南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMPILIER @Liu Yepang 2019

for SUSTech CSE

Project 1

Edited by

汪至圆

11610634

2019

SHENZHEN

1 Lab Demand

In this project, I'm required to implement a SPL parser, specifically, your parser will perform lexical analysis and syntax analysis on the SPL source code. The parser was written by C++/Flex/Bison, and it can output a syntax tree and find some easy grammatical errors for a SPL code.

2 Environment

The environment for my project is g++ 9.1, flex 2.6, bison 3.4. All the work of coding and test was finished in the Manjaro 18.1 which based on Linux 4.19.

3 Lexical Analyzer Generator

The lexical analyzer generator I used is GNU Flex, which is the open source version for the lex. In the lex, I define some token following the guide provided by the TA. And I also define some token by myself to satisfy the requirements of the grammar analyzer. The token I defined are LEXERR, which will produce the error lexeme and throw an type A error to tell the user there is an unknown lexeme. And the comments was also produced in the lex file, I use regular expressions to filter the comment lines and comment blocks. I also can find the nested comment block by the regular expression, which is not allowed in the SPL. For the token which I define by myself, the code of them is:

```
A [/]
B [*]
C [^*/]
comment "/*"{A}*({C}{A}*|{B}|{C})*"*/"
%%
\/\/.*                  {comment line}
"/*"{A}*({C}{A}*|{B}|{C})*{comment}(.|\n)*"*/"  {nested comment block}
{comment}               {comment block}
0x[0-9A-Za-z]+          {yylval=createNode(yytext); return LEXERR; }
('\\x[0-9a-zA-Z]*')     {yylval=createNode(yytext); return LEXERR; }
[0-9][a-zA-Z_0-9]*      {yylval=createNode(yytext); return LEXERR; }
.                       {yylval=createNode(yytext); return LEXERR;}
%%
```

4 Parser Generator

The parser generator I used is GNU Bison. In the bison, I define the syntax rule following the guide and add some syntax rule by myself to catch the error and output the error message. In the bison I define a struct called node, which is used to record the message of the syntax tree, it has a member variable with type string to store the node name and line number, it also have a member variable with type vector to store the the subnode of it. When process each syntax rule except which using to catch the error, I will get the line number of the code and the add the nodes return by the

subrules into the subnode list. If the bison enter a rule which use to catch the error, I will add a error message to a list which is used to storage the error. After analysis the code, I will check the size of the list which used to store the error, if it is empty, I will output the syntax tree Recursively, or I will output the error messages.

4.1 Struct node

```
struct Node{
    vector<Node> subNode;
    string show;
};

Node createNode(string s){
    Node node;
    node.show = s;
    return node;
}
```

4.2 Additional Rules

- Extdef:
  - Specifier ExtDecList error
  - Specifier error
- FunDec:
  - ID LP error
  - ID LP VarList error
  - LEXERR LP VarList RP
  - LEXERR LP RP
- Def:
  - Specifier DecList error
- Exp:
  - ID LP Args error
  - ID LP error
  - LEXERR LP Args error
  - Exp LEXERR Exp
  - LEXERR LP Args RP
  - LEXERR LP RP
  - Exp DOT LEXERR
  - LEXERR

5 Run

My project was compiled by the make. If you want to compile the project, you just need run 'make splc'. And then it will output a executable file splc.out. The result of running splc.out will print in the stdout of console, if you want to input a spl code file and get an out file, you can use 'splc.py' in the bin/, input the path of the code file which will be inputted and the result will be outputted to the file with the name replace the '.spl' in the path to the '.out'.

6 bonus