Gandi Gaman

ES21BTECH11014

Compilers II

Assignment 2

Introduction

This document contains a brief overview of the assignment. I have implemented a syntax analyser which takes a file as input (input file contains the source code of the c like language) and generates two output files. (one contains the sequence of tokens and the other , the parsed file which contains the type of statements in the source code)

Compilation Steps

The source files are in the P2 directory. To test the program on an input file, name the input file as “input.txt” and place it in the same directory as the source files. To compile the program , execute the following commands on the linux terminal .

There is also a bash file in the same directory. Execute this bash file using the command :

```
bash g.sh
```

The bash file “g.sh” contains the following commands

```
lex lex_prog.l

bison -d -t  yacc_prog.y
```

```
gcc lex.yy.c yacc_prog.tab.c
```

```
./a.out input.txt
```

Upon successful execution of the command the required output files will be generated in the same directory.

About the output files

One output file is seq_tokens.txt which contains the stream of tokens for the given input file. Each line in this file contains a recognized token along with its classification. These tokens are then passed to the parser.

Implementation

- Various regular expressions are defined in the lex program to recognize different types of tokens. If a token is recognized then it is printed to the output file and returned. If there is any lexical error then the error message is printed with the line number.
- I have defined the Grammar rules for the c like language using yacc/bison specifications. The token types used in the grammar rules are specified using the %token declaration. All of the tokens are terminal symbols of the grammar. I have defined the grammar rules to recognize functions, classes, different types of statements (declaration, expression, loops, return ….. and more) etc.
- If a syntax error occurs then the program stops processing the remaining code and an error message will be printed to the terminal and also to the output file.

Issues with the Implementation

- As given in the problem statement, if the number of arguments to function is zero, then there is no need to have “[number]” . Although this check comes under semantic analysis, I have added this check to the syntax analysis.
- One difficulty I have encountered is to handle the case that the function/method should have at least 1 return statement. To handle this problem I have declared a global variable count = 0 and incremented it when the return statement is encountered. At the end of the function/method , check if count is > 0 . If true then correct, assign count = 0, else an error message is printed.
- Another difficulty I encountered is designing Grammar for the predicate. Initially I implemented a grammar but there were many shift-reduce and reduce-reduce conflicts. I tried to remove all these conflicts by changing the grammar slightly but it turned out my final grammar for predicates doesn’t handle some cases. Ex : My grammar does not handle the case when there can be multiple parentheses enclosing the predicate.
  And also it doesn’t handle the case when there are multiple “neg” in the same predicate.

***********************************END************************************