编译原理实验三

——语义分析器——

19335253 葉珺明

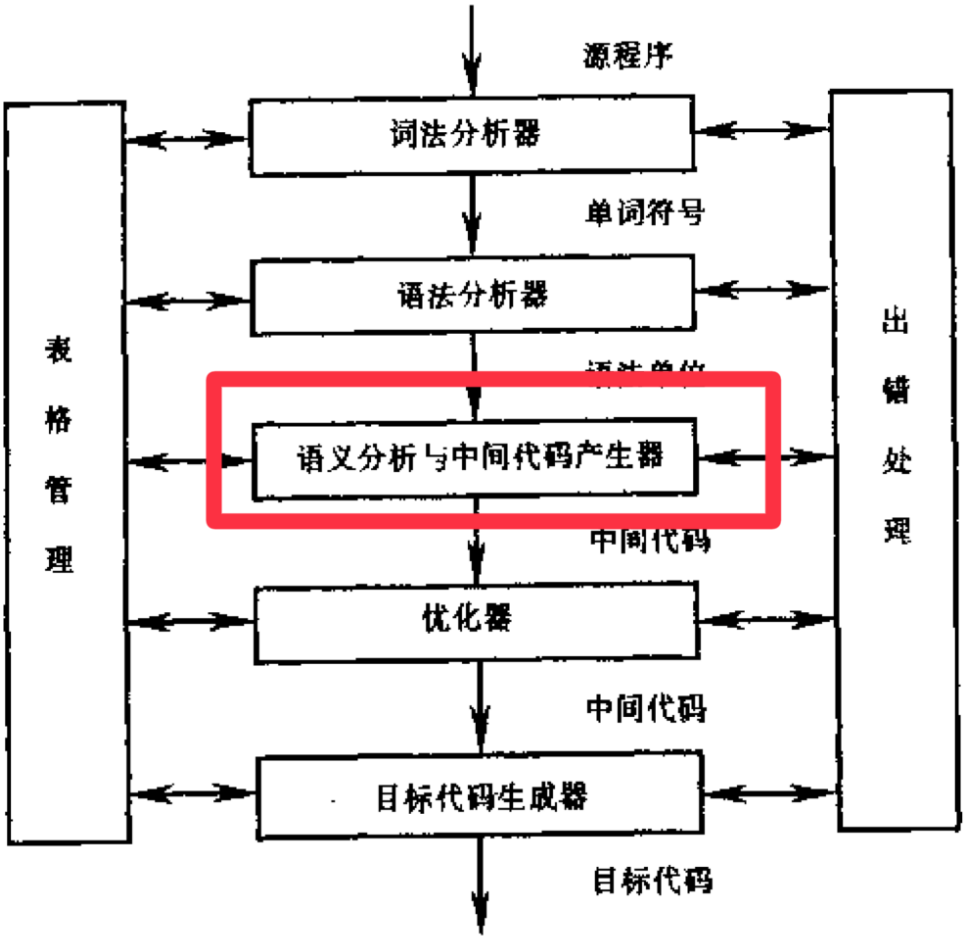
目录

所有文件源码地址： **click here**

一 实验目的

- 设计至少支持加减乘除以及括号操作的算术表达式语义分析器
- 实现语义分析器功能：将算术表达式翻译成四元式的中间语言
- 使用LL(1)翻译法实现

编译程序的总体处理过程：红色框选部分为本次实验要求实现的部分。



二 实验要求

- 用**LL(1)翻译法**设计实现算术表达式的语分析器

三 实验设计

3.1 算术表达式LL(1)文法

3.1.1 LL(1)文法设计

- 算术表达式的文法:

$$
\begin{aligned}
&E \to T|E+T|E-T \\
&T \to F|T*F|T/F \\
&F \to id|(E)
\end{aligned}
$$

其中id代表数字或字母

- 消除上述文法的左递归

$$
\begin{aligned}
&E \to T\{GEQ(+)\}E' && ① \\
&E' \to +T\{GEQ(-)\}E' && ② \\
&E' \to -TE' && ③ \\
&E' \to \epsilon && ④ \\
&T \to FT' && ⑤ \\
&T' \to *F\{GEQ(*)\}T' && ⑥ \\
&T' \to /F\{GEQ(/)\}T' && ⑦ \\
&T' \to \epsilon && ⑧ \\
&F \to id\{PUSH(i)\} && ⑨ \\
&F \to (E) && ⑩
\end{aligned}
$$

其中id代表数字或字母

- 构造LL(1)的预测分析表

	id	$+$	$-$	$*$	$/$	$($	$)$	$\#$
E	①					①		
E'		②	③				④	④
T	⑤					⑤		
T'		⑧	⑧	⑥	⑦		⑧	⑧
F	⑨					⑩		

3.1.2 LL(1)程序

- 语法栈SYN，语义栈SEM，四元式区QT
- 具体实现:
  - 当产生式(逆序)压栈时，动作符号也不例外;
  - 当动作符号位于栈顶时，执行之;
- 伪代码:

```
if syn.peek() == 'geq':
    syn.pop()
    s1 = sem.peek()
    sem.pop()
    s2 = sem.peek()
    sem.pop()
    s3 = 't'+str(j)
    sem.push(s3)
    j = j+1
    s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
    qt.push(s4)
    continue
    ...

if oper == 1:
    status.push(1)
    syn.pop()
    syn.push('E1')
    syn.push('T')
elif oper == 2:
    status.push(2)
    syn.pop()
    syn.push('E1')
    syn.push('geq+')
    syn.push('T')
    syn.push('+')
    ...
```

四 实验代码

4.1 LL(1)翻译法实现算术表达式四元式翻译

- 部分核心代码:

```
for line in exps:
    syn = Stack()
    syn.push('E')
    status = Stack()
    status.push(0)
    sem = Stack()
    qt = Stack()

    while syn.size()!=0:
        token = line[i]
        mark = token
        regular = r'\d+\.\d+|\d+|\w+(\w|\d)*'
        t = re.fullmatch(regular, token)
        if t != None:
            mark = 'id'

        if syn.peek() == token:
            syn.pop()
            i = i+1
```

```
            continue

        if syn.peek() == 'geq+':
            syn.pop()
            s1 = sem.peek()
            sem.pop()
            s2 = sem.peek()
            sem.pop()
            s3 = 't'+str(j)
            sem.push(s3)
            j = j+1
            s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
            qt.push(s4)
            continue
        ...

        if row.get(syn.peek()) == None:
            print("INCORRECR EXPRESSION!!")
            break
        oper = table[row[syn.peek()]][col[mark]]
        if oper == 1:
            status.push(1)
            syn.pop()
            syn.push('E1')
            syn.push('T')
        elif oper == 2:
            status.push(2)
            syn.pop()
            syn.push('E1')
            syn.push('geq+')
            syn.push('T')
            syn.push('+')
        ...
```

五 实验结果

5.1 LL(1)翻译法实验结果

测试文件为$tokens.txt$，内容是算术表达式的$tokens$。

读取文件$tokens.txt$，语义分析器直接输出分析结果：

- 翻译算术表达式：$98.0 + 32.44 - 0.4\#$：

```
['98.0', '+', '32.44', '-', '0.4', '#']
SYN                                                 TOKEN     SEM                 STATUS
['E']                                               98.0      []                  0
['E1', 'T']                                         98.0      []                  1
['E1', 'T1', 'F']                                   98.0      []                  5
['E1', 'T1', 'push', '98.0']                        98.0      []                  9
['E1', 'T1', 'push']                                +         []                  9
['E1', 'T1']                                        +         ['98.0']            9
['E1']                                              +         ['98.0']            8
['E1', 'geq+', 'T', '+']                            +         ['98.0']            2
['E1', 'geq+', 'T']                                 32.44     ['98.0']            2
['E1', 'geq+', 'T1', 'F']                           32.44     ['98.0']            5
['E1', 'geq+', 'T1', 'push', '32.44']               32.44     ['98.0']            9
['E1', 'geq+', 'T1', 'push']                        -         ['98.0']            9
['E1', 'geq+', 'T1']                                -         ['98.0', '32.44']   9
['E1', 'geq+']                                      -         ['98.0', '32.44']   8
['E1']                                              -         ['t0']              8
['E1', 'geq-', 'T', '-']                            -         ['t0']              3
['E1', 'geq-', 'T']                                 0.4       ['t0']              3
['E1', 'geq-', 'T1', 'F']                           0.4       ['t0']              5
['E1', 'geq-', 'T1', 'push', '0.4']                 0.4       ['t0']              9
['E1', 'geq-', 'T1', 'push']                        #         ['t0']              9
['E1', 'geq-', 'T1']                                #         ['t0', '0.4']       9
['E1', 'geq-']                                      #         ['t0', '0.4']       8
['E1']                                              #         ['t1']              8
CORRECT EXPRESSION!!
The result is:  ['(+,98.0,32.44,t0)', '(-,t0,0.4,t1)']
```

- 分析：第一行代表需要分析的tokens，SYN列为语法栈，可以看到翻译过程中，程序能够根据LL(1)分析表进行状态的转化（文法中表达式的选择），这一点也可以从最后一列的状态看出，其最终结果也正确输出；

- 翻译算术表达式：$123/((a*b)-5\#$

```
['123', '/', '(', '(', 'a', '*', 'b', ')', '-', '5', '#']
SYN                                                                 TOKEN     SEM                STATUS
['E']                                                               123       []                 0
['E1', 'T']                                                         123       []                 1
['E1', 'T1', 'F']                                                   123       []                 5
['E1', 'T1', 'push', '123']                                         123       []                 9
['E1', 'T1', 'push']                                                /         []                 9
['E1', 'T1']                                                        /         ['123']            9
['E1', 'T1', 'geq/', 'F', '/']                                      /         ['123']            7
['E1', 'T1', 'geq/', 'F']                                           (         ['123']            7
['E1', 'T1', 'geq/', ')', 'E', '(']                                 (         ['123']            10
['E1', 'T1', 'geq/', ')', 'E']                                      (         ['123']            10
['E1', 'T1', 'geq/', ')', 'E1', 'T']                                (         ['123']            1
['E1', 'T1', 'geq/', ')', 'E1', 'T1', 'F']                          (         ['123']            5
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E', '(']                (         ['123']            10
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E']                     a         ['123']            10
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T']               a         ['123']            1
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'F']         a         ['123']            5
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push', 'a'] a         ['123']            9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push']      *         ['123']            9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1']              *         ['123', 'a']       9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'F', '*'] *    ['123', 'a']       6
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'F']  b        ['123', 'a']       6
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'push', 'b'] b ['123', 'a']       9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*', 'push'] )      ['123', 'a']       9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq*']      )         ['123', 'a', 'b']  9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1', 'T1']              )         ['123', 't0']      9
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')', 'E1']                    )         ['123', 't0']      8
['E1', 'T1', 'geq/', ')', 'E1', 'T1', ')']                          )         ['123', 't0']      4
['E1', 'T1', 'geq/', ')', 'E1', 'T1']                               -         ['123', 't0']      4
['E1', 'T1', 'geq/', ')', 'E1']                                     -         ['123', 't0']      8
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T', '-']                   -         ['123', 't0']      3
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T']                        5         ['123', 't0']      3
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1', 'F']                  5         ['123', 't0']      5
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1', 'push', '5']          5         ['123', 't0']      9
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1', 'push']               #         ['123', 't0']      9
['E1', 'T1', 'geq/', ')', 'E1', 'geq-', 'T1']                       #         ['123', 't0', '5'] 9
['E1', 'T1', 'geq/', ')', 'E1', 'geq-']                             #         ['123', 't0', '5'] 8
['E1', 'T1', 'geq/', ')', 'E1']                                     #         ['123', 't1']      8
['E1', 'T1', 'geq/', ')']                                           #         ['123', 't1']      4
INCORRECR EXPRESSION!!
The result is:  ['(*,a,b,t0)', '(-,t0,5,t1)']
```

- 分析：该算术表达式式不正确的，其圆括号没有匹配上，程序最终也输出“INCORRECT EXPRESSION”提示该表达式是不正确的，前面$(a*b)$，与$(a*b)-5$的部分是正确输出其四元式的表达，该语义分析在翻译遇到错误时能够正确判断并退出。

六 实验心得

本次实验中，主要是实现语义分析器的功能，在此之前需要熟悉语法分析的LL(1)和逆波兰式，结合之前所学的语法分析器和词法分析器，根据所得token进行分析。

本次主要实现LL(1)分析表的逻辑，难度不高。通过设置语义栈和语法栈，token来确定当前状态和下一步的状态转化，对于**PUSH**和**GEQ**的部分则需要做出相应的操作，**PUSH**需要把id存入语法栈中，**GEQ**则需要生成对应的四元式，和语法栈的更新。在实验过程中需要注意的是按照产生式压栈，即逆序分析。

附录A

A.1 LL(1)翻译文法实现代码：

```
from email.policy import default
import re
#         i, +, -, *, /, (, ), #
table = [[1, 0, 0, 0, 0, 1, 0, 0],
         [0, 2, 3, 0, 0, 0, 4, 4],
         [5, 0, 0, 0, 0, 5, 0, 0],
         [0, 8, 8, 6, 7, 0, 8, 8],
         [9, 0, 0, 0, 0,10, 0, 0],
         [11,11,11,11,11,11,11,11]]


row = { 'E' : 0 , 'E1': 1 , 'T' : 2 , 'T1': 3, 'F' : 4 , 'push' : 5}
col = { 'id': 0 , '+' : 1 , '-' : 2 , '*' : 3, '/' : 4 ,
        '(' : 5 , ')' : 6 , '#' : 7 }


exps = []

class Stack:          # 定义一个栈
    def __init__(self):    # 初始化栈为空列表
        self.items = []

    def isempty(self):
        return self.items == []

    def push(self,item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)

    def all(self):
        return self.items

def get_tokens(filepath):
    file = open(filepath, 'r')
    line = file.readlines()
    exp = []
    for i in line:
        i = i.strip()
        exp.append(i)
        if i == '#':
            exps.append(exp)
            exp = []

```

```
52
53  get_tokens("tokens.txt")
54
55
56  for line in exps:
57      print(line)
58
59      syn = Stack()
60      syn.push('E')
61      status = Stack()
62      status.push(0)
63      sem = Stack()
64      qt = Stack()
65      i = 0
66
67      print('%-80s%-10s%-20s%-20s'%('SYN', 'TOKEN', 'SEM', 'STATUS'))
68
69      j = 0
70
71      while syn.size()!=0:
72
73
74          token = line[i]
75          print('%-80s%-10s%-20s%-20s'%(syn.all(), token, sem.all(),
    status.peek()))
76
77          mark = token
78          regular = r'\d+\.\d+|\d+|\w+(\w|\d)*'
79          t = re.fullmatch(regular, token)
80          if t != None:
81              mark = 'id'
82
83          if syn.peek() == token:
84              syn.pop()
85              i = i+1
86              continue
87
88          if syn.peek() == 'geq+':
89              syn.pop()
90              s1 = sem.peek()
91              sem.pop()
92              s2 = sem.peek()
93              sem.pop()
94              s3 = 't'+str(j)
95              sem.push(s3)
96              j = j+1
97              s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
98              qt.push(s4)
99              continue
100         elif syn.peek() == 'geq-':
101             syn.pop()
102             s1 = sem.peek()
103             sem.pop()
104             s2 = sem.peek()
105             sem.pop()
106             s3 = 't'+str(j)
107             sem.push(s3)
108             j = j+1
```

```
            s4 = '(' + '-' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
            qt.push(s4)
            continue
        elif syn.peek() == 'geq*':
            syn.pop()
            s1 = sem.peek()
            sem.pop()
            s2 = sem.peek()
            sem.pop()
            s3 = 't'+str(j)
            sem.push(s3)
            j = j+1
            s4 = '(' + '*' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
            qt.push(s4)
            continue
        elif syn.peek() == 'geq/':
            syn.pop()
            s1 = sem.peek()
            sem.pop()
            s2 = sem.peek()
            sem.pop()
            s3 = 't'+str(j)
            sem.push(s3)
            j = j+1
            s4 = '(' + '/' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
            qt.push(s4)
            continue

        if row.get(syn.peek()) == None:
            print("INCORRECR EXPRESSION!!")
            break

        oper = table[row[syn.peek()]][col[mark]]

        if oper == 1:
            status.push(1)
            syn.pop()
            syn.push('E1')
            syn.push('T')
        elif oper == 2:
            status.push(2)
            syn.pop()
            syn.push('E1')
            syn.push('geq+')
            syn.push('T')
            syn.push('+')
        elif oper == 3:
            status.push(3)
            syn.pop()
            syn.push('E1')
            syn.push('geq-')
            syn.push('T')
            syn.push('-')
        elif oper == 4:
            status.push(4)
            syn.pop()
        elif oper == 5:
            status.push(5)
```

```
                syn.pop()
                syn.push('T1')
                syn.push('F')
            elif oper == 6:
                status.push(6)
                syn.pop()
                syn.push('T1')
                syn.push('geq*')
                syn.push('F')
                syn.push('*')
            elif oper == 7:
                status.push(7)
                syn.pop()
                syn.push('T1')
                syn.push('geq/')
                syn.push('F')
                syn.push('/')
            elif oper == 8:
                status.push(8)
                syn.pop()
            elif oper == 9:
                status.push(9)
                syn.pop()
                syn.push('push')
                syn.push(token)
            elif oper == 10:
                status.push(10)
                syn.pop()
                syn.push(')')
                syn.push('E')
                syn.push('(')
            elif oper == 11:
                syn.pop()
                sem.push(line[i-1])
            elif oper == 0:
                print("ERROR!")
                break
        if syn.size() == 0:
            print("CORRECT EXPRESSION!!")
        print("The result is: " , qt.all())
        print('')


```