编译原理实验指导书

适用专业：计算机应用

制 定 人：童旺宇

计算机科学与技术学院

前言

“编译原理”是一门研究设计和构造编译程序原理和方法的课程，是计算机各专业的一门重要专业基础课。编译原理这门课程蕴含着计算机学科中解决问题的思路、形式化问题和解决问题的方法，对应用软件和系统软件的设计与开发有一定的启发和指导作用。编译程序构造的原理和技术在软件工程、逆向工程、软件再工程、语言转换及其他领域中都有着广泛的应用。

通过本课程的实验教学，使学生加深对编译系统的结构、工作流程及编译程序各组成部分设计原理的理解，使他们能够掌握和应用常用的编译技术和方法，为今后从事应用软件和系统软件的开发打下一定的理论和实践基础。

编译原理实验指导书围绕着实验教学目标，详细阐述了各实验的原理和步骤。希望同学们能够充分利用实验条件，认真完成实验，从实验中得到应有的锻炼和培养。

实验要求

为了顺利完成编译原理课程实验，学生应做到：

（1） 熟练掌握一种高级程序设计语言。

（2） 实验前，认真学习教材以及实验指导书的相关内容，提前做好实验准备。

（3） 每次实验先分析后编程，在实验报告中应写明自己的编程思路和设计流程。

（4） 实验结束一周后提交实验报告。实验报告内容应包括：实验目的、实验内容、设计思路和流程框图，源程序（含注释）清单、测试结果以及实验总结。

（5） 遵守机房纪律，服从辅导教师指挥，爱护实验设备。

实验的验收将分为两个部分。第一部分是上机操作，随机抽查程序运行和即时提问；第二部分是提交书面的实验报告。此外杜绝抄袭现象，一经发现雷同，双方成绩均以0分计算。

目　录

实验 1 词法分析程序设计

【开发语言及实现平台或实验环境】

C/C++/C#

Microsoft Visual Studio 6.0/ Microsoft Visual Studio .NET 2003-2005

【实验目的】

（1）理解词法分析在编译程序中的作用

（2）加深对有穷自动机模型的理解

（3）掌握词法分析程序的实现方法和技术

【实验内容】

对一个简单语言的子集编制一个一遍扫描的词法分析程序。

【实验要求】

（1）待分析的简单语言的词法

1）关键字

begin if then while do end

2）运算符和界符

:= + - * / < <= > >= <> = ; ( ) #

3）其他单词是标识符(ID)和整形常数(NUM)，通过以下正规式定义：

ID=letter(letter|digit)*

NUM=digitdigit*

4）空格由空白、制表符和换行符组成。空格一般用来分隔 ID、NUM、运算符、界符和关键字，词法分析阶段通常被忽略。

（2）各种单词符号对应的种别编码

单词符号	种别码	单词符号	种别码
begin	1	:	17
if	2	:=	18
then	3	<	20
while	4	<>	21
do	5	<=	22
end	6	>	23
letter(letter|digit)*	10	>=	24
digitdigit*	11	=	25
+	13	;	26
-	14	(	27
*	15	)	28
/	16	#	0

（3）词法分析程序的功能

输入：所给文法的源程序字符串

输出：二元组（syn,token 或 sum）构成的序列。

syn 为单词种别码；

token 为存放的单词自身字符串；

sum 为整形常数。

例如：对源程序 begin x:=9;if x>0 then x:=2*x+1/3;end# 经词法分析后输出如下序列：（1，begin）(10,'x') (18,:=) (11,9) (26,;) (2,if)……

【实验步骤】

（1）根据图 1.1 构建主程序框架

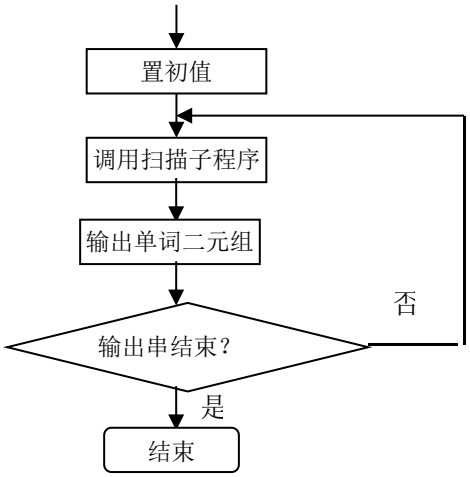


图 1.1 词法分析主程序示意图

代码提示：

```
main()
{
    p=0;
    printf("\n please input string:\n");
    do{
        输入源程序字符串，送到缓冲区 prog[p++]中
    }
   while(ch!='#');
    p=0;
    do
    {
        scanner();//调用扫描子程序
        switch(syn)
        {
            case 11:输出（数的二元组）；break;
            case –1:输出（错误）;break;
            default:输出（其他单词二元组）;
        }
    } while(syn!=0);
}
```

（2）关键字表置初值

关键字作为特殊标识符处理，把它们预先安排在一张表格中（关键字表），当扫描程序识别标识符时，查关键字表。如能查到匹配的单词，则为关键字，否则为一般标识符。

（3）编写扫描子程序

代码提示：

```
scanner()
{
    ……
    读下一个字符送入 ch；
    while(ch= =' ')      读下一个字符;
    if(ch 是字母或数字)
    {
        while((ch 是字母或数字))
        {
            ch=>token;
            读下一个字符;
        }
        token 与关键字表进行比较，确定 syn 的值；

    }
    else
        if(ch 是数字)
        {
            ……..
            syn=11;
        }
    else
        swith(ch)//其他字符情况
        {
            case'<':
                ……
            case'>':
                ……
            ……
            Default:syn=-1;
        }
}
```

（4）调试程序，验证输出结果。

【思考题】

（1）在编程过程中遇到了哪些问题，你是如何解决的。

（2）源程序若存在注释，如何实现词法分析，在现有程序基础上进行扩充。

【参考文献】

1. 胡伦骏、徐兰芳等，编译原理（第 2 版），电子工业出版社，246，2005.7

2. 王雷、刘志成等，编译原理课程设计，机械工业出版社，138，2005.3

实验 2 递归下降语法分析程序设计

【开发语言及实现平台或实验环境】

C/C++/C#

Microsoft Visual Studio 6.0/ Microsoft Visual Studio .NET 2003-2005

【实验目的】

（1）理解语法分析在编译程序中的作用，以及它与词法分析程序的关系

（2）加深对递归下降语法分析原理的理解

（3）掌握递归下降语法分析的实现方法

【实验内容】

编制一个递归下降分析程序，实现对词法分析程序提供的单词序列的语法检查和结构分析。

【实验要求】

（1）待分析的简单语言的词法同实验 1

（2）待分析的简单语言的语法

用扩充的 BNF 表示如下：

1）<程序>::=begin<语句串>end

2）<语句串>::=<语句>{;<语句>}

3）<语句>::=<赋值语句>

4）<赋值语句>::=ID:=<表达式>

5）<表达式>::=<项>{+<项>|-<项>}

6）<项>::=<因子>{*<因子>|/<因子>}

7）<因子>::=ID|NUM|(<表达式>)

（3）语法分析程序的功能

输入单词串以"#"结束，如果是文法正确的句子，输出成功信息；否则输出错误信息。

例如：

输入　begin a:=9; x:=2 * 3; b:=a + x end #

输出　success

输入　x:=a + b * c end #

输出　error

【实验步骤】

（1）根据图 2.1 递归下降分析程序示意图构建主程序框架

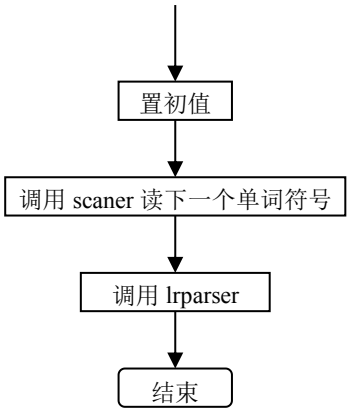


图 2.1 递归下降分析程序示意图

（2）编写各语法单位分析函数

1）编写语句串及语句分析函数

代码提示：

```
yucu()//语句串分析函数
    {
     调用 statement();//语句分析函数
    while(syn=26){
        读入下一个单词符号；
        调用 statement();
    }
    return;
}
statement（）
{
    if(syn=10){
        读入下一个单词符号；
        if(syn=18)
            {
            读入下一个单词符号；
            调用 expression 函数；//表达式分析函数
            }
            else{输出赋值号错误；kk=1//出错标记}
    }
    else{输出语句号错误；kk=1;}
    return;
}
```

2）编写表达式分析过程

3）编写项分析过程

4）编写因子分析过程

（3）调试程序，验证输出结果

【思考题】

（1） 你所编制的程序与实验 1 程序有何联系，如何应用实验 1。

（2） 将源程序放置在文本文件中，运用流操作实现对源程序的扫描和分解，编程实现。

【参考文献】

1．胡伦骏、徐兰芳等，编译原理（第 2 版），电子工业出版社，246，2005.7

2．王雷、刘志成等，编译原理课程设计，机械工业出版社，138，2005.3

实验 3 语义分析程序设计

【实验目的】

构造 LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解 LR（K）分析方法是严格的从左向右扫描，和自底向上的语法分析方法。

【实验内容】

对下列文法，用 LR（1）分析法对任意输入的符号串进行分析：

(1)S->E	**(5)T->F**
(2)E->E+T	**(6)F->(E)**
(3)E->T	**(7)F->i**
(4)T->T*F	

【设计思想】

(1)总控程序，也可以称为驱动程序。对所有的 LR 分析器总控程序都是相同的。

(2)分析表或分析函数，不同的文法分析表将不同，同一个文法采用的 LR 分析器不同时，分析表将不同，分析表又可以分为动作表（ACTION）和状态转换（GOTO）表两个部分，它们都可用二维数组表示。

(3)分析栈，包括文法符号栈和相应的状态栈，它们均是先进后出栈。

分析器的动作就是由栈顶状态和当前输入符号所决定。

- LR 分析器由三个部分组成：
- 其中:SP 为栈指针，S[i]为状态栈，X[i]为文法符号栈。状态转换表用 GOTO[i，X]=j 表示，规定当栈顶状态为 i，遇到当前文法符号为 X 时应转向状态 j，X 为终结符或非终结符。
- ACTION[i，a]规定了栈顶状态为 i 时遇到输入符号 a 应执行。动作有四种可能：

(1)移进：

action[i，a]= Sj：状态 j 移入到状态栈，把 a 移入到文法符号栈，其中 i,j 表示状态号。

(2)归约：

action[i，a]=rk：当在栈顶形成句柄时，则归约为相应的非终结符 A，即文法中有 A- B 的产生式，若 B 的长度为 R(即|B|=R)，则从状态栈和文法符号栈中自顶向下去掉 R 个符号，即栈指针 SP 减去 R，并把 A 移入文法符号栈内，j=GOTO[i,A]移进状态栈，其中 i 为修改指针后的栈顶状态。

(3)接受 acc:

当归约到文法符号栈中只剩文法的开始符号 S 时，并且输入符号串已结束即当前输入符是'#'，则为分析成功。

(4)报错:

当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，则报错，说明输入端不是该文法能接受的符号串。

【实验要求】

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

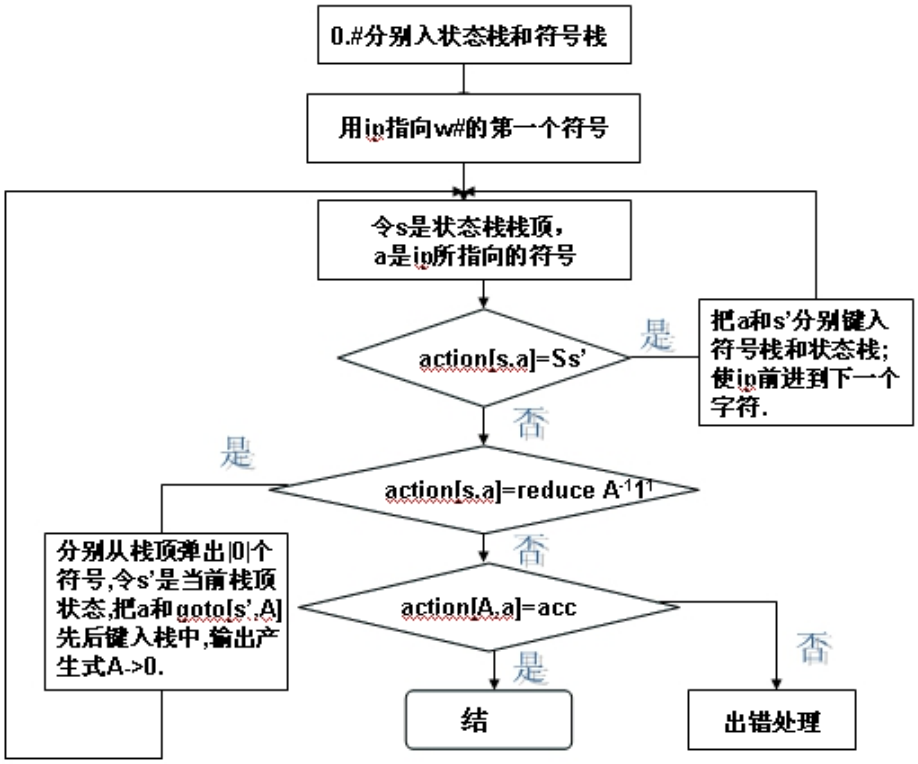
3、程序输入/输出实例：

输入一以#结束的符号串(包括+—*/（）i#)：在此位置输入符号串

输出过程如下：

步骤	状态栈	符号栈	剩余输入串	动 作
1	0	#	i+i*i#	移进

【流程图】



【源代码】

```
#include<stdio.h>
#include<stdlib.h>

int Action[12][6]=
{
105,0,0,104,0,0,
0,106,0,0,0,-1,
0,52,107,0,52,52,
0,54,54,0,54,54,
105,0,0,104,0,0,
0,56,56,0,56,56,
105,0,0,104,0,0,
105,0,0,104,0,0,
0,106,0,0,111,0,
0,51,107,0,51,51,
0,53,53,0,53,53,
0,55,55,0,55,55};
int Goto[12][3]=
```

```
{
	1,2,3,
	0,0,0,
	0,0,0,
	0,0,0,
	8,2,3,
	0,0,0,
	0,9,3,
	0,0,10,
	0,0,0,
	0,0,0,
	0,0,0,
	0,0,0
};
char Grammar[20][10]={'\0'};
char VT[10],VN[10];
char AVT[6]={'i','+','*','(',')','#'};
char GVN[3]={'E','T','F'};
int vnNum,vtNum,stateNum=12;
int VNum[10];
int grammarNum;

typedef struct{
	char *base;
	char *top;
}SymbolStack;

typedef struct{
	int *base;
	int *top;
}StateStack;


StateStack state;
SymbolStack symbol;

int ScanGrammar()
{
	FILE *fp=fopen("SLR 文法.txt","r");
	FILE *tp;
	char singleChar,nextChar;
	int i=0,j=0,k,count;
	while(!feof(fp))
	{
		fscanf(fp,"%c",&singleChar);
		if(singleChar=='?')
		{
			Grammar[i][j]='\0';
			break;
		}
		if(singleChar=='\n')
		{
			Grammar[i][j]='\0';
			i++;
			j=0;
			continue;
		}
		if(singleChar=='-')
		{
			tp=fp;
			fscanf(tp,"%c",&nextChar);
			if(nextChar=='>')
			{
				fp=tp;
				continue;
			}
		}
		if(singleChar=='|')
		{

	Grammar[i+1][0]=Grammar[i][0];
			Grammar[i][j]='\0';
			i++;
			j=1;
			continue;
		}
		Grammar[i][j]=singleChar;

	if(singleChar>='A'&&singleChar<='Z')
		{
			count=0;

	while(VN[count]!=singleChar&&VN[co
unt]!='\0')
			{
				count++;
			}
			if(VN[count]=='\0')
```

```
                {
                    vnNum=count+1;
                    if(singleChar=='S')
                    {
                        j++;
                        continue;
                    }
                    VN[count]=singleChar;
                    vnNum=count+1;
                }
            }
            else
            {
                count=0;

        while(VT[count]!=singleChar&&VT[cou
nt]!='\0')
                {
                    count++;
                }
                if(VT[count]=='\0')
                {
                    VT[count]=singleChar;
                    vtNum=count+1;
                }
            }
            j++;
        }
        printf("输入的文法：\n");
        for(k=0;k<=i;k++)
        {
            j=0;
            while(Grammar[k][j]!='\0')
            {
                if(j==1)
                {
                    printf("->");
                }
                printf("%c",Grammar[k][j]);
                j++;
            }
            printf("\n");
        }
        count=0;
        printf("VT:");
        while(VT[count]!='\0')
        {
            printf("%3c",VT[count]);
            count++;
        }
        VT[count]='#';
        vtNum=count+1;
        printf("%3c",VT[count]);
        printf("\nVN:");
        count=0;
        while(VN[count]!='\0')
        {
            printf("%3c",VN[count]);
            count++;
        }
        printf("\n");
//      printf("\n%d %d\n",vtNum,vnNum);
        fclose(fp);
        grammarNum=i+1;
        return i;
}

int vNumCount()
{
        int i,j;
        for(i=0;i<grammarNum;i++)
        {
            j=1;
            while(Grammar[i][j]!='\0')
            {
                j++;
            }
            VNum[i]=j;
//          printf("%3d",VNum[i]);
        }
        printf("\n");
        return 0;
}

void InitStack()
{
        state.base=(int *)malloc(100*sizeof(int));
        if(!state.base)exit(1);
```

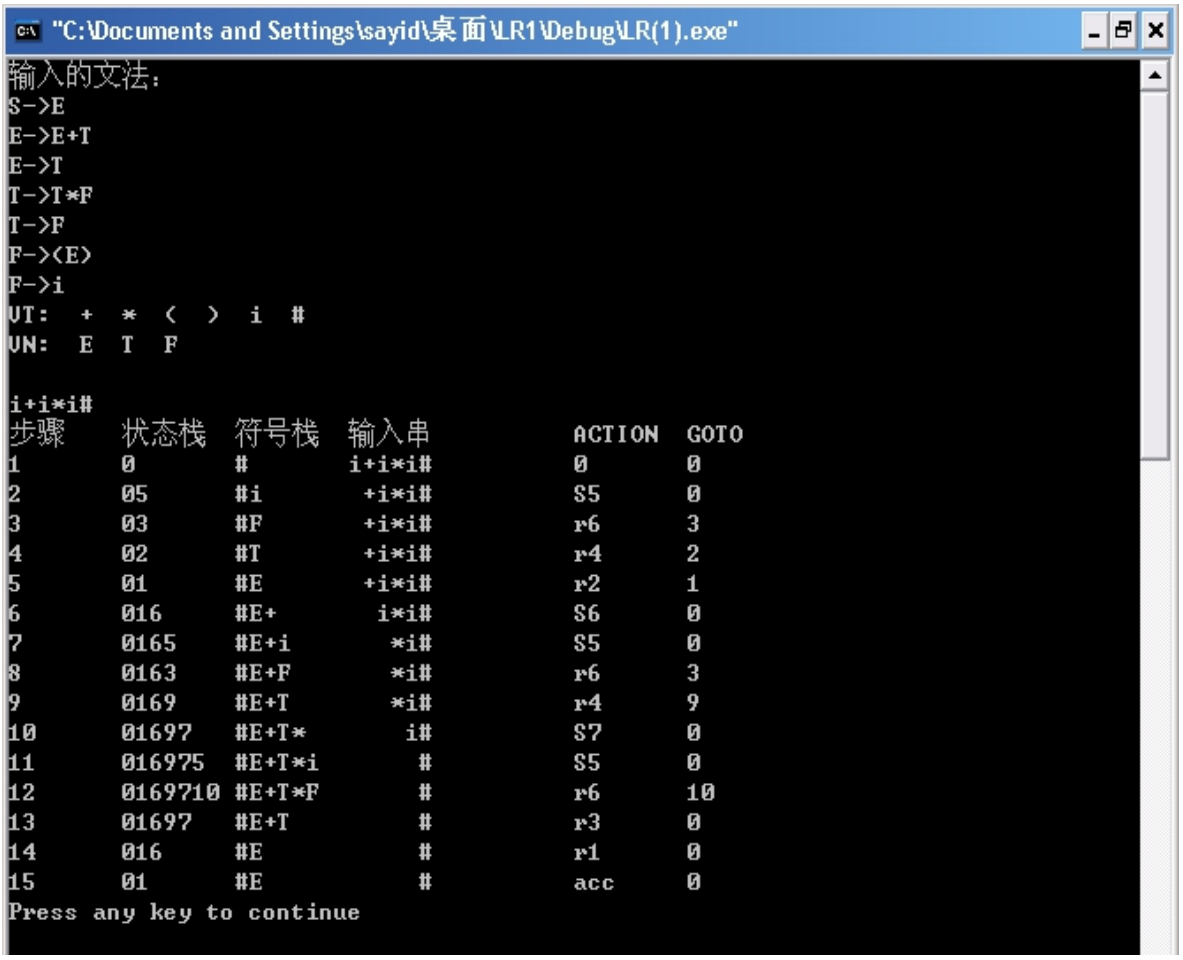
```
    state.top=state.base;
    *state.top=0;
    symbol.base=(char
*)malloc(100*sizeof(char));
    if(!symbol.base)exit(1);
    symbol.top=symbol.base;
    *symbol.top='#';
}

int Judge(int stateTop,char inputChar)
{
    int i,j;
    for(i=0;i<stateNum;i++)
    {
        if(stateTop==i)break;
    }
    for(j=0;j<vtNum;j++)
    {
        if(inputChar==AVT[j])break;
    }
    return Action[i][j];
}
int GetGoto(int stateTop,char inputChar)
{
    int i,j;
    for(i=0;i<stateNum;i++)
    {
        if(stateTop==i)break;
    }
    for(j=0;j<vnNum;j++)
    {
        if(inputChar==GVN[j])break;
    }
    return Goto[i][j];
}

int print(int count,int i,char Input[],int
action,int gt,int sign)
{

    int *p=state.base,stateNum;
    int j,jj;
    char *q=symbol.base,symbolNum;
    printf("%d\t",count);
    while(p!=state.top+1)
    {
        stateNum=*p;
        printf("%d",stateNum);
        p++;
    }
    printf("\t");

    while(q!=symbol.top+1)
    {
        symbolNum=*q;
        printf("%c",symbolNum);
        q++;
    }
    printf("\t");
    j=i;
    jj=0;
    while(jj<j)
    {
        printf(" ");
        jj++;
    }
    while(Input[j]!='\0')
    {
        printf("%c",Input[j]);
        j++;
    }
    printf("\t");
    if(sign==1)
    {
        printf("\tS%d\t%d\n",action,gt);
    }
    if(sign==2)
    {
        printf("\tr%d\t%d\n",action,gt);
    }
    if(sign==3)
    {
        printf("\tacc\t%d\n",gt);
    }
    if(sign==0)printf("\t0\t0\n");
    return 0;
}
```

```
int Pop(int action)
{
	int *p,stateNum,ssValue,i;
	state.top--;
	p=state.top;
	stateNum=*p;
	i=VNum[action]-1;
	while(i!=0)
	{
		symbol.top--;
		i--;
	}
	symbol.top++;
	*symbol.top=Grammar[action][0];
	ssValue=GetGoto(stateNum,Grammar[ac
tion][0]);
	if(ssValue==0)return ssValue;
	state.top++;
	*state.top=ssValue;
	return ssValue;
}
int Reduction()
{
	char Input[20];
	int i=0,count=1;
	int ssValue,action;
	int stateTop,gt;
	int sign=-1;//移进 1，规约 2，接受 3
	scanf("%s",&Input);
	while(Input[i]!='\0')
	{
		if(Input[i]>='A'&&Input[i]<='Z')
		{
			printf("输入的不是有效的表
达式！");
			return 0;
		}
		i++;
	}
	i=0;
	printf("步骤\t 状态栈\t 符号栈\t 输入串
\t\tACTION\tGOTO\n");
	while(Input[i]!='\0')
	{
		if(count==1)
		{
			print(count,i,Input,0,0,0);
			count++;
		}
		stateTop=*state.top;
		ssValue=Judge(stateTop,Input[i]);
		if(ssValue==0)
		{
			state.top--;
			if(*symbol.top=='#')
			{
				printf("规约出错！");
				return 0;
			}
			continue;
		}
		if(ssValue==-1)
		{
			sign=3;

print(count,i,Input,ssValue,0,sign);
			count++;
			return 1;
		}
		if(ssValue>=100)
		{
			sign=1;
			action=ssValue-100;
			state.top++;
			*state.top=action;
			symbol.top++;
			*symbol.top=Input[i];
			i++;

print(count,i,Input,action,0,sign);
			count++;
		}
		if(ssValue>=50&&ssValue<100)
		{
			sign=2;
			action=ssValue-50;
			gt=Pop(action);
```

```
        print(count,i,Input,action,gt,sign);
                count++;
            }
        }
        return 0;
}
```

```
int main()
{
    ScanGrammar();
    vNumCount();
    InitStack();
    Reduction();
    return 0;
}
```

【运行结果】


"C:\Documents and Settings\sayid\桌面\LR1\Debug\LR(1).exe"
输入的文法：
S->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
VT: + * ( ) i #
VN: E T F
i+i*i#
步骤 状态栈 符号栈 输入串 ACTION GOTO
1 0 # i+i*i# 0 0
2 05 #i +i*i# S5 0
3 03 #F +i*i# r6 3
4 02 #T +i*i# r4 2
5 01 #E +i*i# r2 1
6 016 #E+ i*i# S6 0
7 0165 #E+i *i# S5 0
8 0163 #E+F *i# r6 3
9 0169 #E+T *i# r4 9
10 01697 #E+T* i# S7 0
11 016975 #E+T*i # S5 0
12 0169710 #E+T*F # r6 10
13 01697 #E+T # r3 0
14 016 #E # r1 0
15 01 #E # acc 0
Press any key to continue

实验四 自动机应用，利用自动机技术和原理解决实际问题

一、 实验目的

1.理解有限自动机的作用；
2.利用状态图和状态表表示有限自动机；
3.以程序实现有限自动机的运行过程;
4.利用状态表和有限自动机的运行原理编制程序，使得程序能够识别一个输入串是否为一个有效的十六进制；

二、 实验环境

操作系统：window xp
编写环境：visual c++
编写语言：c 语言

三、 实验内容

1. 简单介绍你所设计的有限自动机。

有符号十六进制有限自动机

能识别（+|-）dd*(.dd*| ε)h|H 格式的字符串，其中 d 为 0-9，A-F，a-f;

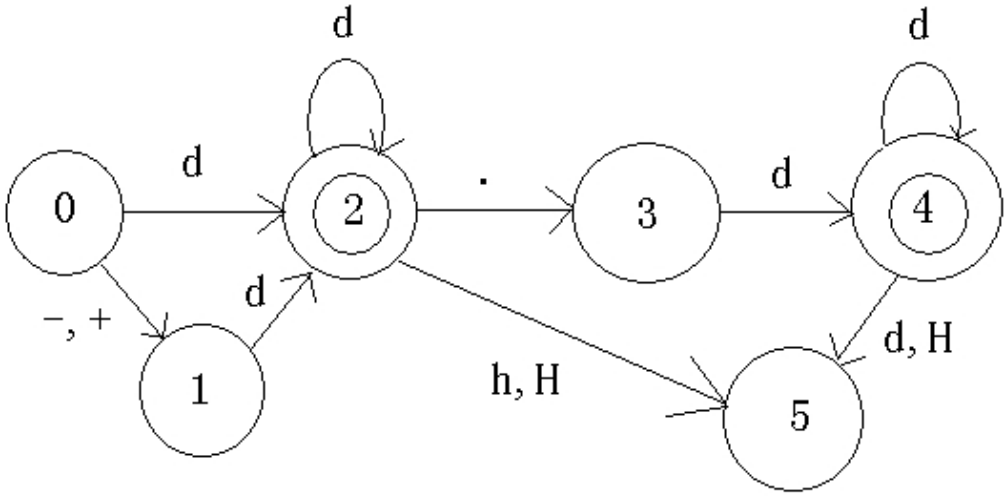
例如，+2.fh, -f.2H, f.fh， 6h 均为合法十六进制数，而 b.h，.ffh， 5.6.fh， zzh， ff 均为不合法十六进制数

2.画出有限自动机的状态表。

符号 / 状态	-，+	d	.	H,h
0	1	2	∉	∉
1	∉	2	∉	∉
2	∉	2	3	5
3	∉	4	∉	∉
4	∉	4	∉	5
5	∉	∉	∉	∉

状态图：

d 为 0-9，A-F，a-f，0 为初始状态，5 为结束状态，输入必须以 h，H 结尾



3.测试数据和结果。

组号	输入数据	结果
1	1.0h	接受
2	-E.eH	接受
3	EH	接受
4	-5h	接受
5	15246eah	接受
6	-100H	接受
7	5.0000ah	接受
8	.12345h	不接受
9	Abxh	不接受
10	333	不接受
11	3.1.4h	不接受
12	123.h	不接受

四、 实验结果

经测试程序均能识别各种有符号十六进制数，各个状态之间的转换也没有出错，程序正确，功能能够实现。

五、 调试分析

一开始程序中字符跟数字的判断是分开的，所以状态表多了完全相同的一列，调中发现这两列可以合并，精简代码和空间，修改代码实现精简。

六、 实验小结

由于有模板，所以思考时间很少，程序编写是思路比较清晰，所以调试基本没有出错，基本上时间均用在编写代码，在无符号，有正符号，有负符号的考虑上无法做到一个很好的缩减状态，所以多扩展了一个状态表示有符号的状态。

了解了自动机的工作方式，能够很有结构的结局一类问题。

附录：源代码

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define F 0
#define RIGHT 1
#define DOT 2
#define H 3
bool IsRightWord(char ch) /*判断是否为a-f,A-F,这里的ch已经被转换成了大写，所以只需判断是否为A-F*/
{
    if(ch >= 'A' && ch <= 'F')
        return true;
    return false;
}
int state[6][4]={   //状态转移表
    1,2,0,0,
    0,2,0,0,
    0,2,3,5,
    0,4,0,0,
    0,4,0,5,
    0,0,0,0,
};
bool Accept(char * str)//判断是否为正确的有符号十六进制数
{
    int s=0;
    int len = strlen(str);
    int i;
    for (i=0;i<len;i++)//按位判断
    {
        if(str[i]=='-'||str[i]=='+')
        {
            if(state[s][F] == 0) return false;
            s = state[s][F];
        }
        else if(isdigit(str[i])||IsRightWord(toupper(str[i])))
        {
            if(state[s][RIGHT] == 0) return false;
            s = state[s][RIGHT];
        }
        else if(str[i] == '.')
        {
```

```
                if(state[s][DOT] == 0) return false;
                s = state[s][DOT];
            }
            else if(toupper(str[i])=='H')
            {
                if(state[s][H] == 0) return false;
                s = state[s][H];
            }
        }
        if(s == 5) return true;
        else return false;
}
int main ()
{
        char str[1005];
        while (gets(str)!=NULL)
        {
            if(Accept(str))
            {
                printf("Accept!\n");
            }
            else
            {
                printf("Not Accept!\n");
            }
        }
        return 0;
}
```