Programming Assignment II

15-李晓畅-凌之寒

1.1 词法分析器实现与 cool.flex 代码简述

按照给出框架，分为个部分分别实现：

1.运算符和关键字：

按照文档给出的要求：

> The keywords of cool are: **class, else, false, fi, if, in, inherits, isvoid, let, loop, pool, then, while, case, esac, new, of, not, true.** Except for the constants **true** and **false**, keywords are case insensitive. To conform to the rules for other objects, the first letter of **true** and **false** must be lowercase; the trailing letters may be upper or lower case.

我们应该对应实现如上所述的关键字和运算符。

运算符只需要对应声明，然后再对应实现方法即可。

```
LE                  <=
DARROW              =>
ASSIGN              <-
```

```
121   {DARROW}      { return (DARROW); }
122   {ASSIGN}      { return (ASSIGN); }
123   {LE}          { return (LE); }
```

但值得注意的是对关键字的处理：

一般关键字应该是不区分大小写的，因而在具体实现上可以类似于：

```
(?i:CLASS)      { return (CLASS); }
(?i:ELSE)       { return (ELSE); }
(?i:FI)         { return (FI); }
```

这种样式。

而布尔变量要求首字母必须小写而其他字母可以忽略大小写；识别出对应的布尔后应该将其加入符号表供下一步使用，因而实现上就类似于：

```
TRUE                t[Rr][Uu][Ee]
FALSE               f[Aa][Ll][Ss][Ee]
```

```
156   {FALSE} {
157     cool_yylval.boolean = false;
158     return (BOOL_CONST);
159   }
```

2. 整数、标识符和特殊标记

对应定义符合其要求的规则：

```
65    INT               [0-9]+
66    NAMECHAR          [0-9a-zA-Z_]
67    TYPE              [A-Z]{NAMECHAR}*
68    OBJECT            [a-z]{NAMECHAR}*
```

再以类似于下面的方式识别，并加入符号表即可。

```
160   {INT} {
161     cool_yylval.symbol = inttable.add_string(yytext);
162     return (INT_CONST);
163   }
```

3. 注释

单行注释容易实现`115  --.* {}`。

实现多行注释需要添加一个状态`%x COMMENT`和一个表示注释层数的计数器`49  static int commentLayer =0;`。

具体来说，正常情况下：在读到(*时将开始 COMMENT 状态，

```
"(*" {
  BEGIN(COMMENT);
  commentLayer = 1;
 }
```

读到大多数字符都可以直接忽略掉；读到(*或*)应该修改注释层数，若是最外层的*)就应该结束注释状态；

```
<COMMENT>"*)" {
  commentLayer --;
  if (commentLayer == 0 ) BEGIN(0);
 }
```

读到\n 应该增加行数。

```
<COMMENT>"\n" {
  curr_lineno ++;
 }
```

特殊地，注释中包含 EOF 符将报告 EOF in comment 错；未启动 COMMENT 状态下读到*)应该报 Unmatched *)错。

4.字符串

使用一个字符串作为缓冲记录识别到的字符`static std::string str="";`，

用 STR 状态和 ERRORSTR 状态标记字符串正在识别字符串和发现错误字符串

```
%x STR
%x ERRORSTR
```

。

具体来说，正常情况下识别到前引号”将开始识别字符串过程；

字符串识别过程对大多数字符而言都是直接加入到缓冲，但对表达转义的字符应该特殊处理；

```
<STR>\\. {
  // std :: cout<<"SPACE: "<<yytext<<endl;
  switch (yytext[1]){
    case 'n': str.append("\n"); break;
    case 'b': str.append("\b"); break;
    case 'f': str.append("\f"); break;
    case 't': str.append("\t"); break;
    default : str.push_back(yytext[1]);
  }
}
```

可以换行表示字符串，但必须加\；

```
<STR>\\\n {
  str.push_back(yytext[1]);
  curr_lineno ++;
}
```

在识别到后引号且长度符合定义即可结束对该字符串的识别。

特殊地，单个的\0 在字符串中总是非法的，将报包含空字符错；

```
<STR>. {
  if (yytext[0]==0) {cool_yylval.error_msg = "String contains null character."; BEGIN(ERRORSTR);}
  str.push_back(yytext[0]);
}
```

对于上面描述的错误，将进入 E R R O R S T R状态，此状态下什么都不做直到找到可以重启的地方再次开始识别；

```
237   <ERRORSTR>[\"\n] {
238     BEGIN(0);
239     return ERROR;
240   }
```

COOL 语言不允许字符串直接换行：

```
229   <STR>[\n] {
230     cool_yylval.error_msg = "Unterminated string constant";
231     curr_lineno ++;
232     BEGIN(0);
233     return ERROR;
234   }
```

字符串未正常结束情况下读到 EOF 将报对应报错；

```
193   <STR><<EOF>> {
194     cool_yylval.error_msg = "EOF in string constant";
195     BEGIN(0);
196     return ERROR;
197   }
```

即使字符串正常识别到后引号，仍有可能因过长而出错：

```
210   <STR>\" {
211     if (str.length()>= MAX_STR_CONST ) {
212       cool_yylval.error_msg="String constant too long";
213       BEGIN(0); return ERROR;}
214     BEGIN(0);
215     cool_yylval.symbol = stringtable.add_string((char*)str.c_str());
216     return (STR_CONST);
217   }
```

5 .其他字符

一般状态下，对读到的空格符直接忽略就可以了

```
63    SPACE             [ \t\f\r\v]
```

```
175   {SPACE} {}
```

COOL 支持的合法字符集并不很多，可以将其单独写出来：

```
245   [\(\)\{\}<=,:;~@] {
246       return yytext[0];
247   }
248
249   [\-\+\*\/\.] {
250       return yytext[0];
251   }
```

那对于剩下的其他字符应该对应报错：

```
253   . {
254     if (yytext[0]==0) {cool_yylval.error_msg = "\000"; return ERROR;}
255     cool_yylval.error_msg = yytext;
256     return ERROR;
257   }
```

1.2 验证

按照要求执行代码，结果如下：

```
wojtek@wojtek-virtual-machine:~/user/Compile-Lab/assignments/PA2$ make lexer
flex -d -ocool-lex.cc cool.flex
/bin/sh -ec 'g++ -MM -I. -I../../include/PA2 -I../../src/PA2 cool-lex.cc | sed '\''s/\(cool-lex\.o\)[ :]*/\1 cool-lex.d : /g'\''
 > cool-lex.d'
/bin/sh -ec 'g++ -MM -I. -I../../include/PA2 -I../../src/PA2 handle_flags.cc | sed '\''s/\(handle_flags\.o\)[ :]*/\1 handle_flag
s.d : /g'\'' > handle_flags.d'
/bin/sh -ec 'g++ -MM -I. -I../../include/PA2 -I../../src/PA2 stringtab.cc | sed '\''s/\(stringtab\.o\)[ :]*/\1 stringtab.d : /g'
\'' > stringtab.d'
/bin/sh -ec 'g++ -MM -I. -I../../include/PA2 -I../../src/PA2 utilities.cc | sed '\''s/\(utilities\.o\)[ :]*/\1 utilities.d : /g'
\'' > utilities.d'
/bin/sh -ec 'g++ -MM -I. -I../../include/PA2 -I../../src/PA2 lextest.cc | sed '\''s/\(lextest\.o\)[ :]*/\1 lextest.d : /g'\'' >
lextest.d'
g++ -g -Wall -Wno-unused -Wno-write-strings -I. -I../../include/PA2 -I../../src/PA2 -c cool-lex.cc
g++ -g -Wall -Wno-unused -Wno-write-strings -I. -I../../include/PA2 -I../../src/PA2 lextest.o utilities.o stringtab.o handle_fla
gs.o cool-lex.o -lfl -o lexer
wojtek@wojtek-virtual-machine:~/user/Compile-Lab/assignments/PA2$ ./checker test.cl
passed
```

同预期一致，测试通过。

2.测试样例构建

对 4.1 和 4.4 要求的错误构造样例：

```
`~!@#$%^&*()_+=[]'<>?/.
___what_am_I
LET_STMT --This is strange! *)
"I am an unescaped
newline"
"I am an escaped\
newline"
*)
(*I am EOF in comment
```

包含对合法/非法字符；变量名；单独出现的*)；unescaped/escaped 换行和 EOF 的检查。

同预期一致

```
wojtek@wojtek-virtual-machine:~/user/Compile-Lab/assignments/PA2$ ./checker test2
passed
```

对 4.3 构造样例：

```
"123\0"
"123\\0"
"123\\\0"
"123\\\\0"
"123164564156156165149874896165498165847986151231645641561561651498748961654981658479861512316456415615
"123\nab"
"
```

包含对字符串中转义字符、超长字符串和 EOF 的检查。

同预期一致

```
wojtek@wojtek-virtual-machine:~/user/Compile-Lab/assignments/PA2$ ./checker test1
passed
```

3.特殊情况和处理

我们发现，如果在正文中出现\0 标准词法分析器会报告`#6 ERROR "\000"`。

我们将其作为特殊情况处理。`if (yytext[0]==0) {cool_yylval.error_msg = "\000"; return ERROR;}`

4.实验总结

> 本次实验中我们运用 flex 工具实现了 COOL 语言的词法分析器。
>
> flex 将符合指定格式的文件，转化为 C 代码，用于进行**词法分析** Lexical Analysis。在.flex 文件中，我们通过设置**正则表达式**，定义了一些**词汇** lexeme，这些词汇将被转化为抽象的符号 token，作为词法分析的结果。
>
> 在词法分析器的设计中我们不难看出 COOL 语言一些独特安排的好处。比如对 type 和 object 的特殊要求、提供较少的运算符和其他符号等设计可以使得我们在构建词法分析器时更简单方便。
>
> 良好的词法分析器需要精巧的设计，正确地规划先后顺序、长度才能正确实现功能并保证正确。

附录

Cool.flex 代码如下：

```
/*
 *  The scanner definition for COOL.
 */

/*
 *  Stuff enclosed in %{ %} in the first section is copied verbatim to the
 *  output, so headers and global definitions are placed here to be visible
 * to the code in the file.  Don't remove anything that was here initially
 */
%{
#include <cool-parse.h>
#include <stringtab.h>
#include <utilities.h>
#include <stdint.h>
#include <string>
#include <iostream>

/* The compiler assumes these identifiers. */
#define yylval cool_yylval
#define yylex  cool_yylex

/* Max size of string constants */
#define MAX_STR_CONST 1025
#define YY_NO_UNPUT   /* keep g++ happy */

extern FILE *fin; /* we read from this file */

/* define YY_INPUT so we read from the FILE fin:
 * This change makes it possible to use this scanner in
 * the Cool compiler.
 */
#undef YY_INPUT
#define YY_INPUT(buf,result,max_size) \
    if ( (result = fread( (char*)buf, sizeof(char), max_size, fin)) < 0) \
        YY_FATAL_ERROR( "read() in flex scanner failed");

char string_buf[MAX_STR_CONST]; /* to assemble string constants */
char *string_buf_ptr;

extern int curr_lineno;
extern int verbose_flag;

extern YYSTYPE cool_yylval;

/*
```

```
 *  Add Your own definitions here
 */

static int commentLayer =0;
static std::string str="";

static int errorflag = 0;

%}

/*
 * Define names for regular expressions here.
 */
LE              <=
DARROW          =>
ASSIGN          <-

SPACE           [ \t\f\r\v]

INT             [0-9]+
NAMECHAR        [0-9a-zA-Z_]
TYPE            [A-Z]{NAMECHAR}*
OBJECT          [a-z]{NAMECHAR}*
TRUE            t[Rr][Uu][Ee]
FALSE           f[Aa][Ll][Ss][Ee]

%x COMMENT
%x STR
%x ERRORSTR

%option noyywrap
%%
 /*
  *  Nested comments
  */

"(*" {
  BEGIN(COMMENT);
  commentLayer = 1;
 }

<COMMENT>"\n" {
  curr_lineno ++;
}

<COMMENT><<EOF>> {
  cool_yylval.error_msg = "EOF in comment";
  BEGIN(0);
  return ERROR;
}

<COMMENT>. {

}

  <COMMENT>"(*" {
    commentLayer ++;
  }

  <COMMENT>"*)" {
    commentLayer --;
    if (commentLayer == 0 ) BEGIN(0);
  }

  \*\) {
      cool_yylval.error_msg = "Unmatched *)";
      return (ERROR);
  }

```

```
--.* {}


 /*
  *  The multiple-character operators.
  */
{DARROW}        { return (DARROW); }
{ASSIGN}    { return (ASSIGN); }
{LE}        { return (LE); }

 /*
  * Keywords are case-insensitive except for the values true and false,
  * which must begin with a lower-case letter.
  */
[\[\]\'>] {
    cool_yylval.error_msg = yytext;
    return (ERROR);
}
(?i:CLASS)      { return (CLASS); }
(?i:ELSE)         { return (ELSE); }
(?i:FI)           { return (FI); }
(?i:IF)           { return (IF); }
(?i:IN)           { return (IN); }
(?i:INHERITS)   { return (INHERITS); }
(?i:LET)          { return (LET); }
(?i:LOOP)         { return (LOOP); }
(?i:POOL)         { return (POOL); }
(?i:THEN)         { return (THEN); }
(?i:WHILE)      { return (WHILE); }
(?i:CASE)         { return (CASE); }
(?i:ESAC)         { return (ESAC); }
(?i:OF)           { return (OF); }
(?i:NEW)          { return (NEW); }
(?i:LE)           { return (LE); }
(?i:NOT)          { return (NOT); }
(?i:ISVOID)     { return (ISVOID); }

{TRUE} {
  cool_yylval.boolean = true;
  return (BOOL_CONST);
}
{FALSE} {
  cool_yylval.boolean = false;
  return (BOOL_CONST);
}
{INT} {
  cool_yylval.symbol = inttable.add_string(yytext);
  return (INT_CONST);
}

{TYPE} {
  cool_yylval.symbol = idtable.add_string(yytext);
  return (TYPEID);
}

{OBJECT} {
  cool_yylval.symbol = idtable.add_string(yytext);
  return (OBJECTID);
}

{SPACE} {}

[\n] {
  curr_lineno ++;
}

 /*
  *  String constants (C syntax)
  *  Escape sequence \c is accepted for all characters c. Except for
```

```
    *  \n \t \b \f, the result is c.
    *
    */

\" {
  BEGIN (STR);
  str = "";
}

<STR><<EOF>> {
  cool_yylval.error_msg = "EOF in string constant";
  BEGIN(0);
  return ERROR;
}

<STR>\\. {
  // std :: cout<<"SPACE: "<<yytext<<endl;
  switch (yytext[1]){
    case 'n': str.append("\n"); break;
    case 'b': str.append("\b"); break;
    case 'f': str.append("\f"); break;
    case 't': str.append("\t"); break;
    case '\0': cool_yylval.error_msg = "String contains escaped null character."; BEGIN(ERRORSTR); break;
    default : str.push_back(yytext[1]);
  }
}

<STR>\" {
  // cout<<"DONE "<<str.length();
  if (str.length()>= MAX_STR_CONST ) {cool_yylval.error_msg="String constant too long"; BEGIN(0); return ERROR;}
  // cout<<"New str: "<<str<<endl;
  BEGIN(0);
  cool_yylval.symbol = stringtable.add_string((char*)str.c_str());
  return (STR_CONST);
}

<STR>\\\n {
  str.push_back(yytext[1]);
  curr_lineno ++;
}

<STR>. {
  if (yytext[0]==0) {cool_yylval.error_msg = "String contains null character."; BEGIN(ERRORSTR);}
  str.push_back(yytext[0]);
}

<STR>[\n] {
  cool_yylval.error_msg = "Unterminated string constant";
  curr_lineno ++;
  // cout<< "EOL in string constant"<< endl;
  BEGIN(0);
  return ERROR;
}

<ERRORSTR>[\"\n] {
  BEGIN(0);
  return ERROR;
}

<ERRORSTR>. {

}

[\(\)\{\}<=,:;~@] {
    return yytext[0];
}

[\-\+\*\/\.] {
    return yytext[0];
```

```
253.  }
254.
255.  . {
256.    if (yytext[0]==0) {cool_yylval.error_msg = "\000"; return ERROR;}
257.    cool_yylval.error_msg = yytext;
258.    return ERROR;
259.  }
260.
261.  %%
```