

实验 1 扫描器的设计（4 学时）

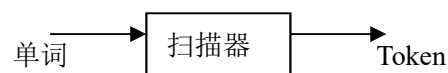
一、 实验目的

熟悉并实现一个扫描器（词法分析程序）。

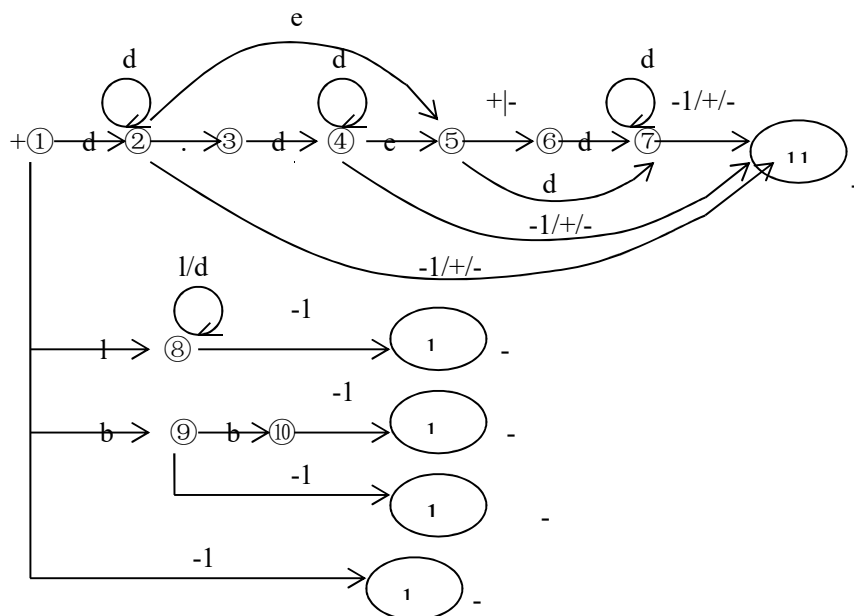
二、 实验内容

- (1) 设计扫描器的有限自动机（识别器）；
- (2) 设计翻译、生成 Token 的算法（翻译器）；
- (3) 编写代码并上机调试运行通过。
 - 输入——源程序文件或源程序字符串；
 - 输出——相应的 Token 序列；
关键字表和界符表；
符号表和常数表；

三、 实验原理及基本步骤



(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）。

状态转换矩阵：

	d	·	E e	+ -	l	b	-1
1	2				8	9	15
2	2	3	5	11			11
3	4						
4	4		5	11			11
5	7			6			
6	7						
7	7			11			11
8	8				8		12
9						10	14
10							13
11							
12							
13							
14							
15							

(2) 关键字表和界符表

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

(3) 主程序流程:

```

初始化;
打开用户源程序文件;
while (文件未结束)
{ 读入一行到 w[i], i=0;
  do                                     //处理一行,每次处理一个单词
  { 滤空格, 直到第一个非空的 w[i];
    i--;
    s=1;                               //处理一个单词开始
  }
}
```

```

        while (s!=0)                                //拼单词并生成相应 Token
        {
            act(s);                                  //执行 qs
            if (s>=11 && s<=14)                      //一个单词处理结束
                break;
            i++;                                      //getchar()
            s=find(s, w[i]);
        }
        if (s==0)
            词法错误;
    }while (w[i]!=换行符);
}
关闭用户源程序文件;
生成 Token 文件;
输出关键字表;
输出 Token 序列;
输出符号表;
输出常数表;

```

四、 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其它变量: 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

```

(4) 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;

//标识符的编码 (code) 为 1, 值 (value) 为其在符号表中的位置; 常数的编码 (code) 为 2, 值 (value) 为其在常数表中的位置; 关键字和界符的编码 (code) 为其在关键字表中的位置, 值 (value) 为 0。

```

    • q11:  $num=n*10e*p-m$ ;          //计算常数值
t[i].code=2;  t[i].value=InsertConst(num);    //生成常数 Token
    • q12: code=Reserve(strTOKEN);          //查关键字表
        if code then { t[i].code=code;  t[i].value=0; } //生成关键字 Token
else {t[i].code=1;  t[i].value=InsertID(strTOKEN); }
//生成标识符 Token
    • q13: code=Reserve(strTOKEN);          //查界符表
        if code then { t[i].code=code;  t[i].value=0; } //生成界符 Token
else {
strTOKEN[strlen(strTOKEN)-1]=' \0' ;    //单界符
源程序缓冲区指针减 1;
        code=Reserve(strTOKEN);          //查界符表
        t[i].code=code;  t[i].value=0;    //生成界符 Token
    }
    • q14: code=Reserve(strTOKEN);          //查界符表
t[i].code=code;  t[i].value=0;          //生成界符 Token
    • q15: stop.

```

(5) 查状态变换表

int find(int s,char ch) //s 是当前状态, ch 是当前字符, 返回值是转换后状态

```

{
查状态转换矩阵 aut[10][7];
返回新状态;
}

```

(6) 主程序流程:

初始化;

打开用户源程序文件;

```

        while (文件未结束)
    { 读入一行到 w[i], i=0;
    do                                     //处理一行，每次处理一个单词
    { 滤空格，直到第一个非空的 w[i];
    i--;
    s=1;                                 //处理一个单词开始
    while (s!=0)                         //拼单词并生成相应 Token
    {
        act(s);                          //执行 qs
        if (s>=11 && s<=14)              //一个单词处理结束
            break;
        i++;                             //getchar()
        s=find(s, w[i]);
    }
    if (s==0)
        词法错误;
    }while (w[i]!=换行符);
}
关闭用户源程序文件;
生成 Token 文件;
输出关键字表;
输出 Token 序列;
输出符号表;
输出常数表;

```

五、 关键代码分析（带注释）及运行结果

```

// scan.cpp : Defines the entry point for the console application.
#include "string.h"
#include "math.h"
#include<stdio.h>
//关键字表和界符表
char keywords[30][12]= {"program","begin","end","var","while","do","repeat",
                        "until","for","to","if","then","else",";",":","(",")"," ",
                        ":", "+", "-", "*", "/", ">", ">=", "==", "<", "<="
                        };

int num_key=28;

//状态转换矩阵
int aut[11][8]= { 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 2, 0, 0, 0, 8, 9, 15,
                  0, 2, 3, 5, 11, 0, 0, 11,
                  0, 4, 0, 0, 0, 0, 0, 0,
                  0, 4, 0, 5, 11, 0, 0, 11,
                  0, 7, 0, 0, 6, 0, 0, 0,
                  0, 7, 0, 0, 0, 0, 0, 0,

```

```

        0, 7, 0, 0, 11, 0, 0, 11,
        0, 8, 0, 0, 0, 8, 0, 12,
        0, 0, 0, 0, 0, 0, 10, 14,
        0, 0, 0, 0, 0, 0, 0, 13
    };

```

//符号表

char ID[50][12]; //表中存有源程序中的标识符

//常数表

int C[20];

int num_ID=0, num_C=0;

//其它变量

struct token {

int code;

int value;

};

//Token 结构

struct token tok[100];

//Token 数组

//int s; //当前状态

int i_token=0, num_token=0;

//Token 计数器和 Token 个数

char strTOKEN[15];

//当前单词//当前已经识别出的单词

int i_str;

//当前单词指针

int n, p, m, e, t;

//尾数值, 指数值, 小数位数, 指数符

号, 类型

double num;

//常数值

char w[50];

//源程序缓冲区

int i;

//源程序缓冲区指针, 当前字符为 w[i]

struct map {

//当前字符到状态转换矩阵列标记的

映射

char str[50];

int col; //状态列

};

struct map col1[4] = {{ "0123456789", 1 }, { ".", 2 }, { "Ee", 3 }, { "+- ", 4 } }; //数字

struct map col2[2] = {{ "abcdefghijklmnopqrstuvwxyz", 5 }, { "0123456789", 1 } }; //关键字或标识符

struct map col3[1] = {{ " ; () , + - * / = > < ", 6 } }; //界符

struct map *ptr;

int num_map; //?

void act(int s);

int find(int s, char ch);

int InsertConst(double num);

```

int Reserve(char *str);
int InsertID(char *str);
void printToken();
void printData();

int main(int argc, char* argv[]) {

    FILE *fp;
    int s; //当前状态 行
    fp=fopen("exa.txt","r");
    while (!feof(fp)) {
//      printf("\n***** 开始之
前行为");
        int zz;
        for (zz=0;zz<50;zz++){
            w[zz]='\n';//手动覆盖所有未清除数据
        }
//      printf("\n");
        fgets(w,50,fp);

        i=0;
//      printf("\n***** 当前行
为");
//      int zz;
//      for (zz=0;zz<50;zz++){
//          printf("%c",w[zz]);
//      }
//      printf("\n");
        do {
            while (w[i]==' ') //滤空格
                i++;
//      printData();
//      printf("#####
当前 w[i]是%c\n",w[i]); //以开头判断单词类型
            if (w[i]>='a' && w[i]<='z') { //判定单词类别
                ptr=col2;
                num_map=2;
                //printf("ptr[num_map] %s\n",ptr->str[num_map]);
            } else if (w[i]>='0' && w[i]<='9') {
                ptr=col1;
                num_map=4;
            } else if (strchr(col3[0].str,w[i])==NULL) {

```

```

        printf("非法字符%c\n",w[i]);
        i++;
        continue;
    } else {
        ptr=col3;
        num_map=1;
    }

    i--;
    s=1;
    //i_str = 0;//

//开始处理一个单词，状态转换
    while (s!=0) {
        act(s);
        if (s>=11 && s<=14)
            break;

        i++;
        s=find(s,w[i]);
    }
    if (s==0) {
        strTOKEN[i_str]='\0';
        printf("词法错误: %s\n",strTOKEN);
    }
//    printToken();
    } while (w[i]!='\n' && w[i]!='\0' );//
}

printf("关键字表: ");
for (i=0; i<30; i++)
    printf("%s ",keywords[i]);
printf("\n");
printf("\n");
printf("Token 序列: ");
for (i=0; i<num_token; i++)
    printf("(%d,%d)",tok[i].code,tok[i].value);
printf("\n");
printf("\n");
printToken();//格式化输出
printf("\n");
printf("符号表: ");
for (i=0; i<num_ID; i++)
    printf("%s ",ID[i]);
printf("\n");

```

//输出结果


```

printf("常数表: ");
for (i=0; i<num_C; i++)
    printf("%d ",C[i]);
printf("\n");

fclose(fp);
printf("Hello World!\n");

return 0;
}

void act(int s) {
//  printf("-----act()-----\n");
    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
//            strTOKEN[15]= {0};
            memset(strTOKEN, '\0', sizeof(strTOKEN));
//            printf("n=%d  m=%d  p=%d  t=%d  e=%d  num=%lf  i_str=%d\n",n,m,p,t,e,num,i_str,strTOKEN);
            strTOKEN=%s \n",n,m,p,t,e,num,i_str,strTOKEN);
//            printf("-----act()---end-----\n");
            break;
        case 2:
            n=10*n+w[i]-48;
//            printf("n=%d  m=%d  p=%d  t=%d  e=%d  num=%lf  i_str=%d\n",n,m,p,t,e,num,i_str,strTOKEN);
            strTOKEN=%s \n",n,m,p,t,e,num,i_str,strTOKEN);
//            printf("-----act()---end-----\n");
            break;
        case 3:
            t=1;
//            printf("-----act()---end-----\n");
            break;
        case 4:
            n=10*n+w[i]-48;
            m++;
    }
}

```

```

//          printf("-----act()---end-----\n");
//          break;
case 5:
    t=1;
//          printf("-----act()---end-----\n");
//          break;
case 6:
    if (w[i]=='-') e=-1;
//          printf("-----act()---end-----\n");
//          break;
case 7:
    p=10*p+w[i]-48;
//          printf("-----act()---end-----\n");
//          break;
case 8:
//          printf("前 w[%d] %c strTOKEN %s i_str %d\n",i,w[i],strTOKEN,i_str);
//          strTOKEN[i_str++]=w[i]; //将 ch 中的符号拼接到 strTOKEN 的尾部;
//
//          printf("后 w[%d] %c strTOKEN %s i_str %d\n",i,w[i],strTOKEN,i_str);
//          printf("n=%d m=%d p=%d t=%d e=%d num=%lf i_str=%d\n",n,m,p,t,e,num,i_str,strTOKEN);
//          printf("-----act()---end-----\n");
//          break;
case 9:
//          printf("前 w[%d] %c strTOKEN %s i_str %d\n",i,w[i],strTOKEN,i_str);
//          strTOKEN[i_str++]=w[i]; //将 ch 中的符号拼接到 strTOKEN 的尾部;
//          printf("后 w[%d] %c strTOKEN %s i_str %d\n",i,w[i],strTOKEN,i_str);
//          printf("n=%d m=%d p=%d t=%d e=%d num=%lf i_str=%d\n",n,m,p,t,e,num,i_str,strTOKEN);
//          printf("-----act()---end-----\n");
//          break;
case 10:
//          printf("前 w[%d] %c strTOKEN %s i_str %d\n",i,w[i],strTOKEN,i_str);
//          strTOKEN[i_str++]=w[i]; //将 ch 中的符号拼接到 strTOKEN 的尾部;
//          printf("后 w[%d] %c strTOKEN %s i_str %d\n",i,w[i],strTOKEN,i_str);
//          printf("n=%d m=%d p=%d t=%d e=%d num=%lf i_str=%d\n",n,m,p,t,e,num,i_str,strTOKEN);
//          printf("-----act()---end-----\n");
//          break;
case 11:
//          printf("num %lf n %d e %d p %d m %d\n",num,n,e,p,m);
//          num=n*pow(10,e*(p-m)); //计算常数值
//          printf("num %lf n %d e %d p %d m %d\n",num,n,e,p,m);
//          tok[i_token].code=2;//常数的编码 (code) 为 2

```

```

        tok[i_token++].value=InsertConst(num); //生成常数 Token
//      printf("tok[i_token++].value %d\n",tok[i_token].value);
        num_token++;
//      printf("-----act()---end-----\n");
        break;
    case 12:
//      printf("case 12:-----\n");
        strTOKEN[i_str]='\0';
//      printf("strTOKEN %s\n",strTOKEN);
        code=Reserve(strTOKEN); //查关键字表//关键字和界符的编码（code）为其在关键字表中的位置，值（value）为0。
//      printf("code %d\n",code);
        if (code) {
            tok[i_token].code=code; //生成关键字 Token
            tok[i_token++].value=0;
        } else {
            tok[i_token].code=1;
            tok[i_token++].value=InsertID(strTOKEN);
        } //生成标识符 Token
        num_token++;
//      printf("case 12:-----\n");
//      printf("n=%d m=%d p=%d t=%d e=%d num=%lf i_str=%d\n",n,m,p,t,e,num,i_str,strTOKEN);
//      printf("-----act()---end-----\n");
        break;
    case 13:
        strTOKEN[i_str]='\0';
        code=Reserve(strTOKEN); //查界符表
        if (code) {
            tok[i_token].code=code; //1 //生成界符 Token
            tok[i_token++].value=0;
        } 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++;
//      printf("n=%d m=%d p=%d t=%d e=%d num=%lf i_str=%d\n",n,m,p,t,e,num,i_str,i_str,strTOKEN);
//      printf("-----act()---end-----\n");
        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++;
//        printf("-----act()---end-----\n");
        break;
    }

}

int find(int s,char ch) {
//    printf("-----find()-----\n");
    int i,col=7;//默认最后一列
    struct map *p;
    p=ptr;
    for (i=0; i<num_map; i++){
//        printf("217 %s %c\n",(p+i)->str,ch);//
        if (strchr((p+i)->str,ch)) {//在 map 中可以找到 ch 的话，值为 ch 第一次出现的位置
            col=(p+i)->col;//将 map 中的 col 赋值给 col，以便找到下一状态
//            printf("220 col %d\n",col);//
            break;
        }
//        printf("223 aut[%d][%d] %d\n",s,col,aut[s][col]);//
//        printf("-----find()---end-----\n");
        return aut[s][col];
    }
}

int InsertConst(double num) {
    int i;
    for (i=0; i<num_C; i++)
        if (num==C[i])
            return i;
    C[i]=num;
    num_C++;
    return num_C;//i
}

int Reserve(char *str) {
//    printf("Reserve----->\n");
    int i;
    for (i=0; i<num_key; i++){
//        printf("keywords[%d] %s str %s\n",i,keywords[i],str);
        if (!strcmp(keywords[i],str))//strcmp(str1,str2)，若 str1=str2，则返回零；若
str1<str2，则返回负数；若 str1>str2，则返回正数。

```

```

        return (i); //return (i+3);
    }
// printf("<-----Reserve\n");
return 0;
}

int InsertID(char *str) {
    int i;
// printf("ID %s\n",ID);
    for (i=0; i<num_ID; i++)
        if (!strcmp(ID[i],str))
            return i + 1; //不是下标
    strcpy(ID[i],str);
    num_ID++;
// printf("ID %s\n",ID);
    return num_ID; //i
}

void printToken(){
    //输出 token
    printf("Token 序列: ");
    for (i=0; i<num_token; i++){
        if(!tok[i].value){
            printf("(%s,%d)",keywords[tok[i].code],tok[i].value); //关键字
        } else if (tok[i].code == 2){
            printf("(%d,%d)",tok[i].code,C[tok[i].value-1]); //常数
        } else if (tok[i].code == 1){
            printf("(%d,%s)",tok[i].code,ID[tok[i].value-1]); //标识符
        } else printf("#%d,%d#",tok[i].code,tok[i].value);
    }
    printf("\n");
}

void printData(){
//printf("*****\n");
int i;
printf("C[] ");
for (i = 0; i < num_C; i++){
    printf(" %d ",C[i]);
}
printf("\nID[] ");
for (i = 0; i < num_ID; i++){
    printf(" %s ",ID[i]);
}
//printf("\n*****\n");
}

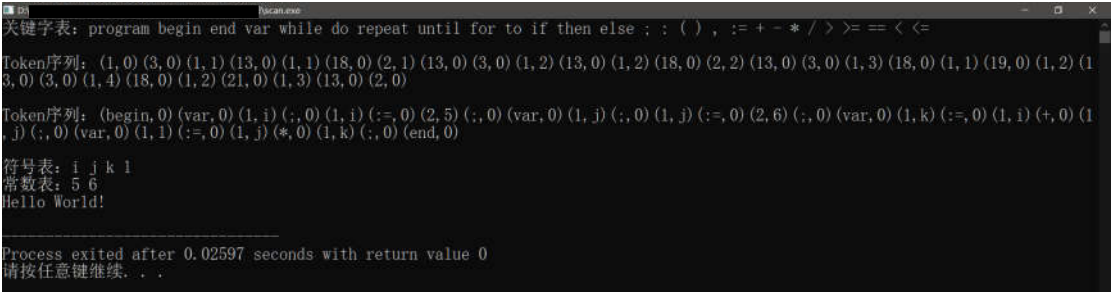
```

运行结果：



```
exa.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
begin
var i;
i:=5;
var j;
j:=6.66;
var k:=i+j;
var l:=j*k;
end
```

图 1 exa.txt 源文件



```
关键字表: program begin end var while do repeat until for to if then else ; ( ) , := + - * / > < >= <=
Token序列: (1,0) (3,0) (1,1) (13,0) (1,1) (18,0) (2,1) (13,0) (3,0) (1,2) (13,0) (1,2) (18,0) (2,2) (13,0) (3,0) (1,3) (18,0) (1,1) (19,0) (1,2) (13,0) (3,0) (1,4) (18,0) (1,2) (21,0) (1,3) (13,0) (2,0)
Token序列: (begin,0) (var,0) (1,i) (:,0) (1,i) (:,0) (2,5) (:,0) (var,0) (1,j) (:,0) (1,j) (:,0) (2,6) (:,0) (var,0) (1,k) (:,0) (1,i) (+,0) (1,j) (:,0) (var,0) (1,l) (:,0) (1,j) (*,0) (1,k) (:,0) (end,0)
符号表: i j k l
常数表: 5 6
Hello World!
Process exited after 0.02597 seconds with return value 0
请按任意键继续. . .
```

图 2 程序运行结果展示

六、 总结与分析

(1) 扫描器的任务是什么？

扫描器读入组成源程序的字符流，并且将他们组织成为有意义的词素的序列。对于每个词素，扫描器产生<token-name, attribute-value>形式的词法单元（token）作为输出。

(2) 扫描器、识别器、翻译器三者之间的关系是怎样的？

扫描器——扫描程序，完成词法分析；识别器——识别单词的有限自动机；翻译器——根据有限自动机所识别出的对象，完成从单词串到单词的 TOKEN 串的翻译。
而扫描器的实现要通过识别器和翻译器的共同实现。

(3) 为什么说有限自动机是词法分析的基础？

有限自动机理论与正规文法,正规式之间在描述语言方面有一一对应的关系。不论是翻译器的识别还是翻译，都离不开自动机。

实验 2 中间代码生成器的设计（4 学时）

七、 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。

八、 实验内容

- 1、 形式语言基础及其文法运算
- 2、 语法分析原理以及 3 种常用的语法分析方法
- 3、 语义分析原理

九、 实验原理及基本步骤

- (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) 本实验已给出递归子程序法的四元式属性翻译文法的设计，鼓励学生在此基础上进行创新，即设计 LL(1)分析法或 LR(0)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。

十、 数据结构设计

(1) 算术表达式文法

$G(E):$

$$\begin{aligned} E &\rightarrow E \omega_0 T \mid T \\ T &\rightarrow T \omega_1 F \mid F \\ F &\rightarrow i \mid (E) \end{aligned}$$

(2) 文法变换

$G'(E)$

$$\begin{aligned} E &\rightarrow T \{ \omega_0 T \} \\ T &\rightarrow F \{ \omega_1 F \} \\ F &\rightarrow i \mid (E) \end{aligned}$$

(3) 属性翻译文法：

$$\begin{aligned} E &\rightarrow T \{ \omega_0 \text{"push(SYN, w)"} T \text{"QUAT"} \} \\ T &\rightarrow F \{ \omega_1 \text{"push(SYN, w)"} F \text{"QUAT"} \} \\ F &\rightarrow i \{ \text{"push(SEM, entry(w))"} \} \mid (E) \end{aligned}$$

其中：· push(SYN, w) — 当前单词 w 入算符栈 SYN；

· push(SEM, entry(w)) — 当前 w 在符号表中的入口值压入语义栈

SEM；

· QUAT — 生成四元式函数

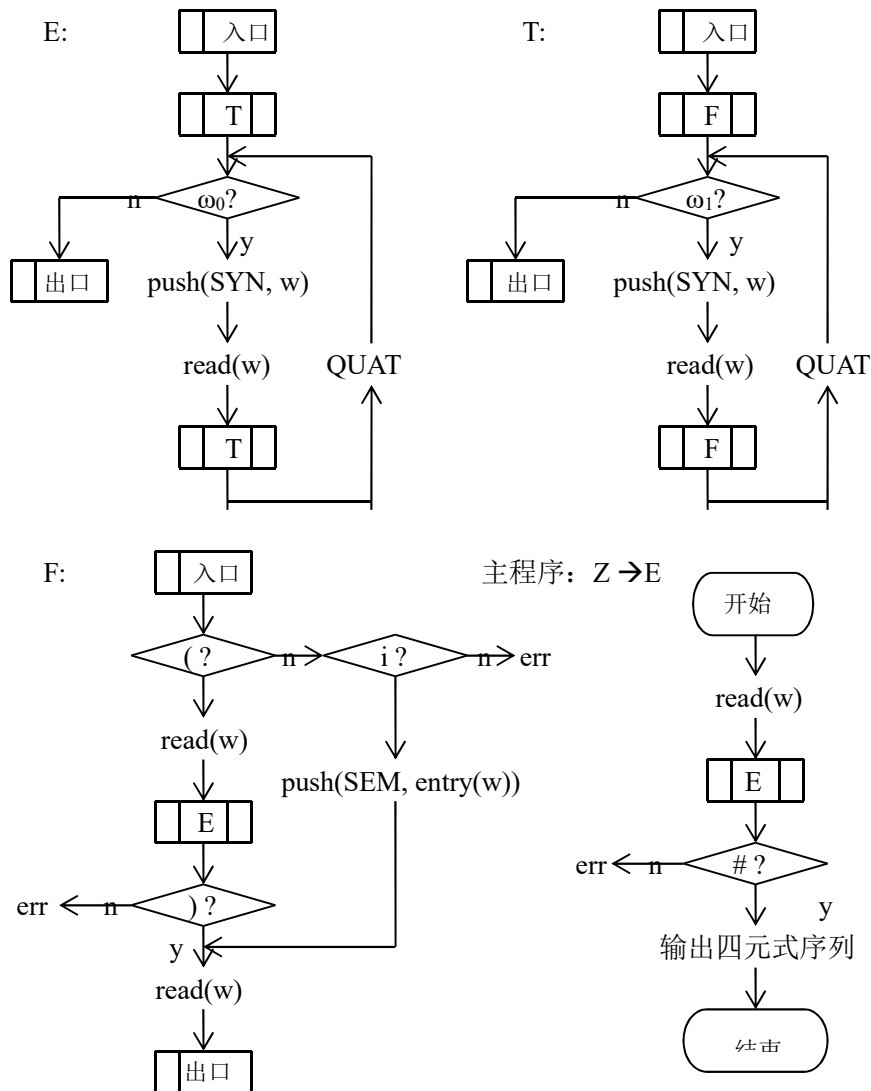
i. $T = \text{newtemp}$;

ii. $QT[j] = (SYN[k], SEM[s-1], SEM[s], T)$; $j++$;

iii. pop(SYN, _); pop(SEM, _); pop(SEM, _);
push(SEM, T);

(4) 递归下降子程序:

·数据结构: SYN — 算符栈;
SEM — 语义栈;



十一、 关键代码分析（带注释）及运行结果

```
#include<stdio.h>
#include<iostream>
using namespace std;
int t;
string stack[50];
int top=0;                                     //符号栈
                                              //栈顶指针
```



```

string its();
char factor(char word);
char expr(char word);
char term(char word);
bool automat();

int main() {
    t=1;
    cout<<"输入表达式，以#结束："<<endl;
    automat();
}

string its(int a) {                //int to string
    string d;
    char b='0',c;
    int i;
    while(a!=0) {
        i=a%10;
        a=a/10;
        c=(int)b+i;
        d=c+d;
    }
    return d;
}

char factor(char word) {           //自动机
    string theWord;
    if (word>='a'&&word<='z'||word>='A'&&word<='automat') { //当前字符是
字母
        theWord=word;
        stack[++top]=theWord;
    } else if (word=='(') {        //(
        cin>>word;                 //读取下一个字符
        word=expr(word);
        if (word!=')') {
            cerr<<"输入错误 1！"<<endl;
            exit(0);
        }
    } else {
        cout<<word<<endl;
        cerr<<"输入错误 2！"<<endl;
        exit(0);
    }
    cin>>word;                     //读取下一个字符

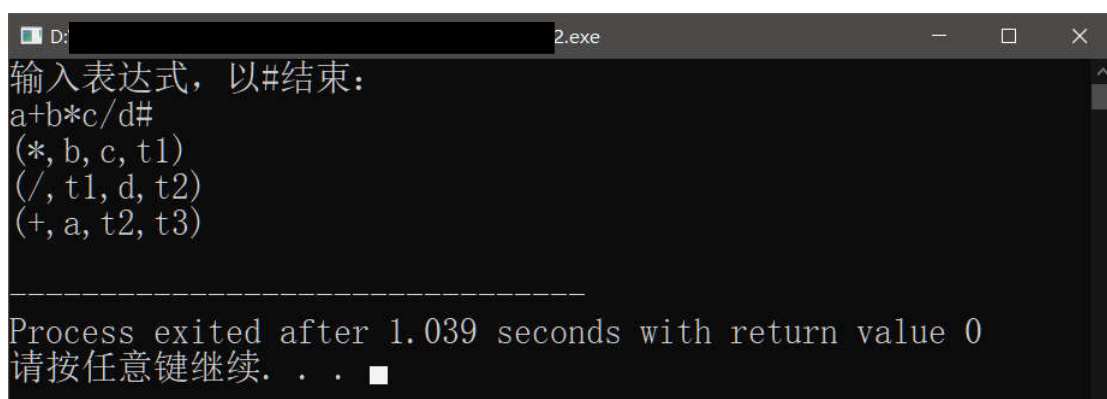
```

```

    return word;
}
char expr(char word) {                                //自动机 expr
    string operate,a,b,c;
    string state[5];
    word=term(word);
    while(word=='+'||word=='-') { //当前是+-
        operate=word;
        cin>>word;
        word=term(word);
        b=stack[top--];
        a=stack[top--];
        cout<<"("<<operate<<","<<a<<","<<b<<","<<t<<")"<<endl;
        c="t"+its(t);
        stack[++top]=c;
        t++;
    }
    return word;
}
char term(char word) {
    string operate,a,b,c;
    string state[5];
    word=factor(word);
    while(word=='*'||word=='/') { //当前是*/
        operate=word;
        cin>>word;
        word=factor(word);
        b=stack[top--];
        a=stack[top--];
        cout<<"("<<operate<<","<<a<<","<<b<<","<<t<<")"<<endl;
        c="t"+its(t);
        stack[++top]=c;
        t++;
    }
    return word;
}
bool automat() {                                     //automat 自动机 s
    char word;
    cin>>word;
    word=expr(word);
    if(word=='#')                                    //结束
        return true;
    else return false;
}

```

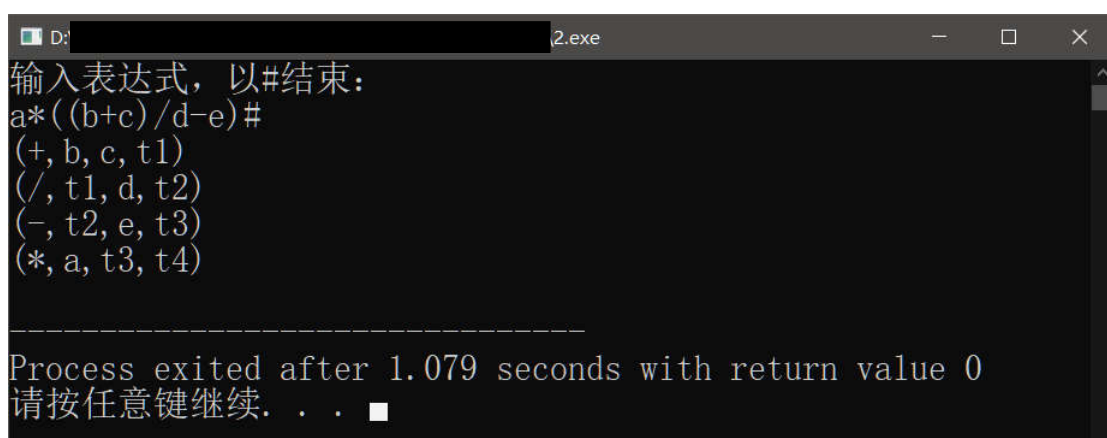
运行结果：



```
D:\2.exe
输入表达式，以#结束：
a+b*c/d#
(*, b, c, t1)
(/, t1, d, t2)
(+, a, t2, t3)

-----
Process exited after 1.039 seconds with return value 0
请按任意键继续. . . ■
```

图 3 程序运行结果展示 1



```
D:\2.exe
输入表达式，以#结束：
a*((b+c)/d-e)#
(+, b, c, t1)
(/, t1, d, t2)
(-, t2, e, t3)
(*, a, t3, t4)

-----
Process exited after 1.079 seconds with return value 0
请按任意键继续. . . ■
```

图 4 程序运行结果展示 2

十二、总结与分析

(1) 语法分析分为几类？其关键技术各是什么？

自顶向下：从语法分析树的顶部（根节点）开始向底部（叶子节点）构造语法分析树；

自底向上：从叶子节点开始，逐渐向根节点构造。

(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？

i. 递归下降子程序法：

1. 递归子程序法属于自顶向下语法分析方法。故又名递归下降法。

ii. LL(1)分析法：

1. LL(1)分析法是指从左到右扫描(第一个 L)、最左推导(第二个 L)和只查看一个当前符号(括号中的 1)之意；

2. 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。

iii. 递归下降子程序法和 LL(1)分析法都要求文法是 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：

1. 都要求文法是 LL(1)文法；

2. 都是自顶向下的分析方法；

3. 都通过分析下个字符来判断该进入哪个状态或者调用哪个函数。

不同点:

1. 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;
2. 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, 每识别出一个语法结构, 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(m)--语义栈(属性传递、赋值场所);

QT[q]-四元式区;

$G(E): E \rightarrow T \mid E+T \{GEQ(+)\} \mid E-T \{GEQ(-)\} \mid T \rightarrow F \mid T * F \{GEQ(*)\} \mid T / F \{GEQ(/)$

$F \rightarrow i \{PUSH(i)\} \mid (E)$

其中:

(1)PUSH(i)-压栈函数(把当前 i 压入语义栈);

(2)GEQ(w)-表达式四元式生成函数:

生成一个四元式送 QT[q]过程:

1. $t := NEWT; \{ \text{申请临时变量函数}; \}$
2. $SEND(w, SEM[m-1], SEM[m], t)$
3. $POP; POP; PUSH(t)$