编译原理源码

实验一

```
%{
/*********************************************************************************
mylexer.l
ParserWizard generated Lex file.

Date: 2020年6月7日
*********************************************************************************/

#include "myparser.h"
%}

//////////////////////////////////////////////////////////////////////
// declarations section

// lexical analyser name
%name mylexer

// class definition
{
    // place any extra class members here
}

// constructor
```

```
{
    // place any extra initialisation code here
}

// destructor
{
    // place any extra cleanup code here
}

// place any declarations here

headfile  #include(" "*)[<"](([a-z]|[A-Z]|\.)+)[>"]
numbers ([0-9]+[\.|0-9]*[e|E]?[0-9]+)|0[x|X][0-9|A-F|a-f]+[\.|0-9|a-f|A-F]*
string \"(.)*\"
reserved auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long|register|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volatile|while
delimiter  ,|;|\(|\)|\{|\}|\[|\]|\'|\"|\<|\>
identifier [_|a-zA-Z]+[_|a-zA-Z]*[0-9]*
backspace \r\n
operator \>\>=|\<\<=|\+\+|--|==|\>=|\<=|!=|&&|\|\||\<\<|\>\>|\+=|-=|\*=|\/=|\%=|&=|\|=|\^=|\+|-|\*|\/|\%|\>|\<|!|&~|\^|=
%%
////////////////////////////////////////////////////////////////////////
// rules section
```

```
{headfile} {printf("%s是第%d行的 头文件",yytext,yylineno);}
{numbers} {printf("%s是第%d行的 数字",yytext,yylineno);}
{string} {printf("%s是第%d行的 字符",yytext,yylineno);}
{reserved} {printf("%s是第%d行的 保留字",yytext,yylineno);}
{delimiter} {printf("%s是第%d行的 分隔符",yytext,yylineno);}
{identifier} {printf("%s是第%d行的 标识符",yytext,yylineno);}
{backspace} {printf("%s是第%d行的 换行符",yytext,yylineno);}
{operator} {printf("%s是第%d行的 操作符",yytext,yyineno);}
%%

////////////////////////////////////////////////////////////////////////
// programs section
int main()
{
//create a lexer, and call the lex function.
//it will read from stdin and parser the tokens.
 YYLEXERNAME lexer;
 if(lexer.yycreate()){
     lexer.yylex();
 }
}
```

实验二

```
#include <iostream>
#include <cstring>
#include<string>
#include<vector>

#define charNum 26
using namespace std;
vector<string> keywords = {"auto", "break", "case", "char", "const", "continue", "default", "do", "double",
                           "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return",
                           "short",
                           "signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void",
                           "volatile", "while"};

int identifyString(char tstr);

int identifyString(char tstr) {
    if (tstr == ',' || tstr == ';' || tstr == '(' || tstr == ')' ||
        tstr == '{' || tstr == '}' || tstr == '[' || tstr == ']' ||
        tstr == '\"' || tstr == '\'' || tstr == '<' || tstr == '>') {
```

```
20          cout << endl << "//" << tstr << " is
    delimite" << endl;
21          return 1;
22      }
23      return 0;
24  }
25
26  int identifyHead(string tstr) {
27      if (tstr[0] == '#') {
28          //if (tstr.substr(1, 8) == "include<")
29          {
30              int flag = 0;
31              for (int i = 0; i < tstr.size();
    i++) {
32                  char c = tstr[i];
33                  if (c == '<') {
34                      flag = i;
35                      break;
36                  }
37              }
38              cout << endl << "//" <<
    tstr.substr(flag + 1, tstr.size() - flag - 2)
    << " is headfile" << endl;
39              return 1;
40          }
41      }
42      return 0;
43  }
44
45  int identifyKey(string tstr) {
46
47      for (auto it = keywords.begin(); it <
    keywords.end(); it++) {
48          if (tstr == *it) {
```

```
            cout << endl << "//" << *it << " is keywords." << endl;
            return 1;
        }
        return 0;
    }
    return 0;
}

int identifyStr(string tstr) {
    int flag = 0;
    for (int i = 0; i < tstr.size(); i++) {
        char c = tstr[i];
        if (!isalpha(c) || c == '_') {
            if (identifyString(c)) {
                if (i > 0)
                    cout << endl << "//" << tstr.substr(flag, i - flag) << "  is inversed word." << endl;
                flag = i + 1;
            }
        }
    }
    if (flag != tstr.size())cout << endl << "//" << tstr.substr(flag, tstr.size()) << " is inverted word." << endl;
    return 0;
}


int main() {
    string tstr;
    while (cin >> tstr) {
        char c = tstr[0];
```

```
78          if (identifyString(c))continue;
79          if (identifyHead(tstr))continue;
80          if (identifyKey(tstr))continue;
81          else {
82              identifyStr(tstr);
83
84          }
85
86      }
87
88      return 0;
89  }
```

实验三

```
1  %{
2  /*********************************************************
   ******************************
3  mylexer.l
4  ParserWizard generated Lex file.
5
6  Date: 2020年6月7日
7  *********************************************************
   ******************************/
8
9  #include "myparser.h"
10 #include<iostream>
11 using namespace std;
12 %}
13
```

```
14 ////////////////////////////////////////////////////////////////////////////
15 // declarations section
16
17 // lexical analyser name
18 %name mylexer
19
20 // class definition
21 {
22     // place any extra class members here
23 }
24
25 // constructor
26 {
27     // place any extra initialisation code here
28 }
29
30 // destructor
31 {
32     // place any extra cleanup code here
33 }
34
35 // place any declarations here
36 delim [ \t]
37 ws    {delim}+
38 letter [a-zA-Z]
39 digit [0-9]
40 id    {letter}({letter}|{digit})*
41 /* can support 12.34 */
42 number {digit}+(\.{digit}+)?
43 %%
44
```

```
45 ////////////////////////////////////////////////////////////////////////
46 // rules section
47
48 %{
49     // extract yylval for use later on in actions
50     YYSTYPE YYFAR& yylval = *(YYSTYPE YYFAR*)yyparserptr->yylvalptr;
51 %}
52
53 // place your Lex rules here
54 {ws} {/*do nothing*/}
55
56 "+" {return PLUS;}
57 "-" {return MINUS;}
58 "*" {return MULTI;}
59 "(" {return LP;}
60 ")" {return RP;}
61 "=" {return EQUAL;}
62 "\n" {return EOL;}
63 "/" {return DIVIDE;}
64 {id} {return ID;}
65 {number}    { yylval = atof(yytext);return NUMBER;}
66 %%
67
68 ////////////////////////////////////////////////////////////////////////
69 // programs section
70
```