实验五 对多条执行语句的递归下降分析

一、实验内容

【任务介绍】根据给定的上下文无关文法，对高级程序设计语言中常见的几种执行语句进行语法分析

【输入】一串执行语句，其中包括：赋值语句、选择语句和循环语句

【输出】如果分析正确，则输出为正确，否则输出错误

【题目】设计一个程序，根据给定的上下文无关文法，对于输入的一串源程序语句，判读语法是否正确或者报告错误。要求：

1. 基础文法以<BLOCK>为开始符号（此处为原文法消除左递归后的结果）

```
<BLOCK> -> { <DECLS> <STMTS> }
<DECLS> -> <DECLS1>
<DECLS1> -> <DECL> <DECLS1> | empty
<DECL> -> <TYPE> <NAMES>;
<NAMES> -> <NAME> <NAMES1>
<NAMES1> -> , <NAME> <NAMES1> | empty
<TYPE> -> int
<NAME> -> id

<STMTS> -> <STMTS1>
<STMTS1> -> <STMT> <STMTS1> | empty
<STMT> -> id = <EXPR>;
<STMT> -> if(<BOOL>) <STMT>
<STMT> -> if(<BOOL>) <STMT> else <STMT>
<STMT> -> while(<BOOL>) <STMT>
<STMT> -> <BLOCK>

<EXPR> -> <TERM> <EXPR1>
<EXPR1> -> <ADD> <TERM> <EXPR1> | empty
<ADD> -> + | -
<TERM> -> <FACTOR> <TERM1>
<TERM1> -> <MUL> <FACTOR> <TERM1> | empty
<MUL> -> * | /
<FACTOR> -> (<EXPR>) | id | number
<BOOL> -> <EXPR> <ROP> <EXPR>
<ROP> -> > | >= | < | <= | == | !=
```

2. 语法规则
    1. ）名字：由字母打头后跟字母、数字任意组合的字符串；长度不超过20；不区分大小写；把下划线看作第27个字母
    2. ）常数：完全由数字组成的字符串；正数和0前面不加符号，负数在正数前面加-构成；长度不超过15
    3. ）关键字、运算符、分隔符仅包含在文法定义中出现过的单词。
    4. ）字母表定义为 1.）~ 3.）中出现的字符的集合；不在该集合中的符号都以非法字符对待
3. 语法分析方法采用递归子程序法
4. 输入：一串（3~5句）执行语句，其中包括：赋值语句、选择语句和循环语句
5. 输出：如果分析正确，则输出为正确，否则输出错误

6. 赋值语句：做不为1个简单变量（假定都为整型），有部为1个算术表达式；可以调用实验四中的程序来完成对这个算术表达式的分析
7. 选择语句：包含if-then单分支和if-then-else双分支两种结构。只考虑分支判定条件为1个简单的关系运算表达式的情况，暂不处理逻辑运算
8. 循环语句：while-do

二、实验思路

1. 词法分析

- 调用实验三的词法分析程序提取出token

2. 语法分析

- 按照实验四的思路，将词法分析出的token作为输入，进行递归下降分析，为每一个产生式编写一个过程，如果检测到不合法语句就直接输出"Error"。如果所有语句都合法，则输出正确。
- FIRST

```
FIRST(BLOCK)  = {'{'}
FIRST(DECLS)  = {empty, int}
FIRST(DECLS1) = {empty, int}
FIRST(DECL)   = {int}
FIRST(NAMES)  = {id}
FIRST(NAMES1) = {',', empty}
FIRST(TYPE)   = {int}
FIRST(NAME)   = {id}
FIRST(STMTS)  = {empty, id, if, while, '{'}
FIRST(STMTS1) = {empty, id, if, while, '{'}
FIRST(STMT)   = {id, if, while, '{'}
FIRST(EXPR)   = {(, id, number}
FIRST(EXPTR1) = {empty, +,  -}
FIRST(ADD)    = {+, -}
FIRST(TERM)   = {(, id, number}
FIRST(TERM1)  = {empty, *, /}
FIRST(MUL)    = {*, /}
FIRST(FACTOR) = {(, id, number}
FIRST(BOOL)   = {(, id, number}
FIRST(ROP)    = {>, >=, <, <=, ==, !=}
```

- FOLLOW

```
FOLLOW(BLOCK)  = {$, id, if, while, '{', '}', else}
FOLLOW(DECLS)  = {id, if, while, '{', '}'}
FOLLOW(DECLS1) = {id, if, while, '{', '}'}
FOLLOW(DECL)   = {int, id, if, while, '{', '}'}
FOLLOW(NAMES)  = {';', ','}
FOLLOW(NAMES1) = {';', ','}
FOLLOW(TYPE)   = {id}
FOLLOW(NAME)   = {';', ','}
FOLLOW(STMTS)  = {'}'}
FOLLOW(STMTS1) = {'}'}
FOLLOW(STMT)   = {id, if, while, '{', '}', else}
```

```
FOLLOW(EXPR)   = {;, ), >, >=, <, <=, ==, !=}
FOLLOW(EXPR1)  = {;, ), >, >=, <, <=, ==, !=}
FOLLOW(ADD)    = {(, id, number}
FOLLOW(TERM)   = {+, -, ;, ), >, >=, <, <=, ==, !=}
FOLLOW(TERM1)  = {+, -, ;, ), >, >=, <, <=, ==, !=}
FOLLOW(MUL)    = {(, id, number}
FOLLOW(FACTOR) = {*, /, +, -, ;, ), >, >=, <, <=, ==, !=}
FOLLOW(BOOL)   = {)}
FOLLOW(ROP)    = {(, id, number}
```

3. 产生式处理过程（伪代码举例）

- 举例<DECLS1> -> <DECL> <DECLS1> | empty

```
procedure DECLS1(token t)
BEGIN
    IF(t属于DECLS1的FOLLOW) 直接返回
    DECL(t)
    DECLS(t的后继token)
END
```

三、实验环境

- 语言：C++
- 编译环境：Code Blocks 17.12 自带MinGW中的g++，需要在编译器设置中勾选C++14
- 输入文件（input1-4.txt）和代码放在一个目录下面
- 输入1：input1.txt (赋值，算术表达式)

  输出1：词法分析完成，正确

  语法分析完成，正确

  一个词法分析结果Word_List.txt
- 输入2：input2.txt（if-else语句）

  输出1：词法分析完成，正确

  语法分析完成，正确

  一个词法分析结果Word_List.txt
- 输入3：input3.txt（while-do语句）

  输出3：词法分析完成，正确

  语法分析完成，正确

  一个词法分析结果Word_List.txt
- 输入4：input4.txt （语法错误）

  输出1：词法分析完成，正确

  Error!