add_arr(set_T* set, void **more_set, size_t size
;int i, c
} (for(i = 0; i < size; ++i
;([c += set_add(set, more_set[i
```

```
{
;return c
{

Function to check if an element exists in the set //
} (int set_contains(set_T* set, void* data
;set_node_T* current = set->head

} (while (current != NULL
} (if (set->compare(current->data, data) == 0
return 1;  // Element found
{
;current = current->next
{
return 0;  // Element not found
{

Function to remove an element from the set //
} (int set_remove(set_T* set, void* data
;set_node_T* current = set->head
;set_node_T* prev = NULL

Find the node to remove and the node before it //
} (while (current != NULL
} (if (set->compare(current->data, data) == 0
;break
{
;prev = current
;current = current->next
{

(If element not found, return 0 (failure //
} (if (current == NULL
;return 0
{

Handle removing the head node //
} (if (prev == NULL
;set->head = current->next
} else {
;prev->next = current->next
{

(free(current->data);  // Free data if needed (depending on data type
```

```
;(free(current
;--set->size
return 1;  // Element removed successfully
{

} (void set_flip(set_T *set
;set_node_T *current = set->head
;set_node_T *prev = NULL
;set_node_T *next = NULL

} (while (current != NULL
;next = current->next
;current->next = prev
;prev = current
;current = next
{

;set->head = prev
{

} (void set_free(set_T* set
;set_node_T* current = set->head
} (while (current != NULL
;set_node_T* temp = current
;current = current->next
(free(temp->data);  // Free data if needed (depending on data type
;(free(temp
{
;(free(set
{
"include "include/hashset.h#
"include "../err/err.h#
<include <stdio.h#
<include <stdlib.h#
<include <stdbool.h#

Create a new hash set //
hashset_T *init_hash_set(int initial_capacity, float load_factor, unsigned int (*hash_func)(void *),
} ((* int (*compare_func)(void *, void
;((hashset_T *set = (hashset_T *)malloc(sizeof(hashset_T
(if(!set
;(thrw(ALLOC_ERR

;set->size = 0
```

```
set->capacity = initial_capacity > 0 ? initial_capacity : 4;
// Minimum capacity of 4
set->load_factor = load_factor > 0 ? load_factor : 0.75;
// load factor
//     ;((* set->buckets = (hashset_node_T **)calloc(set->capacity, sizeof(hashset_node_T
(if(!set->buckets
;(thrw(ALLOC_ERR
//                set->hash_func = hash_func != NULL ? hash_func : hash_set_hash;
apply default hash func if not given one
set->compare_func = compare_func; // apply default cmp func if not given one

;return set
{

a random hash function //
} (unsigned int hash_set_hash(void *data
;unsigned long long key = (unsigned long long)data
;(key += ~(key << 15
;key *= 224682251
;(key ^= (key >> 13
;key *= 1831564602
;((return (unsigned int)(key ^ (key >> 16
{

(Default comparison function (simple pointer comparison //
} (int default_compare_func(void* a, void* b
;return a == b
{

Add an element to the hash set //
} (int hash_set_add(hashset_T *set, void* data
;int index = set->hash_func(data) % set->capacity
hashset_node_T *new_node = (hashset_node_T *)malloc(sizeof(hashset_node_T)), *curr =
;[set->buckets[index

} (while (curr != NULL
} ((if (set->compare_func(curr->data, data
return 0; // Element already exists, don't add again
{
;curr = curr->next
{

(if(!new_node
;(thrw(ALLOC_ERR
```

```
;new_node->data = data
;[new_node->next = set->buckets[index
;set->buckets[index] = new_node
;++set->size

Check for resize condition //
} (if ((float)set->size / set->capacity >= set->load_factor
} ((if (!hash_set_resize(set, set->capacity * 2
thrw(ALLOC_ERR); // Resize failed
{
{

;return 1
{

} (int hash_set_add_all(hashset_T *set, hashset_T *other_set
;hashset_node_T *curr
;int add_c

} (if (other_set == NULL || other_set->size == 0
return 0;  // Nothing to add
{

} (++for (int i = 0; i < other_set->capacity; i
;[curr = other_set->buckets[i
} (while (curr != NULL
add_c += hash_set_add(set, curr->data);  // Add each element from the other set
;curr = curr->next
{
{

;return add_c
{

Check if an element exists in the hash set //
} (int hash_set_contains(hashset_T *set, void* data
;int index = set->hash_func(data) % set->capacity
;[hashset_node_T *current = set->buckets[index

} (while (current != NULL
} ((if (set->compare_func(current->data, data
;return 1
{
;current = current->next
```

```
{
;return 0
{

// Resize the hash table dynamically
} (int hash_set_resize(hashset_T *set, int new_capacity
;hashset_node_T **temp_buckets, *current, *temp_node
;int new_index

;((* temp_buckets = (hashset_node_T **)calloc(new_capacity, sizeof(hashset_node_T
(if (!temp_buckets
;(thrw(ALLOC_ERR


// Rehash all elements into the new table
} (++for (int i = 0; i < set->capacity; i
;[current = set->buckets[i
} (while (current != NULL
;new_index = set->hash_func(current->data) % new_capacity
;temp_node = current
;current = current->next
;[temp_node->next = temp_buckets[new_index
;temp_buckets[new_index] = temp_node
{
{

// Free old table memory
;(free(set->buckets
;set->buckets = temp_buckets
;set->capacity = new_capacity

;return true
{

} (void hash_set_free(hashset_T *set
;hashset_node_T *current, *temp

} (++for (int i = 0; i < set->capacity; i
;[current = set->buckets[i
} (while (current != NULL
;temp = current
;current = current->next
;(free(temp
{
```

```
{
;(free(set->buckets
;(free(set
"include "include/stack.h#{
"include "../err/err.h#
<include <stdio.h#

} ()stack_T* stack_init
;((stack_T* s = (stack_T*)malloc(sizeof(stack_T
(if(!s
;(thrw(ALLOC_ERR

;s->top = NULL
;s->size = 0

;return s
{

} (void stack_push(stack_T* s, void* data
;((stack_node_T* new_node = (stack_node_T*)malloc(sizeof(stack_node_T
(if(!new_node
;(thrw(ALLOC_ERR

;new_node->data = data
;new_node->next = s->top
;s->top = new_node
;++s->size
{

} (void* stack_pop(stack_T* s
} ((if (IS_EMPTY(s
;return NULL
{
;void* data = s->top->data

;s->top = s->top->next
;--s->size

;return data
{

} (void* stack_peek(stack_T* s
} ((if (IS_EMPTY(s
;return NULL
```

```
{

;return s->top->data
{

} (void stack_clear(stack_T* s
} ((while (!IS_EMPTY(s
;(stack_pop(s
{
{

} (void stack_flip(stack_T* s

{

} (size_t stack_size(stack_T* s
;return s->size
{

===----------= Lexer =--===//
//
.The lexer of the compiler //
//
//
WORK IN PROGRESS //
//
===----------===//

"include "../include/lexer/lexer.h#
"include "../include/lexer/lexer_automata.h#
"include "../include/macros.h#
"include "../include/lexer/token.h#
<include <string.h#
<include <stdlib.h#
<include <ctype.h#
<include <stdio.h#

.advance lexer to the next ch aracter in its source //
} (void lexer_advance(lexer_T *lex
} ('if(lex->i < lex->src_size && lex->c != '\0
;++lex->i
;[lex->c = lex->src[lex->i
{
{
```

```
} (void lexer_skip_whitespace(lexer_T *lex
while (lex->c == '\r'   // carriage return
lex->c == '\n'   // newline ||
lex->c == '\t'   // tab ||
(' ' == lex->c ||
lexer_advance(lex); // skip
{

} (void lexer_skip_bullshit(lexer_T *lex
} (while(lex->c > lex->automata->n_symbols
;("printf("Unexcepted symbol\n //
;(lexer_advance(lex
{
{

} (static token_T* get_token(lexer_T *lex
;unsigned int curr_state = 0, buffer_len = 1
;((char *buffer = malloc(sizeof(char

} (if(lex->automata->automata[curr_state][lex->c] == -1
;((buffer = (char *) realloc(buffer, (++buffer_len) * sizeof(char
;'buffer[buffer_len - 1] = '\0
;buffer[buffer_len - 2] = lex->c

;(lexer_advance(lex

;([return init_token(buffer, lex->automata->state_type[curr_state
{

} do
;((buffer = (char *) realloc(buffer, (++buffer_len) * sizeof(char
;'buffer[buffer_len - 1] = '\0
;buffer[buffer_len - 2] = lex->c

;[curr_state = lex->automata->automata[curr_state][lex->c

;(lexer_advance(lex

;('while(lex->automata->automata[curr_state][lex->c] != -1 && lex->c != '\0 {

;([return init_token(buffer, lex->automata->state_type[curr_state
{
```

```
} (token_T* lexer_next_token(lexer_T *lex
} ('while (lex->c != '\0
;(lexer_skip_whitespace(lex
;(lexer_skip_bullshit(lex
;(lexer_skip_whitespace(lex

;(return get_token(lex
{
;(return init_token("$", TOK_eof
{


.initilize lexer with a source //
TODO: dont need dfa src //
} (lexer_T* init_lexer(char *src
;((lexer_T *lex = malloc(sizeof(lexer_T

;lex->src = src
;(lex->src_size = strlen(src
;lex->i = 0
;[lex->c = src[lex->i
lex->automata = init_lexer_automata(LEXER_DFA_PATH,
;(LEXER_DFA_STATES_PATH

;return lex
{
"include "../include/lexer/token.h#
"include "../include/macros.h#
<include <stdlib.h#
<include <string.h#

} (token_T* init_token(char* value, int type
;((token_T* tk = calloc(1, sizeof(token_T
;(tk->value = strdup(value
;tk->type = type

;return tk
{

} (int token_cmp(const token_T *tok1, const token_T *tok2
;int delta

;delta = tok1->type - tok2->type
;(IF_SIGN(delta
```

```
;(delta = strcmp(tok1->value, tok2->value
;(IF_SIGN(delta

;return 0
{

} (int token_cmp_generic(const void *tok1, const void *tok2
;(return token_cmp((token_T *) tok1, (token_T *) tok2
"include "../include/lexer/lexer_automata.h#{
"include "../utils/lexer_DFA/include/lexer_DFA.h#
<include <stdio.h#
<include <stdlib.h#


} (static void init_automata_state_type(lexer_automata_T *automata, const char *states_src
;("FILE *fp = fopen(states_src, "r
;size_t i, state_index
;int type

handle file not opening //
} (if(fp == NULL
;("perror("Error opening states file
;(exit(EXIT_FAILURE
{

TODO: need to handle file not opening //
automata->state_type = (token_type_E *)malloc(automata->n_state *
;((sizeof(token_type_E
} (for(i = 0; i < automata->n_state; ++i
;(fscanf(fp, "%zd ", &state_index
;([fscanf(fp, "%d", &automata->state_type[state_index
{

;return
{

} (static void init_automata_mat(lexer_automata_T *automata, const char *auto_src
;("FILE *fp = fopen(auto_src, "r
;size_t rows, cols, i, j

handle file not opening //
} (if(fp == NULL
;("perror("Error opening automata file
```

```
;(exit(EXIT_FAILURE
{

get rows and columns of the dfa's 2d arr //
;(fscanf(fp, "%zd", &rows
;(fscanf(fp, "%zd", &cols

init auto self //
;automata->n_state = rows
;automata->n_symbols = cols
;((* automata->automata = (short **)malloc(automata->n_state * sizeof(short

handle malloc not working properly //
} (if(automata->automata == NULL
;("perror("Error mallocing
;(exit(EXIT_FAILURE
{

malloc rows of automata mat //
} (for(i = 0; i < automata->n_state; ++i
;((automata->automata[i] = (short *)malloc(cols * sizeof(short
{

} (for(i = 0; i < automata->n_state; ++i
} (for(j = 0; j < cols; ++j
if theres an error reading from the file, exit //
} (if(fscanf(fp, "%hd", &automata->automata[i][j]) != 1
;("perror("Error reading from file
;(fclose(fp

(for(i = 0; i < automata->n_state; ++i
;([free(automata->automata[i
;(free(automata->automata

;(exit(1
;return
{
{
{

;return
{

} (lexer_automata_T* init_lexer_automata(const char *auto_src, const char *states_src
```

```
;((lexer_automata_T *automata = malloc(sizeof(lexer_automata_T

;(init_automata_mat(automata, auto_src
;(init_automata_state_type(automata, states_src

;return automata
{
#ifndef NON_TERM
#define NON_TERM(name, symbol)
#endif

("" ,NON_TERM(null
("'"NON_TERM(start, "S

("NON_TERM(program, "program
("NON_TERM(statement_list, "statement_list
("NON_TERM(statement, "statement
("NON_TERM(declaration, "declaration
("NON_TERM(type_specifier, "type_specifier
("NON_TERM(function_definition, "function_definition
("NON_TERM(parameter_list, "parameter_list
("NON_TERM(expression_statement, "expression_statement
("NON_TERM(constant_expression, "constant_expression
("NON_TERM(logical_or_expression, "logical_or_expression
("NON_TERM(logical_and_expression, "logical_and_expression
("NON_TERM(inclusive_or_expression, "inclusive_or_expression
("NON_TERM(exclusive_or_expression, "exclusive_or_expression
("NON_TERM(and_expression, "and_expression
("NON_TERM(equality_expression, "equality_expression
("NON_TERM(relational_expression, "relational_expression
("NON_TERM(shift_expression, "shift_expression
("NON_TERM(additive_expression, "additive_expression
("NON_TERM(multiplicative_expression, "multiplicative_expression
("NON_TERM(unary_expression, "unary_expression
("NON_TERM(postfix_expression, "postfix_expression
("NON_TERM(assignment_expression, "assignment_expression
("NON_TERM(primary_expression, "primary_expression
("NON_TERM(expression, "expression
("NON_TERM(assignment_operator, "assignment_operator
("NON_TERM(unary_operator, "unary_operator
("NON_TERM(compound_statement, "compound_statement
("NON_TERM(selection_statement, "selection_statement
("NON_TERM(iteration_statement, "iteration_statement
("NON_TERM(labeled_statement, "labeled_statement
```

```
undef NON_TERM#ifndef QUEST_BNF_H#
define QUEST_BNF_H#

;(void bnf_make_non_terminals(char *src, char *dest

endif#ifndef QUEST_SYMBOL_SET#
define QUEST_SYMBOL_SET#

"include "symbol.h#
<include <stddef.h#

TODO: make a generic set, a hashset maybe //

} typedef struct SYMBOL_SET_STRUCT
;symbol_T **set
;size_t size
;symbol_set_T {

;()symbol_set_T *init_symbol_set
;(symbol_set_T *init_symbol_set_with_symbols(symbol_T **syms, const size_t size
;(int add_symbol(symbol_set_T *set, symbol_T *item
;(int remove_symbol(symbol_set_T *set, symbol_T *item

===----------= endif//===--= action_table.h#
//
//
//
===----------===//

ifndef QUEST_ACTION_TABLE_H#
define QUEST_ACTION_TABLE_H#

"include "non_terminal.h#
"include "../lexer/token.h#
<include <stddef.h#

} typedef struct ACTION_TABLE_STRUCT
// a matrix of string describing the actions the parser should do char ***actions;
// an array of terminals, a terminals index is its index in the action token_T **terminals;
table
// number of terminals size_t n_terminals;
// number of states in the action table size_t n_states;
;action_tbl_T {
```

```
;(action_tbl_T *init_action_tbl(token_T **terminals, size_t n_terminals, size_t n_states

;(int action_tbl_find_terminal(action_tbl_T *tbl, token_T *term

;(void action_tbl_print_to_file(action_tbl_T *tbl, char *dest
;(void action_tbl_pretty_print_to_file(action_tbl_T *tbl, char *dest

endif#ifndef QUEST_GRAMMER_H#
define QUEST_GRAMMER_H#

"include "non_terminal.h#
"include "rule.h#
"include "symbol.h#
"include "../lexer/token.h#
"include "../../utils/DS/include/generic_set.h#
<include <stddef.h#

} typedef struct GRAMMER_STRUCT
;set_T *rules
;set_T *symbols
;grammer_T {

;(grammer_T *init_grammer(set_T *rules, set_T *symbols

;(token_T **terminals_in_symbol_set(set_T *symbols
;(token_T **terminals_in_symbol_set_and_dollar(set_T *symbols
;(size_t n_terminals_in_symbol_set(set_T *symbols
;(non_terminal_T **non_terminals_in_symbol_set(set_T *symbols
;(size_t n_non_terminals_in_symbol_set(set_T *symbols

;(size_t find_right_grammer_index(const grammer_T *gram, symbol_T **right, size_t right_size

===----------= endif//===--= goto_table.h#
//
A struct representing a goto table. A goto table is used after reducing //
a production. Its main purpose is to map a state and a non-terminal //
.to a state //
//
===----------===//

ifndef QUEST_GOTO_TABLE_H#
define QUEST_GOTO_TABLE_H#
```

```
"include "non_terminal.h#
<include <stddef.h#

} typedef struct GOTO_TABLE_STRUCT
// a matrix of integers describing the goto the parser should do int **gotos;
non_terminal_T **non_terminals; // an array of terminals, a terminals index is its index in
the action table
// number of terminals size_t n_non_terminals;
// number of states in the goto table size_t n_states;
;goto_tbl_T {

goto_tbl_T *init_goto_tbl(non_terminal_T **non_terminals, size_t n_non_terminals, size_t
;(n_states

;(size_t goto_tbl_find_non_terminal(goto_tbl_T *tbl, non_terminal_T *nterm

;(void goto_tbl_print_to_file(goto_tbl_T *tbl, char *dest
;(void goto_tbl_pretty_print_to_file(goto_tbl_T *tbl, char *dest


endif#ifndef QUEST_NON_TERMINAL#
define QUEST_NON_TERMINAL#

} typedef enum NON_TERMINAL_ENUM
,define NON_TERM(name, symbol) NON_TERM_##name#
"include "non_terminals_bnf.h#
undef NON_TERM#
NUM_NON_TERM
;non_terminal_E {


} typedef struct NON_TERMINAL_STRUCT
;non_terminal_E type
;char *value
;non_terminal_T {

;(non_terminal_T *init_non_terminal(char *value, non_terminal_E type

;(int non_terminal_cmp(const non_terminal_T *nt1, const non_terminal_T *nt2

endif#ifndef QUEST_SLR_H#
define QUEST_SLR_H#

"include "../../utils/DS/include/generic_set.h#
```

```
"include "action_table.h#
"include "goto_table.h#
"include "grammer.h#

} typedef struct SLR_STRUCT
// cannonical collection of all lr0 items. the reason its an array of set_T **lr0_cc;
.items is because its more efficient and easy to work with
// size of cc size_t lr0_cc_size;
action_tbl_T *action; // action table
// goto table goto_tbl_T *go_to;
// grammer of the grammer_T *grammer;
;slr_T {

;(slr_T *init_slr(set_T *lr0, grammer_T *gram

;(void slr_write_to_bin(slr_T *slr, char *dest
;(slr_T *slr_read_from_bin(char *src

;()slr_T *init_default_slr

endif#ifndef QUEST_LR_STACK_H#
define QUEST_LR_STACK_H#

<include <stddef.h#

(define LR_IS_EMPTY(s) ((s)->size == 0#


} typedef struct LR_STACK_STRUCT
// top of the stack int *top;
// size of stack size_t size;
// capacity of stack size_t capacity;
size_t alloc_size; // how many bytes to allocate when stack is full
;lr_stack_T {

;(lr_stack_T *init_lr_stack(size_t alloc_size

;(void lr_stack_push(lr_stack_T *s, int data
;(int lr_stack_pop(lr_stack_T *s
;(int lr_stack_peek(lr_stack_T *s
;(int lr_stack_peek_inside(lr_stack_T *s, int n
;(int lr_stack_full(lr_stack_T *s
;(void lr_stack_clear(lr_stack_T *s
```

```
endif /* QUEST_LR_STACK_H */#ifndef QUEST_SYMBOLS#
define QUEST_SYMBOLS#

"include "../lexer/token.h#
"include "non_terminal.h#


} typedef enum SYMBOL_TYPES_ENUM
,TERMINAL
NON_TERMINAL
;symbol_type_E {

} typedef union SYMBOL_UNION
;token_T *terminal
;non_terminal_T *non_terminal
;symbol_U {

} typedef struct SYMBOL_STRUCT
;symbol_type_E sym_type
;symbol_U *symbol
;symbol_T {

;(symbol_T *init_symbol(symbol_U *symbol, symbol_type_E type
;(symbol_T *init_symbol_terminal(token_T *tok
;(symbol_T *init_symbol_non_terminal(non_terminal_T *nt

;(int symbol_equals(const symbol_T *sym1, const symbol_T *sym2
;(int symbol_cmp(const symbol_T *sym1, const symbol_T *sym2
;(int symbol_cmp_generic(const void *sym1, const void *sym2

endif#ifndef QUEST_LR_ITEM_H#
define QUEST_LR_ITEM_H#

"include "grammer.h#
"include "non_terminal.h#
"include "rule.h#
"include "symbol_set.h#
"include "symbol.h#
"include "symbol_set.h#
"include "../../utils/DS/include/generic_set.h#

} typedef struct LR_ITEM_STRUCT
;rule_T *rule
```

```
;size_t dot_index
;set_T *lookaheads
;lr_item_T {

;(lr_item_T *init_lr_item(rule_T *rule, size_t dot_index, set_T *lookaheads

;(int lr_item_cmp(const lr_item_T *item1, const lr_item_T *item2
;(int lr_item_cmp_generic(const void *item1, const void *item2
;(int lr_item_set_cmp(const set_T *set1, const set_T *set2
;(int lr_item_set_cmp_generic(const void *item1, const void *item2

;(set_T *first(const grammer_T *gram, const symbol_T *sym
;(set_T *follow(const grammer_T *gram, const non_terminal_T *nt

;(set_T *closure(grammer_T *grammer, set_T *items
;(set_T *go_to(grammer_T *grammer, set_T *items, symbol_T *symbol
;(set_T *closure_lookahead(grammer_T *grammer, set_T *items
;(set_T *go_to_lookahead(grammer_T *grammer, set_T *items, symbol_T *symbol

;(set_T *lr0_items(grammer_T *grammer, lr_item_T *starting_item
;(set_T *lr1_items(grammer_T *grammer, lr_item_T *starting_item
;(set_T *lalr_items(grammer_T *grammer, lr_item_T *starting_item

endif#ifndef QUEST_PARSE_TREE_NODE_H#
define QUEST_PARSE_TREE_NODE_H#

"include "rule.h#
"include "symbol.h#
<include <stddef.h#
<include <sys/types.h#

} typedef struct PARSE_TREE_NODE_STRUCT
// symbol in curr node on parse tree symbol_T *symbol;
// index of rule in rules arr (or -1 indicating that it ssize_t rule_index;
(has no rule where its in the ledt side
struct PARSE_TREE_NODE_STRUCT **children; // arr of children of parse tree node
// number of children the node has size_t n_children;
;parse_tree_node_T {

} typedef struct PARSE_TREE_STRUCT
// root node of parse tree parse_tree_node_T *root;
;(// array of all rules. in array so getting of rule is theta(1 rule_T **rules;
// number of rules size_t n_rules;
;parse_tree_T {
```

```
;(parse_tree_T *init_parse_tree(parse_tree_node_T *root, rule_T **rules, size_t n_rules

;(void parse_tree_free(parse_tree_node_T *tree

parse_tree_node_T *init_parse_tree_node(symbol_T *sym, ssize_t rule_index,
;(parse_tree_node_T **children, size_t n_children
;(parse_tree_node_T *init_parse_tree_leaf(symbol_T *sym

;(void parse_tree_traverse_preorder(parse_tree_node_T *tree, int layer
;(void parse_tree_traverse_postorder(parse_tree_node_T *tree, int layer

endif /* QUEST_PARSE_TREE_NODE_H */#ifndef NON_TERM#
(define NON_TERM(name, symbol#
endif#

("" ,NON_TERM(null
("'NON_TERM(start, "S

undef NON_TERM#
ifndef QUEST_PARSER_H#
define QUEST_PARSER_H#

define LETTERS_SIZE 26#

"include "action_table.h#
"include "goto_table.h#
"include "grammer.h#
"include "lr_stack.h#
"include "parse_tree.h#
"include "rule.h#
"include "slr.h#
"include "../../utils/DS/include/queue.h#
"include "symbol.h#

} typedef enum PARSE_STATUS_TYPE_ENUM
,ERR
,ACCEPT
,SHIFT
,REDUCE
N_PARSE_STATUS
;parse_status_type_E {

} typedef struct PARSER_STRUCT
```

```
action_tbl_T *action;        // action table
goto_tbl_T *go_to;           // goto table
rule_T **rules;              // array of all rules. in array
;(so getting of rule is theta(1
size_t n_rules;              // number of rules
lr_stack_T *stack;           // lr stack of parser
;parser_T {

;(parser_T *init_parser(slr_T *slr

;(parse_tree_T *parse(parser_T *prs, queue_T *queue_tok
;(void parser_shift(parser_T *prs, int data
;(symbol_T *parser_reduce(parser_T *prs, rule_T *rule

===----------= endif//===--= Rule.h#
//
(A grammer rule. (add a rule explanation and grammer and maybe example //
//
//
//
//
===----------===//

ifndef QUEST_RULE_H#
define QUEST_RULE_H#

"include "../lexer/token.h#
"include "non_terminal.h#
"include "symbol.h#
<include <stddef.h#

} typedef struct RULE_STRUCT
;non_terminal_T *left
;symbol_T **right
;size_t right_size
;rule_T {

;(rule_T *init_rule(non_terminal_T *left, symbol_T **right, size_t right_size

;(int rule_cmp(const rule_T *rule1, const rule_T *rule2
;(int rule_cmp_generic(const void *rule1, const void *rule2

;(int find_first_nt(const rule_T *rule, int offset
```

```
endif#ifndef QUEST_MACROS_H#
define QUEST_MACROS_H#

"define TOKS_PATH "/home/goodman/school/Quest/src/include/lexer/tokens.h#
"define LEXER_DFA_PATH "/home/goodman/school/Quest/build/lexer_dfa.dat#
define LEXER_DFA_STATES_PATH#
""/home/goodman/school/Quest/build/lexer_dfa_states.dat
define LEXER_DFA_STATES_DETAILS_PATH#
""/home/goodman/school/Quest/build/lexer_dfa_states_details.dat

"define PARSER_ACTION_PATH "/home/goodman/school/Quest/build/parser_action.dat#
define PARSER_ACTION_PRETTY_PATH#
""/home/goodman/school/Quest/build/parser_action_pretty.dat
"define PARSER_GOTO_PATH "/home/goodman/school/Quest/build/parser_goto.dat#
define PARSER_GOTO_PRETTY_PATH#
""/home/goodman/school/Quest/build/parser_goto_pretty.dat
"define PARSER_BNF "/home/goodman/school/Quest/resources/language/quest.ebnf#
define PARSER_BNF_NON_TERMINALS#
""/home/goodman/school/Quest/src/include/parser/non_terminals_bnf.h
"define PARSER_BNF_XLAT "/home/goodman/school/Quest/resources/slr_xlat#
"define PARSER_SLR_BIN "/home/goodman/school/Quest/build/slr.bin#

\" define PARSER_NON_TERMINALS_HEADER#
\ifndef NON_TERM \n#
\define NON_TERM(name, symbol) \n#
\endif \n#
\n\
\NON_TERM(null, \"\") \n
\NON_TERM(start, \"S'\") \n
"
"define PARSER_NON_TERMINALS_FOOTER "\n#undef NON_TERM#


define MAX(a, b) a > b ? a : b#
define MIN(a, b) a < b ? a : b#
{;(((define IF_SIGN(x) if((x)) {return ((x) / (abs(x#

define print_item(x) printf("[%s -> %s {%zu}] rs: %d\n", x->rule->left->value,#
x->rule->right[0]->sym_type == TERMINAL ? x->rule->right[0]->symbol->terminal->value :
;(x->rule->right[0]->symbol->non_terminal->value, x->dot_index ,x->rule->right_size


endif#ifndef QUEST_LEXER_AUTOMATA_H#
define QUEST_LEXER_AUTOMATA_H#
```

```
#include "token.h"

typedef struct LEXER_AUTOMATA_STRUCT {
    short **automata;    // mat representing the automata
    unsigned int n_symbols;    // number of symbols from the start of the
    unsigned int n_state;    // number of states
    token_type_E *state_type;    // arr representing states types: deny, accept...
};lexer_automata_T

lexer_automata_T *init_lexer_automata(const char *auto_src, const char *states_src);

#ifndef QUEST_LEXER_H#endif
#define QUEST_LEXER_H
#include "lexer_automata.h"
#include "token.h"
#include <stddef.h>

typedef struct LEXER_STRUCT {
    char* src;    // src code
    size_t src_size;    // size of src code
    char c;    // curr character the lexer is on
    unsigned int i;    // curr index the lexer is on
    lexer_automata_T *automata;    // automata of the lexer
};lexer_T

lexer_T* init_lexer(char *src);

void lexer_advance(lexer_T* lex);

token_T* lexer_advance_with(lexer_T* lex, token_T* tk);

token_T* lexer_advance_current(lexer_T* lex, int type);

char lexer_peek(lexer_T* lex, int offset);

void lexer_skip_whitespace(lexer_T* lex);

token_T* lexer_parse_id(lexer_T* lex);

token_T* lexer_parse_number(lexer_T* lex);

token_T* lexer_next_token(lexer_T* lex);
```

```
endif#ifndef QUEST_TOKEN_H#
define QUEST_TOKEN_H#

} typedef enum TOKEN_ENUM
,define TOK(name, lexeme, val, is_kw) TOK_##name val#
"include "tokens.h#
undef TOK#
NUM_TOK = 80
;token_type_E {


} typedef struct TOKEN_STRUCT
;char *value
;token_type_E type
;token_T {


;(token_T *init_token(char *value, int type
;(int token_cmp(const token_T *tok1, const token_T *tok2
;(int token_cmp_generic(const void *tok1, const void *tok2

===---------- endif//===--- The Quest Language: Tokens#
//
.In here all of Quest's tokens are defined //
Macros are used to define the tokens for external //
.use, like defining enums and functions //
//
//
WORK IN PROGRESS //
//
...TODO: not necessary, but maybe organize by type, defenitions, etc - //
===----------===//

ifndef TOK#
(define TOK(name, lexeme, val, is_kw#
endif#
ifndef BASETOK#
(, 0 , TOK(name, define BASETOK(name)#
endif#
ifndef DEBUG#
(, 0 , TOK(name, define DEBUG(name)#
endif#
ifndef KEYWORD#
(, 1 TOK(name, lexeme, define KEYWORD(name, lexeme)#
```

```
endif#
ifndef PUNCTUATOR#
(, 0   TOK(name, lexeme,       define PUNCTUATOR(name, lexeme)#
endif#
ifndef PUNCTUAVAL#
(define PUNCTUAVAL(name, lexeme, val) TOK(name, lexeme, =val, 0#
endif#
ifndef OPERATOR#
(, 1   TOK(name, lexeme,           define OPERATOR(name, lexeme)#
endif#
ifndef OPERAVAL#
(define OPERAVAL(name, lexeme, val)   TOK(name, lexeme, =val, 1#
endif#

------------------//
(Debug Tokens (4 //
------------------//

(DEBUG(null
(DEBUG(UNKNOWN
(DEBUG(eof
(DEBUG(COMMENT

------------------//
(Base (4 //
------------------//

(BASETOK(IDENTIFIER
(BASETOK(NUMBER_CONSTANT
(BASETOK(CHAR_CONSTANT
(BASETOK(STRING_LITERAL

------------------//
(Keywords (24 //
------------------//

(""bool    KEYWORD(BOOL,
(""break  KEYWORD(BREAK,
(""case    KEYWORD(CASE,
(""char    KEYWORD(CHAR,
(""const  KEYWORD(CONST,
("KEYWORD(CONTINUE, "continue
(""do        KEYWORD(DO,
("KEYWORD(DOUBLE,   "double
```

```
(""else     KEYWORD(ELSE,
(""false   KEYWORD(FALSE,
(""float   KEYWORD(FLOAT,
(""for     KEYWORD(FOR,
(""if      KEYWORD(IF,
(""int     KEYWORD(INT,
(""long    KEYWORD(LONG,
(""ret     KEYWORD(RET,
(""short   KEYWORD(SHORT,
("KEYWORD(SIGNED,   "signed
("KEYWORD(SWITCH,   "switch
(""true    KEYWORD(TRUE,
("KEYWORD(TYPEDEF,  "typedef
("KEYWORD(UNSIGNED, "unsigned
(""void    KEYWORD(VOID,
(""while   KEYWORD(WHILE,

------------------//
(Operators (38 //
------------------//

one character symbols have theire ASCII value used as theire token's value //
('.' ,"."      ,OPERAVAL(DOT
(',' ,","      ,OPERAVAL(COMMA
('~' ,"~"      ,OPERAVAL(TILDE
('!' ,"!"      ,OPERAVAL(NOT
('&' ,"&"      ,OPERAVAL(BITAND
('|' ,"|"      ,OPERAVAL(BITOR
('^' ,"^"      ,OPERAVAL(BITXOR
('!' ,"!"      ,OPERAVAL(BITNOT
('*' ,"*"      ,OPERAVAL(STAR
('/' ,"/"      ,OPERAVAL(SLASH
('+' ,"+"      ,OPERAVAL(PLUS
('-' ,"-"      ,OPERAVAL(MINUS
('%' ,"%"      ,OPERAVAL(PRECENT
('<' ,"<"      ,OPERAVAL(GREATER
('>' ,">",OPERAVAL(LESSER
('=' ,"="      ,OPERAVAL(EQUEL

("..."         ,OPERATOR(ELLIPSES
("<-"          ,OPERATOR(ARROW
("&&"          ,OPERATOR(AND
("||"          ,OPERATOR(OR
("^^"          ,OPERATOR(XOR
```

```
(">>"      ,OPERATOR(SHIFTLEFT
("<<"   ,OPERATOR(SHIFTRIGHT
("=="          ,OPERATOR(EQUELEQUEL
("=<"      ,OPERATOR(GREATEREQUEL
("=>"         ,OPERATOR(LESSEREQUEL
("=+"   ,OPERATOR(PLUSEQUEL
("=-"          ,OPERATOR(MINUSEQUEL
("=*"   ,OPERATOR(STAREQUEL
("=/"          ,OPERATOR(SLASHEQUEL
("=%"      ,OPERATOR(PRECENTEQUEL
("=&"     ,OPERATOR(ANDEQUEL
("=|"       ,OPERATOR(OREQUEL
("=^"     ,OPERATOR(XOREQUEL
("=!"     ,OPERATOR(NOTEQUEL
("=~"   ,OPERATOR(TILDEEQUEL
("=>>"  ,OPERATOR(SHIFTLEFTEQUEL
("=<<" ,OPERATOR(SHIFTRIGHTEQUEL

------------------//
(Punctuators (10 //
------------------//

one character symbols have theire ASCII value used as theire token's value //
(')' ,")"       ,PUNCTUAVAL(LPAREN
('(' ,"("       ,PUNCTUAVAL(RPAREN
(']' ,"]"       ,PUNCTUAVAL(LBRACK
('[' ,"["       ,PUNCTUAVAL(RBRACK
('}' ,"}"       ,PUNCTUAVAL(LBRACE
('{' ,"{"       ,PUNCTUAVAL(RBRACE
('>' ,">"  ,PUNCTUAVAL(LCHEVRON
('<' ,"<"  ,PUNCTUAVAL(RCHEVRON
(':' ,":"        ,PUNCTUAVAL(COLON
(';' ,";" ,PUNCTUAVAL(SEMICOLON


#undef ALIAS
#undef OPERAVAL
#undef OPERATOR
#undef PUNCTUAVAL
#undef PUNCTUATOR
#undef KEYWORD
#undef TOK
#ifndef QUEST_CODE_GENERATOR_H
#define QUEST_CODE_GENERATOR_H
```

```
#include "TTS.h"
#include "register.h"
#include "../semantic_analizer/AST.h"
#include "../../utils/symbol_table/include/symbol_table_tree.h"
#include <stdint.h>

typedef struct CODE_GENERATOR_STRUCT {
    register_pool_T **registers;
    tts_T *tts;
    symbol_table_tree_T *sym_tbl;
    uint32_t label_counter;
    char *output;
} code_gen_T;

code_gen_T *init_code_gen(register_pool_T **registers, tts_T *tts, symbol_table_tree_T *sym_tbl);

char *generate_code(ast_node_T *ast, code_gen_T *cg);


// ----------------==
// Registers
// ----------------==

static const register_pool_T NASM_REGS[NUM_REG] = {
    {RAX, 0, DATA},
    {RBX, 0, DATA},
    {RCX, 0, DATA},
    {RDX, 0, DATA},
    {RSP, 0, POINTER},
    {RBP, 0, POINTER},
    {RSI, 0, INDEX},
    {RDI, 0, INDEX},
    {R8,  0, SCRATCH},
    {R9,  0, SCRATCH},
    {R10, 0, SCRATCH},
    {R11, 0, SCRATCH},
    {R12, 0, SCRATCH},
    {R13, 0, SCRATCH},
    {R14, 0, SCRATCH},
    {R15, 0, SCRATCH},
};
```

```
endif /* QUEST_CODE_GENERATOR_H */#ifndef QUEST_NASM_MACROS_H#
define QUEST_NASM_MACROS_H#

==----------== //
op-code //
==----------== //

"define DATA_SECTION "section .data\n#
"define TEXT_SECTION "section .text\n#
"define BSS_SECTION "section .bss\n#

"define GLOBAL(x) "global " x "\n#
"define EXTERN(x) "extern " x "\n#

"define LABEL(x) x ":\n#

"define DB(x) "%s db " x "\n#
"define DW(x) "%s dw " x "\n#
"define DD(x) "%s dd " x "\n#
"define DQ(x) "%s dq " x "\n#

"define RESB(x) "%s resb " x "\n#
"define RESW(x) "%s resw " x "\n#
"define RESD(x) "%s resd " x "\n#
"define RESQ(x) "%s resq " x "\n#

"[" (define MEM(x) "[" (x#

==----------== //
Instructions //
==----------== //

"define NOP "nop\n#

"define MOV "mov %s, %s\n#
"define MOV_MEM "mov [%s], %s\n#
"define MOV_MEM_TYPE "mov %s [%s], %s\n#
"define MOV_REG_TYPE "mov %s, [%s]\n#

"define OR "or %s, %s\n#
"define XOR "xor %s, %s\n#
"define AND "and %s, %s\n#
"define NOT "not %s\n#
"define NEG "neg %s\n#
```

```
"define ADD "add %s, %s\n#
"define SUB "sub %s, %s\n#
"define MUL "mul %s\n#
"define DIV "div %s\n#

"define SHL "shl %s, %s\n#
"define SHR "shr %s, %s\n#
"define SAR "sar %s, %s\n#

"define CMP "cmp %s, %s\n#

"define JE "je %s\n#
"define JNE "jne %s\n#
"define JG "jg %s\n#
"define JL "jl %s\n#
"define JGE "jge %s\n#
"define JLE "jle %s\n#
"define JZ "jz %s\n#
"define JNZ "jnz %s\n#
"define JMP "jmp %s\n#

"define SETG "setg %s\n#
"define SETL "setl %s\n#
"define SETE "sete %s\n#
"define SETNE "setne %s\n#

"define PUSH "push %s\n#
"define POP "pop %s\n#

==----------== //
misc //
==----------== //
#type(TYPE_TO_SIZE(define) )        \
DWORD ? TOK_INT == type :   \
BYTE ? TOK_CHAR == type :   \
0   \
)

#size(SIZE_TO_STR(define) )        \
"DWORD ? DWORD == size : \
"WORD ? WORD == size :   \
"BYTE ? BYTE == size :   \
0   \
```

```
(

"define END_PROGRAM "mov rax, 60\n" "mov rdi, 0\n" "syscall\n#


endif /* QUEST_NASM_MACROS_H *///===--= TTS.h#
===----------------------------------------------------=
//
the header for the tree translation scheme //
//
===-----------------------------------------------------------------===//

ifndef QUEST_TTS_H#
define QUEST_TTS_H#

"include "translation_rule.h#
} typedef struct TREE_TRANSLATION_SCHEME_STRUCT
;translation_rule_T **tok_translation
;size_t n_tok
;translation_rule_T **non_term_translation
;size_t n_non_term
;tts_T {

tts_T *init_tts(translation_rule_T **tok_translation, size_t n_tok, translation_rule_T
;(**non_term_translation, size_t n_non_term
tts_T *create_tts(translation_rule_T **tok_translation, size_t n_tok, translation_rule_T
;(**non_term_translation, size_t n_non_term

endif /* QUEST_TTS_H */#ifndef QUEST_REGISTER_H#
define QUEST_REGISTER_H#

"include "../lexer/token.h#
<include <stdint.h#

// 0000 0011 define LOW_BITS 0x3#
define HIGH_BITS 0xC // 0000 1100#

define NUM_REG 16#
define NUM_SIZE 4#

} typedef enum REGISTER_ENUM
,RAX
,RCX
,RDX
```

```
,RBX
,RSP
,RBP
,RSI
,RDI
,R8
,R9
,R10
,R11
,R12
,R13
,R14
,R15
;register_E {

} typedef enum REGISTER_SIZE
,BYTE = 1
,WORD = 2
,DWORD = 4
QWORD = 8
;register_size_E {

} typedef enum REGISTER_TYPE
,DATA = 1
,POINTER = 2
,INDEX = 4
,SEGMENT = 8
,CONTROL = 16
,FLAGS = 32
,STACK = 64
SCRATCH = 128
;register_type_E {

} = [static const char *REGS_STR[NUM_REG][NUM_SIZE
,{"al", "ax", "eax", "rax"}
,{"cl", "cx", "ecx", "rcx"}
,{"dl", "dx", "edx", "rdx"}
,{"bl", "bx", "ebx", "rbx"}
,{"spl", "sp", "esp", "rsp"}
,{"bpl", "bp", "ebp", "rbp"}
,{"sil", "si", "esi", "rsi"}
,{"dil", "di", "edi", "rdi"}
,{"r8b", "r8w", "r8d", "r8"}
,{"r9b", "r9w", "r9d", "r9"}
```

```
,{"r10b", "r10w", "r10d", "r10"}
,{"r11b", "r11w", "r11d", "r11"}
,{"r12b", "r12w", "r12d", "r12"}
,{"r13b", "r13w", "r13d", "r13"}
,{"r14b", "r14w", "r14d", "r14"}
{"r15b", "r15w", "r15d", "r15"}
;{

} = [static const char *DATA_REGS_STR[4][2
,{"al", "ah"}
,{"cl", "ch"}
,{"dl", "dh"}
{"bl", "bh"}
;{

} typedef struct REGISTER_STRUCT
// reg type register_E reg;
register_size_E size; // reg size
// name char *name;
;register_T {

} typedef struct REGISTER_POOL_STRUCT
// reg pool value uint64_t value;
// is reg or part of reg in use uint64_t in_use;
register_type_E type; // type of reg
;register_pool_T {

;(register_T *init_register(uint8_t type, uint8_t size, char *name
;(register_pool_T *init_register_pool(uint8_t type

;(register_T *get_register(register_pool_T **regs, uint8_t type, uint8_t size
;(char *get_register_name(uint8_t type, uint8_t size
;(char *get_byte_data_reg_name(uint8_t type, uint8_t size


;(void reg_alloc(register_T *reg, uint64_t value
;(void reg_free(register_pool_T **regs, register_T *reg

/* endif /* QUEST_REGISTER_H#
ifndef QUEST_TRANSLATION_RULE_H#
define QUEST_TRANSLATION_RULE_H#

"include "../parser/symbol.h#
"include "../semantic_analizer/AST.h#
```

```
"include "../../utils/DS/include/stack.h#
"include "register.h#

} typedef struct TRANSLATION_RULE_STRUCT
;symbol_T *symbol
char *(*translation)(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
;(register_pool_T **regs
;translation_rule_T {

translation_rule_T *init_translation_rule(symbol_T *symbol, char *(*translation)(ast_node_T *ast,
;((stack_T *astack, stack_T *code_stack, register_pool_T **regs


endif /* QUEST_TRANSLATION_RULE_H */#ifndef QUEST_OPERAND_H#
define QUEST_OPERAND_H#

"include "register.h#
"include "../parser/symbol.h#

} typedef enum OPERAND_TYPES_ENUM
,SYMBOL
REGISTER
;operand_type_E {

} typedef union OPERAND_UNION
;symbol_T *sym
;register_T *reg
;operand_U {

} typedef struct OPERAND_STRUCT
;operand_type_E type
;operand_U *operand
;operand_T {

;(operand_T *init_operand_symbol(symbol_T *sym
;(operand_T *init_operand_register(register_T *reg

;(void operand_free(operand_T *op

endif /* QUEST_OPERAND_H *///===--= translations.h#
===----------------------------------------------------=
//
,here are all the translations that are used in the code generator //
they are stored in the translation table and each translation correlates to //
```

```
a pattern in the AST //
//
===----------------------------------------------------------------------------===//

ifndef QUEST_TRANSLATIONS_H#
define QUEST_TRANSLATIONS_H#

"include "../semantic_analizer/AST.h#
"include "register.h#
"include "../../utils/DS/include/stack.h#

tok //
char *trans_num_const(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
;(register_pool_T **regs
;(char *trans_id(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T **regs
char *trans_plus(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
;(**regs
char *trans_minus(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
;(**regs
char *trans_assign(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
;(**regs
char *trans_greater(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
;(**regs
char *trans_less(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
;(**regs


nt //
char *trans_decl(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
;(**regs
char *trans_selection_stmt(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
;(register_pool_T **regs
char *trans_iteration_stmt(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
;(register_pool_T **regs

endif /* QUEST_TRANSLATIONS_H */#ifndef QUEST_SDT_H#
define QUEST_SDT_H#

"include "../../utils/DS/include/generic_set.h#
"include "semantic_rule.h#

} typedef struct SDT_STRUCT
;semantic_rule_T **definitions
;size_t n_defenitions
```

```
;sdt_T {

;(sdt_T *init_sdt(semantic_rule_T **definitions, size_t n_defenitions
;(sdt_T *init_default_sdt(rule_T **rules, size_t n_rules

/* endif /* QUEST_SDT_H#
===----------------------------------------= semantic_rule.h =--===//
//
a semantic rule //
//
===----------------------------------------------------------------===//

ifndef QUEST_SEMANTIC_RULE_H#
define QUEST_SEMANTIC_RULE_H#

"include "../parser/parse_tree.h#
"include "../../utils/DS/include/stack.h#
"include "../../utils/symbol_table/include/symbol_table.h#
"include "AST.h#

} typedef struct SEMANTIC_RULE_STRUCT
// production rule, for example, E -> rule_T *rule;
E + T
void (*definition)(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);   // definition
of rule, for example. E.val = E1.val + T.val, implemented by function ptr
;semantic_rule_T {

semantic_rule_T *init_sementic_rule(rule_T *rule, void (*definition)(stack_T *astack,
;((parse_tree_node_T *tree, stack_T *st_s

endif /* QUEST_SEMANTIC_RULE_H *///===--= definitions.h#
===----------------------------------------------=
//
Here are all the prototypes for definition for each rule, written to build an AST //
.using the parse tree //
//
===----------------------------------------------------------------===//

ifndef QUEST_DEFINITIONS_H#
define QUEST_DEFINITIONS_H#

"include "AST.h#
"include "../parser/parse_tree.h#
"include "../../utils/DS/include/stack.h#
```

```
#include "../../utils/symbol_table/include/symbol_table.h"


/*
each rule should have a corresponding definition
*/
void definition_start_r(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_program_sl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_sl_s(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_sl_sl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_s_exp_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_s_compound_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_s_selection_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_s_iteration_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_s_labeled_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_s_decl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_decl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_type_int(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_type_char(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_type_float(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_type_void(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);

void definition_exp_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_cnstnt_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);

void definition_assignment_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
stack_T *st_s);
void definition_assignment_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_logical_or_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
*st_s);
void definition_logical_or_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_logical_and_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
stack_T *st_s);
void definition_logical_and_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_inclusive_or_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
stack_T *st_s);
void definition_inclusive_or_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_exclusive_or_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
stack_T *st_s);
void definition_exclusive_or_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_and_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_and_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_equality_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
*st_s);
```

```
void definition_equality_equal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_equality_notequal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_relational_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_relational_less_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_relational_less_equal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_relational_greater_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_relational_greater_equal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_shift_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_shift_lshift_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_shift_rshift_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_additive_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_additive_plus_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_additive_minus_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_multiplicative_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_multiplicative_multiply_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_multiplicative_divide_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_multiplicative_mod_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_primary_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_primary_exp_id(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_primary_exp_constant(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_primary_exp_str(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_exp_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_exp_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);

void definition_compount_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);

void definition_selection_stmt_if(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
void definition_selection_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);

void definition_iteration_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s);
```

```
} = (static void (*def_fns[])(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
                                                                    ,definition_start_r
                                                                    ,definition_program_sl
                                                                    ,definition_sl_s
                                                                    ,definition_sl_sl
                                                                    ,definition_s_exp_stmt
                                                                    ,definition_s_compound_stmt
                                                                    ,definition_s_selection_stmt
                                                                    ,definition_s_iteration_stmt
                                                                    ,definition_s_labeled_stmt
                                                                    ,definition_s_decl
                                                                    ,definition_decl
                                                                    ,definition_type_int
                                                                    ,definition_type_char
                                                                    ,definition_type_float
                                                                    ,definition_type_void

                                                                    ,definition_exp_stmt
                                                                    ,definition_cnstnt_exp

                                                                    ,definition_assignment_exp_precedence
                                                                    ,definition_assignment_exp
                                                                    ,definition_logical_or_exp_precedence
                                                                    ,definition_logical_or_exp
                                                                    ,definition_logical_and_exp_precedence
                                                                    ,definition_logical_and_exp
                                                                    ,definition_inclusive_or_exp_precedence
                                                                    ,definition_inclusive_or_exp
                                                                    ,definition_exclusive_or_exp_precedence
                                                                    ,definition_exclusive_or_exp
                                                                    ,definition_and_exp_precedence
                                                                    ,definition_and_exp
                                                                    ,definition_equality_exp_precedence
                                                                    ,definition_equality_equal_exp
                                                                    ,definition_equality_notequal_exp
                                                                    ,definition_relational_exp_precedence
                                                                    ,definition_relational_less_exp
                                                                    ,definition_relational_less_equal_exp
                                                                    ,definition_relational_greater_exp
                                                                    ,definition_relational_greater_equal_exp
                                                                    ,definition_shift_exp_precedence
                                                                    ,definition_shift_lshift_exp
                                                                    ,definition_shift_rshift_exp
                                                                    ,definition_additive_exp_precedence
```

```
,definition_additive_plus_exp
,definition_additive_minus_exp
,definition_multiplicative_exp_precedence
,definition_multiplicative_multiply_exp
,definition_multiplicative_divide_exp
,definition_multiplicative_mod_exp
,definition_primary_exp_precedence
,definition_primary_exp_id
,definition_primary_exp_constant
,definition_primary_exp_str
,definition_exp_exp_precedence
,definition_exp_exp

,definition_compount_stmt

,definition_selection_stmt_if
,definition_selection_stmt

,definition_iteration_stmt
;{


endif#ifndef QUEST_AST_H#
define QUEST_AST_H#

"include "../parser/symbol.h#
"include "../../utils/symbol_table//include/symbol_table.h#
<include <stddef.h#

} typedef struct ABSTRACT_SYNTAX_TREE_NODE_STRUCT
;symbol_T *symbol
;struct ABSTRACT_SYNTAX_TREE_NODE_STRUCT **children
;size_t n_children
;symbol_table_entry_T *st_entry
;ast_node_T {

;(ast_node_T *init_ast_node(symbol_T *symbol, ast_node_T **children, size_t n_children
;(ast_node_T *init_ast_leaf(symbol_T *symbol

;(void ast_add_to_node(ast_node_T *ast, ast_node_T *child

;(void traverse_ast(ast_node_T *ast, int layer

endif /* QUEST_AST_H */#ifndef QUEST_SEMANTIC_ANALYZER_H#
```

```
define QUEST_SEMANTIC_ANALYZER_H#

"include "../parser/parse_tree.h#
"include "../quest.h#
"include "AST.h#
"include "sdt.h#

;(ast_node_T *build_ast(parse_tree_T *tree, quest_T *q

endif#ifndef QUEST_IO_H#
define QUEST_IO_H#

;(char* read_file(const char *filename
;(void write_file(const char *filename, const char *content
;(void write_file_append(const char *filename, char *content
;(void print_to_stdout(char *content
;(void print_to_stderr(char *content
;(char *get_new_filename(const char *filename, const char *new_name


endif#ifndef QUEST_QUEST_H#
define QUEST_QUEST_H#

"include "code_gen/code_generator.h#
"include "lexer/lexer.h#
"include "parser/parser.h#
"include "semantic_analizer/sdt.h#

} typedef struct QUEST_STRUCT
;char *srcfile
;char *destfile
;char *src
;lexer_T *lexer
;parser_T *parser
;sdt_T *sdt
;code_gen_T *code_gen
;quest_T {

;(void compile(quest_T *q

;(void compile_file(const char *filename

/* endif /* QUEST_QUEST_H#
```

```
ifndef QUEST_LANG_H#
define QUEST_LANG_H#

"include "quest.h#

;(quest_T *init_quest(const char *filename

"endif#include "include/io.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#
<include <sys/types.h#

reads the contents of a file //
} (char* read_file(const char *filename
;FILE *fp
;char *line = NULL
;size_t len = 0
;ssize_t read
;char* buffer

;("fp = fopen(filename, "rb
} (if(fp == NULL
;(printf("Couldn't read file %s\n", filename
;(exit(1
{

;((buffer = malloc(sizeof(char
;'buffer[0] = '\0

} ((while((read = getline(&line, &len, fp) != -1
;((buffer = realloc(buffer, (strlen(buffer) + strlen(line) + 1) * sizeof(char
;(strcat(buffer, line
{

;(fclose(fp
(if(line
;(free(line

;return buffer
{

writes to a file //
```

```
} (void write_file(const char *filename, const char *content
;FILE *fp
;("fp = fopen(filename, "w
} (if(fp == NULL
;("perror("Couldn't write to file
;(exit(1
{

} (if(fputs(content, fp) == EOF
;("perror("Error writing to file
;(exit(1
{
;(fclose(fp
{

append to a file //
} (void write_file_append(const char *filename, char *content
;FILE *fp
;("fp = fopen(filename, "a
} (if(fp == NULL
;(printf("Couldn't write to file %s\n", filename
;(exit(1
{
;(fprintf(fp, "%s", content
;(fclose(fp
{

prints to stdout //
} (void print_to_stdout(char *content
;(printf("%s", content
{

prints to stderr //
} (void print_to_stderr(char *content
;(fprintf(stderr, "%s", content
{

get filename and a new name, create new file to be in the same folder as filename and the //
name of the file is provided, unless the name is NULL then its .out
} (char *get_new_filename(const char *filename, const char *new_name
;char *new_filename
;(size_t len = strlen(filename
;('/' ,const char *last_slash = strrchr(filename
```

```
(if(last_slash
;len = last_slash - filename + 1

new_filename = malloc(len + (new_name ? strlen(new_name) : 5) + 1); // +1 for null terminator

;(strncpy(new_filename, filename, len
;'new_filename[len] = '\0

(if(new_name
;(strcat(new_filename, new_name
else
;("strcat(new_filename, ".out

;return new_filename
{

"include "include/lang.h#
"include "include/code_gen/TTS.h#
"include "include/code_gen/translation_rule.h#
"include "include/code_gen/translations.h#
"include "include/io.h#
"include "include/lexer/token.h#
"include "include/parser/lr_item.h#
"include "include/parser/parser.h#
"include "include/parser/rule.h#
"include "include/parser/slr.h#
"include "include/macros.h#
"include "include/parser/symbol.h#
"include "include/quest.h#
"include "include/semantic_analizer/definitions.h#
"include "include/semantic_analizer/sdt.h#
"include "include/semantic_analizer/semantic_rule.h#
"include "utils/DS/include/generic_set.h#
"include "utils/err/err.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#


TODO: make a sysmtem that doesnt need to have all these defs //
} (static slr_T *init_default_lang(quest_T *q
;int i
*/
```

```
symbols
/*
;(set_T *syms = set_init(symbol_cmp_generic

;((symbol_T *start = init_symbol_non_terminal(init_non_terminal("S'", NON_TERM_start
symbol_T *labeled_statement =
init_symbol_non_terminal(init_non_terminal("labeled_statement",
;((NON_TERM_labeled_statement
symbol_T *iteration_statement =
init_symbol_non_terminal(init_non_terminal("iteration_statement",
;((NON_TERM_iteration_statement
symbol_T *selection_statement =
init_symbol_non_terminal(init_non_terminal("selection_statement",
;((NON_TERM_selection_statement
symbol_T *compound_statement =
init_symbol_non_terminal(init_non_terminal("compound_statement",
;((NON_TERM_compound_statement
symbol_T *unary_operator =
;((init_symbol_non_terminal(init_non_terminal("unary_operator", NON_TERM_unary_operator
symbol_T *assignment_operator =
init_symbol_non_terminal(init_non_terminal("assignment_operator",
;((NON_TERM_assignment_operator
symbol_T *expression = init_symbol_non_terminal(init_non_terminal("expression",
;((NON_TERM_expression
symbol_T *primary_expression =
init_symbol_non_terminal(init_non_terminal("primary_expression",
;((NON_TERM_primary_expression
symbol_T *assignment_expression =
init_symbol_non_terminal(init_non_terminal("assignment_expression",
;((NON_TERM_assignment_expression
symbol_T *postfix_expression =
init_symbol_non_terminal(init_non_terminal("postfix_expression",
;((NON_TERM_postfix_expression
symbol_T *unary_expression =
init_symbol_non_terminal(init_non_terminal("unary_expression",
;((NON_TERM_unary_expression
symbol_T *multiplicative_expression =
init_symbol_non_terminal(init_non_terminal("multiplicative_expression",
;((NON_TERM_multiplicative_expression
symbol_T *additive_expression =
init_symbol_non_terminal(init_non_terminal("additive_expression",
;((NON_TERM_additive_expression
symbol_T *shift_expression =
;((init_symbol_non_terminal(init_non_terminal("shift_expression", NON_TERM_shift_expression
```

```
symbol_T *relational_expression =
init_symbol_non_terminal(init_non_terminal("relational_expression",
((NON_TERM_relational_expression);
symbol_T *equality_expression =
init_symbol_non_terminal(init_non_terminal("equality_expression",
((NON_TERM_equality_expression);
symbol_T *and_expression =
((init_symbol_non_terminal(init_non_terminal("and_expression", NON_TERM_and_expression);
symbol_T *exclusive_or_expression =
init_symbol_non_terminal(init_non_terminal("exclusive_or_expression",
((NON_TERM_exclusive_or_expression);
symbol_T *inclusive_or_expression =
init_symbol_non_terminal(init_non_terminal("inclusive_or_expression",
((NON_TERM_inclusive_or_expression);
symbol_T *logical_and_expression =
init_symbol_non_terminal(init_non_terminal("logical_and_expression",
((NON_TERM_logical_and_expression);
symbol_T *logical_or_expression =
init_symbol_non_terminal(init_non_terminal("logical_or_expression",
((NON_TERM_logical_or_expression);
symbol_T *constant_expression =
init_symbol_non_terminal(init_non_terminal("constant_expression",
((NON_TERM_constant_expression);
symbol_T *expression_statement =
init_symbol_non_terminal(init_non_terminal("expression_statement",
((NON_TERM_expression_statement);
symbol_T *parameter_list =
((init_symbol_non_terminal(init_non_terminal("parameter_list", NON_TERM_parameter_list);
symbol_T *function_definition =
init_symbol_non_terminal(init_non_terminal("function_definition",
((NON_TERM_function_definition);
symbol_T *type_specifier = init_symbol_non_terminal(init_non_terminal("type_specifier",
((NON_TERM_type_specifier);
symbol_T *declaration = init_symbol_non_terminal(init_non_terminal("declaration",
((NON_TERM_declaration);
symbol_T *statement = init_symbol_non_terminal(init_non_terminal("statement",
((NON_TERM_statement);
symbol_T *statement_list = init_symbol_non_terminal(init_non_terminal("statement_list",
((NON_TERM_statement_list);
symbol_T *program = init_symbol_non_terminal(init_non_terminal("program",
((NON_TERM_program);

(set_add(syms, start);
(set_add(syms, labeled_statement);
```

```
;(set_add(syms, iteration_statement
;(set_add(syms, selection_statement
;(set_add(syms, compound_statement
;(set_add(syms, unary_operator
;(set_add(syms, assignment_operator
;(set_add(syms, expression
;(set_add(syms, primary_expression
;(set_add(syms, assignment_expression
;(set_add(syms, postfix_expression
;(set_add(syms, unary_expression
;(set_add(syms, multiplicative_expression
;(set_add(syms, additive_expression
;(set_add(syms, shift_expression
;(set_add(syms, relational_expression
;(set_add(syms, equality_expression
;(set_add(syms, and_expression
;(set_add(syms, exclusive_or_expression
;(set_add(syms, inclusive_or_expression
;(set_add(syms, logical_and_expression
;(set_add(syms, logical_or_expression
;(set_add(syms, constant_expression
;(set_add(syms, expression_statement
;(set_add(syms, parameter_list
;(set_add(syms, function_definition
;(set_add(syms, type_specifier
;(set_add(syms, declaration
;(set_add(syms, statement
;(set_add(syms, statement_list
;(set_add(syms, program


;((symbol_T *logical_or = init_symbol_terminal(init_token("||", TOK_OR
;((symbol_T *logical_and = init_symbol_terminal(init_token("&&", TOK_AND
;((symbol_T *bitor = init_symbol_terminal(init_token("|", TOK_BITOR
;((symbol_T *bitxor = init_symbol_terminal(init_token("^", TOK_BITXOR
;((symbol_T *bitand = init_symbol_terminal(init_token("&", TOK_BITAND
;((symbol_T *equal_equal = init_symbol_terminal(init_token("==", TOK_EQUELEQUEL
;((symbol_T *not_equal = init_symbol_terminal(init_token("!=", TOK_NOTEQUEL
;((symbol_T *less = init_symbol_terminal(init_token("<", TOK_LESSER
;((symbol_T *less_equal = init_symbol_terminal(init_token("<=", TOK_LESSEREQUEL
;((symbol_T *greater = init_symbol_terminal(init_token(">", TOK_GREATER
symbol_T *greater_equal = init_symbol_terminal(init_token(">=",
;((TOK_GREATEREQUEL
;((symbol_T *lshift = init_symbol_terminal(init_token("<<", TOK_SHIFTLEFT
```

```
;((symbol_T *rshift = init_symbol_terminal(init_token(">>", TOK_SHIFTRIGHT
;((symbol_T *plus = init_symbol_terminal(init_token("+", TOK_PLUS
;((symbol_T *minus = init_symbol_terminal(init_token("-", TOK_MINUS
;((symbol_T *divide = init_symbol_terminal(init_token("/", TOK_SLASH
;((symbol_T *multiply = init_symbol_terminal(init_token("*", TOK_STAR
;((symbol_T *mod = init_symbol_terminal(init_token("%", TOK_PRECENT
;((symbol_T *not = init_symbol_terminal(init_token("!", TOK_NOT
;((symbol_T *bitnot = init_symbol_terminal(init_token("~", TOK_BITNOT

;((symbol_T *assign = init_symbol_terminal(init_token("=", TOK_EQUEL

;((symbol_T *comma = init_symbol_terminal(init_token(",", TOK_COMMA
;((symbol_T *semicolon = init_symbol_terminal(init_token(";", TOK_SEMICOLON
;((symbol_T *lparen = init_symbol_terminal(init_token("(", TOK_LPAREN
;((symbol_T *rparen = init_symbol_terminal(init_token(")", TOK_RPAREN
;((symbol_T *lbracket = init_symbol_terminal(init_token("[", TOK_LBRACK
;((symbol_T *rbracket = init_symbol_terminal(init_token("]", TOK_RBRACK
;((symbol_T *lbrace = init_symbol_terminal(init_token("{", TOK_LBRACE
;((symbol_T *rbrace = init_symbol_terminal(init_token("}", TOK_RBRACE

;((symbol_T *if_tok = init_symbol_terminal(init_token("if", TOK_IF
;((symbol_T *else_tok = init_symbol_terminal(init_token("else", TOK_ELSE
;((symbol_T *while_tok = init_symbol_terminal(init_token("while", TOK_WHILE

;((symbol_T *id = init_symbol_terminal(init_token("id", TOK_IDENTIFIER
;((symbol_T *num = init_symbol_terminal(init_token("num", TOK_NUMBER_CONSTANT
;((symbol_T *string = init_symbol_terminal(init_token("str", TOK_STRING_LITERAL

;((symbol_T *int_tok = init_symbol_terminal(init_token("int", TOK_INT
;((symbol_T *char_tok = init_symbol_terminal(init_token("char", TOK_CHAR

;((symbol_T *return_tok = init_symbol_terminal(init_token("ret", TOK_RET

;(set_add(syms, logical_or
;(set_add(syms, logical_and
;(set_add(syms, bitor
;(set_add(syms, bitxor
;(set_add(syms, bitand
;(set_add(syms, equal_equal
;(set_add(syms, not_equal
;(set_add(syms, less
;(set_add(syms, less_equal
;(set_add(syms, greater
;(set_add(syms, greater_equal
```

```
;(set_add(syms, lshift
;(set_add(syms, rshift
;(set_add(syms, plus
;(set_add(syms, minus
;(set_add(syms, divide
;(set_add(syms, multiply
;(set_add(syms, mod
;(set_add(syms, not
;(set_add(syms, bitnot
;(set_add(syms, assign
;(set_add(syms, comma
;(set_add(syms, semicolon
;(set_add(syms, lparen
;(set_add(syms, rparen
;(set_add(syms, lbracket
;(set_add(syms, rbracket
;(set_add(syms, lbrace
;(set_add(syms, rbrace
;(set_add(syms, if_tok
;(set_add(syms, else_tok
;(set_add(syms, while_tok
;(set_add(syms, id
;(set_add(syms, num
;(set_add(syms, string
;(set_add(syms, int_tok
;(set_add(syms, char_tok
;(set_add(syms, return_tok

*/
symbol lists
/*

;{symbol_T *stat_list[] = {statement_list, statement

symbol_T *declaration_list[] = {type_specifier, id, assign, constant_expression,
;{semicolon

;{symbol_T *exp_stmt_list[] = {constant_expression, semicolon

symbol_T *assignment_expression_list[] = {assignment_expression, assign,
;{logical_or_expression

symbol_T *logic_or_exp_list[] = {logical_or_expression, logical_or,
;{logical_and_expression
```

```
symbol_T *logic_and_exp_list[] = {logical_and_expression, logical_and,
;{inclusive_or_expression

symbol_T *inclusive_or_expression_list[] = {inclusive_or_expression, bitor,
;{exclusive_or_expression

symbol_T *exclusive_or_expression_list[] = {exclusive_or_expression, bitxor,
;{and_expression

;{symbol_T *and_expression_list[] = {and_expression, bitand, equality_expression

symbol_T *equality_expression_equal_list[] = {equality_expression, equal_equal,
;{relational_expression
symbol_T *equality_expression_notequal_list[] = {equality_expression, not_equal,
;{relational_expression

symbol_T *relational_expression_less_list[] = {relational_expression, less,
;{shift_expression
symbol_T *relational_expression_greater_list[] = {relational_expression, greater,
;{shift_expression
symbol_T *relational_expression_equal_greater_list[] = {relational_expression,
;{greater_equal, shift_expression
symbol_T *relational_expression_equal_less_list[] = {relational_expression, less_equal,
;{shift_expression

;{symbol_T *shift_expression_lshift_list[] = {shift_expression, lshift, additive_expression
;{symbol_T *shift_expression_rshift_list[] = {shift_expression, rshift, additive_expression

symbol_T *additive_expression_plus_list[] = {additive_expression, plus,
;{multiplicative_expression
symbol_T *additive_expression_minus_list[] = {additive_expression, minus,
;{multiplicative_expression

symbol_T *multiplicative_expression_multiply_list[] = {multiplicative_expression, multiply,
;{primary_expression
symbol_T *multiplicative_expression_divide_list[] = {multiplicative_expression, divide,
;{primary_expression
symbol_T *multiplicative_expression_mod_list[] = {multiplicative_expression, mod,
;{primary_expression

;{symbol_T *primary_expression_list[] = {lparen, expression, rparen

;{symbol_T *expression_list[] = {expression, comma, constant_expression
```

```
;{symbol_T *compound_stmt_list[] = {lbrace, statement_list, rbrace

symbol_T *selection_stmt_if_list[] = {if_tok, lparen, constant_expression, rparen,
;{compound_statement
symbol_T *selection_stmt_list[] = {if_tok, lparen, constant_expression, rparen,
;{compound_statement, else_tok, compound_statement

symbol_T *iteration_stmt_while[] = {while_tok, lparen, constant_expression, rparen,
;{compound_statement


*/
rules
/*
;(set_T *rules = set_init(rule_cmp_generic

;(rule_T *start_r = init_rule(start->symbol->non_terminal, &program, 1
;(rule_T *program_sl = init_rule(program->symbol->non_terminal, &statement_list, 1

;(rule_T *sl_s = init_rule(statement_list->symbol->non_terminal, &statement, 1
;(rule_T *sl_sl = init_rule(statement_list->symbol->non_terminal, stat_list, 2

rule_T *s_exp_stmt = init_rule(statement->symbol->non_terminal,
;(&expression_statement, 1
rule_T *s_compound_stmt = init_rule(statement->symbol->non_terminal,
;(&compound_statement, 1
rule_T *s_selection_stmt = init_rule(statement->symbol->non_terminal,
;(&selection_statement, 1
rule_T *s_iteration_stmt = init_rule(statement->symbol->non_terminal,
;(&iteration_statement, 1
rule_T *s_jump_stmt = init_rule(statement->symbol->non_terminal, &jump_statement, //
1); dont need for now
rule_T *s_labeled_stmt = init_rule(statement->symbol->non_terminal,
;(&labeled_statement, 1
;(rule_T *s_decl = init_rule(statement->symbol->non_terminal, &declaration, 1

;(rule_T *decl = init_rule(declaration->symbol->non_terminal, declaration_list, 5

;(rule_T *type_int = init_rule(type_specifier->symbol->non_terminal, &int_tok, 1
;(rule_T *type_char = init_rule(type_specifier->symbol->non_terminal, &char_tok, 1

*/
```

```
function
param-list
param
/*

rule_T *exp_stmt = init_rule(expression_statement->symbol->non_terminal,
;(exp_stmt_list, 2

rule_T *cnstnt_exp = init_rule(constant_expression->symbol->non_terminal,
;(&assignment_expression, 1

rule_T *assignment_exp_precedence =
;(init_rule(assignment_expression->symbol->non_terminal, &logical_or_expression, 1
rule_T *assignment_exp = init_rule(assignment_expression->symbol->non_terminal,
;(assignment_expression_list, 3

rule_T *logical_or_exp_precedence =
;(init_rule(logical_or_expression->symbol->non_terminal, &logical_and_expression, 1
rule_T *logical_or_exp = init_rule(logical_or_expression->symbol->non_terminal,
;(logic_or_exp_list, 3

rule_T *logical_and_exp_precedence =
;(init_rule(logical_and_expression->symbol->non_terminal, &inclusive_or_expression, 1
rule_T *logical_and_exp = init_rule(logical_and_expression->symbol->non_terminal,
;(logic_and_exp_list, 3

rule_T *inclusive_or_exp_precedence =
;(init_rule(inclusive_or_expression->symbol->non_terminal, &exclusive_or_expression, 1
rule_T *inclusive_or_exp = init_rule(inclusive_or_expression->symbol->non_terminal,
;(inclusive_or_expression_list, 3

rule_T *exclusive_or_exp_precedence =
;(init_rule(exclusive_or_expression->symbol->non_terminal, &and_expression, 1
rule_T *exclusive_or_exp = init_rule(exclusive_or_expression->symbol->non_terminal,
;(exclusive_or_expression_list, 3

rule_T *and_exp_precedence = init_rule(and_expression->symbol->non_terminal,
;(&equality_expression, 1
rule_T *and_exp = init_rule(and_expression->symbol->non_terminal,
;(and_expression_list, 3

rule_T *equality_exp_precedence =
;(init_rule(equality_expression->symbol->non_terminal, &relational_expression, 1
```

```
rule_T *equality_equal_exp = init_rule(equality_expression->symbol->non_terminal,
;(equality_expression_equal_list, 3
rule_T *equality_notequal_exp = init_rule(equality_expression->symbol->non_terminal,
;(equality_expression_notequal_list, 3

rule_T *relational_exp_precedence =
;(init_rule(relational_expression->symbol->non_terminal, &shift_expression, 1
rule_T *relational_less_exp = init_rule(relational_expression->symbol->non_terminal,
;(relational_expression_less_list, 3
rule_T *relational_greater_exp = init_rule(relational_expression->symbol->non_terminal,
;(relational_expression_greater_list, 3
rule_T *relational_less_equal_exp =
init_rule(relational_expression->symbol->non_terminal, relational_expression_equal_less_list,
;(3
rule_T *relational_greater_equal_exp =
init_rule(relational_expression->symbol->non_terminal,
;(relational_expression_equal_greater_list, 3

rule_T *shift_exp_precedence = init_rule(shift_expression->symbol->non_terminal,
;(&additive_expression, 1
rule_T *shift_lshift_exp = init_rule(shift_expression->symbol->non_terminal,
;(shift_expression_lshift_list, 3
rule_T *shift_rshift_exp = init_rule(shift_expression->symbol->non_terminal,
;(shift_expression_rshift_list, 3

rule_T *additive_exp_precedence =
;(init_rule(additive_expression->symbol->non_terminal, &multiplicative_expression, 1
rule_T *additive_plus_exp = init_rule(additive_expression->symbol->non_terminal,
;(additive_expression_plus_list, 3
rule_T *additive_minus_exp = init_rule(additive_expression->symbol->non_terminal,
;(additive_expression_minus_list, 3

rule_T *multiplicative_exp_precedence =
;(init_rule(multiplicative_expression->symbol->non_terminal, &primary_expression, 1
rule_T *multiplicative_multiply_exp =
init_rule(multiplicative_expression->symbol->non_terminal,
;(multiplicative_expression_multiply_list, 3
rule_T *multiplicative_divide_exp =
init_rule(multiplicative_expression->symbol->non_terminal,
;(multiplicative_expression_divide_list, 3
rule_T *multiplicative_mod_exp =
init_rule(multiplicative_expression->symbol->non_terminal, multiplicative_expression_mod_list,
;(3
```

```
rule_T *primary_exp_precedence =
;(init_rule(primary_expression->symbol->non_terminal, primary_expression_list, 3
;(rule_T *primary_exp_id = init_rule(primary_expression->symbol->non_terminal, &id, 1
rule_T *primary_exp_constant = init_rule(primary_expression->symbol->non_terminal,
;(&num, 1
rule_T *primary_exp_str = init_rule(primary_expression->symbol->non_terminal, &string,
;(1

rule_T *exp_exp_precedence = init_rule(expression->symbol->non_terminal,
;(&constant_expression, 1
;(rule_T *exp_exp = init_rule(expression->symbol->non_terminal, expression_list, 3

rule_T *compount_stmt = init_rule(compound_statement->symbol->non_terminal,
;(compound_stmt_list, 3

rule_T *selection_stmt_if = init_rule(selection_statement->symbol->non_terminal,
;(([selection_stmt_if_list, sizeof(selection_stmt_if_list) / sizeof(selection_stmt_if_list[0
rule_T *selection_stmt = init_rule(selection_statement->symbol->non_terminal,
;(([selection_stmt_list, sizeof(selection_stmt_list) / sizeof(selection_stmt_list[0

rule_T *iteration_stmt = init_rule(iteration_statement->symbol->non_terminal,
;(([iteration_stmt_while, sizeof(iteration_stmt_while) / sizeof(iteration_stmt_while[0

;(set_add(rules, start_r
;(set_add(rules, program_sl
;(set_add(rules, sl_s
;(set_add(rules, sl_sl
;(set_add(rules, s_exp_stmt
;(set_add(rules, s_compound_stmt
;(set_add(rules, s_selection_stmt
;(set_add(rules, s_iteration_stmt
;(set_add(rules, s_labeled_stmt
;(set_add(rules, s_decl
;(set_add(rules, decl
;(set_add(rules, type_int
;(set_add(rules, type_char

;(set_add(rules, exp_stmt
;(set_add(rules, cnstnt_exp

;(set_add(rules, assignment_exp_precedence
;(set_add(rules, assignment_exp
;(set_add(rules, logical_or_exp_precedence
;(set_add(rules, logical_or_exp
```

```
;(set_add(rules, logical_and_exp_precedence
;(set_add(rules, logical_and_exp
;(set_add(rules, inclusive_or_exp_precedence
;(set_add(rules, inclusive_or_exp
;(set_add(rules, exclusive_or_exp_precedence
;(set_add(rules, exclusive_or_exp
;(set_add(rules, and_exp_precedence
;(set_add(rules, and_exp
;(set_add(rules, equality_exp_precedence
;(set_add(rules, equality_equal_exp
;(set_add(rules, equality_notequal_exp
;(set_add(rules, relational_exp_precedence
;(set_add(rules, relational_less_exp
;(set_add(rules, relational_less_equal_exp
;(set_add(rules, relational_greater_exp
;(set_add(rules, relational_greater_equal_exp
;(set_add(rules, shift_exp_precedence
;(set_add(rules, shift_lshift_exp
;(set_add(rules, shift_rshift_exp
;(set_add(rules, additive_exp_precedence
;(set_add(rules, additive_plus_exp
;(set_add(rules, additive_minus_exp
;(set_add(rules, multiplicative_exp_precedence
;(set_add(rules, multiplicative_multiply_exp
;(set_add(rules, multiplicative_divide_exp
;(set_add(rules, multiplicative_mod_exp
;(set_add(rules, primary_exp_precedence
;(set_add(rules, primary_exp_id
;(set_add(rules, primary_exp_constant
;(set_add(rules, primary_exp_str
;(set_add(rules, exp_exp_precedence
;(set_add(rules, exp_exp

;(set_add(rules, compount_stmt

;(set_add(rules, selection_stmt_if
;(set_add(rules, selection_stmt

;(set_add(rules, iteration_stmt


;(set_flip(rules

;(grammer_T *gram = init_grammer(rules, syms
```

```
;((set_T *itms = lr0_items(gram, init_lr_item(start_r, 0, NULL
;(slr_T *slr = init_slr(itms, gram

;(action_tbl_print_to_file(slr->action, PARSER_ACTION_PATH
;(action_tbl_pretty_print_to_file(slr->action, PARSER_ACTION_PRETTY_PATH
;(goto_tbl_print_to_file(slr->go_to, PARSER_GOTO_PATH
;(goto_tbl_pretty_print_to_file(slr->go_to, PARSER_GOTO_PRETTY_PATH

;(q->parser = init_parser(slr

*/
sdt
/*

;((* semantic_rule_T **srs = calloc(q->parser->n_rules, sizeof(semantic_rule_T

} (for(i = 0; i < q->parser->n_rules; ++i
;([srs[i] = init_sementic_rule(q->parser->rules[i], def_fns[i
{

;(q->sdt = init_sdt(srs, q->parser->n_rules

*/
code_gen
/*
} = []translation_rule_T *tts_tok
,(init_translation_rule(num, &trans_num_const
,(init_translation_rule(id, &trans_id
,(init_translation_rule(plus, &trans_plus
,(init_translation_rule(minus, &trans_minus
,(init_translation_rule(assign, &trans_assign
,(init_translation_rule(greater, &trans_greater
,(init_translation_rule(less, &trans_less

;{

} = []translation_rule_T *tts_nt
,(init_translation_rule(declaration, &trans_decl
,(init_translation_rule(selection_statement, &trans_selection_stmt
(init_translation_rule(iteration_statement, &trans_iteration_stmt
;{

)q->code_gen = init_code_gen
```

```
                                                                    ,NULL
create_tts(tts_tok, sizeof(tts_tok) / sizeof(tts_nt[0]), tts_nt, sizeof(tts_nt) /
                                                                    ,(([sizeof(tts_nt[0
                                                                    NULL
                                                                    ;(

                                                                    ;return slr
                                                                    {

                                                                    } (quest_T *init_quest(const char *filename
                                                                    ;((quest_T *q = malloc(sizeof(quest_T
                                                                    (if(!q
                                                                    ;(thrw(ALLOC_ERR

                                                                    ;(q->srcfile = strdup(filename
                                                                    ;("q->destfile = get_new_filename(filename, "out.asm
                                                                    ;(q->src = read_file(filename
                                                                    ;(q->lexer = init_lexer(q->src

                                                                    ;(init_default_lang(q

                                                                    ;return q
                                                                    "include "../include/code_gen/TTS.h#{
                                                                    "include "../utils/err/err.h#

                                                                    <include <stdlib.h#

tts_T *init_tts(translation_rule_T **tok_translation, size_t n_tok, translation_rule_T
                                                                    } (**non_term_translation, size_t n_non_term
                                                                    ;((tts_T *tts = malloc(sizeof(tts_T
                                                                    (if(!tts
                                                                    ;(thrw(ALLOC_ERR

                                                                    ;tts->tok_translation = tok_translation
                                                                    ;tts->n_tok = n_tok
                                                                    ;tts->non_term_translation = non_term_translation
                                                                    ;tts->n_non_term = n_non_term

                                                                    ;return tts
                                                                    {

tts_T *create_tts(translation_rule_T **tok_translation, size_t n_tok, translation_rule_T
                                                                    } (**non_term_translation, size_t n_non_term
                                                                    ;int i
```

```
;((tts_T *tts = malloc(sizeof(tts_T
(if(!tts
;(thrw(ALLOC_ERR

;tts->n_tok = NUM_TOK
;tts->n_non_term = NUM_NON_TERM

;((* tts->tok_translation = calloc(tts->n_tok, sizeof(translation_rule_T
(if(!tts->tok_translation
;(thrw(ALLOC_ERR

tts->non_term_translation = calloc(tts->n_non_term, sizeof(translation_rule_T *) *
;(n_non_term
(if(!tts->non_term_translation
;(thrw(ALLOC_ERR

(for(i = 0; i < n_tok; ++i
tts->tok_translation[tok_translation[i]->symbol->symbol->terminal->type] =
;[tok_translation[i

(for(i = 0; i < n_non_term; ++i

tts->non_term_translation[non_term_translation[i]->symbol->symbol->non_terminal->type] =
;[non_term_translation[i

;return tts
{
"include "../include/code_gen/code_generator.h#
"include "../include/code_gen/nasm_macros.h#
"include "../utils/err/err.h#
"include "../utils/DS/include/stack.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#

} ()static register_pool_T **copy_nasm_regs
;int i = 0
;(register_pool_T **tmp = malloc(sizeof(register_pool_T *) * NUM_REG
(if(!tmp
;(thrw(ALLOC_ERR

} (for(;i < NUM_REG; ++i
;((tmp[i] = malloc(sizeof(register_T
([if(!tmp[i
```

```
;(thrw(ALLOC_ERR

;[tmp[i] = NASM_REGS[i*
{
;return tmp
{

} (static void generate_text_section(code_gen_T *cg
cg->output = realloc(cg->output, strlen(cg->output) + strlen(TEXT_SECTION) +
;(strlen(GLOBAL("_start")) + strlen(LABEL("_start")) + 1
;(strcat(cg->output, TEXT_SECTION
;(("strcat(cg->output, GLOBAL("_start
;(("strcat(cg->output, LABEL("_start
{

} (static char *generate_global_variables(code_gen_T *cg
;int i
;symbol_table_entry_T *cur
;char *tmp

;(cg->output = realloc(cg->output, strlen(cg->output) + strlen(BSS_SECTION) + 1
;(strcat(cg->output, BSS_SECTION

} (for(i = 0; i < cg->sym_tbl->root->table->capacity; ++i
;[cur = cg->sym_tbl->root->table->buckets[i

} (while(cur
;(tmp = malloc(strlen(RESB("1")) + strlen(cur->name) + 1

} (switch (cur->type
:case TOK_INT
;(sprintf(tmp, RESD("1"), cur->name
;(strcat(cg->output, tmp
;break

:case TOK_CHAR
;(sprintf(tmp, RESB("1"), cur->name
;(strcat(cg->output, tmp
;break

:default
;break
{
```

```
;cur = cur->next
{
{

;return cg->output
{

:reverser code string using its lines. for exampla //
mov DWORD [a], ecx //
mov ecx, 94 //
[mov eax, [a //
should be //
[mov eax, [a //
mov ecx, 94 //
mov DWORD [a], ecx //
} (static char *reverse_code(char *code
;()stack_T *code_stack = stack_init
;("char *tmp = strtok(code, "\n

} (while(tmp
;((stack_push(code_stack, strdup(tmp
;("tmp = strtok(NULL, "\n
{

;(free(code
;(code = calloc(1, 1
} ((while(!IS_EMPTY(code_stack
;(tmp = stack_pop(code_stack
;(code = realloc(code, strlen(code) + strlen(tmp) + 2
;(strcat(code, tmp
;("strcat(code, "\n
{

;return code
{

static char *generate_code_rec(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
} (code_gen_T *cg, char *tmp
(if(!ast
;return NULL

;int i
} (for(i = 0; i < ast->n_children ; ++i
;(tmp = generate_code_rec(ast->children[i], astack, stack_init(), cg, NULL
```

```
(if(tmp
;((stack_push(code_stack, strdup(tmp

;(free(tmp
{

;tmp = NULL
} (if(ast->symbol->sym_type == TERMINAL
([if(cg->tts->tok_translation[ast->symbol->symbol->terminal->type
tmp = cg->tts->tok_translation[ast->symbol->symbol->terminal->type]->translation(ast,
;(astack, code_stack, cg->registers
{
} else
([if(cg->tts->non_term_translation[ast->symbol->symbol->non_terminal->type
tmp =
cg->tts->non_term_translation[ast->symbol->symbol->non_terminal->type]->translation(ast,
;(astack, code_stack, cg->registers
{

flip stack //
;char *tmp2

} ((while(!IS_EMPTY(code_stack
;(tmp2 = stack_pop(code_stack

(if(!tmp
;(tmp = strdup(tmp2
} else
;(tmp = realloc(tmp, strlen(tmp) + strlen(tmp2) + 1
;((strcat(tmp, strdup(tmp2
{
{

;return tmp
{

} (static char *gen_code_instructions(ast_node_T *ast, code_gen_T *cg
;()stack_T *astack = stack_init
;()stack_T *code_stack = stack_init

;((return reverse_code(generate_code_rec(ast, astack, code_stack, cg, NULL
{
```

```
code_gen_T *init_code_gen(register_pool_T **registers, tts_T *tts, symbol_table_tree_T
} (*sym_tbl
;((code_gen_T *cg = malloc(sizeof(code_gen_T
(if(!cg
;(thrw(ALLOC_ERR

cg->registers = registers
registers ?
;()copy_nasm_regs :
;cg->tts = tts
;cg->sym_tbl = sym_tbl
;cg->label_counter = 0
;(cg->output = calloc(1, 1
(if(!cg->output
;(thrw(ALLOC_ERR

;return cg
{

} (char *generate_code(ast_node_T *ast, code_gen_T *cg
;((cg->output = strdup(generate_global_variables(cg
;(generate_text_section(cg

;((strcat(cg->output, gen_code_instructions(ast, cg
;(strcat(cg->output, END_PROGRAM

;return cg->output
"include "../include/code_gen/operand.h#{
"include "../utils/err/err.h#
<include <stdlib.h#

} (operand_T *init_tts_node_symbol(symbol_T *sym
;((operand_T *node = malloc(sizeof(operand_T
(if(!node
;(thrw(ALLOC_ERR

;node->type = SYMBOL
;node->operand->sym = sym

;return node
{

} (operand_T *init_tts_node_register(register_T *reg
;((operand_T *node = malloc(sizeof(operand_T
```

```
(if(!node
;(thrw(ALLOC_ERR

;node->type = REGISTER
;node->operand->reg = reg

;return node
{
"include "../include/code_gen/register.h#
"include "../utils/err/err.h#
<include <stdint.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#

special case for data byte registers //
} (static register_T *get_data_byte_reg(register_pool_T **regs
;int i = 0

} (for(;i < NUM_REG; ++i
(if(regs[i]->type != DATA
;continue

} ((if(!(regs[i]->in_use & LOW_BITS
;regs[i]->in_use = regs[i]->in_use | LOW_BITS
;((return init_register(i, BYTE, get_byte_data_reg_name(i, LOW_BITS
} ((else if(!(regs[i]->in_use & HIGH_BITS {
;regs[i]->in_use = regs[i]->in_use | HIGH_BITS
;((return init_register(i, BYTE, get_byte_data_reg_name(i, HIGH_BITS
{
{

;(thrw(REG_NOT_FOUND_ERR
;return NULL
{

general case //
} (static register_T *get_reg_reg(register_pool_T **regs, uint8_t type, uint8_t size
;int i = 0
;uint64_t bits_on = (1 << (size * 2)) - 1

} (for(;i < NUM_REG; ++i
} ((if(regs[i]->type == type && !(regs[i]->in_use & bits_on
;regs[i]->in_use = regs[i]->in_use | bits_on
```

```
;(return init_register(i, size, NULL
{
{

;(thrw(REG_NOT_FOUND_ERR
;return NULL
{

} (register_T *init_register(uint8_t type, uint8_t size, char *name
;((register_T *reg = malloc(sizeof(register_T
(if(!reg
;(thrw(ALLOC_ERR

;reg->reg = type
;reg->size = size
;(reg->name = name ? name : get_register_name(type, size

;return reg
{

} (register_pool_T *init_register_pool(uint8_t type
;((register_pool_T *pool = malloc(sizeof(register_pool_T
(if(!pool
;(thrw(ALLOC_ERR

;pool->value = 0
;pool->in_use = 0
;pool->type = type

;return pool
{

} (register_T *get_register(register_pool_T **regs, uint8_t type, uint8_t size
return type == DATA  && size == BYTE
(get_data_byte_reg(regs ?
;(get_reg_reg(regs, type, size :
{

} (char *get_register_name(uint8_t type, uint8_t size
;int size_num = 0

(while (size >>= 1
;++size_num
```

```
;([return strdup(REGS_STR[type][size_num
{

} (char *get_byte_data_reg_name(uint8_t type, uint8_t bits
;([return strdup(DATA_REGS_STR[type][bits == HIGH_BITS
{

} (void reg_alloc(register_T *reg, uint64_t value //
;reg->value = value //
;reg->is_used = 1 //

;return //
{ //

} (void reg_free(register_pool_T **regs, register_T *reg
;(uint64_t bits_off = ~((1 << (reg->size * 2)) - 1

;regs[reg->reg]->in_use &= bits_off
{

"include "../include/code_gen/translations.h#
"include "../include/code_gen/nasm_macros.h#
"include "../include/code_gen/register.h#
"include "../utils/err/err.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#

} ()static char *alloc_instruction_mem
;(char *tmp = malloc(1
;'tmp = '\0*
(if(!tmp
;(thrw(ALLOC_ERR

;return tmp
{

} (static char *reverse_code(char *code
;()stack_T *code_stack = stack_init
;("char *tmp = strtok(code, "\n

} (while(tmp
;((stack_push(code_stack, strdup(tmp
;("tmp = strtok(NULL, "\n
```

```
{

;(free(code
;(code = calloc(1, 1
} ((while(!IS_EMPTY(code_stack
;(tmp = stack_pop(code_stack
;(code = realloc(code, strlen(code) + strlen(tmp) + 2
;(strcat(code, tmp
;("strcat(code, "\n
{

;return code
{


char *trans_num_const(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
} (register_pool_T **regs
;()char *tmp = alloc_instruction_mem

register_T *reg = get_register(regs, DATA, DWORD);  // TODO: maybe have access to
paren to see type size? or have symbols have types
;(sprintf(tmp, MOV, reg->name, ast->symbol->symbol->terminal->value
;(stack_push(astack, reg

;return tmp
{

(char *trans_id(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T **regs
}
;()char *tmp = alloc_instruction_mem

;(register_T *reg = get_register(regs, DATA, DWORD
;(sprintf(tmp, MOV_REG_TYPE, reg->name, ast->symbol->symbol->terminal->value
;(stack_push(astack, reg

;return tmp
{

char *trans_plus(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
} (**regs
;()char *tmp = alloc_instruction_mem

;(register_T *reg2 = stack_pop(astack
;(register_T *reg1 = stack_pop(astack
```

```
;(sprintf(tmp, ADD, reg1->name, reg2->name
;(stack_push(astack, reg1
;(reg_free(regs, reg2

;return tmp
{

char *trans_minus(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
} (**regs
;()char *tmp = alloc_instruction_mem

;(register_T *reg2 = stack_pop(astack
;(register_T *reg1 = stack_pop(astack

;(sprintf(tmp, SUB, reg1->name, reg2->name
;(stack_push(astack, reg1
;(reg_free(regs, reg2

;return tmp
{

char *trans_assign(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
} (**regs
;()char *tmp = alloc_instruction_mem

;(register_T *reg_const = stack_pop(astack
;((reg_free(regs, stack_pop(astack

sprintf(tmp, MOV_MEM_TYPE, SIZE_TO_STR(reg_const->size),
;(ast->children[0]->symbol->symbol->terminal->value, reg_const->name
;(reg_free(regs, reg_const

;return tmp
{

char *trans_greater(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
} (**regs
;()char *tmp1 = alloc_instruction_mem
;()char *tmp2 = alloc_instruction_mem

;(register_T *reg1 = stack_pop(astack
;(register_T *reg2 = stack_pop(astack
```

```
;(sprintf(tmp1, CMP, reg2->name, reg1->name
;(reg_free(regs, reg1
;(reg_free(regs, reg2

;(reg1 = get_register(regs, DATA, BYTE
;(sprintf(tmp2, SETG, reg1->name
;(strcat(tmp1, tmp2
;(stack_push(astack, reg1

;(free(tmp2
;((return strdup(reverse_code(tmp1
{

char *trans_less(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
} (**regs
;()char *tmp1 = alloc_instruction_mem
;()char *tmp2 = alloc_instruction_mem
;(register_T *reg1 = stack_pop(astack
;(register_T *reg2 = stack_pop(astack

;(sprintf(tmp1, CMP, reg2->name, reg1->name
;(reg_free(regs, reg1
;(reg_free(regs, reg2

;(reg1 = get_register(regs, DATA, BYTE
;(sprintf(tmp2, SETL, reg1->name
;(strcat(tmp1, tmp2
;(stack_push(astack, reg1

;(free(tmp2
;((return strdup(reverse_code(tmp1
{


char *trans_decl(ast_node_T *ast, stack_T *astack, stack_T *code_stack, register_pool_T
} (**regs
;()char *tmp = alloc_instruction_mem

;(register_T *reg_const = stack_pop(astack
;((reg_free(regs, stack_pop(astack

sprintf(tmp, MOV_MEM_TYPE, SIZE_TO_STR(reg_const->size),
;(ast->children[1]->symbol->symbol->terminal->value, reg_const->name
;(reg_free(regs, reg_const
```

```
;return tmp
{

char *trans_selection_stmt(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
} (register_pool_T **regs
;()char *tmp1 = alloc_instruction_mem
;()char *tmp2 = alloc_instruction_mem
;()char *tmp3 = alloc_instruction_mem
;()char *tmp4 = alloc_instruction_mem
;()char *tmp5 = alloc_instruction_mem

;char *tmp_exp = NULL
;char *tmp_if = NULL
;char *tmp_else = NULL

;(register_T *reg_exp = stack_pop(astack

if without else // //
} (if(ast->n_children == 2
;(tmp_if = stack_pop(code_stack
;(tmp_exp = stack_pop(code_stack

;("sprintf(tmp1, CMP, reg_exp->name, "0
;("sprintf(tmp2, JZ, "end_if
;(("sprintf(tmp3, LABEL("end_if

tmp3 = realloc(tmp3, strlen(tmp_exp) + strlen(tmp1) + strlen(tmp2) + strlen(tmp_if) +
;(strlen(tmp3) + 1
;(strcat(tmp3, tmp_if
;(strcat(tmp3, tmp2
;(strcat(tmp3, tmp1
;(strcat(tmp3, tmp_exp

;(tmp5 = strdup(tmp3
{
if with else //
} (else if(ast->n_children == 3
;(tmp_else = stack_pop(code_stack
;(tmp_if = stack_pop(code_stack
;(tmp_exp = stack_pop(code_stack

;("sprintf(tmp1, CMP, reg_exp->name, "0
;("sprintf(tmp2, JZ, "else
```

```
;("sprintf(tmp3, JMP, "end
;(("sprintf(tmp4, LABEL("else
;(("sprintf(tmp5, LABEL("end

tmp5 = realloc(tmp5, strlen(tmp_else) + strlen(tmp4) + strlen(tmp3) + strlen(tmp2) +
;(strlen(tmp1) + strlen(tmp_exp) + strlen(tmp5) + 1
;((strcat(tmp5, strdup(tmp_else
;(strcat(tmp5, tmp4
;(strcat(tmp5, tmp3
;((strcat(tmp5, strdup(tmp_if
;(strcat(tmp5, tmp2
;(strcat(tmp5, tmp1
;((strcat(tmp5, strdup(tmp_exp
{

;(free(tmp1
;(free(tmp2
;(free(tmp3
;(free(tmp4
;(free(tmp_exp
;(free(tmp_if
;(free(tmp_else


;(reg_free(regs, reg_exp
;return tmp5
{

char *trans_iteration_stmt(ast_node_T *ast, stack_T *astack, stack_T *code_stack,
} (register_pool_T **regs
;()char *tmp1 = alloc_instruction_mem
;()char *tmp2 = alloc_instruction_mem
;()char *tmp3 = alloc_instruction_mem
;()char *tmp4 = alloc_instruction_mem
;()char *tmp5 = alloc_instruction_mem

;(char *tmp_while = stack_pop(code_stack
;(char *tmp_exp = stack_pop(code_stack

;(register_T *reg_exp = stack_pop(astack

;(("sprintf(tmp1, LABEL("loop
;("sprintf(tmp2, CMP, reg_exp->name, "0
;("sprintf(tmp3, JZ, "end_loop
```

```
;("sprintf(tmp4, JMP, "loop
;(("sprintf(tmp5, LABEL("end_loop

;(printf("%s\n", tmp_while //

tmp5 = realloc(tmp5, strlen(tmp_while) + strlen(tmp4) + strlen(tmp3) + strlen(tmp2) +
;(strlen(tmp1) + strlen(tmp_exp) + strlen(tmp5) + 1
;(strcat(tmp5, tmp_exp
;(strcat(tmp5, tmp4
;(strcat(tmp5, tmp3
;(strcat(tmp5, tmp2
;(strcat(tmp5, tmp1
;(strcat(tmp5, tmp_while

;(free(tmp1
;(free(tmp2
;(free(tmp3
;(free(tmp4

;(reg_free(regs, reg_exp

;return tmp5
{
"include "../include/code_gen/translation_rule.h#
"include "../utils/err/err.h#

<include <stdlib.h#

translation_rule_T *init_translation_rule(symbol_T *symbol, char *(*translation)(ast_node_T *ast,
} ((stack_T *astack, stack_T *code_stack, register_pool_T **regs
;((translation_rule_T *rule = malloc(sizeof(translation_rule_T
(if(!rule
;(thrw(ALLOC_ERR

;rule->symbol = symbol
;rule->translation = translation
;return rule
"include "../include/semantic_analizer/AST.h#{
"include "../utils/err/err.h#
<include <stdio.h#
<include <stdlib.h#
<include <string.h#

**/
```

```
* @brief Initialize a new AST node
*
* @param symbol The symbol that represents the node
* @param children Array of children nodes
* @param n_children Number of children nodes
* @return ast_node_T* The new AST node
*/
ast_node_T *init_ast_node(symbol_T *symbol, ast_node_T **children, size_t n_children) {
    ast_node_T *astn = malloc(sizeof(ast_node_T));
    if(!astn)
        throw(ALLOC_ERR);

    astn->symbol = symbol;
    astn->children = children;
    astn->n_children = n_children;
    astn->st_entry = NULL;

    return astn;
}

/**
* @brief Initialize a new AST leaf node
*
* @param symbol The symbol that represents the leaf
* @return ast_node_T* The new AST leaf node
*/
ast_node_T *init_ast_leaf(symbol_T *symbol) {
    ast_node_T *astn = malloc(sizeof(ast_node_T));
    if(!astn)
        throw(ALLOC_ERR);

    astn->symbol = symbol;
    astn->children = NULL;
    astn->n_children = 0;
    astn->st_entry = NULL;

    return astn;
}

/**
* @brief Add a node to another node
*
* @param ast The node to add the child to
```

```
 * @param child The node to be added
 */
void ast_add_to_node(ast_node_T *ast, ast_node_T *child) {
    ++ast->n_children;
    ast->children = realloc(ast->children, sizeof(ast_node_T *) * ast->n_children);
    ast->children[ast->n_children - 1] = child;
}

/**
 * @brief Traverse the AST and print it
 *
 * @param ast The root node of the AST
 * @param layer The level of the node (0 being the root)
 */
void traverse_ast(ast_node_T *ast, int layer) {
    if(ast == NULL)
        return;

    int i;
    char *val = ast->symbol->sym_type == TERMINAL ?
        ast->symbol->symbol->terminal->value :
        ast->symbol->symbol->non_terminal->value;

    printf("%c\n", val[0]);
    for(i = 0; i < layer; ++i) {
        printf("  ");
    }

    printf("%s\n", val);

    for(i = 0; i < ast->n_children; ++i) {
        traverse_ast(ast->children[i], layer + 1);
    }
}
#include "../include/semantic_analizer/sdt.h"
#include "../utils/err/err.h"
#include <stdlib.h>


static sdt_T *match_parser_sdt_rules(rule_T **rules, size_t n_rules, semantic_rule_T **srs,
    size_t n_sr) {
    int i, j, flag;
    semantic_rule_T **definitions = calloc(n_rules, sizeof(semantic_rule_T *));
    if(!definitions)
```

```
;(thrw(ALLOC_ERR

} (for(i = 0; i < n_rules; ++i
;((definitions[i] = calloc(1, sizeof(semantic_rule_T
;[definitions[i]->rule = rules[i
;flag = 0

} (for(j = 0; j < n_sr; ++j
} ((if(!flag && !rule_cmp(srs[j]->rule, definitions[i]->rule
;definitions[i]->definition = srs[j]->definition
;flag = 1
{
{
{

;(return init_sdt(definitions, n_rules
{

} (sdt_T *init_sdt(semantic_rule_T **definitions, size_t n_defenitions
;((sdt_T *sdt = malloc(sizeof(sdt_T

;sdt->definitions = definitions
;sdt->n_defenitions = n_defenitions

;return sdt
{

} (sdt_T *init_default_sdt(rule_T **rules, size_t n_rules
;(semantic_rule_T **srs = malloc(sizeof(semantic_rule_T *) * 6

;(return match_parser_sdt_rules(rules, n_rules, srs, 6
{
"include "../include/semantic_analizer/definitions.h#
"include "../include/parser/parse_tree.h#
"include "../utils/err/err.h#
"include "../utils/symbol_table/include/symbol_table_tree.h#
<include <stdio.h#
<include <stdlib.h#

} (static ast_node_T *binop_node(ast_node_T *left, ast_node_T *node, ast_node_T *right
;(ast_add_to_node(node, right
;(ast_add_to_node(node, left

;return node
```

```
{


{} (void definition_start_r(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_program_sl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
} (void definition_sl_s(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(ast_node_T *stmt_list = init_ast_leaf(tree->symbol
;((ast_add_to_node(stmt_list, stack_pop(astack
;(stack_push(astack, stmt_list
{
} (void definition_sl_sl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(ast_node_T *stmt_list = stack_pop(astack
;((ast_add_to_node(stmt_list, stack_pop(astack

;(stack_push(astack, stmt_list
{
{} (void definition_s_exp_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_s_compound_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_s_selection_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
} (void definition_s_iteration_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s

{
{} (void definition_s_labeled_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_s_decl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
} (void definition_decl(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(symbol_table_tree_node_T *sym_tbl = stack_pop(st_s
;(ast_node_T **children = malloc(sizeof(ast_node_T *) * 3
(if(!children
;(thrw(ALLOC_ERR
;ast_node_T *node

children[0] = stack_pop(astack); // type
children[1] = stack_pop(astack); // id
// exp stack_pop(astack);
children[2] = stack_pop(astack); // value
; // ;(stack_pop(astack

;(node = init_ast_node(tree->symbol, children, 3
)node->st_entry = init_symbol_table_entry
,children[1]->symbol->symbol->terminal->value
,children[0]->symbol->symbol->terminal->type
,NULL
;(GLOBAL
```

```
;(stack_push(astack, node
;(symbol_table_insert(sym_tbl->table, node->st_entry
;(stack_push(st_s, sym_tbl
{

{} (void definition_type_int(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_type_char(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_type_float(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{} (void definition_type_void(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s

} (void definition_exp_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(ast_node_T *exp = stack_pop(astack
;(stack_pop(astack

;(stack_push(astack, exp
{
{} (void definition_cnstnt_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s

void definition_assignment_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
{} (stack_T *st_s
} (void definition_assignment_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_logical_or_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
{} (*st_s
} (void definition_logical_or_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_logical_and_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
{} (stack_T *st_s
} (void definition_logical_and_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_inclusive_or_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
{} (stack_T *st_s
```

} (void definition_inclusive_or_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
);
{
void definition_exclusive_or_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
{} (stack_T *st_s
} (void definition_exclusive_or_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
);
{
(void definition_and_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{}
} (void definition_and_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
);
{
void definition_equality_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
{} (*st_s
} (void definition_equality_equal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
);
{
void definition_equality_notequal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
);
{
void definition_relational_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
{} (*st_s
} (void definition_relational_less_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
);

```
{
void definition_relational_less_equal_exp(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_relational_greater_exp(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_relational_greater_equal_exp(stack_T *astack, parse_tree_node_T *tree,
} (stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
(void definition_shift_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{}
} (void definition_shift_lshift_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
} (void definition_shift_rshift_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_additive_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
{} (*st_s
} (void definition_additive_plus_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
```

```
} (void definition_additive_minus_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_multiplicative_exp_precedence(stack_T *astack, parse_tree_node_T *tree,
{} (stack_T *st_s
void definition_multiplicative_multiply_exp(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_multiplicative_divide_exp(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_multiplicative_mod_exp(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
)stack_push
,astack
((binop_node(stack_pop(astack), stack_pop(astack), stack_pop(astack
;(
{
void definition_primary_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T
} (*st_s
;ast_node_T *exp

) // ;(stack_pop(astack
;(exp = stack_pop(astack
( // ;(stack_pop(astack

;(stack_push(astack, exp
{
{} (void definition_primary_exp_id(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
(void definition_primary_exp_constant(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{}
{} (void definition_primary_exp_str(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
```

```
(void definition_exp_exp_precedence(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
{}
{} (void definition_exp_exp(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s

{} (void definition_compount_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s

} (void definition_selection_stmt_if(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(ast_node_T *sec = init_ast_leaf(tree->symbol

// if                    stack_pop(astack);
) //                     ;(stack_pop(astack
// exp   ast_add_to_node(sec, stack_pop(astack));
( //                     ;(stack_pop(astack
} //                     ;(stack_pop(astack
// stmts ast_add_to_node(sec, stack_pop(astack));
{ //                     ;(stack_pop(astack

;(stack_push(astack, sec
{

} (void definition_selection_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(ast_node_T *sec = init_ast_leaf(tree->symbol

// if                    stack_pop(astack);
) //                     ;(stack_pop(astack
// exp   ast_add_to_node(sec, stack_pop(astack));
( //                     ;(stack_pop(astack
} //                     ;(stack_pop(astack
// stmts ast_add_to_node(sec, stack_pop(astack));
{ //                     ;(stack_pop(astack
// else                  stack_pop(astack);
} //                     ;(stack_pop(astack
// stmts ast_add_to_node(sec, stack_pop(astack));
{ //                     ;(stack_pop(astack

;(stack_push(astack, sec
{

} (void definition_iteration_stmt(stack_T *astack, parse_tree_node_T *tree, stack_T *st_s
;(ast_node_T *it = init_ast_leaf(tree->symbol

// while                 stack_pop(astack);
) //                     ;(stack_pop(astack
ast_add_to_node(it, stack_pop(astack)); // exp
```

```
;(stack_pop(astack ( //
;(stack_pop(astack } //
ast_add_to_node(it, stack_pop(astack)); // stmts
;(stack_pop(astack { //

;(stack_push(astack, it
{
#include "../include/semantic_analizer/semantic_analyzer.h"
#include "../utils/DS/include/stack.h"
#include "../utils/symbol_table/include/symbol_table_tree.h"
#include <stdio.h>

static void build_ast_rec(parse_tree_node_T *tree, sdt_T *sdt, stack_T *ast_s,
} (symbol_table_tree_node_T *st_node, stack_T *st_s
(if(tree == NULL
;return

} (if(tree->symbol->sym_type == TERMINAL
;((stack_push(ast_s, init_ast_leaf(tree->symbol

TODO: think of a better way to add scopes //
(if(tree->symbol->symbol->terminal->type == TOK_RBRACE
;((()stack_push(st_s, init_symbol_table_tree_leaf(init_symbol_table_default

} (else if (tree->symbol->symbol->terminal->type == TOK_LBRACE
;(st_node = stack_pop(st_s
;(symbol_table_tree_node_add(stack_peek(st_s), st_node
{

;return
{

;int i

(for(i = 0; i < tree->n_children; ++i
;(build_ast_rec(tree->children[i], sdt, ast_s, st_node, st_s

(if(sdt->definitions[tree->rule_index]->definition
;(sdt->definitions[tree->rule_index]->definition(ast_s, tree, st_s
{


} (ast_node_T *build_ast(parse_tree_T *tree, quest_T *q
;()stack_T *ast_s = stack_init
```

```
;()stack_T *st_s = stack_init
;((()stack_push(st_s, init_symbol_table_tree_leaf(init_symbol_table_default
;(build_ast_rec(tree->root, q->sdt, ast_s, NULL, st_s

q->code_gen->sym_tbl = init_symbol_table_tree(((symbol_table_tree_node_T *)
;(((stack_peek(st_s

;(return stack_pop(ast_s
{

"include "../include/semantic_analizer/semantic_rule.h#

<include <stdlib.h#


semantic_rule_T *init_sementic_rule(rule_T *rule, void (*definition)(stack_T *astack,
} ((parse_tree_node_T *tree, stack_T *st_s
;((semantic_rule_T *sr = malloc(sizeof(semantic_rule_T

;sr->rule = rule
;sr->definition = definition

;return sr
"include "include/quest.h#{
"include "include/lang.h#
<include <stdio.h#

} ([]int main(int argc, char* argv
} (if(argc < 2
;("printf("Please specify input file. \n
;return 1
{

;([compile_file(argv[1

;return 0
"include "include/quest.h#{
"include "include/lang.h#
"include "include/lexer/lexer.h#
"include "include/io.h#
"include "include/parser/parse_tree.h#
"include "include/parser/parser.h#
"include "include/semantic_analizer/semantic_analyzer.h#
"include "utils/DS/include/queue.h#
```

```
#include "utils/lexer_DFA/include/lexer_DFA.h"

void compile(quest_T *q) {
    lexer_T* lex = q->lexer;
    parser_T *prs = q->parser;

    token_T* tk = 0;
    queue_T *queue = queue_init();

    do {
        tk = lexer_next_token(lex);

        if(tk->type == TOK_UNKNOWN) {
            printf("token is unknown: TOKEN(%s) (%d)\n", tk->value, tk->type); // exit(EXIT_FAILURE);
        } else {
            queue_enqueue(queue, tk);
        }
    } while(tk->type != TOK_eof);

    parse_tree_T *tree = parse(prs, queue);
    parse_tree_traverse_postorder(tree->root, 0); // 

    ast_node_T *ast = build_ast(tree, q);
    traverse_ast(ast, 0);

    write_file(q->destfile, generate_code(ast, q->code_gen));
}

void compile_file(const char *filename) {
    quest_T *q = init_quest(filename);
    compile(q);

    init_default_dfa(); // 
}
```

נספחים

- קישור לפרויקט בגיטהאב: https://github.com/EthanChartoff/Quest
- תוכלו למצוא את כל הנספחים ועוד בתוך הפרויקט בתיקיית resources!