《编译原理与技术》
实验报告

姓　　名：李勐

班　　级：**07409**

学　　号：**071202**

班内序号：**1**

一、实验目的

通过手工编制一个C语言词法分析器，进一步理解词法分析的原理、过程以及在编译中的作用。

二、实验要求

设计并实现C语言的词法分析程序，要求如下：

1. 可以识别出C语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。
2. 可以识别并读取源程序的注释。
3. 可以统计源程序中的语句行数、单词个数和字符个数，其中标点和空格不计算为单词，并输出统计结果。
4. 检查源程序中存在的错误后，并可以报告错误所在的行列位置。
5. 发现源程序的错误后，进行适当的恢复使词法分析器可以继续进行，通过一次词法分析处理，可以检查和报告源程序中存在的所有错误。

三、实验环境

Linux 2.6.31-ARCH
flex 2.5.35
gcc 4.4.1
zsh 4.3.10-10

四、源代码

```
%{
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>

using namespace std;

void install_num(char * token);
void install_str(char * token);
void install_opt(char * token);
void install_id(char * token);
int is_existed(string token);

vector<string>  id_table, kw_table, opt_table, num_table, str_table;
int lineCnt = 1, wlineCnt = 0, punctCnt = 0;
%}

/* ============================================================= */
```

```
/*                         Regex Defination                      */
/* ============================================================= */
ws           [\t ]+
wl           ^[\t ]*\n
eol          \n
punct        [,;{}]
letter       [_A-Za-z]
digit        [0-9]
string       \".*[^\\]\"
char         '.*'
id           {letter}({letter}|{digit})*
num          -?({digit}+)|({digit}*(\.{digit}+)([Ee][+\-]?{digit}+)?)
opt          \(|\)|\.|\+[=+]?|-[-=]?|\*[=]?|\/[=]?|%[=]?|\[|\]|(![=]?)|(<[<=]?)|(>[>=]?)|
       (&[&=]?)|(\|[|=]?)|(^[=]?)|(sizeof)|(=[=]?)
marco        #.*{eol}
comment      (\/\*[^\\]*\*\/)|(\/\/[^{eol}]*)

%%
{wl}                {wlineCnt++;}
{eol}               {lineCnt++;}
{punct}             {punctCnt++;}
{comment}           {fprintf(yyout, "{comment} ");}
{string}|{char}     {install_str(yytext);}
{marco}             {fprintf(yyout, "{marco} ");}
{ws}                {}
{id}                {install_id(yytext);}
{num}               {install_num(yytext);}
{opt}               {install_opt(yytext);}

%%
/* ================================================================= */
/*                        Functions' Definition                      */
/* ================================================================= */

/* keywords table */
char keywords[100][10] = {
    "int", "char", "long", "float", "double", "auto", "register",
    "struct", "union", "const", "case", "switch", "default",
    "if", "then", "else", "for", "while", "break", "continue",
    "do", "signed", "unsigned", "short", "goto", "return", "void"
};


bool is_keyword(char * token)
{
    for (int i = 0; i < 100; ++i)
        if (strcmp(token, keywords[i]) == 0)
            return true;
    return false;
}

/* judge a id whether it is existed in id table */
int is_existed(string token)
{
    for (int i = 0; i < id_table.size(); ++i)
        if (id_table[i] == token)
            return i;
    return -1;
```

```
}

void install_num(char * token)
{
    num_table.push_back(token);
    fprintf(yyout, "{num%d} ", num_table.size());
}

void install_str(char * token)
{
    str_table.push_back(token);
    fprintf(yyout, "{str%d} ", str_table.size());
}

void install_opt(char * token)
{
    opt_table.push_back(token);
    fprintf(yyout, "%s{opt} ", token);
}

void install_id(char * token)
{
    if (is_keyword(token))
    {
        kw_table.push_back(token);
        fprintf(yyout, "%s{kw} ", token);
    }
    else
    {
        int idx;
        if ((idx = is_existed(token)) != -1)
            fprintf(yyout, "{id%d} ", idx);
        else
        {
            id_table.push_back(token);
            fprintf(yyout, "{id%d} ", id_table.size());
        }
    }
}

int yywrap()
{
      return 1;
}

int main(int argc, char * argv[])
{
    yyin = fopen(argv[1], "r");


    /* Lexical Analysis */
    yylex();

    /* print statistics */
    fprintf(yyout, "\n\nTotally %d lines, %d white lines.",
            lineCnt + wlineCnt, wlineCnt);
    fprintf(yyout, " %d punctations, %d operators, %d keywords, %d identifiers.\n",
            punctCnt, opt_table.size(), kw_table.size(), id_table.size());
```

```
    /* print identifiers */
    fprintf(yyout, "\nIdentifires:\n");
    for (int i = 0; i < id_table.size(); ++i)
        fprintf(yyout, "%s ", id_table[i].c_str());
    fprintf(yyout, "\n");

    /* print keywords */
    fprintf(yyout, "Keywords:\n");
    for (int i = 0; i < kw_table.size(); ++i)
        fprintf(yyout, "%s ", kw_table[i].c_str());
    fprintf(yyout, "\n");

    /* print operators*/
    fprintf(yyout, "Operators:\n");
    for (int i = 0; i < opt_table.size(); ++i)
        fprintf(yyout, "%s ", opt_table[i].c_str());
    fprintf(yyout, "\n");

    /* print numbers */
    fprintf(yyout, "Numbers:\n");
    for (int i = 0; i < num_table.size(); ++i)
        fprintf(yyout, "%s ", num_table[i].c_str());
    fprintf(yyout, "\n");

    /* print strings */
    fprintf(yyout, "Strings:\n");
    for (int i = 0; i < str_table.size(); ++i)
        fprintf(yyout, "%s\n", str_table[i].c_str());
    fprintf(yyout, "\n");
$ lex lex.l

$ g++ lex.yy.c -o LAoC

$ ./LAoC reverse_token.c

    fclose(yyin);
}
```

五、实验完成情况

运行方法

打开终端，切换至 lex 规则文件 lex.l 目录，输入以下命令：

```
$ lex lex.l
$ g++ lex.yy.c -o LAoC
$ ./LAoC reverse_token.c
```

生成可执行文件 LAoC，然后执行：

```
$ ./LAoC reverse_token.c
```

输入文件内容

```
#include <stdio.h>
#include <string.h>
#define MAX “I am a marco, leave me alone”

char * a = "this is a \"string with quotes\"";
void reverse_token()
{
  char str[MAX] = {0};
  if (scanf("%[^#|\0]", str) != EOF) {
      getchar(); /* * this is a multi-line
                    comment*/
      reverse_token(); // this is also a comment
      printf("%s ", str);
  }
}

int main()
{
    reverse_token();
}
```

输出

```
{marco} {marco} {marco} char{kw} *{opt} {id1} ={opt} {str1} void{kw}
{id2} ({opt} ){opt} char{kw} {id3} [{opt} {id4} ]{opt} ={opt} {num1}
if{kw} ({opt} {id5} ({opt} {str2} {id2} ){opt} !={opt} {id6} ){opt}
{id7} ({opt} ){opt} {comment} {id1} ({opt} ){opt} {comment} {id8}
({opt} {str3} {id2} ){opt} int{kw} {id9} ({opt} ){opt} {id1} ({opt} )
{opt}

Totally 20 lines, 2 white lines. 16 punctations, 22 operators, 5
keywords, 9 identifiers.

Identifires:
a reverse_token str MAX scanf EOF getchar printf main
Keywords:
char void char if int
Operators:
* = ( ) [ ] = ( ( ) != ) ( ) ( ) ( ) ( ) ( )
Numbers:
0
Strings:
"this is a \"test\""
"%[^#|\0]"
"%s "
```

*lex*统计信息

```
flex version 2.5.35 usage statistics:
  scanner options: -lvI8 -Cem
  170/2000 NFA states
  72/1000 DFA states (300 words)
  10 rules
  Compressed tables always back-up
```

```
Beginning-of-line patterns used
1/40 start conditions
100 epsilon states, 67 double epsilon states
30/100 character classes needed 190/500 words of storage, 0 reused
684 state/nextstate pairs created
145/539 unique/duplicate transitions
79/1000 base-def entries created
158/2000 (peak 289) nxt-chk entries created
28/2500 (peak 238) template nxt-chk entries created
0 empty table entries
7 protos created
7 templates created, 23 uses
34/256 equivalence classes created
4/256 meta-equivalence classes created
0 (1 saved) hash collisions, 76 DFAs equal
0 sets of reallocations needed
764 total table entries needed
```

六、实验总结

通过本次实验，初步掌握了 lex 使用方法。通过 lex，可以方便的生成基于正则表达式的词法分析器。通过阅读 lex 生成的 C 代码，学习了如何用 C 语言实现 DFA，以及 lex 生成的词法分析器的大致工作流程；

七、参考资料

[1] Thomas Niemann. *A COMPACT GUIDE TO LEX & YACC.*
[2] John R. Levine, Tony Mason, Doug Brown. *Lex and Yacc, 2nd Edition.* O'Reilly.