实验八九 对多条执行语句构造递归下降翻译器

一、实验内容

【任务性质】必做任务，分值10分。

【任务介绍】对能处理多条执行语句的递归下降分析器进行改造，使其能够一遍处理，同时完成语 法分析和中间代码翻译。

【输入】一串语句组成的语句块，其中包括：赋值语句、选择语句和循环语句。

【输出】与输入对应的一个四元式序列。

【题目】对实验五的程序进行升级改造，使得程序对于输入的任意一串语句，在对其做递归下降分 析的同时，生成等价的四元式序列，一遍完成。要求：

1. 基础文法：同实验五

(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)要求

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

2. 语法分析：沿用实验五的程序框架。
3. 语义处理：生成四元式序列。
4. 一遍处理：把语义处理的代码插入到语法分析的代码中。
5. 为简化问题，不考虑输入有错误的情况，不考虑语义检查

二、实验思路

1. 声明语句

- 在声明语句产生式<NAMES1> -> , <NAME> <NAMES1> | empty 和

  <NAMES> -> <NAME> <NAMES1>

  中会遇到NAME所指标识符，所以在NAME中添加语义动作

- <NAME> **{产生四元式(int, _, _, id)}**，id就是NAME中token的id
- ```
  void NAME(token t) {
      results.push_back(quad{"int", "_", "_", t.val});
      rescnt++;
  }
  ```

2. 算术表达式

- 与实验七思路相同，在有EXPR的产生式中EXPR后面加上最后对栈中剩余元素的处理
- <STMT> -> id = <EXPR>; {处理EXPR栈中剩余元素}
- <BOOL> -> <EXPR>{处理EXPR栈中剩余元素} <ROP> <EXPR>{处理EXPR栈中剩余元素}
- 将在下面赋值语句和if语句中出现

3. 赋值语句

- 观察文法赋值语句对应<STMT> -> id = <EXPR>; **{处理EXPR栈中剩余元素，产生四元式}**
- 在**STMT**函数最后加上**语义动作**：

  <STMT> -> id = <EXPR>; **{处理EXPR栈中剩余元素，产生四元式}**

  产生四元式如下

  ```
  (=, ti, _, id)       //ti是EXPR最后的结果，id即为STMT产生式中的id
  ```

- C++实现
  ```

```
if(t.type=="id") {  //id = <EXPR>;
    //语法分析处理
/*********************************************************************************
*********/
    ///对EXPR最后的处理
    while(!ope.empty()) {
        string a1 = num.top(); num.pop();
        string a2 = num.top(); num.pop();
        string o = ope.top(); ope.pop();
        string tmp = geneari(o, a2, a1);
        num.push(tmp);
    }
    ///语义动作，产生四元式(=, ti, _, id)
    results.push_back(quad{"=", num.top() "_", t.val});
    num.pop();
    rescnt++;
}
```

4. if-then

- <STMT> -> if(<BOOL>) **{产生BOOL.FALSE四元式}** <STMT>
- ```
  (jmp, _, _, BOOL.FALSE) //直接跳转到BOOL.FALSE
  ```

<BOOL>

- <BOOL> -> <EXPR1> **{产生EXPR1四元式}** <ROP> <EXPR2> **{产生EXPR2四元式}** **{产生BOOL四元式}**
- /因为此产生式后面会跟上表达式为假的跳转语句，所以表达式为真跳转到现在地址+2的位置
- ```
  (jROP, EXPR1结果, EXPR2结果, 表达式为真时跳转到 现地址+2)
  ```

5. if-then-else

- <STMT> -> if(<BOOL>) **{产生BOOL.FALSE四元式}** <STMT1> **{产生STMT1.NEXT四元式}**

  else **{回填BOOL.FALSE}** <STMT2> **{回填STMT1.NEXT}**
- ```
  else if(t.val=="if") {  ///<STMT>->if(<BOOL>)<STMT1>
          t = getNext();
          if(t.val!="(") {
              cout << "Error!" << endl;
              exit(0);
          }
          BOOL(getNext());
          t = getNext();
          if(t.val!=")") {
              cout << "Error!" << endl;
              exit(0);
          }
          //产生四元式
          results.push_back(quad{"jmp","_", "_","BOOL.FALSE"});
          rescnt++;
          int BOOL_FALSE = rescnt-2;
  ```
  ```

```
        STMT(getNext());

        //产生四元式
        results.push_back(quad{"jmp","_", "_","STMT.NEXT"});
        rescnt++;
        int STMT_NEXT = rescnt-2;

        t = getNext();
        if(t.val!="else") {
            //不是if-else，删除STMT_NEXT
            results.erase(results.begin()+STMT_NEXT);
            rescnt--;
            //回填BOOL.FALSE
            results[BOOL_FALSE].res = to_string(rescnt);
            cnt2--;
            return;
        }
        else {//<STMT> -> if(<BOOL>) <STMT1> else <STMT2>
            //回填BOOL.FALSE
            results[BOOL_FALSE].res = to_string(rescnt);
            STMT(getNext());
        }
        //回填STMT.NEXT
        results[STMT_NEXT].res = to_string(rescnt);
    }
```

5. while-do

- <STMT> -> **{创建并赋值WHILE_BEGIN}** while(<BOOL>) **{产生BOOL.FALSE四元式}**

  <STMT> **{产生跳至WHILE_BEGIN四元式}** **{回填BOOL.FALSE}**

- ```
  else if(t.val=="while") {//<STMT> -> while(<BOOL>) <STMT>
          int WHILE_BEGIN = rescnt;   //WHILE_BEGIN
          t = getNext();
          if(t.val!="(") {
              cout << "Error!" << endl;
              exit(0);
          }
          BOOL(getNext());
          t = getNext();
          if(t.val!=")") {
              cout << "Error!" << endl;
              exit(0);
          }
          //产生四元式
          results.push_back(quad{"jmp","_", "_","BOOL.FALSE"});
          rescnt++;
          int BOOL_FALSE = rescnt-2;

          STMT(getNext());
          //产生四元式
          results.push_back(quad{"jmp","_","_",to_string(WHILE_BEGIN)});
          rescnt++;
          //回填BOOL.FLASE
          results[BOOL_FALSE].res = to_string(rescnt);
      }
  ```
  ```

三、实验环境和结果

- 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1：input1.txt （赋值，算术表达式）

输出1：

```
Please input the filename: input1.txt
词法分析完成，正确
语法分析完成，正确
四元式如下：
 1  int        _        _        a
 2  int        _        _        b
 3  int        _        _        c
 4  int        _        _        d
 5    =        0        _        a
 6    +        a        1       t1
 7    =       t1        _        b
 8    -       10        b       t2
 9    =       t2        _        c
10    *        b        c       t3
11    =       t3        _        d
12    /        c        3       t4
13    =       t4        _        d
14
```

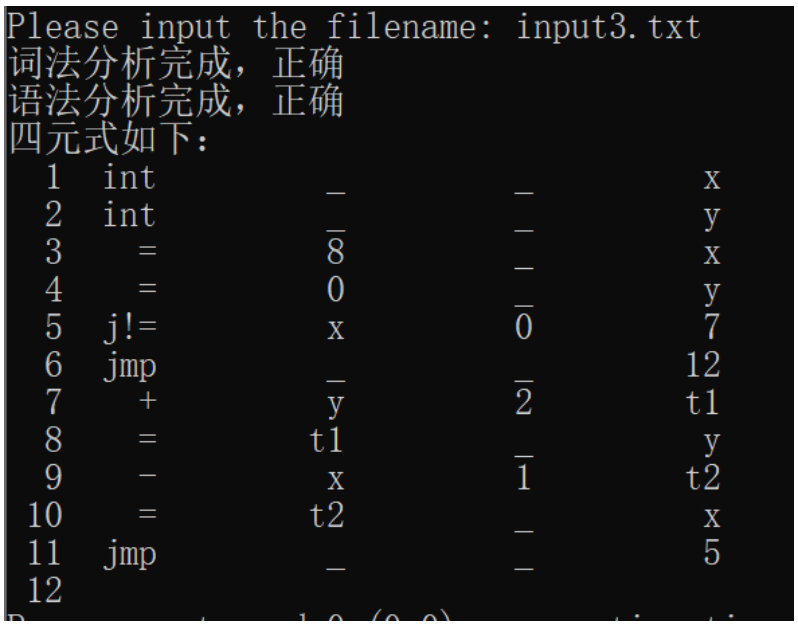
- 输入2：input2.txt (if-else语句)

输出2：

```
Please input the filename: input2.txt
词法分析完成，正确
语法分析完成，正确
四元式如下：
 1  int        _        _        x
 2    =        1        _        x
 3  j!=        x        0        5
 4  jmp        _        _        7
 5    =        0        _        x
 6  jmp        _        _        9
 7    -        x        1       t1
 8    =       t1        _        x
 9
```

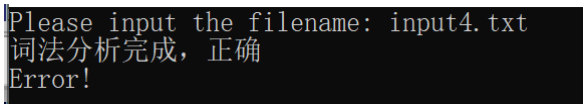
- 输入3：input3.txt (while-do语句)

输出3：



- 输入4：input4.txt （语法错误）

输出4：



- 输入5：input5.txt (while里面有if-else)

输出5：