Compiler Construction Design Document

Tyler Ray

September 2023

Contents

1 Symbols

Types

int float char void
: Basic data types.

Type Modifiers

long, unsigned, signed, short
: Type modifiers.

Keywords

if else while for return break continue
: Reserved keywords.

Whitespace

Whitespace characters are not tokenized.

Identifiers

Identifiers should match the regular expression: `r'^[a-zA-Z_][\w_]*$'`

Numbers

Numbers should match one of the following regular expressions:

- `r'^[\d]+$'`
- `r'^[\d]*\.[\d]+$'`
- `r'^0[xX][\dA-Fa-f]+$'` (Hexadecimal)
- `No negative Numbers`

Include

Include statements should match the regular expression: `r'^[A-Za-z_]*[.][h]$'`

Multiline Strings

Multiline strings are allowed, but they should not allow for variable printing.

2 Grammar

Program → DeclList

DeclList → Decl | Decl DeclList

Decl → Type id (Args) {Local_Decls stmtList} Type id = endofDecl;

Args → Arg | Arg , Args

Arg → Type id | ϵ

Local_Decls → Local_Decl | Local_Decl Local_Decls

Local_Decl → Type id; | Type id = EndOfDecl; | ϵ

StmtList → Stmt | Stmt StmtList

Stmt → ReturnStmt | AssignStmt | WhileStmt | IfStmt | ϵ

ReturnStmt → return num; | return id; | return; | return expr; | return character | return string

AssignStmt → id = EndOfDecl;

WhileStmt → while (Conditional_Expr) {StmtList}

IfStmt → if (Conditional_Expr) {StmtList} | if (Conditional_Expr) {StmtList} else {StmtList}

Conditional_Expr → expr Relop expr

Relop →== | != | > | >= | < | <=

EndOfDecl → Expr | string | character

Expr → Term + Expr | Term - Expr | Term

Term → Factor * Term | Factor / Term | Factor

Factor → num | (Expr) | id

Type → int | void | float | char

3 ParseTree

My Parse Tree is a AST Tree that shows 4 overall things within a function declaration:

- Id - The name of the function
- Type - What type is the function
- Args - What arguments the function has
- Local Declarations - What declarations it creates at the beginning of a declaration
- Statement List - All of the statements after our local declarations

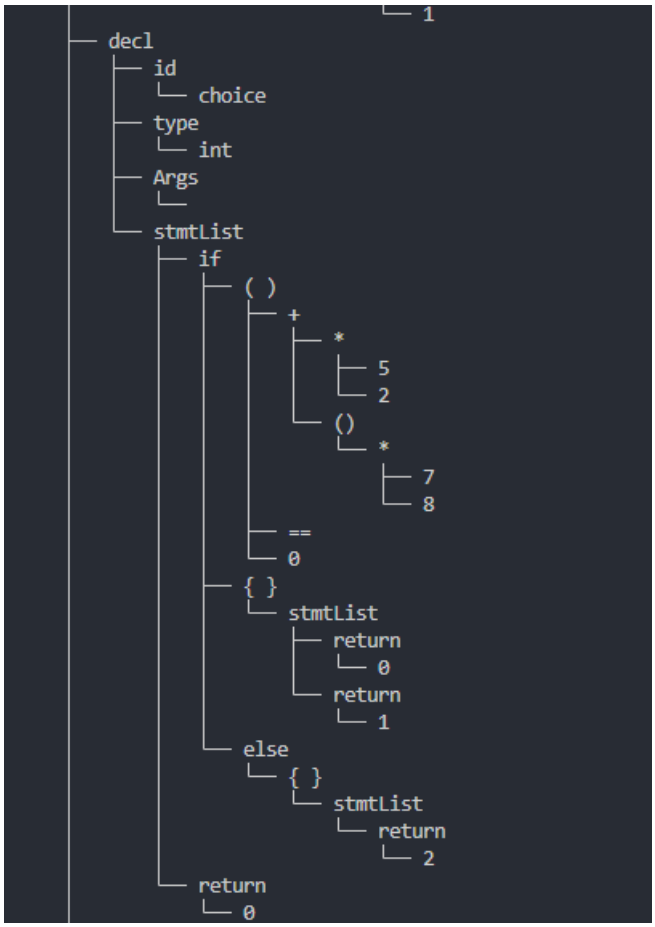

Figure 1: Parse Tree Example

In figure 1 is an example Parse Tree output within our program. At the top is our Id, type, and arguments. Within this example there are no local declarations so it doesn't show up on our tree. I decided to include the 'Args' just to show that there are none regardless. Within the first statement we see an if statement. The nodes after are the parenthesis brackets and in this case, an else statement. The () contain an expression. Our expression tree shows the operation, followed by the two numbers that it will perform that operation on. With multiple operations it will show the operation you must make before you make that one. Our parser will follow the order of operations

4 Usage

```
Compiler.py [-h] [-t] [-p] [-s] File
```

4.1 Positional Arguments

File
A valid C input file.

4.2 Options

-h, –help Show this help message and exit.

-t Outputs a tokenized version of the input file.

-p Outputs a parse tree of the input file.

-s Outputs a symbol table of the input table.

5 Caveats

- Not all valid tokens will be allow within our parser, I left them in for reference to the tokenizer. The parser grammar will be the best source on what tokens are allowed within our compiler.