

中南林业科技大学

实验报告

课程名称：编译原理

专业班级：2021 级计算机科学与技术 3 班

姓名：刘军

学号：20212753

2024 年 6 月 4 日

实验一 词法分析

一、实验目的

编制一个读单词过程，从输入的源程序中，识别出各个具有独立意义的单词，即基本保留字、标识符、常数、运算符、分隔符五大类。并依次输出各个单词的内部编码及单词符号自身值。

二、实验题目

如源程序为C语言。输入如下一段：

```
main()
{
int  a=-5,b=4,j;
if(a>=b)
j=a-b;
else  j=b-a;
}
```

运行结果输出如下：

```
("main", 1, 1)
("(" , 5)
(")", 5)
("{", 5)
("int", 1, 2)
("a", 2)
("=", 4)
("-5", 3)
(",", 5)
("b", 2)
("=", 4)
("4", 3)
(",", 5)
("j", 2)
(",", 5)
("if", 1,3)
("(" , 5)
("a", 2)
(">=", 4)
("b", 2)
(")", 5)
("j", 2)
("=", 4)
("a", 2)
("-", 4)
("b", 2)
(",", 5)
("else", 1,4)
("j", 2)
("=", 4)
("b", 2)
```

- ("-", 4)
- ("a", 2)
- (";", 5)
- ("}", 5)

三、实验理论依据

(一)识别各种单词符号

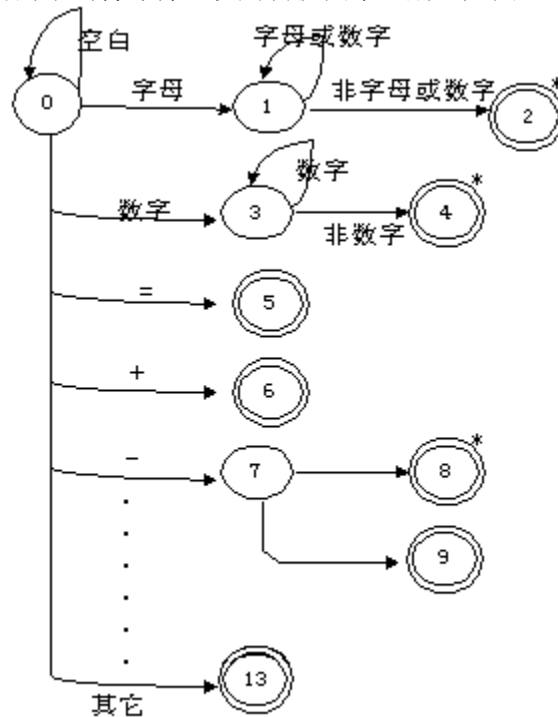
1、程序语言的单词符号一般分为五种：

- (1) 关键字（保留字/ 基本字）if 、 while 、 int...
- (2) 标识符：常量名、变量名...
- (3) 常数：34 、 56.78 、 ...
- (4) 运算符：+ 、 - 、 * 、 / 、 < 、 || 、 &&....
- (5) 界限符：, ; () { } /*...

2、识别单词：掌握单词的构成规则很重要

- (1) 标识符的识别：字母| 下划线+(字母/ 数字/ 下划线)
- (2) 关键字的识别：与标识符相同，最后查表
- (3) 常数的识别
- (4) 界符和算符的识别

3、大多数程序设计语言的单词符号都可以用转换图来识别，如图1-1



对简单语言进行词法分析的状态转换图

图 1-1

4、词法分析器输出的单词符号常常表示为二元式：（单词种别，单词符号的属性值）

- (1) 单词种别通常用整数编码，如1 代表关键字，2 代表标识符等
- (2) 关键字可视其全体为一种，也可以一字一种。采用一字一种得分法实际处理起来较为方便。
- (3) 标识符一般统归为一种
- (4) 常数按类型（整数、实数等）分种

(5) 运算符可采用一符一种的方法。

(6) 界符一般一符一种的分法。

(二)超前搜索方法

1、词法分析时，常常会用到超前搜索方法。

如当前待分析字符串为“a>+”，当前字符为“>”，此时，分析器到底是否将其分析为大于关系运算符还是大于等于关系运算符呢？

显然，只有知道下一个字符是什么才能下结论。于是分析器读入下一个字符‘+’，这时可知应将‘>’解释为大于运算符。但此时，超前读了一个字符‘+’，所以要回退一个字符，词法分析器才能正常运行。又比如：‘+’分析为正号还是加法符号

(三)预处理

预处理工作包括对空白符、跳格符、回车符和换行符等编辑性字符的处理，及删除注解等。由一个预处理子程序来完成。

四、词法分析器的设计

1、设计方法：

(1) 写出该语言的词法规则。

(2) 把词法规则转换为相应的状态转换图。

(3) 把各转换图的初态连在一起，构成识别该语言的自动机

(4) 设计扫描器

2、把扫描器作为语法分析的一个过程，当语法分析需要一个单词时，就调用扫描器。

扫描器从初态出发，当识别一个单词后便进入终态，送出二元式

五、完整程序

```
#include <stdio.h>
#include <ctype.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#define NULL 0
FILE *fp;
char cbuffer;
//增加了关键字
char *key[24]=
{"main","int","if","else","for","while","do","return","break","continue","auto","double","struct",
"long","switch","case","register","typedef","char","extern","union","const","short","null"
};
int atype,id=4;

/*判断单词是保留字还是标识符*/
int search(char searchchar[], int wordtype)
{
    int i=0;
    int p;
    switch (wordtype)
    {
        case 1:
            for (i=0; i<=21; i++)
            {
                if (strcmp(key[i],searchchar)==0)
```

```

        {
            p=i+1;    /*是保留字则 p 为非 0 且不重复的整数*/
            break;
        }
        else p=0;      /*不是保留字则用于返回的 p=0*/
    }
    return(p);
}
}

```

```

char  alphaprocess(char buffer)
{
    int  atype;    /*保留字数组中的位置*/
    int  i=-1;
    char  alhatp[20];
    while ((isalpha(buffer))||(isdigit(buffer))||buffer=='_')
    {
        alhatp[++i]=buffer;
        buffer=fgetc(fp);
    } /*读一个完整的单词放入 alhatp 数组中*/
    alhatp[i+1]='\0';
    atype=search(alhatp,1);/*对此单词调用 search 函数判断类型*/
    if(atype!=0)
    {
        printf("(%s, 1,%d)\n",alhatp,atype-1);

        id=1;
    }
    else
    {
        printf("(%s ,2)\n",alhatp);
        id=2;
    }
    return(buffer);
}

```

```

char digitprocess(char buffer) {
    int i = -1;
    char digittp[20];
    while (isdigit(buffer) || ((buffer == '.') && i != -1) || buffer == 'e' || buffer == 'E' || buffer == '+' ||
buffer == '-') {
        digittp[++i] = buffer;
        buffer = fgetc(fp);
    }
    digittp[i + 1] = '\0';
    printf("(%s ,3)\n", digittp);
    id = 3;
    return (buffer);
}

```

```

//char digitprocess(char buffer)
//{

```

```

//  int i=-1;
//  char digittp[20];
//  while (isdigit(buffer)||((buffer=='.')&&i!=-1))
//  {
//      digittp[++i]=buffer;
//      buffer=fgetc(fp);
//  }
//  digittp[i+1]='\0';
//  printf("(%s ,3)\n",digittp);
//  id=3;
//  return(buffer);
//}

```

```

char otherprocess(char buffer)
{
    char ch[20];
    ch[0]=buffer;
    ch[1]='\0';

```

// 检查是否为解引用运算符

```

if (ch[0] == '*') {
    printf("(%s ,4)\n", ch);
    id = 4;
    return fgetc(fp);
}

```

// 检查是否为成员访问运算符

```

if (ch[0] == '.' && fgetc(fp) == '>') {
    printf("(-> ,4)\n");
    id = 4;
    return fgetc(fp);
}

```

```

if(ch[0]=='|'||ch[0]=='='||ch[0]=='{'||ch[0]=='}'||ch[0]=='('||ch[0]==')')
{

```

```

    printf("(%s ,5)\n",ch);
    buffer=fgetc(fp);
    id=4;
    return(buffer);
}

```

```

if(ch[0]=='*'||ch[0]=='/')
{

```

```

    buffer=fgetc(fp);
    ch[1]=buffer;
    if(ch[1]=='=')
    {
        ch[2]='\0';
        printf("(%s ,4)\n",ch);
        buffer=fgetc(fp);
        id=4;
        return(buffer);
    }
}

```

```

        //增加判断 * /=
        else
        {
            ch[1]='\0';
            printf("(%s ,4)\n",ch);
        }
        id=4;
        return(buffer);
    }
    if(ch[0]=='='||ch[0]=='!'||ch[0]=='<'||ch[0]=='>')
    {
        buffer=fgetc(fp);
        if(buffer=='=')
        {
            ch[1]=buffer;
            ch[2]='\0';
            printf("(%s ,4)\n",ch);
        }
        //增加判断 >> <<
        else if(buffer == '>' || buffer == '<')
        {
            ch[1]=buffer;
            ch[2]='\0';
            printf("(%s ,4)\n",ch);
        }
        else
        {
            printf("(%s ,4)\n",ch);
            id=4;
            return(buffer);
        }
        buffer=fgetc(fp);
        id=4;
        return(buffer);
    }

    if(ch[0]=='+'||ch[0]=='-')
    {
        if(id==4) /*在当前符号以前是运算符，则此时为正负号*/
        {
            buffer=fgetc(fp);
            ch[1]=buffer;
            ch[2]='\0';
            printf("(%s ,3)\n",ch);
            id=3;
            buffer=fgetc(fp);
            return(buffer);
        }
        //增加判断 += -=
        else if(ch[1] == '=')
        {
            ch[2]='\0';

```



```

        printf("(%s ,4)\n",ch);
    }
    //增加判断 ++ --
    else if(ch[1] == '+' || ch[1] == '-')
    {
        ch[2]='\0';
        printf("(%s ,4)\n",ch);
    }
    else
    {
        ch[1]='\0';//此时为加减运算符
        printf("(%s ,4)\n",ch);
        id=4;
        return(buffer);
    }
    id=4;
    buffer=fgetc(fp);
    return(buffer);
}

//逻辑运算符和位运算符
if (ch[0] == '&' || ch[0] == '|') {
    buffer = fgetc(fp);
    // 如果运算符的两侧是整数类型的操作数，则它是位运算符
    if (isdigit(buffer) || buffer == '_') {
        // 位运算符
        ch[1] = '\0';
        printf("(%s ,4)\n", ch); // 用 6 表示位运算符
    } else {
        // 逻辑运算符
        if (ch[0] == buffer) {
            ch[1] = buffer;
            ch[2] = '\0';
            printf("(%s ,4)\n", ch);
        } else {
            printf("(%s ,4)\n", ch);
            id = 4;
            return (buffer);
        }
    }
    buffer = fgetc(fp);
    id = 4;
    return (buffer);
}
}

```

```

int    main()
{
    if ((fp=fopen("1.c","r"))==NULL)    /*只读方式打开一个文件*/
        printf("error");
    else
    {

```

```

cbuffer = fgetc(fp); /*fgetc()函数：从磁盘文件读取一个字符*/
while (cbuffer!=EOF)
{
    while(cbuffer==' '||cbuffer=='\n') /*掠过空格和回车符*/
        cbuffer=fgetc(fp);
    if(isalpha(cbuffer))
        {cbuffer=alphaprocess(cbuffer);}
    else if (isdigit(cbuffer))
        cbuffer=digitprocess(cbuffer);
    else {cbuffer=otherprocess(cbuffer);}
}
}
}

```

六、运行结果截图

1.cpp	1.c
1	
2	<code>int main()</code>
3	<code>{</code>
4	<code>int a=-5,b=4,j, e;</code>
5	<code>double c = 1.23e3;</code>
6	<code>int *d;</code>
7	<code>e = a++;</code>
8	<code>e = a->b;</code>
9	<code>a+=b;</code>
10	<code>if(a>=b) j=a-b;</code>
11	<code>else j=b-a;</code>
12	<code>}</code>

```

(int, 1,1)
(main, 1,0)
( (, 5)
) , 5)
{ , 5)
(int, 1,1)
(a , 2)
(= , 4)
(- , , 3)
(b , 2)
(= , 4)
(4 , 3)
( , , 5)
(j , 2)
( , , 5)
(e , 2)
( ; , 5)
(double, 1,11)
(c , 2)
(= , 4)
(1.23e3 , 3)

```

实验二 LL(1) 分析法

一、实验目的

根据某一文法编制调试 LL(1)分析程序，以便对任意输入的符号串进行分析。本次实验的目的主要是加深对预测分析 LL(1)分析法的理解。

二、实验题目

实验规定对下列文法，用 LL(1) 分析法对任意输入的符号串进行分析：

- (1) $E::=TG$
- (2) $G::=+TG$
- (3) $G::=\epsilon$
- (4) $T::=FS$
- (5) $S::=*FS$
- (6) $S::=\epsilon$
- (7) $F::=(E)$
- (8) $F::=i$

若输入串为 $i+i*i\#$ ，则输出为：

步骤	分析栈	剩余串	产生式
1	#E	$i+i*i\#$	$E \rightarrow TG$
2	#GT	$i+i*i\#$	$T \rightarrow FS$
3	#GSF	$i+i*i\#$	$F \rightarrow i$
4	#GSi	$i+i*i\#$	i匹配
5	#GS	$+i*i\#$	$S \rightarrow \epsilon$
6	#G	$+i*i\#$	$G \rightarrow +TG$
7	#GT+	$+i*i\#$	+
...
...

LL(1)的分析表为：

	i	+	*	()	#	说 明
E	e			e			Select($E \rightarrow TG$)= { (,i }
G		g			g1	g1	Select ($G \rightarrow +TG$)= { + } Select ($G \rightarrow \epsilon$)= { #,) }
T	t			t			Select ($T \rightarrow FS$)= { (,i }
S		s1	s		s1	s1	Select ($S \rightarrow *FS$)= { * } Select ($S \rightarrow \epsilon$)= { #,) + }
F	f1			f			Select ($F \rightarrow (E)$)= { (} Select ($F \rightarrow i$)= { i }

三、完整程序

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

/*
(1) E::=TG
(2) G::=+TG
(3) G::=  $\epsilon$ 
(4) T::=FS
(5) S::=*FS
(6) S::=  $\epsilon$ 
(7) F::=(E)
(8) F::=i
*/
char A[20];/*分析栈*/
char B[20];/*剩余串*/
char v1[20]={'I','+','*','(',')','#'};/*终结符 */
char v2[20]={'E','G','T','S','F'};/*非终结符 */

int j=0,b=0,top=0,l,m,n;/* 为输入串长度 */
char ch,x;/*x 为当前栈顶字符*/
int k=1,flag=0,finish=0;

typedef struct type/*产生式类型定义 */
{
    char origin;/*大写字符 */
    char array[5];/*产生式右边字符 */
    int length;/*字符个数 */
}type;

void print()/*输出分析栈 */
{
    int a; /*指针*/
    for(a=0;a<=top+1;a++)
        printf("%c",A[a]);
    printf("\t\t");
}

void print1()/*输出剩余串*/
{
    int j;
    for(j=0;j<b;j++)
        printf(" ");
    for(j=b;j<=l;j++)
        printf("%c",B[j]);
    printf("\t\t\t");
}
```

```
}
```

```
int main(){
    type e,t,g,g1,s,s1,f,f1,cha; /*结构体变量 */
    type C[10][10]; /*预测分析表
        */
    /*把文法产生式赋值结构体*/
    e.origin='E';
    strcpy(e.array,"TG");
    e.length=2; /*更改 length 赋值*/
    t.origin='T';
    strcpy(t.array,"FS");
    t.length=2; /*更改 length 赋值*/
    g.origin='G';
    strcpy(g.array,"+TG");
    g.length=3; /*更改 length 赋值*/
    g1.origin='G';
    g1.array[0]='^';
    g1.array[1]='\0'; /*更改 字符串结束标志*/
    g1.length=1; /*更改 length 赋值*/
    s.origin='S';
    strcpy(s.array,"*FS");
    s.length=3; /*更改 length 赋值*/
    s1.origin='S';
    s1.array[0]='^';
    s1.array[1]='\0'; /*更改 字符串结束标志*/
    s1.length=1; /*更改 length 赋值*/
    f.origin='F';
    strcpy(f.array,"(E)");
    f.length=3; /*更改 length 赋值*/
    f1.origin='F';
    f1.array[0]='i';
    f1.array[1]='\0'; /*更改 字符串结束标志*/
    f1.length=1; /*更改 length 赋值*/

    for(m=0;m<=4;m++) /*初始化分析表*/
        for(n=0;n<=5;n++)
            C[m][n].origin='N'; /*全部赋为空*/
    /*填充分析表*/
    C[0][0]=e; C[0][3]=e;
    C[1][1]=g; C[1][4]=g1; C[1][5]=g1;
    C[2][0]=t; C[2][3]=t;
    C[3][1]=s1; C[3][2]=s; C[3][4]=C[3][5]=s1;
    C[4][0]=f1; C[4][3]=f;

    printf("可输入的字符为'i','+', '*', '(', ')', '#'注意: 以第一个'#'作为字符串结束:\n");
    do /*读入分析串*/ {
```

```

scanf("%c",&ch);
if ((ch!='i') &&(ch!='+') &&(ch!='*')&&(ch!='(')&&(ch!='')&&(ch!='#'))
{
    printf("Error!输入串中有非法字符'%c'\n", ch);
    exit(1);
}
B[j]=ch;
j++;
}while(ch!='#');
printf("对字符串的分析过程如下: \n");
l=j; /*分析串长度*/
ch=B[0]; /*当前分析字符*/
A[top]='#'; A[++top]='E'; /*'#','E'进栈*/
printf("步骤\t\t 分析栈\t\t 剩余字符\t\t 所用产生式 \n");
/*推导过程如下*/
do
{
    x=A[top--]; /*x 为当前栈顶字符*/
    printf("%d",k++); /*步骤序号*/
    printf("\t\t");
    for(j=0;j<=5;j++) /*判断是否为终结符*/
        if(x==v1[j])
        {
            flag=1;
            break;
        }
    if(flag==1) /*如果是终结符*/
    {
        if(x=='#')
        {
            if(ch=='#')
            {

                finish=1; /*结束标记*/
                print(); /*更改 最后成功时对齐*/
                print1();

                printf("acc!\n"); /*接受 */
                printf("输入串符合该文法! \n");
                getchar();
                getchar();
                exit(1);
            }
            else
            {
                print();
                print1();
                printf("Error!\n");
                printf("当前分析栈中是#, 但是剩余串还有字符未匹配。 \n");
            }
        }
    }
}

```

```

        printf("输入串不符合该文法! \n");
        getchar();
        getchar();
        exit(0);
    }
}
if(x==ch)
{
    print();
    print1();
    printf("%c 匹配\n",ch);
    ch=B[++b];/*下一个输入字符*/
    flag=0;/*恢复标记*/
}
else
{
    if(ch=='#')
    {
        print();
        print1();
        printf("Error!\n");
        printf("剩余串中是#, 但是分析栈还有字符未匹配。 \n");
        printf("输入串不符合该文法! \n");
        getchar();
        getchar();
        exit(0);
    }
    else
    {
        print();
        print1();
        printf("Error!%c 匹配出错\n",ch);/*输出出错终结符*/
        printf("与当前输入字符不匹配! \n");
        printf("输入串不符合该文法! \n");
        getchar();
        getchar();
        exit(1);
    }
}
}
else/*非终结符处理*/
{
    for(j=0;j<=4;j++)
        if(x==v2[j])
        {
            m=j;/*行号*/
            break;
        }
    for(j=0;j<=5;j++)
        if(ch==v1[j])
        {
            n=j;/*列号*/

```

```

        break;
    }
    cha=C[m][n];/*记录应用的产生式*/
    if(cha.origin!='N')
    {
        print();
        printl();
        printf("%c->",cha.origin);
        for(j=0;j<cha.length;j++)
            printf("%c",cha.array[j]);
        printf("\n");
        for(j=(cha.length-1);j>=0;j--)
        {
            A[++top]=cha.array[j];
            if(A[top]=='^')/*为空则不进栈*/
                top--;
        }
    }
    else
    {
        print();
        printl();
        printf("Error!\n");
        printf("当前栈顶元素是非终结符，但是找不到当前输入字符相应的产生式！\n");

        printf("输入串不符合该文法！\n");
        getchar();
        getchar();
        exit(0);
    }
}
}while(true);
return 0;
}

```

四、运行结果截图

```

D:\LJ189\Documents\2.exe
可输入的字符为 'i', '+', '*', '(', ')', '#', 注意：以第一个 '#' 作为字符串结束：
i+i*i#
对字符串的分析过程如下：
步骤    分析栈    剩余字符    所用产生式
1        #E        i+i*i#     E->TG
2        #GT       i+i*i#     T->FS
3        #GSF      i+i*i#     F->i
4        #GSi      i+i*i#     i匹配
5        #GS       +i*i#     S->^
6        #G        +i*i#     G->+TG
7        #GT+      +i*i#     +匹配
8        #GT       i*i#     T->FS
9        #GSF      i*i#     F->i
10       #GSi      i*i#     i匹配
11       #GS       *i#       S->*FS
12       #GSF*     *i#       *匹配
13       #GSF      i#        F->i
14       #GSi      i#        i匹配
15       #GS       #         S->^
16       #G        #         G->^
17       #         #         acc!
输入串符合该文法！

```



```
D:\LJ189\Documents\2.exe  ×  +  ∨

可输入的字符为 'i', '+', '*', '(', ')', '#' 注意：以第一个 '#' 作为字符串结束：
i+)i#
对字符串的分析过程如下：
步骤      分析栈      剩余字符      所用产生式
1          #E          i+)i#        E->TG
2          #GT        i+)i#        T->FS
3          #GSF        i+)i#        F->i
4          #GSi        i+)i#        i匹配
5          #GS          +)i#        S->^
6          #G           +)i#        G->+TG
7          #GT+         +)i#        +匹配
8          #GT          )i#        Error!
当前栈顶元素是非终结符，但是找不到当前输入字符相应的产生式！
输入串不符合该文法！
```

实验三 逆波兰式的产生及计算

一、实验目的

将用中缀式表示的算术表达式转换为用逆波兰式表示的算术表达式，并计算用逆波兰式来表示的算术表达式的值

二、实验题目

如输入如下：21+((42-2)*15+6)-18#

输出为：

原来表达式：21+((42-2)*15+6)-18#

后缀表达式：21&42&2&-15&*6&+18&-

计算结果：609

三、算法流程图

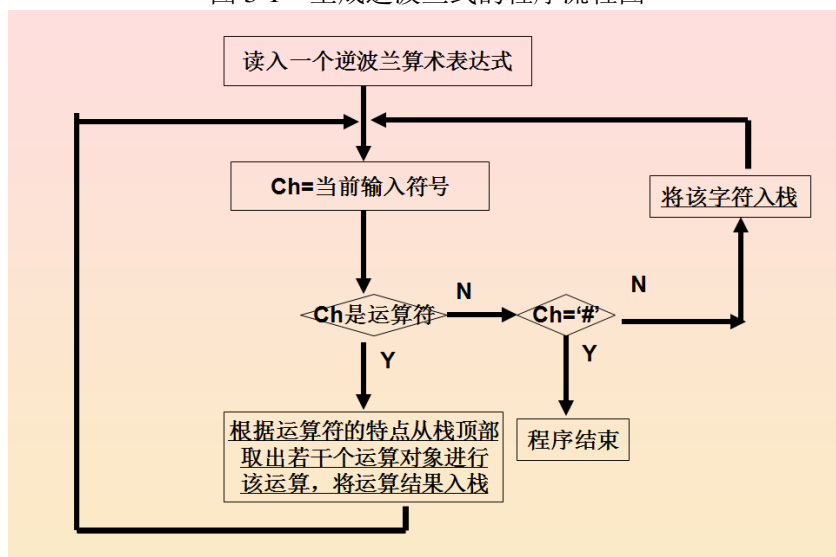
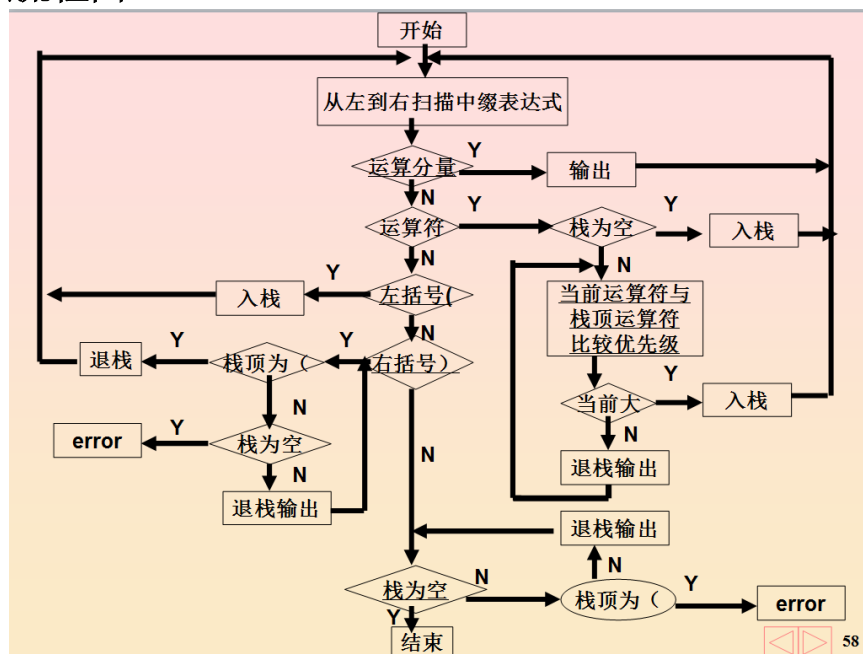


图 3-2 计算逆波兰式的程序流程图

四、完整程序

```
#include <stdio.h>
#include <stdlib.h>
#define max 100 // 定义栈和数组的最大容量为 100

char ex[max]; // 存储后缀表达式的全局数组

// 将中缀表达式转换为后缀表达式的函数
void trans() {
    char str[max]; // 存储输入的中缀表达式
    char stack[max]; // 用于操作符存储的栈
    char ch; // 当前处理的字符
    int sum, i, j, t, top = 0; // 变量初始化
    printf("请输入一个求值的表达式，以#结束。\\n");
    printf("算术表达式: ");

    // 读取输入的中缀表达式直到遇到 '#' 或达到最大长度
    i = 0;
    do {
        i++;
        scanf("%c", &str[i]);
    } while (str[i] != '#' && i != max);

    sum = i; // 表达式的长度
    t = 1; // 后缀表达式的当前索引
    i = 1; // 中缀表达式的当前索引
    ch = str[i];
    i++;
    // 开始转换中缀表达式到后缀表达式
    while (ch != '#') {
        switch (ch) {
            case '(': // 遇到左括号，入栈
                top++;
                stack[top] = ch;
                break;
            case ')': // 遇到右括号，出栈直到左括号
                while (stack[top] != '(') {
                    ex[t] = stack[top];
                    top--;
                    t++;
                }
                top--; // 弹出左括号
                break;
            case '+':
            case '-': // 遇到加号或减号，出栈直到遇到左括号或栈空
                while (top != 0 && stack[top] != '(') {
                    ex[t] = stack[top];
                    top--;
                    t++;
                }
            }
        }
    }
}
```

```

        }
        top++;
        stack[top] = ch; // 当前操作符入栈
        break;
    case '*':
    case '/': // 遇到乘号或除号，处理优先级
        while (stack[top] == '*' || stack[top] == '/') {
            ex[t] = stack[top];
            top--;
            t++;
        }
        top++;
        stack[top] = ch; // 当前操作符入栈
        break;
    case ' ': // 忽略空格
        break;
    default: // 处理操作数（包括小数点）
        while ((ch >= '0' && ch <= '9') || ch == '.') {
            ex[t] = ch;
            t++;
            ch = str[i];
            i++;
        }
        i--; // 回退一个字符
        ex[t] = '&'; // 操作数结束标志
        t++;
        break;
    }
    ch = str[i];
    i++;
}
// 将栈中剩余的操作符全部出栈
while (top != 0) {
    ex[t] = stack[top];
    top--;
    t++;
}
ex[t] = '#'; // 后缀表达式结束标志
ex[t + 1] = '\0'; // 字符串结束符

// 输出原始中缀表达式
printf("\n\t 原来表达式: ");
for (j = 1; j < sum; j++)
    printf("%c", str[j]);

// 输出转换后的后缀表达式
printf("\n\t 后缀表达式: ");
for (j = 1; j < t; j++)
    printf("%c", ex[j]);
}

```

// 计算后缀表达式的值

```
void compvalue() {
    float stack[max], d;
    char ch;
    int t = 1, top = 0;
    ch = ex[t];
    t++;
    // 遍历后缀表达式并计算
    while (ch != '#') {
        switch (ch) {
            case '+': // 处理加法
                if (top > 0) {
                    stack[top - 1] = stack[top - 1] + stack[top];
                    top--;
                }
                break;
            case '-': // 处理减法
                if (top > 0) {
                    stack[top - 1] = stack[top - 1] - stack[top];
                    top--;
                }
                break;
            case '*': // 处理乘法
                if (top > 0) {
                    stack[top - 1] = stack[top - 1] * stack[top];
                    top--;
                }
                break;
            case '/': // 处理除法
                if (top > 0) {
                    if (stack[top] != 0) {
                        stack[top - 1] = stack[top - 1] / stack[top];
                    } else {
                        printf("\n\t 除 0 错误!\n");
                        exit(0);
                    }
                }
                top--;
            }
            break;
        default: // 处理操作数
            d = 0;
            float decimalPlace = 0.1; // 用于处理小数位
            int isDecimal = 0; // 标识是否在处理小数部分
            while ((ch >= '0' && ch <= '9') || ch == '.') {
                if (ch == '.') {
                    isDecimal = 1;
                } else {
                    if (!isDecimal) {
                        d = d * 10 + (ch - '0');
                    } else {
                        d = d + (ch - '0') * decimalPlace;
                        decimalPlace *= 0.1; // 移动到下一个小数位
                    }
                }
            }
        }
```

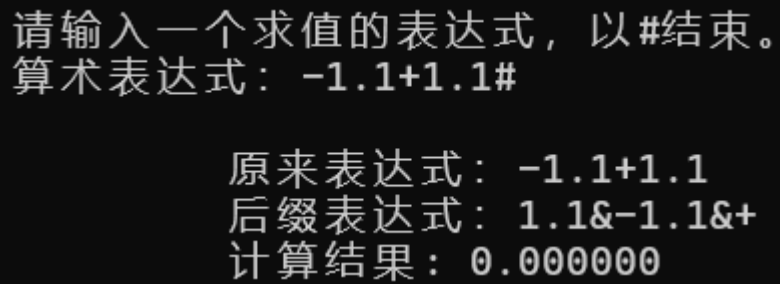
```

        }
    }
    ch = ex[t];
    t++;
}
top++;
stack[top] = d;
}
ch = ex[t];
t++;
}
// 输出计算结果
printf("\n\t 计算结果: %f\n", stack[top]); // 使用 %f 打印浮点数
getchar();
}

int main() {
    trans(); // 调用转换函数
    compvalue(); // 调用计算函数
    getchar(); // 等待用户输入
    return 0;
}

```

五、运行结果截图



```

请输入一个求值的表达式，以#结束。
算术表达式： -1.1+1.1#

          原来表达式： -1.1+1.1
          后缀表达式： 1.1&-1.1&+
          计算结果： 0.000000

```

实验四 LR(1) 分析法

一、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

二、实验题目

1、对下列文法，用 LR (1) 分析法对任意输入的符号串进行分析：

(0) $E \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

2、LR(1)分析表为：

状态	ACTION			GOTO	
	a	b	#	S	B
S0	S3	S4		1	2
S1			acc		
S2	S6	S7			5
S3	S3	S4			8
S4	r3	r3			
S5			r1		
S6	S6	S7			9
S7			r3		
S8	r2	r2			
S9			r2		

(1)若输入 baba#，则输出为：

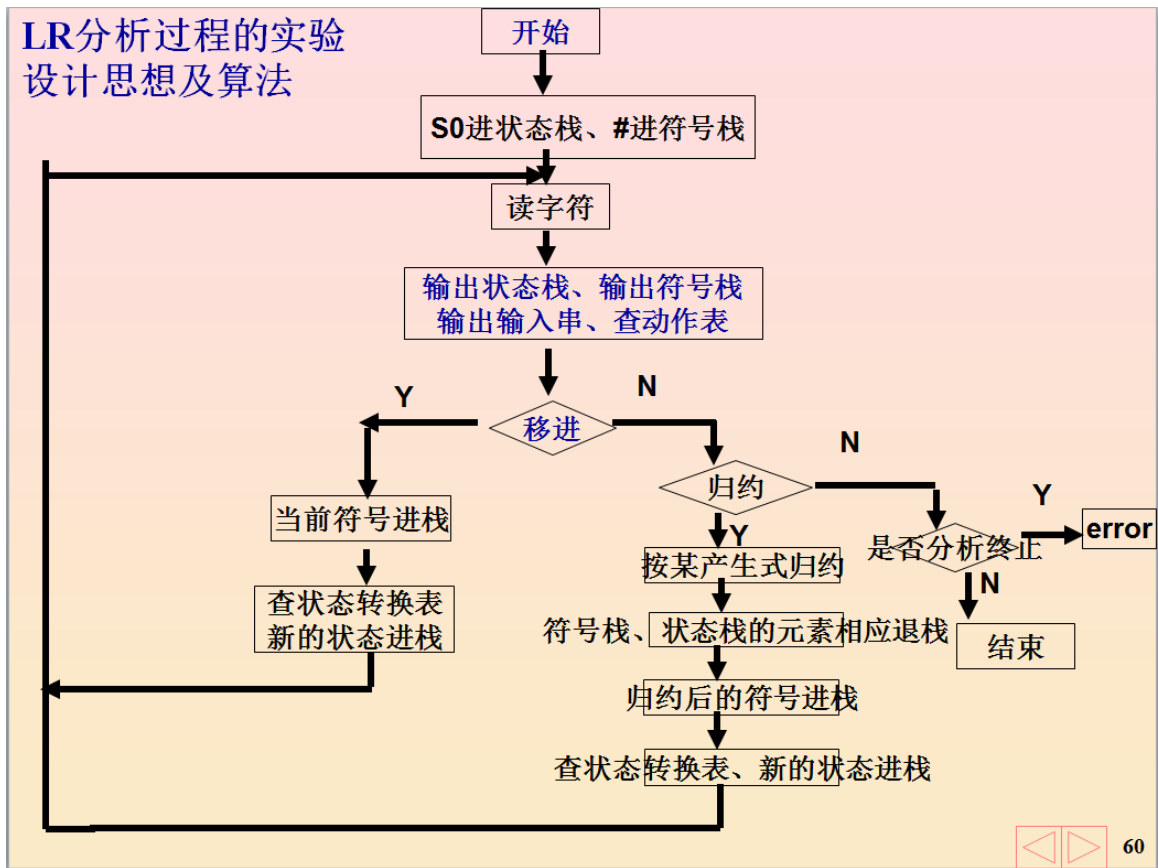
步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	baba#	S4	
2	04	#b	aba#	r3	2
3	02	#B	aba#	S6	
4	026	#Ba	ba#	S7	
5	0267	#Bab	a#	error	

(2)若输入 bb#，则输出为：

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	bb#	S4	
2	04	#b	b#	r3	2
3	02	#B	b#	S7	
4	027	#Bb	#	r3	5
5	025	#BB	#	r1	1
6	01	#S	#	acc	

三、算法流程图

LR分析过程的实验 设计思想及算法



四、完整程序

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

```
// 动作表: action
```

```
char *action[10][3] = {
    "S3#", "S4#", NULL, // 状态 0
    NULL, NULL, "acc", // 状态 1
    "S6#", "S7#", NULL, // 状态 2
    "S3#", "S4#", NULL, // 状态 3
    "r3#", "r3#", NULL, // 状态 4
    NULL, NULL, "r1#", // 状态 5
    "S6#", "S7#", NULL, // 状态 6
    NULL, NULL, "r3#", // 状态 7
    "r2#", "r2#", NULL, // 状态 8
    NULL, NULL, "r2#" // 状态 9
};
```

```
// GOTO 表: goto1
```

```
int goto1[10][2] = {
    {1, 2}, // 状态 0
    {0, 0}, // 状态 1
    {0, 5}, // 状态 2
    {0, 8}, // 状态 3
};
```



```

        {0, 0}, // 状态 4
        {0, 0}, // 状态 5
        {0, 9}, // 状态 6
        {0, 0}, // 状态 7
        {0, 0}, // 状态 8
        {0, 0}  // 状态 9
    };

    // 终结符: vt
    char vt[3] = {'a', 'b', '#'};
    // 非终结符: vn
    char vn[2] = {'S', 'B'};
    // 产生式: LR
    char *LR[4] = {
        "E->S#", // 产生式 0
        "S->BB#", // 产生式 1
        "B->aB#", // 产生式 2
        "B->b#"  // 产生式 3
    };

    // 状态栈、符号栈、输入串及其他相关变量
    int a[10];
    char b[10], c[10], c1;
    int top1, top2, top3, top, m, n;

    int main() {
        int g, h, i, j, k, l, p, y, z, count;
        char x, copy[10], copy1[10];

        // 初始化栈顶指针
        top1 = 0; // 状态栈顶
        top2 = 0; // 符号栈顶
        top3 = 0; // 输入串栈顶
        top = 0;  // 输入串指针

        // 初始状态入栈
        a[0] = 0;
        y = a[0];
        b[0] = '#';

        count = 0;
        z = 0;

        printf("请输入表达式\n");
        // 输入表达式
        do {
            scanf("%c", &c1);
            c[top3] = c1;
            top3 = top3 + 1;
        } while (c1 != '#');
    }

```

```
printf("步骤\t 状态栈\t\t 符号栈\t\t 输入串\t\tACTION\t\tGOTO\n");
```

```
do {
    y = z;
    m = 0;
    n = 0; // y,z 指向状态栈栈顶
    g = top;
    j = 0;
    k = 0;
    x = c[top];
    count++;
    printf("%d\t", count);
    // 输出状态栈
    while (m <= top1) {
        printf("%d", a[m]);
        m = m + 1;
    }
    printf("\t\t");
    // 输出符号栈
    while (n <= top2) {
        printf("%c", b[n]);
        n = n + 1;
    }
    printf("\t\t");
    // 输出输入串
    while (g <= top3) {
        printf("%c", c[g]);
        g = g + 1;
    }
    printf("\t\t");

    // 查找动作表
    if (x == 'a') j = 0;
    if (x == 'b') j = 1;
    if (x == '#') j = 2;
    if (action[y][j] == NULL) {
        printf("error\n");
        getchar();
        getchar();
        return 0;
    } else {
        strcpy(copy, action[y][j]);
    }
    // 处理移进操作
    if (copy[0] == 'S') {
        z = copy[1] - '0';
        top1 = top1 + 1;
        top2 = top2 + 1;
        a[top1] = z;
        b[top2] = x;
        top = top + 1;
    }
}
```

```

        i = 0;
        while (copy[i] != '#') {
            printf("%c", copy[i]);
            i++;
        }
        printf("\n");
    }
    // 处理归约操作
    if (copy[0] == 'r') {
        i = 0;
        while (copy[i] != '#') {
            printf("%c", copy[i]);
            i++;
        }
        h = copy[1] - '0';
        strcpy(copy1, LR[h]);
        if (copy1[0] == 'S') k = 0;
        if (copy1[0] == 'B') k = 1;
        l = strlen(LR[h]) - 4;
        top1 = top1 - l + 1;
        top2 = top2 - l + 1;
        y = a[top1 - 1];
        p = goto1[y][k];
        a[top1] = p;
        b[top2] = copy1[0];
        z = p;
        printf("\t\t%d\n", p);
    }
} while (action[y][j] != "acc");
printf("acc\n");
getchar();
getchar();
return 0;
}

```

五、运行结果截图

请输入表达式						
bbaa#						
步骤	状态栈	符号栈	输入串	ACTION	GOTO	
1	0	#	bbaa#	S4		
2	04	#b	baa#	r3	2	
3	02	#B	baa#	S7		
4	027	#Bb	aa#	error		

请输入表达式						
bb#						
步骤	状态栈	符号栈	输入串	ACTION	GOTO	
1	0	#	bb#	S4		
2	04	#b	b#	r3	2	
3	02	#B	b#	S7		
4	027	#Bb	#	r3	5	
5	025	#BB	#	r1	1	
6	01	#S	#	acc		