词法分析实验

代码逻辑

正则表达式

首先，给所有单词符号编写对应的正则表达式

这一步中的关键问题有以下几个

- 负数的处理：一种方案是在十进制整数前加上 `-?`，将其在词法阶段处理。但是在 sysY 的定义中，数值常量不包含负号，负号任何时候都作为一个单目运算符处理。所以我们采用的是后者。
- 0 的处理：值得注意的是，0 在 sysY 中被认定为八进制数，我们遵守了这点。

> 以上两点的相关代码
>
> ```
> OctConst 0[0-7]*
> DecConst [1-9][0-9]*
> HexConst 0[xX][0-9a-fA-F]+
> IntConst {DecConst}|{OctConst}|{HexConst}
> ```

- 错误的标识符的处理：常见的错误是以数字开头，我们虽然可以在词法阶段处理一部分，但类似的错误是不可穷尽的，所以我们更应该在语法阶段处理，保证逻辑的一致性（不会类似的错误，一会是词法报错，一会是语法报错）。
- 错误的特殊字符的处理：接上一点，除了以数字开头的标识符这种错误。还有例如代码中出现 `@` 等符号这种错误（因为 sysY 不含字符串，所以特殊符号不应该出现），这种错误是适合在词法阶段处理的，只要出现规则之外的字符，我们用 `.` 匹配就可以报错。
- 单行注释的处理：`\/\/[^\n]*` 两个斜杠开头，尽量匹配，直到遇到换行符。根据 sysY 的定义，单行注释不包含换行符，所以以上正则表达式是刚好吻合的。
- 多行注释的处理：利用 flex 提供了一种有条件地激活规则的机制，文档中叫 `Start Conditions`

> ```
> commentbegin "/*"
> commentelement .
> commentline \n
> commentend "*/"
> {commentbegin} {BEGIN COMMENT; UPDATE;}
> <COMMENT>{commentline} {current_line++; UPDATE;}
> <COMMENT>{commentelement} {UPDATE;}
> <COMMENT>{commentend} {BEGIN INITIAL; UPDATE;}
> ```

- 跨系统换行符的问题：在 Windows, Linux, Mac 中，其换行符分别是 `\r\n`, `\n`, `\r`。如果要跨系统，一种方案是 `NEWLINE \r\n|\r|\n`，这样它在贪婪匹配的时候会吸收 `\r\n` 为一个换行，其他情况亦然。但这样其实也有问题，例如在 Windows 环境中，我故意输出了一个 `\r`，那么词法分析器也会把它认为是一个换行，然而此时 `\r` 是一个非法字符。我们不可能通过词法分析检测出对应的文件是哪个系统的，所以事实上只能说：在得到的文件是正常输入得到的情况下，上面的方案是可行的。

以上之外，就是简单的匹配了。

统计行列的逻辑

flex 其实提供了一个宏 `yylineno`，在开启后（`%option yylineno`），可以直接获取当前行号，但是，经过测试，它并不识别 Mac 系统下的 `\r`。所以我们自己实现了一个 `current_line` 变量，用来记录当前行号。

行列的统计逻辑是：遇到换行符 `NEWLINE`，行号加一，列号归一。否则，列号加 `yyleng`。

记录和输出的逻辑

考虑到之后的实验可能需要用到词法分析的结果，所以事先实现了一个 `struct token` 来保存词的属性和值。

其具体实现类似于：

```
typedef enum token_type {
    K, // keyword
    I, // identifier
    C, // constant
    O, // operator
    D, // delimiter
    T, // other
} token_type;
typedef enum keyword_value {
    INT,
    VOID,
    IF,
    ELSE,
    WHILE,
    RETURN,
    BREAK,
    CONST,
    MAIN,
} keyword_value;
typedef struct token{
    token_type type;
    char *text;
    int value;
} token;
```

这里 `value` 可以承载所有类型的值，不同类型的值可能重复，但类型不同就可以区分。

为了方便，定义了两个宏用来更新列和输出。

```
#define UPDATE current_column += yyleng;
#define OUTPUT fprintf(yyout, "%s: %c, (%d, %d)\n", yytext,
get_type_name(tmp.type), current_line, current_column);
```

这样在识别到一个单词符号的时候，就会形如以下创建一个临时局部变量，然后输出。

```
{Main} {token tmp = create_token(K, MAIN); OUTPUT; UPDATE;}
{Const} {token tmp = create_token(K, CONST); OUTPUT; UPDATE;}
{Int} {token tmp = create_token(K, INT); OUTPUT; UPDATE;}
```

实验结果

用 `Makefile` 编写测试脚本，运行所有的测试用例，然后比较。

> 这里找其他热心同学交换了一下输出结果来测试，因为其系统是 `Windows`，我是 `Linux`，所以还要用 `--strip-trailing-cr` 处理一下换行符不一致的问题。

```
all: lex

lex: lex.l
    flex lex.l
    gcc lex.yy.c -o lex

test1:
    ./lex < dataset/1.sy > 1.out
    diff --strip-trailing-cr 1.out output/my1.out

test2:
    ./lex < dataset/2.sy > 2.out
    diff --strip-trailing-cr 2.out output/my2.out

test3:
    ./lex < dataset/3.sy > 3.out
    diff --strip-trailing-cr 3.out output/my3.out

test4:
    ./lex < dataset/4.sy > 4.out
    diff --strip-trailing-cr 4.out output/my4.out

test5:
    ./lex < dataset/5.sy > 5.out
    diff --strip-trailing-cr 5.out output/my5.out

test6:
    ./lex < dataset/6.sy > 6.out
    diff --strip-trailing-cr 6.out output/my6.out

testall: test1 test2 test3 test4 test5 test6
```

完整代码

源代码下载

```
%option noyywrap
%{
    #include <stdio.h>
    #include <string.h>
    typedef enum token_type {
        K, // keyword
        I, // identifier
        C, // constant
        O, // operator
        D, // delimiter
        T, // other
```

```
} token_type;
typedef enum keyword_value {
    INT,
    VOID,
    IF,
    ELSE,
    WHILE,
    RETURN,
    BREAK,
    CONST,
    MAIN,
} keyword_value;
typedef enum operator_value {
    PLUS,
    MINUS,
    STAR,
    DIV,
    MOD,
    LT,
    GT,
    LE,
    GE,
    EQ,
    NE,
    AND,
    OR,
    NOT,
    ASSIGN,
} operator_value;
typedef enum delimiter_value {
    LP,
    RP,
    LS,
    RS,
    LC,
    RC,
    COMMA,
    SEMI,
} delimiter_value;
typedef struct token{
    token_type type;
    char *text;
    int value;
} token;
token* create_token_ptr(token_type type, int value) {
    char* text = yytext;
    token *tmp = malloc(sizeof(token));
    tmp->type = type;
    tmp->text = malloc(strlen(text) + 1);
    strcpy(tmp->text, text);
    tmp->value = value;
    return tmp;
}
token create_token(token_type type, int value) {
    char* text = yytext;
```

```
        token tmp;
        tmp.type = type;
        tmp.text = malloc(strlen(text) + 1);
        strcpy(tmp.text, text);
        tmp.value = value;
        return tmp;
    }
    char get_type_name(token_type type) {
        switch (type) {
            case K: return 'K';
            case I: return 'I';
            case C: return 'C';
            case O: return 'O';
            case D: return 'D';
            case T: return 'T';
            default: return '?';
        }
    }
    int current_line = 1;
    int current_column = 1;
    #define UPDATE current_column += yyleng;
    #define OUTPUT fprintf(yyout, "%s: %c, (%d, %d)\n", yytext,
get_type_name(tmp.type), current_line, current_column);
%}

NEWLINE \r\n|\r|\n
SPACE [ \t]+
Main main
Const const
Int int
Void void
If if
Else else
While while
Break break
Return return
LP \(
RP \)
LS \[
RS \]
LC \{
RC \}
COMMA ,
SEMI ;
PLUS \+
MINUS -
STAR \*
DIV \/
MOD %
LT <
GT >
LE <=
GE >=
EQ ==
NE !=
```

```
AND &&
OR \|\|
NOT !
ASSIGN =
Ident [a-zA-Z_][a-zA-Z_0-9]*
WOGIdent [0-9][a-zA-Z_0-9]*
OctConst 0[0-7]*
DecConst [1-9][0-9]*
HexConst 0[xX][0-9a-fA-F]+
IntConst {DecConst}|{OctConst}|{HexConst}
LineComment \/\/[^\n]*
commentbegin "/*"
commentelement .
commentline \n
commentend "*/"
WOG .|{WOGIdent}

%x COMMENT

%%


{commentbegin} {BEGIN COMMENT; UPDATE;}
<COMMENT>{commentline} {current_line++; UPDATE;}
<COMMENT>{commentelement} {UPDATE;}
<COMMENT>{commentend} {BEGIN INITIAL; UPDATE;}

{LineComment} {UPDATE;}
{Main} {token tmp = create_token(K, MAIN); OUTPUT; UPDATE;}
{Const} {token tmp = create_token(K, CONST); OUTPUT; UPDATE;}
{Int} {token tmp = create_token(K, INT); OUTPUT; UPDATE;}
{Void} {token tmp = create_token(K, VOID); OUTPUT; UPDATE;}
{If} {token tmp = create_token(K, IF); OUTPUT; UPDATE;}
{Else} {token tmp = create_token(K, ELSE); OUTPUT; UPDATE;}
{While} {token tmp = create_token(K, WHILE); OUTPUT; UPDATE;}
{Break} {token tmp = create_token(K, BREAK); OUTPUT; UPDATE;}
{Return} {token tmp = create_token(K, RETURN); OUTPUT; UPDATE;}
{LP} {token tmp = create_token(D, LP); OUTPUT; UPDATE;}
{RP} {token tmp = create_token(D, RP); OUTPUT; UPDATE;}
{LS} {token tmp = create_token(D, LS); OUTPUT; UPDATE;}
{RS} {token tmp = create_token(D, RS); OUTPUT; UPDATE;}
{LC} {token tmp = create_token(D, LC); OUTPUT; UPDATE;}
{RC} {token tmp = create_token(D, RC); OUTPUT; UPDATE;}
{COMMA} {token tmp = create_token(D, COMMA); OUTPUT; UPDATE;}
{SEMI} {token tmp = create_token(D, SEMI); OUTPUT; UPDATE;}
{PLUS} {token tmp = create_token(O, PLUS); OUTPUT; UPDATE;}
{MINUS} {token tmp = create_token(O, MINUS); OUTPUT; UPDATE;}
{STAR} {token tmp = create_token(O, STAR); OUTPUT; UPDATE;}
{DIV} {token tmp = create_token(O, DIV); OUTPUT; UPDATE;}
{MOD} {token tmp = create_token(O, MOD); OUTPUT; UPDATE;}
{LT} {token tmp = create_token(O, LT); OUTPUT; UPDATE;}
{GT} {token tmp = create_token(O, GT); OUTPUT; UPDATE;}
{LE} {token tmp = create_token(O, LE); OUTPUT; UPDATE;}
{GE} {token tmp = create_token(O, GE); OUTPUT; UPDATE;}
{EQ} {token tmp = create_token(O, EQ); OUTPUT; UPDATE;}
```

```
{NE} {token tmp = create_token(O, NE); OUTPUT; UPDATE;}
{AND} {token tmp = create_token(O, AND); OUTPUT; UPDATE;}
{OR} {token tmp = create_token(O, OR); OUTPUT; UPDATE;}
{NOT} {token tmp = create_token(O, NOT); OUTPUT; UPDATE;}
{ASSIGN} {token tmp = create_token(O, ASSIGN); OUTPUT; UPDATE;}

{IntConst} {token tmp = create_token(C, 0); OUTPUT; UPDATE;}
{Ident} {token tmp = create_token(I, 0); OUTPUT; UPDATE;}

{SPACE} {UPDATE; }
{NEWLINE} {current_line++; current_column = 1; }
{WOG} {token tmp=create_token(T, 0); OUTPUT; UPDATE;}

%%
int main(int argc, char **argv) {
    yylex();
    return 0;
}
```