SRL文法实例以其C语言代码介绍

10011603-2016302731-冯宇

首先是文法介绍，样例文法如下：

文法为:

0:S->E

1:E->E+E

2:E->E\*E

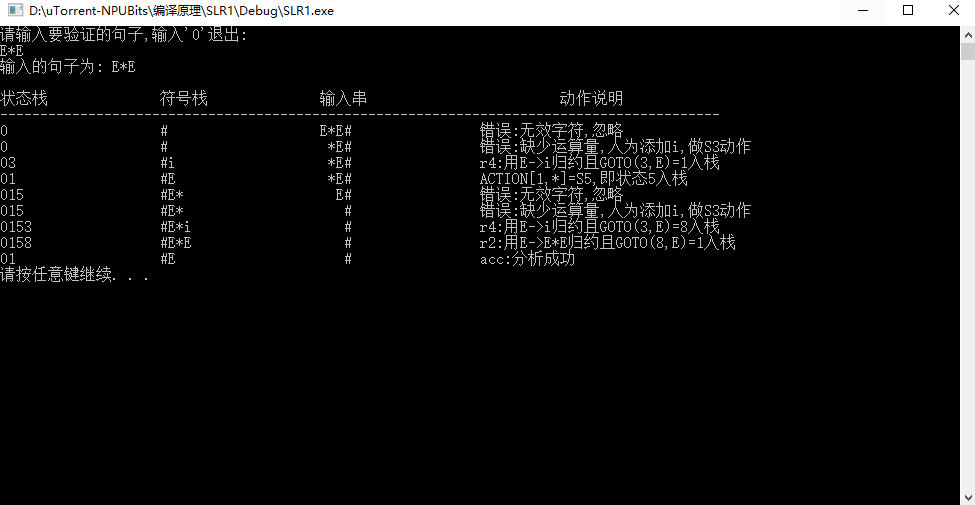
3:E->(E)

4:E->id

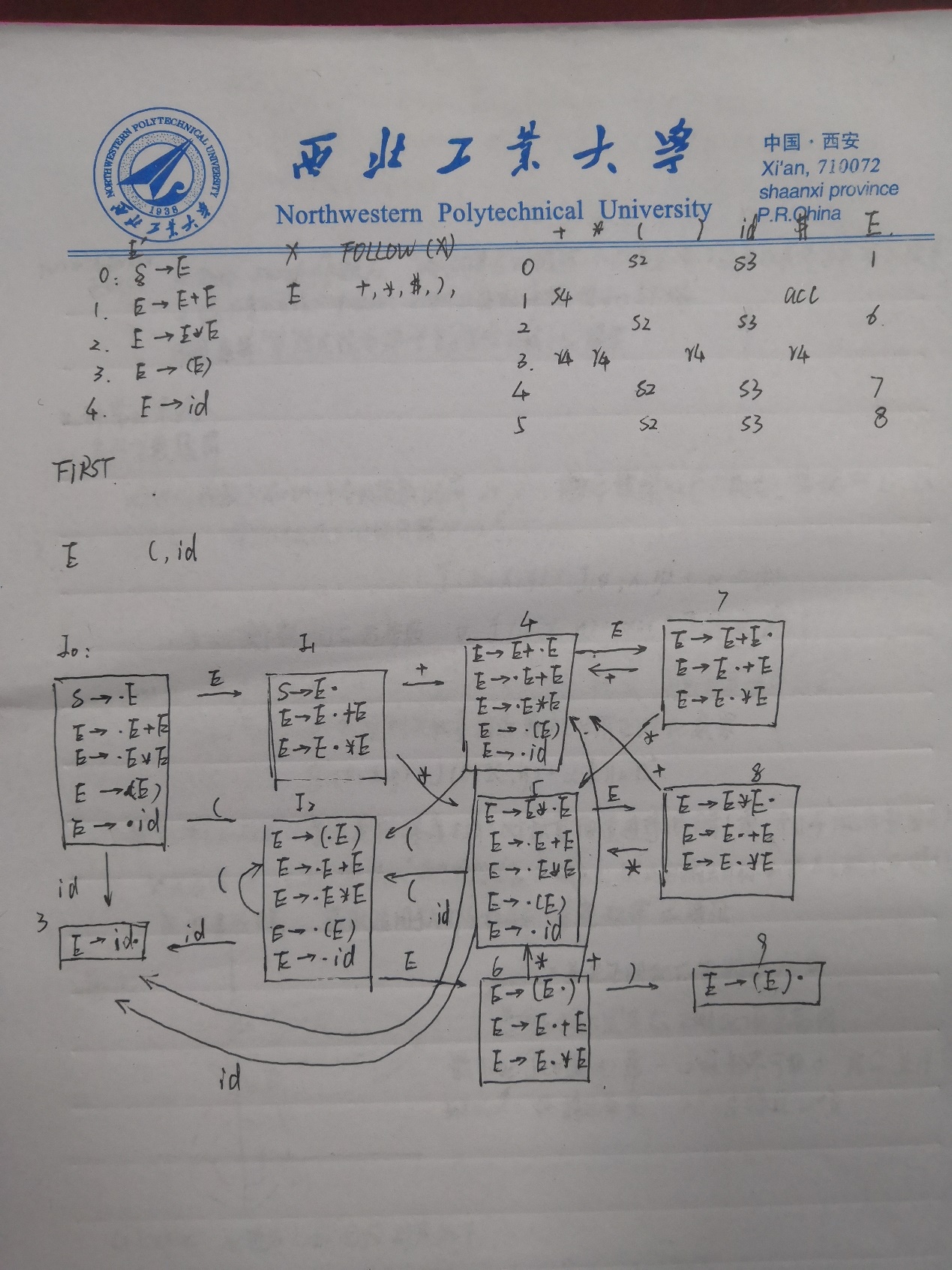
该程序实现的效果是：

运行时只需输入待验证的句子验证该语句是否符合该文法，同时展示出过程

效果图如下：



首先，在纸上用已知的SRL文法，将目标文法的FOLLOW集和FIRST集以及分析表画出来，最后结果如下：



从图中可以看出，该文法共有10个状态，对应的文法分析表也在草稿纸中展示。对于文法分析表中的ACTION部分，我们通过以下规则利用二维数组进行表示：

//ACTION表 小于100正数n代表移进状态n 大于100的整数n代表利用文法n-100归约，等于100代表acc 负数代表出错

int action\_tab[10][6]={

3,-1,-1,2,-2,-1,

-3,4,5,-3,-2,100,

3,-1,-1,2,-2,-1,

-3,104,104,-3,104,104,

3,-1,-1,2,-2,-1,

3,-1,-1,2,-2,-1,

-3,4,5,-3,9,-4,

-3,101,5,-3,101,101,

-3,102,102,-3,102,102,

-3,103,103,-3,103,103

};

GOTO操作中的状态转化部分采用二维数组进行记录

int goto\_tab[10][2]={

1,1,0,0,6,6,0,0,7,7,8,8,0,0,0,0

};

文章定义结构体：

typedef struct LNode

{

int status; //状态

char symbol; //符号

struct LNode \*next;

struct LNode \*prev;

}LNode,\*LinkList;

作为节点，并完成pop,push等函数

并采用带头结点的双向链表存储符号及状态

LinkList L; //带头结点的双向链表

LinkList tail; //记录链表的尾

程序运行过程：

程序通过main()函数直接进入菜单函数

void menu() //菜单函数

主要功能：

1.完成链表的初始化，并将空字符#压入头结点

2.界面提示信息以及输入待验证数据

3.将输入字符串的末尾\0后移一位，并加入空字符#作为结束标志

随后调用analyze()函数进行验证

核心代码：

while(thetext[ip]!='\0') //在句子末尾加入#

ip++;

thetext[ip]='#';

thetext[ip+1]='\0';

ip=0;

void analyze();函数：

主要功能：分析输入语句是否符合文法，是重点中的重点

void analyze() //分析

{

int result\_tab=0; //记录查表结果

int location\_tab=0; //记录在终结符表中的位置

int status\_new=0; //记录查询goto后获取的状态

int i=0,fre=0; //出入栈时用于循环

int flag=0; //判断ip是否下移(flag为1,代表归约,ip不++)

char character; //记录归约产生的非终结符

while(thetext[ip]!='\0') //句子没有结束

{

gettop();

location\_tab=search\_vt(thetext[ip]);//查找对应终结元素的下标，没有返回-1

flag=0;

if(location\_tab!=-1) //在终结符表中有这个元素

{

result\_tab=action\_tab[status\_top][location\_tab]; //获取ACTION表内容

if(result\_tab>=0) //无错误

{

if(result\_tab>100) //归约

{

output(2,result\_tab-101);//该函数用作处理规约和移进操作具体讲解在后面

fre=fa\_num[result\_tab-101]; //记录对应产生式右部元素数量

for(i=0;i<fre;i++) //产生式右部元素出栈

{

pop();

}

gettop();

character=Fa\_vn[result\_tab-101]; //获得产生式左部的非终结符 然后查goto表

i=search\_vn(character); //找到对应非终结元素下标

status\_new=goto\_tab[status\_top][i]; //在goto表中找到对应状态

push(status\_new,character); //产生式左部入栈

flag=1; //是规约操作的话，flag为1，下一步仍考虑该输入元素

}

else if(result\_tab<100) //移进

{

output(1,result\_tab);

push(result\_tab,thetext[ip]);

}

else

{

output(3,0);

}

}

else //错误处理

{

flag=0;

error\_handle(result\_tab);

}

}

else //在终结符表中没有这个元素

{

flag=0;

error\_handle(-5);

}

if(flag==0)//不是规约操作的话，flag保持为0，下一步考虑下一个输入元素

ip++;

}

//句子已经结束 只剩下'#'

}

主题思路大致如上图所示，接下来讲解下规约和移进操作的函数output（）

其中输入参数type为1对应移进操作，2对应规约操作，3对应分析成功

void output(int type,int remark) //输出函数

{

int i=0; //记录归约后非终结符下标

int count=0; //记录归约后状态栈栈顶符号

LinkList p;

out(); //打印空格是最后输出对应整齐

gettop();

if(type==1) //移进操作

{

cout<<"ACTION["<<status\_top<<','<<thetext[ip]<<"]=S"<<remark<<",即状态"<<remark<<"入栈";

}

else if(type==2) //规约操作

{

p=L->next;

i=fa\_num[remark];//对应产生式右部字符数量

for(count=0;count<i;count++)//查找规约后栈顶的状态

{

p=p->next;

}

count=p->status;//并记录

i=search\_vn(Fa\_vn[remark]); //归约后非终结符下标用于GOTO查表

cout<<"r"<<remark+1<<":用"<<Fa\_vn[remark]<<"->"<<Fa[remark]<<"归约且GOTO("<<status\_top<<','

<<Fa\_vn[remark]<<")="<<goto\_tab[count][i]<<"入栈";

}

else

{

cout<<"acc:分析成功";

}

cout<<endl;

}