The Lucas Parser

Group-3

1. JATIN TARACHANDANI
2. PRASHANTH SRIRAM S
3. S GOUTHAM SAI
4. SHANTANU PANDEY
5. ARAVINDA KUMAR REDDY T
6. ANIRUDH SRINIVASAN
7. T SRIVATSAN
8. JULAKUNTLA MADHURI

GOAL OF THE PROJECT

- Goal is to construct compiler to support Calculus related stuff.

- The following are the usp's of our Language:
  - ➢ Supports Calculus calculations
  - ➢ Supports Multi-returning functions
  - ➢ Support arbitrary-precision integers

Design Overview of the Implementation

- From the language specification, Parse rules were formulated
  [If constructs, Loop constructs, Function constructs, etc.]
- Antlr was setup as per the documentation.
- The formulated Parse rules were implemented in Antlrv4.
- Makefile was made to generate the Parser using Antlr from the grammar file and to run Parser on test cases.

ANTLR V4

- ANTLR - ANother Tool for Language Recognition.
- Antlr can be used for both lexing and parsing.
- We need to provide a target language in which the Parser should be generated.
  [For example: C++, C#, Java, Go, Swift, PHP]
- We have used Antlr to generate the Parser in Java.

ANTLR

- ANTLR uses LL(k) parsing to analyse the grammar.
- Parsers, lexers, and tree-parsers are accepted grammar specifications.
- Commands: (‘LucasGrammar.g4’ be the grammar file)
  - ➢ antlr4 LucasGrammarr.g4
  - ➢ javac LucasGrammar*.java
  - ➢ grun LucasGrammar tokens -tokens < test.txt

CHALLENGES FACED

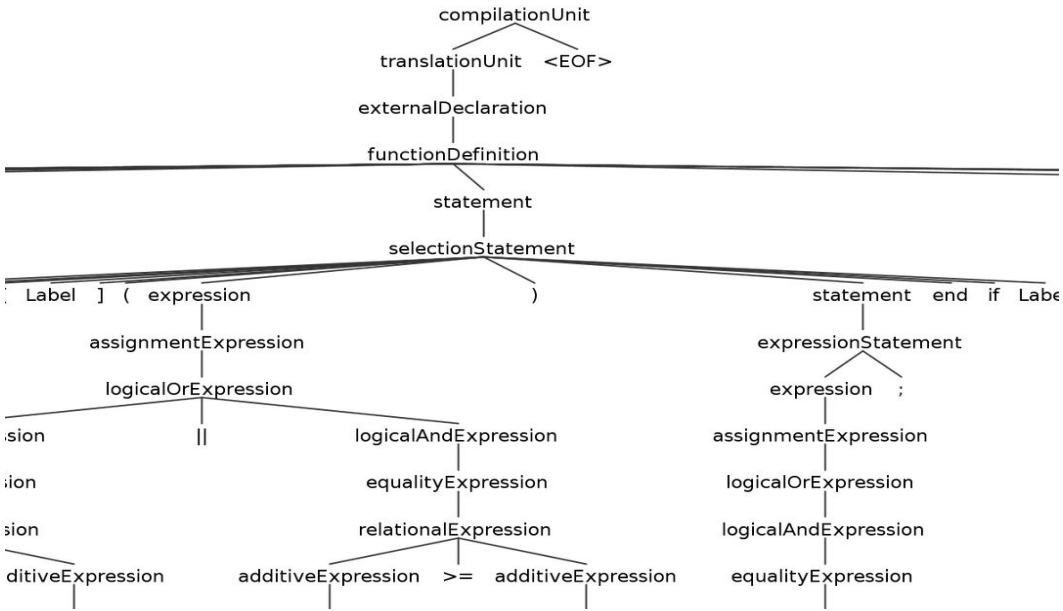
- We got an issue due to an indirect left recursion.
- This was caused because we used begin(instead of ‘{‘) and end(instead of ‘}’).
- Fixed this by removing compound statement, and replacing it with (statement|declaration)*.

Grammar File

```
grammar LucasGrammar;


primaryExpression
    :   Identifier
    |   Literal
    |   StringLiteral+
    |   '(' expression ')'
    ;

postfixExpression
    :
    (   primaryExpression
    |   '__extension__'? '(' typeName ')' '{' initializerList ','? '}'
    )
    ('[' expression ']'
    | '(' argumentExpressionList? ')'
    | ('.' | '->') Identifier
    | ('++' | '--')
    )*
    ;

argumentExpressionList
    :   assignmentExpression (',' assignmentExpression)*
    ;

unaryExpression
    :
    ('++' |  '--' |  'sizeof')*
    (postfixExpression
    |   unaryOperator castExpression
    |   'sizeof' '(' typeName ')'
    |   '&&' Identifier // GCC extension address of label
    )
    ;
```

A small snippet of our Grammar(related to parser)

Sample Output



This is how our output parse tree looks like