XI'AN JIAOTONG UNIVERSITY

2022-2023 学年第二学期

《编译器设计专题实验》

实验报告 3

学　　院：	电信学部
班　　级：	[redacted]
学　　号：	[redacted]
姓　　名：	[redacted]

二〇二三年 四月

目录

《实验 3-词法分析(一),熟悉流程和 flex 工具》

一、实验内容（必做）

1. 统计输入文本的行数、单词数及字符数；

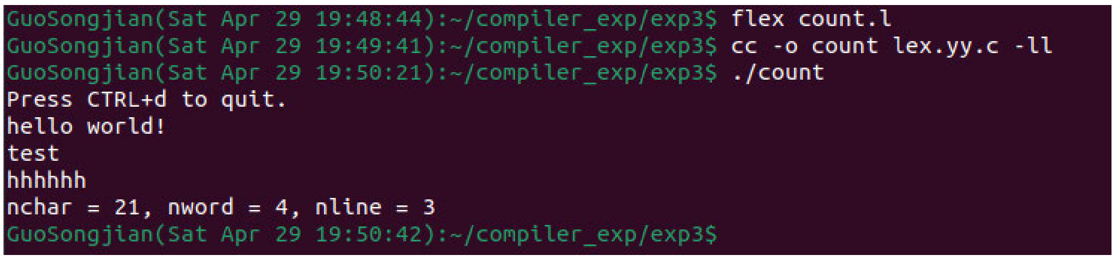
2. 设计 2 个 COOL 程序,包括正确可运行的和错误不可运行的，要求使用词法分析器识别每个不同的关键字并统计不同关键字的个数，以及定位错误 COOL 程序的出错位置。

二、实验内容（选做）

扩展完成 COOL 程序的词法分析，可分析关键字、标识符和注释三种词素，使用 makefile 自动运行。将上述两个 COOL 程序输入，可以通过不同的 make+参数实现不同的词法分析过程。

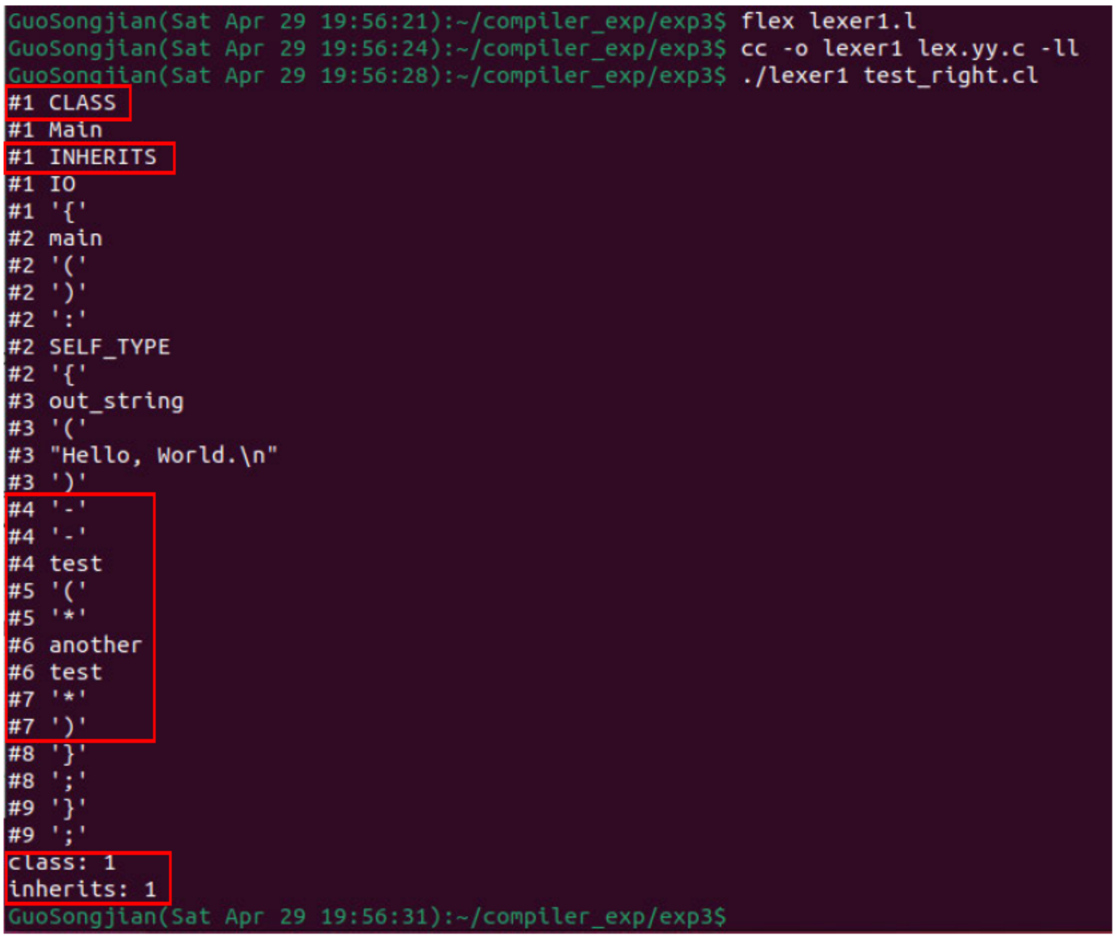
三、实验结果

（1）必做 1-统计文本信息



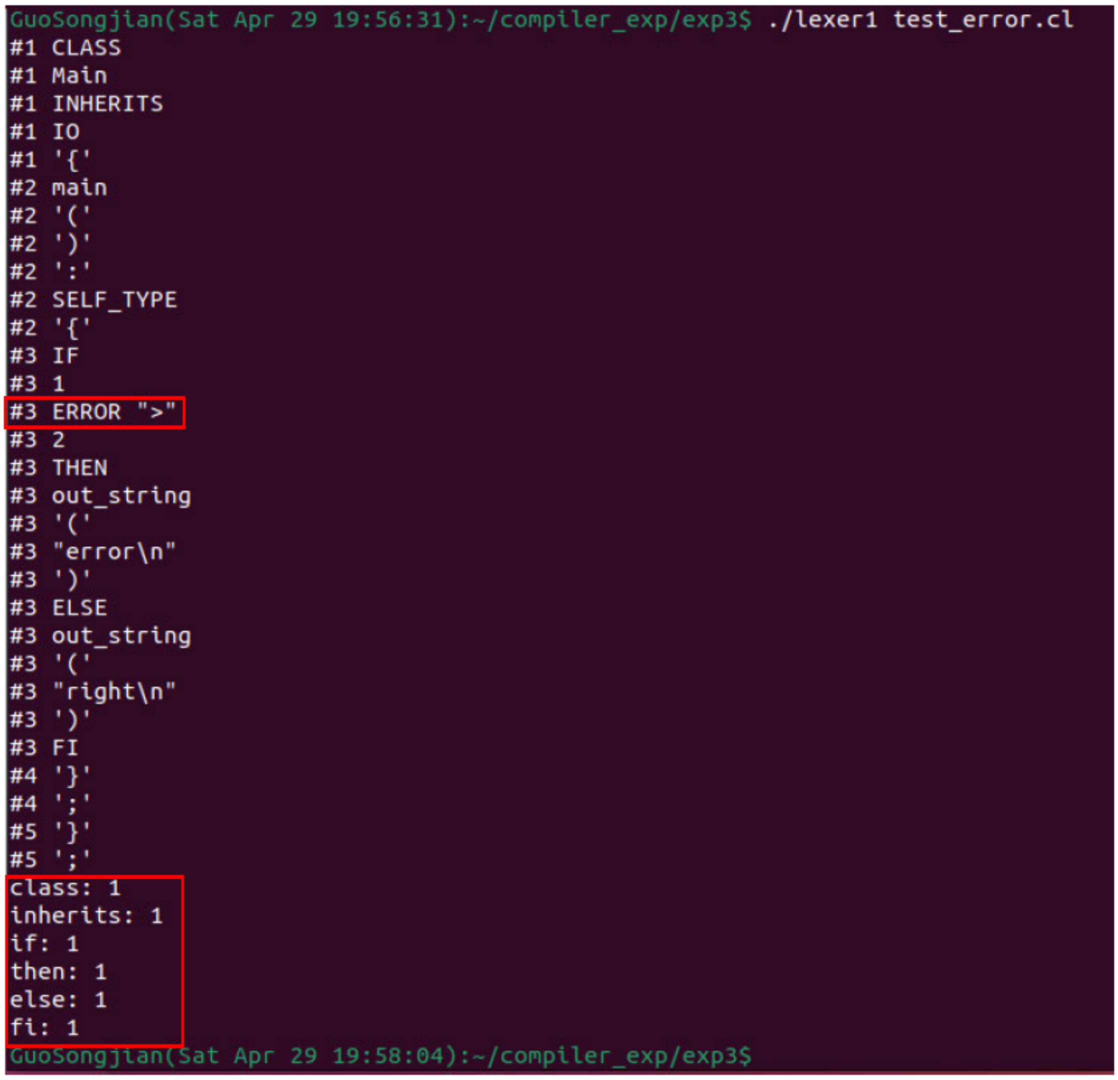
```
GuoSongjian(Sat Apr 29 19:48:44):~/compiler_exp/exp3$ flex count.l
GuoSongjian(Sat Apr 29 19:49:41):~/compiler_exp/exp3$ cc -o count lex.yy.c -ll
GuoSongjian(Sat Apr 29 19:50:21):~/compiler_exp/exp3$ ./count
Press CTRL+d to quit.
hello world!
test
hhhhhh
nchar = 21, nword = 4, nline = 3
GuoSongjian(Sat Apr 29 19:50:42):~/compiler_exp/exp3$
```

（2）必做 2-统计关键字个数并定位错误



```
GuoSongjian(Sat Apr 29 19:56:21):~/compiler_exp/exp3$ flex lexer1.l
GuoSongjian(Sat Apr 29 19:56:24):~/compiler_exp/exp3$ cc -o lexer1 lex.yy.c -ll
GuoSongjian(Sat Apr 29 19:56:28):~/compiler_exp/exp3$ ./lexer1 test_right.cl
#1 CLASS
#1 Main
#1 INHERITS
#1 IO
#1 '{'
#2 main
#2 '('
#2 ')'
#2 ':'
#2 SELF_TYPE
#2 '{'
#3 out_string
#3 '('
#3 "Hello, World.\n"
#3 ')'
#4 '-'
#4 '-'
#4 test
#5 '('
#5 '*'
#6 another
#6 test
#7 '*'
#7 ')'
#8 '}'
#8 ';'
#9 '}'
#9 ';'
class: 1
inherits: 1
GuoSongjian(Sat Apr 29 19:56:31):~/compiler_exp/exp3$
```

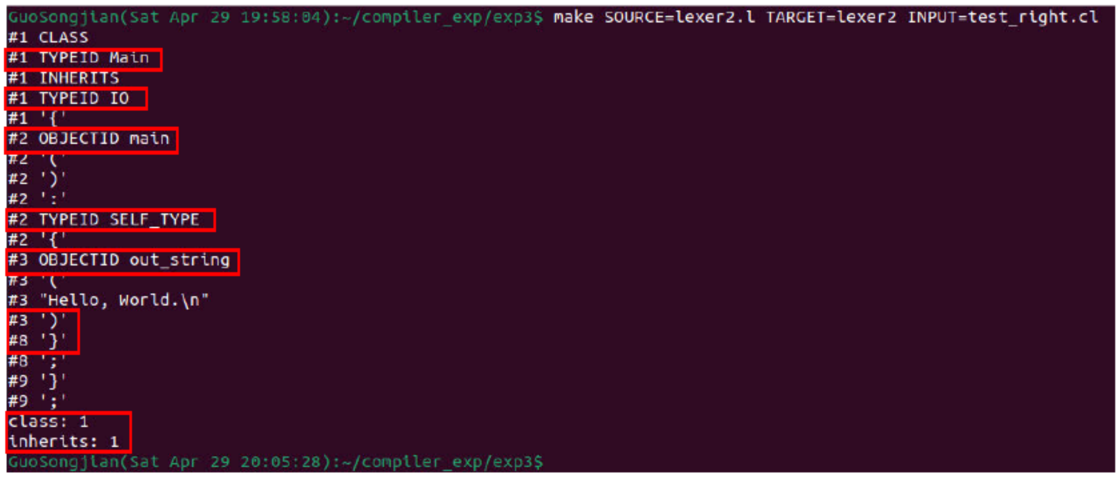
词法分析器正确识别出了关键字并将其转换为大写输出，但并未处理注释（将在选作部分解决该问题）。



```
GuoSongjian(Sat Apr 29 19:56:31):~/compiler_exp/exp3$ ./lexer1 test_error.cl
#1 CLASS
#1 Main
#1 INHERITS
#1 IO
#1 '{'
#2 main
#2 '('
#2 ')'
#2 ':'
#2 SELF_TYPE
#2 '{'
#3 IF
#3 1
#3 ERROR ">"
#3 2
#3 THEN
#3 out_string
#3 '('
#3 "error\n"
#3 ')'
#3 ELSE
#3 out_string
#3 '('
#3 "right\n"
#3 ')'
#3 FI
#4 '}'
#4 ';'
#5 '}'
#5 ';'
class: 1
inherits: 1
if: 1
then: 1
else: 1
fi: 1
GuoSongjian(Sat Apr 29 19:58:04):~/compiler_exp/exp3$
```

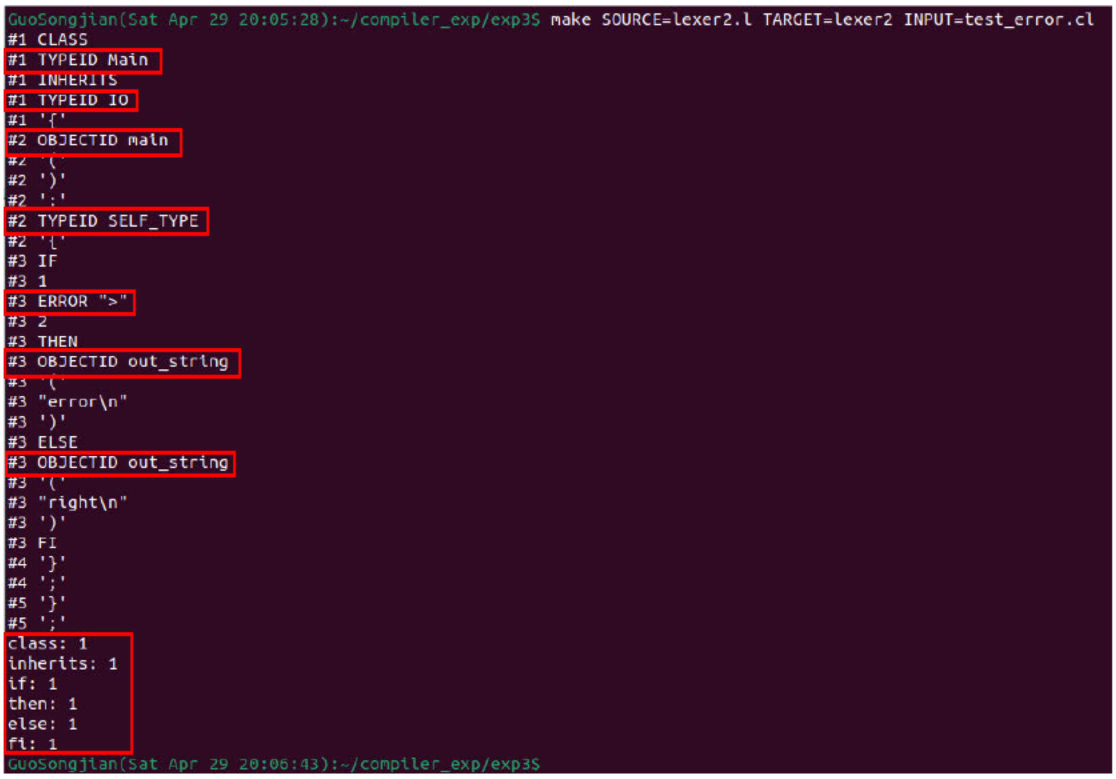
对于存在词法错误的 COOL 程序，词法分析器正确识别出了错误并定位了错误的行数。

(3) 选做-分析关键字、标识符、注释



```
GuoSongjian(Sat Apr 29 19:58:04):~/compiler_exp/exp3$ make SOURCE=lexer2.l TARGET=lexer2 INPUT=test_right.cl
#1 CLASS
#1 TYPEID Main
#1 INHERITS
#1 TYPEID IO
#1 '{'
#2 OBJECTID main
#2 '('
#2 ')'
#2 ':'
#2 TYPEID SELF_TYPE
#2 '{'
#3 OBJECTID out_string
#3 '('
#3 "Hello, World.\n"
#3 ')'
#8 '}'
#8 ';'
#9 '}'
#9 ';'
class: 1
inherits: 1
GuoSongjian(Sat Apr 29 20:05:28):~/compiler_exp/exp3$
```

从上图可以看出，词法分析器正确识别出了两种标识符：TYPEID 和 OBJECTID，同时与必做 2 相比较，词法分析器正确识别出了注释并不再将其输出（COOL 源程序中 4-7 行为注释部分）。



```
GuoSongjian(Sat Apr 29 20:05:28):~/compiler_exp/exp3$ make SOURCE=lexer2.l TARGET=lexer2 INPUT=test_error.cl
#1 CLASS
#1 TYPEID Main
#1 INHERITS
#1 TYPEID IO
#1 '{'
#2 OBJECTID main
#2 '('
#2 ')'
#2 ':'
#2 TYPEID SELF_TYPE
#2 '{'
#3 IF
#3 1
#3 ERROR ">"
#3 2
#3 THEN
#3 OBJECTID out_string
#3 '('
#3 "error\n"
#3 ')'
#3 ELSE
#3 OBJECTID out_string
#3 '('
#3 "right\n"
#3 ')'
#3 FI
#4 '}'
#4 ';'
#5 '}'
#5 ';'
class: 1
inherits: 1
if: 1
then: 1
else: 1
fi: 1
GuoSongjian(Sat Apr 29 20:06:43):~/compiler_exp/exp3$
```

上图是对存在词法错误的 COOL 源程序进行词法分析。由于 COOL 语言中未定义“>”，因此词法分析器将报错并给出错误行数（#3），其余分析与先前一样。

四、源代码

(1) 统计文本信息

```
%{

#include <stdio.h>
int nchar = 0, nword = 0, nline = 0;

%}

%%
```

```
[ \t]
\n          {++nline;}
[^ \t\n]+   {++nword; nchar += yyleng;}

%%

int main(){
    printf("Press CTRL+d to quit.\n");
    yylex();
    printf("nchar = %d, nword = %d, nline = %d\n", nchar, nword,
nline);
    return 0;
}
```

(2) 词法分析器测试样例

```
class Main inherits IO {
    main(): SELF_TYPE {
        out_string("Hello, World.\n")
        --test
        (*
        another test
        *)
    };
};
```

```
class Main inherits IO {
    main(): SELF_TYPE {
        if 1 > 2 then out_string("error\n") else
out_string("right\n") fi
    };
};
```

(3) 统计关键字个数并定位错误

```
%{

#include <stdio.h>
#include <string.h>
#define KEYWORD_NUM 17
int line = 0;
enum {CLASS, INHERITS, IF, THEN, ELSE, FI, WHILE, LOOP, POOL, CASE,
ESAC, OF, LET, IN, NEW, ISVOID, NOT};
char * key[] =
{"class","inherits","if","then","else","fi","while","loop","pool","
case","esac","of","let","in","new","isvoid","not"};
int keyword[KEYWORD_NUM] = {0};
```

```
void keyword_handle(char *str);
void keyword_show(void);

%}

KEYWORD
"class"|"inherits"|"if"|"then"|"else"|"fi"|"while"|"loop"|"pool"|"c
ase"|"esac"|"of"|"let"|"in"|"new"|"isvoid"|"not"
ID          [a-zA-Z]+[_a-zA-Z0-9]*
STRING      \"[^"\n]*\"
INTEGER     [0-9]+
WHITE       [ \t]+
LINE        \n
OPERATOR    "+"|"-"|"*"|"/"|"="|"<"|"<-"|"."|"~"
DELIMITERS  "("|")"|"{"|"}"|";"|","|":"
ERROR       .

%%

{KEYWORD}       {keyword_handle(yytext);}
{ID}           {printf("#%d %s\n", line+1, yytext);}
{STRING}        {printf("#%d %s\n", line+1, yytext);}
{INTEGER}       {printf("#%d %s\n", line+1, yytext);}
{WHITE}         {/* do nothing */}
{LINE}          {++line;}
{OPERATOR}      {printf("#%d '%s'\n", line+1, yytext);}
{DELIMITERS}    {printf("#%d '%s'\n", line+1, yytext);}
{ERROR}         {printf("#%d ERROR \"%s\"\n", line+1, yytext);}


%%

void keyword_handle(char *str){
   if(!strcmp(str, "class")){
      printf("#%d CLASS\n", line + 1);
      keyword[CLASS]++;
   }else if(!strcmp(str, "inherits")){
      printf("#%d INHERITS\n", line + 1);
      keyword[INHERITS]++;
   }else if(!strcmp(str, "if")){
      printf("#%d IF\n", line + 1);
      keyword[IF]++;
   }else if(!strcmp(str, "then")){
      printf("#%d THEN\n", line + 1);
```

```
        keyword[THEN]++;
    }else if(!strcmp(str, "else")){
        printf("#%d ELSE\n", line + 1);
        keyword[ELSE]++;
    }else if(!strcmp(str, "fi")){
        printf("#%d FI\n", line + 1);
        keyword[FI]++;
    }else if(!strcmp(str, "while")){
        printf("#%d WHILE\n", line + 1);
        keyword[WHILE]++;
    }else if(!strcmp(str, "loop")){
        printf("#%d LOOP\n", line + 1);
        keyword[LOOP]++;
    }else if(!strcmp(str, "pool")){
        printf("#%d POOL\n", line + 1);
        keyword[POOL]++;
    }else if(!strcmp(str, "case")){
        printf("#%d CASE\n", line + 1);
        keyword[CASE]++;
    }else if(!strcmp(str, "esac")){
        printf("#%d ESAC\n", line + 1);
        keyword[ESAC]++;
    }else if(!strcmp(str, "of")){
        printf("#%d OF\n", line + 1);
        keyword[OF]++;
    }else if(!strcmp(str, "let")){
        printf("#%d LET\n", line + 1);
        keyword[LET]++;
    }else if(!strcmp(str, "in")){
        printf("#%d IN\n", line + 1);
        keyword[IN]++;
    }else if(!strcmp(str, "new")){
        printf("#%d NEW\n", line + 1);
        keyword[NEW]++;
    }else if(!strcmp(str, "isvoid")){
        printf("#%d ISVOID\n", line + 1);
        keyword[ISVOID]++;
    }else if(!strcmp(str, "not")){
        printf("#%d NOT\n", line + 1);
        keyword[NOT]++;
    }
}

void keyword_show(void){
    for(int i = 0; i < KEYWORD_NUM; i++)
        if(keyword[i])
            printf("%s: %d\n", key[i], keyword[i]);
}

int main(int argc, char** argv) {
    if (argc == 2) {
        if ((yyin = fopen(argv[1], "r")) == NULL) {
            printf("cannot open %s\n", argv[1]);
            return 0;
        }
    } else {
        printf("usage: %s <file>\n", argv[1]);
        return 0;
    }
    yylex();
    keyword_show();
    return 0;
}
```

(4）分析关键字、标识符、注释

```
%{

#include <stdio.h>
#include <string.h>
#define KEYWORD_NUM 17
int line = 0;
enum {CLASS, INHERITS, IF, THEN, ELSE, FI, WHILE, LOOP, POOL, CASE,
ESAC, OF, LET, IN, NEW, ISVOID, NOT};
char * key[] =
{"class","inherits","if","then","else","fi","while","loop","pool","
case","esac","of","let","in","new","isvoid","not"};
int keyword[KEYWORD_NUM] = {0};
void keyword_handle(char *str);
void keyword_show(void);

%}

KEYWORD
"class"|"inherits"|"if"|"then"|"else"|"fi"|"while"|"loop"|"pool"|"c
ase"|"esac"|"of"|"let"|"in"|"new"|"isvoid"|"not"
TYPEID      [A-Z]+[_A-Za-z0-9]*
OBJECTID    [a-z]+[_a-zA-Z0-9]*
STRING      \"[^"\n]*\"
```

```
INTEGER     [0-9]+
WHITE       [ \t]+
LINE        \n
OPERATOR    "+"|"-"|"*"|"/"|"="|"<"|"<-"|"."|"~"
DELIMITERS  "("|")"|"{"|"}"|";"|","|":"
ERROR       .
%x MutiCom

%%

"--"[^\n]*            {/* do nothing */}
"(*"                  {BEGIN(MutiCom);}
<MutiCom>[^\n*]*       {/* do nothing */}
<MutiCom>"*"+[^*)\n]*   {/* do nothing */}
<MutiCom>\n           {++line;}
<MutiCom>"*"+")"       {BEGIN(0);}
{KEYWORD}             {keyword_handle(yytext);}
{TYPEID}              {printf("#%d TYPEID %s\n", line+1, yytext);}
{OBJECTID}            {printf("#%d OBJECTID %s\n", line+1,
yytext);}
{STRING}              {printf("#%d %s\n", line+1, yytext);}
{INTEGER}             {printf("#%d %s\n", line+1, yytext);}
{WHITE}               {/* do nothing */}
{LINE}                {++line;}
{OPERATOR}            {printf("#%d '%s'\n", line+1, yytext);}
{DELIMITERS}          {printf("#%d '%s'\n", line+1, yytext);}
{ERROR}               {printf("#%d ERROR \"%s\"\n", line+1,
yytext);}

%%

void keyword_handle(char *str){
   if(!strcmp(str, "class")){
      printf("#%d CLASS\n", line + 1);
      keyword[CLASS]++;
   }else if(!strcmp(str, "inherits")){
      printf("#%d INHERITS\n", line + 1);
      keyword[INHERITS]++;
   }else if(!strcmp(str, "if")){
      printf("#%d IF\n", line + 1);
      keyword[IF]++;
   }else if(!strcmp(str, "then")){
      printf("#%d THEN\n", line + 1);
      keyword[THEN]++;
```

```
    }else if(!strcmp(str, "else")){
        printf("#%d ELSE\n", line + 1);
        keyword[ELSE]++;
    }else if(!strcmp(str, "fi")){
        printf("#%d FI\n", line + 1);
        keyword[FI]++;
    }else if(!strcmp(str, "while")){
        printf("#%d WHILE\n", line + 1);
        keyword[WHILE]++;
    }else if(!strcmp(str, "loop")){
        printf("#%d LOOP\n", line + 1);
        keyword[LOOP]++;
    }else if(!strcmp(str, "pool")){
        printf("#%d POOL\n", line + 1);
        keyword[POOL]++;
    }else if(!strcmp(str, "case")){
        printf("#%d CASE\n", line + 1);
        keyword[CASE]++;
    }else if(!strcmp(str, "esac")){
        printf("#%d ESAC\n", line + 1);
        keyword[ESAC]++;
    }else if(!strcmp(str, "of")){
        printf("#%d OF\n", line + 1);
        keyword[OF]++;
    }else if(!strcmp(str, "let")){
        printf("#%d LET\n", line + 1);
        keyword[LET]++;
    }else if(!strcmp(str, "in")){
        printf("#%d IN\n", line + 1);
        keyword[IN]++;
    }else if(!strcmp(str, "new")){
        printf("#%d NEW\n", line + 1);
        keyword[NEW]++;
    }else if(!strcmp(str, "isvoid")){
        printf("#%d ISVOID\n", line + 1);
        keyword[ISVOID]++;
    }else if(!strcmp(str, "not")){
        printf("#%d NOT\n", line + 1);
        keyword[NOT]++;
    }
}

void keyword_show(void){
    for(int i = 0; i < KEYWORD_NUM; i++)
```

```
      if(keyword[i])
         printf("%s: %d\n", key[i], keyword[i]);
}

int main(int argc, char** argv) {
   if (argc == 2) {
      if ((yyin = fopen(argv[1], "r")) == NULL) {
         printf("cannot open %s\n", argv[1]);
         return 0;
      }
   } else {
      printf("usage: %s <file>\n", argv[1]);
      return 0;
   }
   yylex();
   keyword_show();
   return 0;
}
```

(5) Makefile

```
CC = cc
LEX = flex

SOURCE = default
TARGET = default
INPUT = default
C_FILE = lex.yy.c

.PHONY: all
all: ${SOURCE}
	@${LEX} ${SOURCE}
	@${CC} -o ${TARGET} ${C_FILE} -ll
	@./${TARGET} ${INPUT}
	@rm ${C_FILE}
```