**课程编号：C0801003030**

**编译方法实验报告**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓名** | **张文祥** | **学号** | | **20175088** |
| **班级** |  | **指导教师** | | **刘洪娟** |
| **实验名称** | **扫描器设计及中间代码生成器设计** | | | |
| **开设学期** | **2019-2020春季学期** | | | |
| **开设时间** | **第4周和第8周** | | | |
| **报告日期** |  | | | |
| **评定成绩** |  | | **评定人** | **刘洪娟** |
| **评定日期** | **2020年4月24日** |

**东北大学软件学院**

1. 实验目的

熟悉并实现一个扫描器(词法分析程序)

1. 实验内容
2. 设计扫描器的有限自动机（识别器）
3. 设计翻译，生成Token的算法
4. 编写代码上机调试运行通过。
   1. 输入—源程序文件或源程序
   2. 输出—相应的token序列；

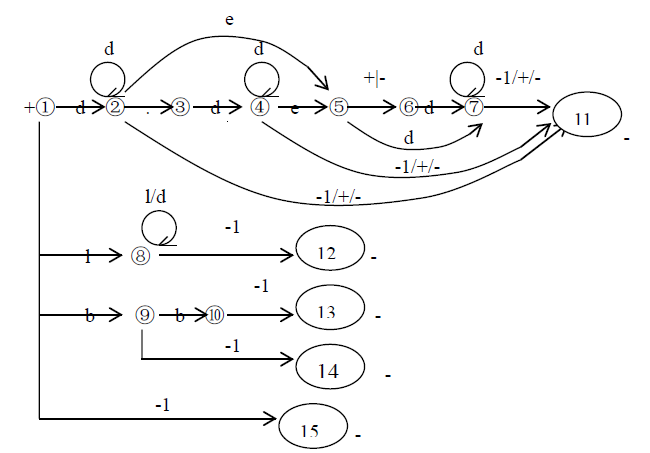
关键字表和界符表

符号表和常数表

1. 实验原理及基本步骤

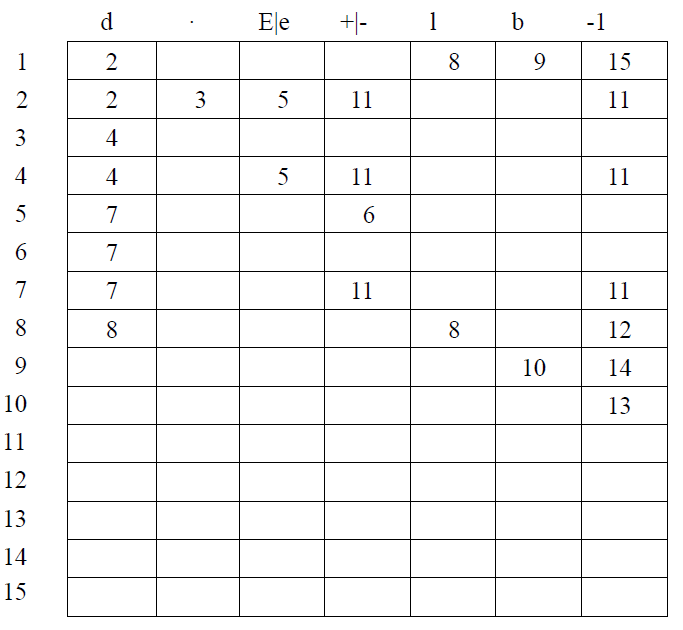
构造有限自动机作为扫描器，可以从源程序中识别出数字和单词，还有界符，根据每个单词的最终识别状态来判断单词类型（翻译器），将每个单词存储在对应的符号表中（用户变量表，常量表），并将单词的类型，和单词在所在的符号表中的索引存在Token数组里。

1. 有限自动机的状态转换图



其中：d为数字，l为字母，b为界符，-1代表其它符号（如在状态8处遇到了非字母或数字的其它符号，会变换到状态12）

1. 状态转换矩阵



1. 关键字和界符表

|  |  |
| --- | --- |
| Program | ; |
| Begin | : |
| End | ( |
| Var | ) |
| While | , |
| Do | := |
| Repeat | + |
| Until | - |
| For | \* |
| To | / |
| If | > |
| Then | >= |
| Else | == |
|  | < |
|  | <= |

1. 数据结构设计

(1)状态转换矩阵

int aut[10][7]={ 2, 0, 0, 0, 8, 9, 15,

2, 3, 5,11, 0, 0, 11,

4, 0, 0, 0, 0, 0, 0,

4, 0, 5,11, 0, 0, 11,

7, 0, 0, 6, 0, 0, 0,

7, 0, 0, 0, 0, 0, 0,

7, 0, 0,11, 0, 0, 11,

8, 0, 0, 0, 8, 0, 12,

0, 0, 0, 0, 0, 10, 14,

0, 0, 0, 0, 0, 0, 13};

(2)关键字表

char keywords[30][12]={“program”,”begin”,”end”,”var”,”while”,”do”,

”repeat”,”until”,”for”,”to”,”if”,”then”,”else”, “;”, ”:”, ”(“, ”)”, ”,”, ”:=”, ”+”, ”-“, ”\*”, ”/”,

”>”, ”>=”, ”==”, “<”, “<=”};

(3)符号表

char ID[50][12]; //表中存有源程序中的标识符

(4)常数表

float C[20];

(5)其他变量

struct token {

int code;

int value

};//Token结构

struct token tok[100]; //Token数组

int s; //当前状态

int n,p,m,e,t; //尾数值，指数值，小数位数，指数符号，类型

float num; //常数值

char w[50]; //源程序缓冲区

int i; //源程序指针,当前字符为w[i]

char strTOKEN[12]; //当前已经识别出的单词

(6)语义动作

q1：n=m=p=t=0; e=1; num=0; 其它变量初始化；

q2：n=10\*n+(w[i]);

q3：t=1;

q4：n=10\*n+( w[i]); m++;

q5：t=1;

q6：if ‘-‘ then e=-1;

q7：p=10\*p+(w[i]);

q8：将w[i]中的符号拼接到strTOKEN的尾部；

q9：将w[i]中的符号拼接到strTOKEN的尾部；

q10：将w[i]中的符号拼接到strTOKEN的尾部；

五、关键代码分析（带注释）及运行结果

(1)关键代码分析

识别单词或界符或常数

//开始处理一个单词

s=1;

//当s=0时表示当前状态s遇到当前单词状态为0，也就是无效状态

while (s!=0)

{

//根据当前的状态执行当前状态所对应的语义

act(s);

//此时表示已经识别完毕一个关键字或界符或常数

if (s>=11 && s<=14)

break;

i++;

//状态变换

s=find(s,w[i]);

}

每种状态语义的定义

void act(int s)

{

int code;

switch (s)

{

case 1:n=0;m=0;p=0;t=0;e=1;num=0;i\_str=0;

strTOKEN[i\_str]='\0'; //其它变量初始化

break;

case 2:n=10\*n+w[i]-48; //根据有限自动机 2-7为常数识别过程

break;

case 3:t=1;

break;

case 4:n=10\*n+w[i]-48; m++;

break;

case 5:t=1;

break;

case 6:if (w[i]=='-') e=-1;

break;

case 7:p=10\*p+w[i]-48;

break;

case 8:strTOKEN[i\_str++]=w[i]; //将ch中的符号拼接到strTOKEN的尾部；

break;

case 9:strTOKEN[i\_str++]=w[i]; //将ch中的符号拼接到strTOKEN的尾部；

break;

case 10:strTOKEN[i\_str++]=w[i]; //将ch中的符号拼接到strTOKEN的尾部；

break;

case 11:num=n\*pow(10,e\*p-m); //计算常数值

tok[i\_token].code=2; tok[i\_token++].value=InsertConst(num); //生成常数Token

num\_token++;

break;

case 12:strTOKEN[i\_str]='\0';

code=Reserve(strTOKEN); //查关键字表

if (code) //关键字

{ tok[i\_token].code=code; tok[i\_token++].value=0; } //生成关键字Token

else

{ tok[i\_token].code=1;

tok[i\_token++].value=InsertID(strTOKEN);

} //生成标识符Token

num\_token++;

break;

case 13:strTOKEN[i\_str]='\0';

code=Reserve(strTOKEN); //查界符表

if (code)

{ tok[i\_token].code=code; tok[i\_token++].value=0; } //生成界符Token

else

{

strTOKEN[strlen(strTOKEN)-1]='\0'; //单界符

i--;

code=Reserve(strTOKEN); //查界符表

tok[i\_token].code=code; tok[i\_token++].value=0; //生成界符Token

}

num\_token++;

break;

case 14:strTOKEN[i\_str]='\0';

code=Reserve(strTOKEN); //查界符表

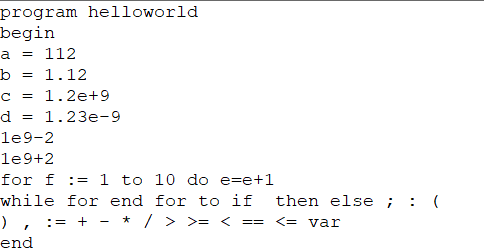
tok[i\_token].code=code; tok[i\_token++].value=0; //生成界符Token

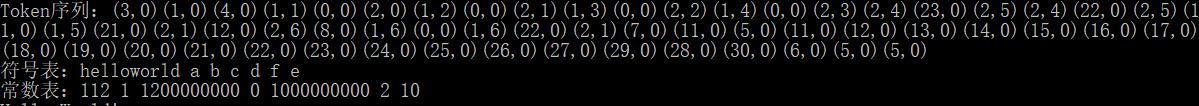
num\_token++;

break;

}

(2)测试用例及结果：





六、总结与分析

1. 扫描器的任务是什么？

词法分析器又称扫描器，任务有：1，识别单词，从用户的源程序中把单词分离出来。2，翻译单词，把单词转换成机内表示，便于后续处理

(2) 扫描器，识别器，翻译器三者之间的关系是怎样的？

扫描器是识别器和翻译器实现的

(3) 为什么说有限自动机是词法分析的基础

因为词法分析包括：识别-识别单词的有限自动机，和翻译-根据有限自动机识别出的对象，完成从单词串到单词的token串的翻译。因此只有有限自动机的存在，才会有词法分析。所以有限自动机是词法分析的基础

实验二

一 实验目的

熟悉算术表达式的语法分析与中间代码生成原理，实现算数表达式的中间代码生成器。

二、 实验内容

(1) 设计语法制导翻译生成表达式的四元式的算法；

(2) 编写代码并上机调试运行通过。

·输入——算术表达式

·输出——语法分析结果

相应的四元式序列

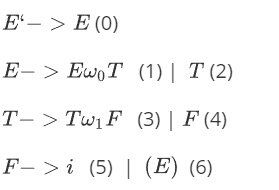
(3) 利用递归子程序法或LL(1)分析法或LR()分析法的属性翻译文法，并根据这些属性翻译文法，使用扩展的语法分析器实现语法制导翻译。

三、 实验原理和基本步骤

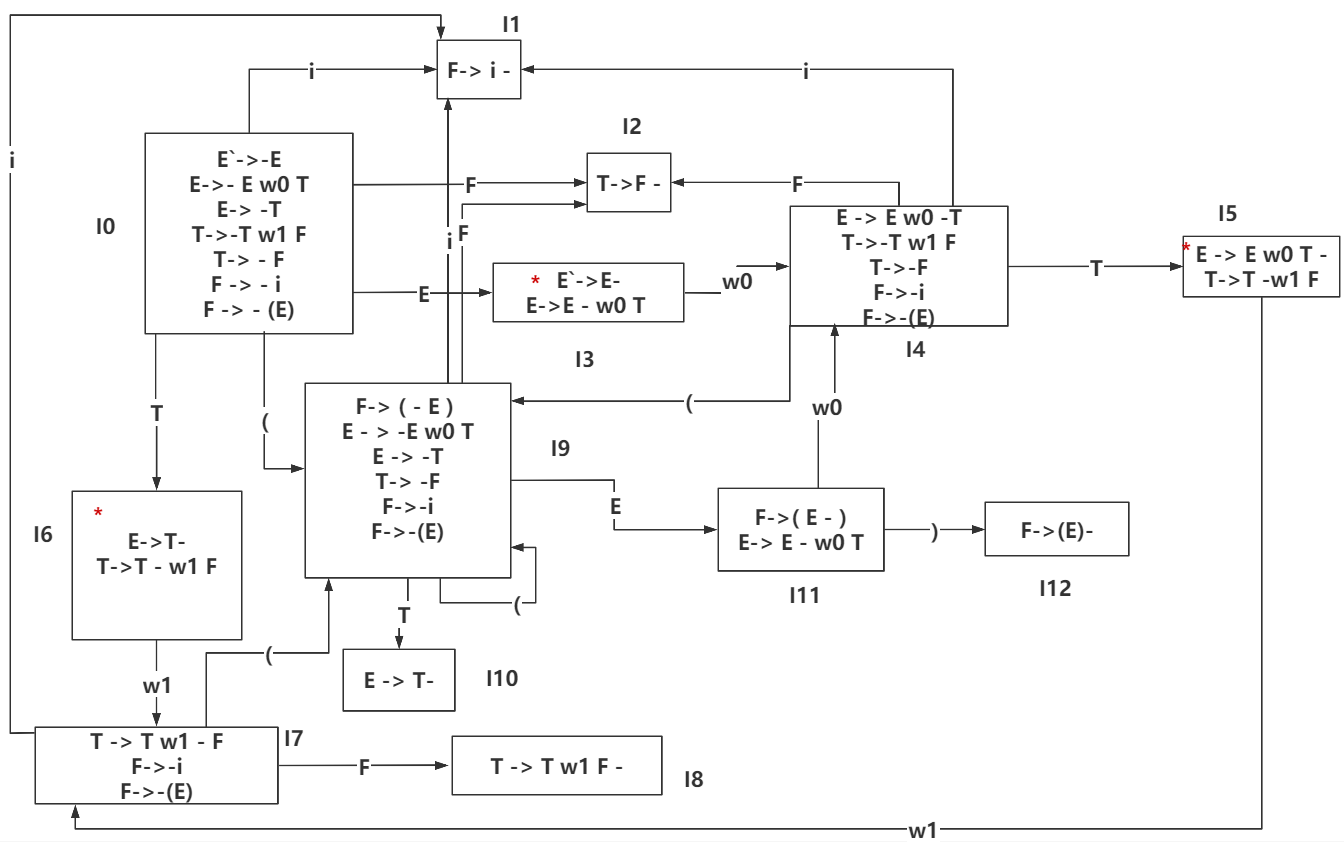
使用SLR(1) 分析法来识别算术表达式，首先需要判断文法是否是LR(0)文法（句柄识别器中无状态冲突），如果不是，则可以考虑SLR（1）分析法，经判断得知该文法为SLR(1)文法，利用LR控制器和LR分析表判断语句是否为算术表达式，并且在识别过程中使用算符栈和语义栈生成表达式的四元式，将其存进四元式数组

(1)构造LR（）分析表

首先将文法变换成如下形式

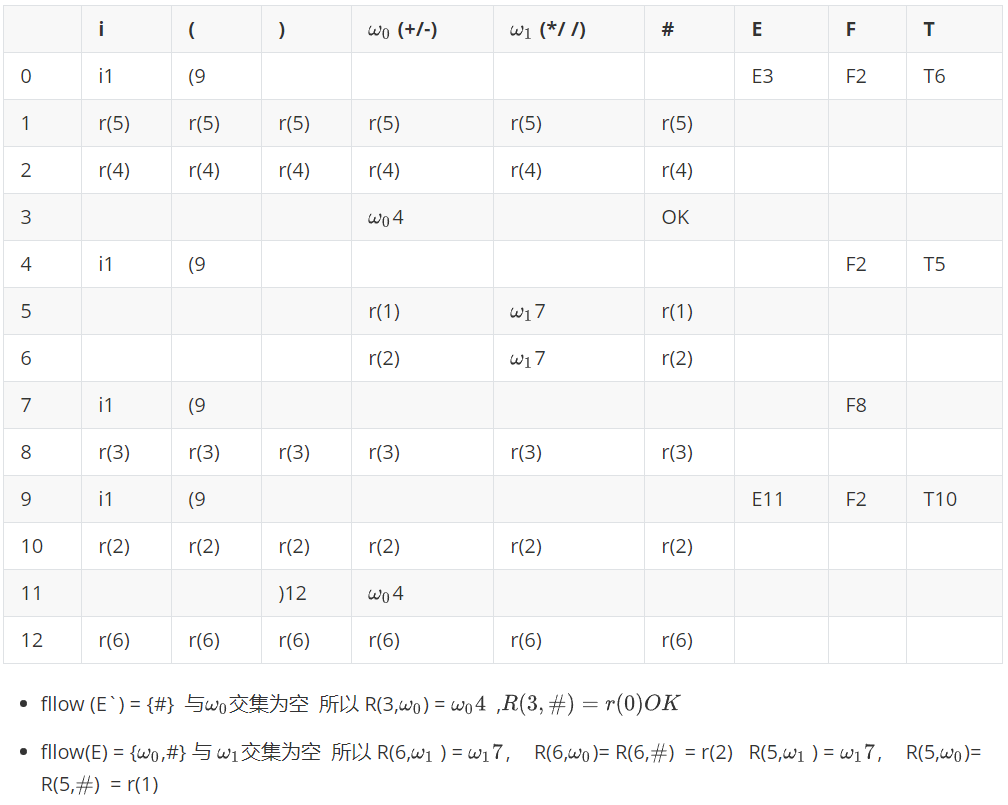


(2)画出 句柄识别器



红色部分为发生冲突的状态，所以该文法为SLR(1)文法，采用SLR(1)分析法

(3)构造LR()分析表



(4)构造控制程序

步骤一 初始将#0压栈

步骤二 读取单词

步骤三 利用栈顶的状态编码（k）和当前取到的单词（w）去分析表中取结果R（k，w）

R为空 报错

R = ok 结束状态 程序结束

R = wi 表示移进状态 将wi进栈 进入步骤二

R = r(j) 表示规约状态，利用第j个产生式进行规约，将产生式右侧连同状态编码一同出栈，将产生式左侧压栈（同时连同左侧的状态编码一同压栈），进入步骤三

四、数据结构设计

char R[13][9] ={{'1','9',' ',' ',' ',' ','3','2','6'},

{'E','E','E','E','E','E',' ',' ',' '},

{'D','D','D','D','D','D',' ',' ',' '},

{' ',' ',' ','4',' ','Y',' ',' ',' '},

{'1','9',' ',' ',' ',' ',' ','2','5'},

{' ',' ','A','A','7','A',' ',' ',' '},

{' ',' ','B','B','7','B',' ',' ',' '},

{'1','9',' ',' ',' ',' ',' ','8',' '},

{'C','C','C','C','C','C',' ',' ',' '},

{'1','9',' ',' ',' ',' ',':','2','6'},

{' ',' ',';','4',' ',' ',' ',' ',' '},

{'F','F','F','F','F','F',' ',' ',' '}};//句柄识别器

stack<char> SYN; //算符栈

stack<char> SEM; //语意栈

vector<string> quats; //产生的四元式

stack<char> analyse\_stack; //分析栈

char var;//生成临时变量

五、关键代码分析及结果

(1) 关键代码分析

控制器实现

bool build(string &s)

{

int i;

for(i =0; i < s.size();)

{

char cur\_word = s[i];

char after\_change = state\_transfer(analyse\_stack.top(),cur\_word);

if(after\_change=='Y') return true;

if(after\_change==' '){ return false; }

//移进状态，将当前单词和状态编码都压进栈,并读取下一个单词

else if(after\_change>='0' && after\_change<=';')

{

analyse\_stack.push(cur\_word);

analyse\_stack.push(after\_change);

i++;

}

//规约状态，将待规约的表达式归约，不读取单词

else if(after\_change>='A' && after\_change<='F')

{

reduce(after\_change);

}

}

return false;

}

规约操作

void reduce(char expression\_num)

{

char ch\_SYN;

char ch\_SEM;

char next\_state;

switch (expression\_num){

case 'A':

analyse\_stack.pop();

analyse\_stack.pop();//出栈 T

analyse\_stack.pop();

ch\_SYN = analyse\_stack.top();

SYN.push(ch\_SYN);

if(SEM.size()>=2 && SYN.size()>=1) generate\_quat();

analyse\_stack.pop(); //出栈w0

analyse\_stack.pop();

analyse\_stack.pop(); //出栈E

next\_state = state\_transfer(analyse\_stack.top(),'E');

analyse\_stack.push('E');

analyse\_stack.push(next\_state); // 将 产生式右侧 符号入栈

break;

case 'B':

analyse\_stack.pop();

analyse\_stack.pop(); //出栈 T

next\_state = state\_transfer(analyse\_stack.top(),'E');

analyse\_stack.push('E');

analyse\_stack.push(next\_state);//进栈E

break;

case 'C':

analyse\_stack.pop();

analyse\_stack.pop(); //出栈F

analyse\_stack.pop();

ch\_SYN = analyse\_stack.top();

SYN.push(ch\_SYN);

if(SEM.size()>=2 && SYN.size()>=1) generate\_quat();

analyse\_stack.pop(); //出栈 w1

analyse\_stack.pop();

analyse\_stack.pop(); //出栈 T

next\_state = state\_transfer(analyse\_stack.top(),'T');

analyse\_stack.push('T');

analyse\_stack.push(next\_state); //进栈T

break;

case 'D':

analyse\_stack.pop();

analyse\_stack.pop(); //出栈 F

next\_state = state\_transfer(analyse\_stack.top(),'T');

analyse\_stack.push('T');

analyse\_stack.push(next\_state); //进栈T

break;

case 'E':

analyse\_stack.pop();

ch\_SEM = analyse\_stack.top();

SEM.push(ch\_SEM);

if(SEM.size()>=2 && SYN.size()>=1) generate\_quat();

analyse\_stack.pop(); //出栈 i

next\_state = state\_transfer(analyse\_stack.top(),'F');

analyse\_stack.push('F');

analyse\_stack.push(next\_state); //进栈F

break;

case 'F':

analyse\_stack.pop();

analyse\_stack.pop(); //出栈 ）

analyse\_stack.pop();

analyse\_stack.pop(); //出栈E

analyse\_stack.pop();

analyse\_stack.pop(); //出栈（

next\_state = state\_transfer(analyse\_stack.top(),'F');

analyse\_stack.push('F');

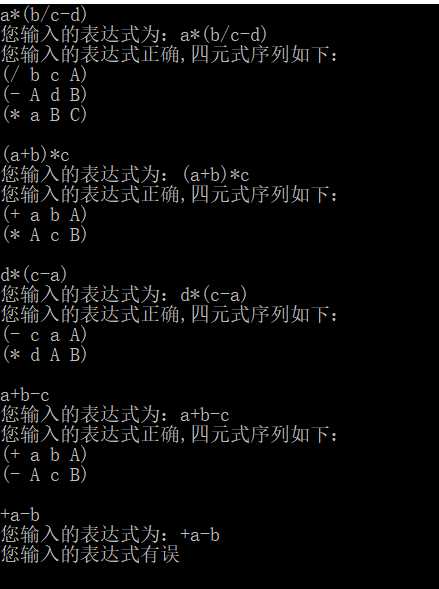
analyse\_stack.push(next\_state);

break;

}

}

(2) 测试结果

****

**六、总结与分析**

(1) 语法分析分为几类？其关键技术各是什么？

1，自顶向下法（推导法）

从开始符号出发，采用推导运算，试图自顶向下构造语法树

2，自底向上法（归约法）

从给定的符号串出发，采用归约运算，试图自底向上构造语法树

(2) 什么是递归下降子程序法，什么是LL(1)分析法？二者对文法各有什么要求？

递归下降子程序法：递归子程序法属于自顶向下的语法分析方法，故名递归下降法。要求文法是LL(1)文法

LL(1)分析法：LL(1)分析法是从左到右扫描(第一个L)，最左推导(第二个L)和之查看一个当前符号(括号中的1)之意；LL(1)分析法又称预测分析法，属于自顶向下确定性语法分析方法。要求文法是LL(1)文法。

(3) 比较LL(1)分析法和递归下降子程序法的异同。

相同点：要求文法都是LL(1)文法；都是自顶向下的分析方法;都通过分析下个字符来判断该进入哪个状态或调用哪个函数

不同点：LL(1)分析法先建立起预测分析表,通过对分析栈的不断操作(出栈，入栈)来进行;递归下降子程序法是通过函数间的函数调用来实现不同状态间的转换，并简化了代码

(4) 什么是语法制导翻译技术？其核心技术是什么？

语法制导翻译是在语法分析过程中，随着分析(推导或归约)的逐步进展，没事别出一个语法结构，根据文法的每个规则所对应的语义子程序进行翻译的方法；核心技术是构造属性翻译文法

(5) 表达式的四元式属性翻译文法如何设计？

假定： SEM—语义栈(属性传递，赋值场所)

QT[q] – 四元式区

G(E): E->T | E+T{GEQ(+)} | E-T{GEQ(-)}

T->F | T \* F{GEQ(\*)} | T/F{GEQ(/)}

F-> i {PUSH(i)} | (E)

其中：PUSH(i) – 压栈函数（把当前i压入语义栈）

GEQ(w) – 表达式四元式生成函数 生成一个四元式 QT[q]过程

1，t:=NEWT 申请临时变量函数

2，SEND(w,SEM[m-1],SEM[m],t)

3, POP; POP; PUS(t)

**成绩评价表格**：

|  |  |
| --- | --- |
| 考核标准 | 得分 |
| （1）正确理解和掌握实验所涉及的基本概念和原理；（10%）； |  |
| （2）按实验要求合理设计数据结构和程序结构（20%）； |  |
| （3）实验中要设计各种测试用例，考虑词法、语法和语义正确和不正确等各种情况（20%）； |  |
| （4）认真记录实验结果，对实验结果的分析准确（20%）； |  |
| （5）实验过程中，具有严谨、认真的学习态度，不做与实验内容无关的其他学习工作（10%）； |  |
| （6）实验2应具有一定的创新性（10%）； |  |
| （7）实验报告内容充实、格式符合规范（10%）。 |  |