

# 天津理工大学实验报告

学院名称：计算机科学与工程学院

姓名	王路耀	学号	20152216	专业	计算机科学与技术
班级	15 级 2 班	实验项目	实验一：词法分析		
课程名称	编译原理		课程代码	0668056	
实验时间	2018 年 5 月 23 日 第*、*节 2018 年 5 月 28 日 第*、*节		实验地点	软件实验室 7-219 软件实验室 7-219	
实验成绩考核评定分析					
实验过程 综合评价 30 分	实验目标 结果评价 20 分	程序设计 规范性评价 20 分	实验报告 完整性评价 30 分	实验报告 雷同分析 分类标注	实验 成绩
■实验过程认真专注，能独立完成设计与调试任务 30 分 ■实验过程认真，能较好完成设计与编成调试任务 25 分 ■实验过程较认真，能完成设计与编成调试任务 20 分 ■实验过程态度较好，基本完成设计与编成调试任务 15 分 ■实验过程态度欠端正，未完成设计与编成调试任务 10 分	■功能完善，且人机交互界面友好 20 分 ■满足功能要求，但人机交互界面一般 15 分 ■基本满足功能需求，人机交互界面欠缺 10 分 ■功能缺失 5 分	■程序易读性好 20 分 ■程序易读性较好 15 分 ■程序易读性欠缺 10 分 ■程序易读性较差 5 分 **注：易读性要求标识符命名见名知意，程序编制采用嵌套方式，层次结构清晰可读，关键部分具有简明注释。	■报告完整 30 分 ■报告较完整 25 分 ■报告内容一般 20 分 ■报告内容极少 10 分	凡雷同报告将不再重复评价前四项考核内容，实验成绩将按低学号雷同学生成绩除雷同人数计算而定。 标记为： <b>S 组号-人数(组分)</b>	前四项评价分数之总和 (**雷同报告按第五项标准核算**)

**实验内容：**  
实现标准 C 语言词法分析器

**实验目的：**

1. 掌握程序设计语言词法分析的设计方法；
2. 掌握 DFA 的设计与使用方法；
3. 掌握正规式到有限自动机的构造方法；

**实验要求：**

1. 单词种别编码要求  
基本字、运算符、界符：一符一种；识符：统一为一种；常量：按类型编码；
2. 词法分析工作过程中建立符号表、常量表，并以文本文件形式输出；
3. 词法分析的最后结果以文本文件形式输出；
4. 完成对所设计词法分析器的功能测试，并给出测试数据和实验结果；
5. 为增加程序可读性，请在程序中进行适当注释说明；
6. 整理上机步骤，总结经验和体会；
7. 认真完成并按时提交实验报告。

## 词法分析器的作用

词法分析是编译的第一阶段。词法分析器的主要任务是读入源程序的输入字符，将它们组成词素，生成并输出一个词法单元序列，这个词法单元序列被输出到语法分析器进行语法分析。另外，由于词法分析器在编译器中负责读取源程序，因此除了识别词素之外，它还会完成一些其他任务，比如过滤掉源程序中的注释和空白，将编译器生成的错误消息与源程序的位置关联起来等。总而言之，词法分析器的作用如下：

1. 读入源程序的输入字符，将它们组成词素，生成并输出一个词法单元序列；
2. 过滤掉源程序中的注释和空白；
3. 将编译器生成的错误消息与源程序的位置关联起来；
4. 其它。

## 词法分析过程

首先，对某个正则语言  $L$ ，构造能够描述其的正则表达式  $r$ ；

然后，需要将  $r$  转换成一个有穷自动机。这里有三种方法，一是直接转换成 NFA，而是直接转换成 DFA，三是先转换成 NFA，再把 NFA 转换成 DFA；

最后，如果将  $r$  转换成了一个 DFA，需要将此 DFA 的状态数最小化。

## 正则表达式

正则表达式可以用来描述词素的模式，一个正则表达式可以由较小的正则表达式递归的构建

。对于符号集合  $\Sigma = \{a, b\}$ ，有：

- 正则表达式  $a$  表示语言  $\{a\}$ ；
- 正则表达式  $a|b$  表示语言  $\{a, b\}$ ；
- 正则表达式  $(a|b)(a|b)$  表示语言  $\{aa, ab, ba, bb\}$ ；
- 正则表达式  $a^*$  表示语言  $\{\epsilon, a, aa, aaa, \dots\}$ ；
- 正则表达式  $(a|b)^*$  表示语言  $\{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$ ；
- 正则表达式  $a|a^*b$  表示语言  $\{a, b, ab, aab, aaab, \dots\}$ 。

上面通过基本的并、连接和闭包运算递归定义了正则表达式

## 有穷自动机

一个有穷自动机可以把一个描述词素的模式变成一个词法分析器，从本质上来讲，有穷自动机是与状态转换图相类似的图，它有以下特点：

有穷自动机是一个识别器，它只能对每个输入符号串简单的输出 “yes” 或 “no”，表示是否能够识别此符号串；

有穷自动机和状态转换图类似，它具有有限个数的结点，每个结点表示一个状态，并且这些状态中有一个初始状态和若干个终止状态。从一个状态  $s$  开始，经过被某个符号  $a$ （可能包括  $\epsilon$ ）标记的有向边，可以到达另一个状态  $t$  或者回到状态  $s$ （成环）；

有穷自动机分为不确定的有穷自动机和确定的有穷自动机，不确定的和确定的有穷自动机能识别的语言集合是相同的。

## 从 NFA 到 DFA 的转换

NFA 抽象地表示了用来识别某个语言中的串的算法，而相应的 DFA 则是一个简单具体的识别串的算法。在构造词法分析器时，真正实现或模拟的是 DFA。本节先不论如何从一个正则表达式构建一个有穷自动机，而是讨论如何从一个 NFA 转换得到相应的 DFA。

从 NFA 转换得到一个 DFA 通常使用子集构造法 (subset construction)。子集构造法的基本思想是让构造得到的 DFA 的每个状态对应于 NFA 的一个状态集合，下面对其进行说明。

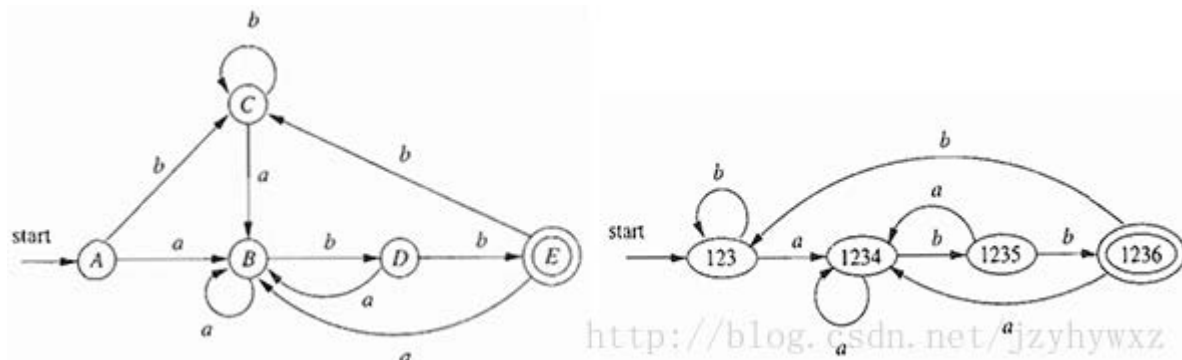
由 NFA 构建 DFA 的子集构造算法，输入一个 NFA  $N$ ，输出一个接受同样语言的 DFA  $D$ 。在此期间会为  $D$  构造一个转换表  $Dtran$ ， $D$  的每个状态是  $N$  的状态的一个子集，也就是说， $D$  的一个状态，是在  $N$  中从状态  $s$  开始经过标号为  $a$  或者  $\epsilon$  的边能够到达的所有状态的集合，为此，我们需要认识在 NFA 状态集上的操作：

我们必须找出当  $N$  读入了某个输入串之后可能位于的所有状态集合。首先，在读入第一个输入符号之前， $N$  可以位于集合  $\epsilon\text{-closure}(s_0)$  中的任何状态上 ( $s_0$  是  $N$  的初始状态)；接着，假定  $N$  在读入串  $x$  之后位于集合  $T$  中的状态上，如果下一个输入符号为  $a$ ，那么  $N$  可以移动到集合  $move(T, a)$  中的任何状态上，又因为  $N$  可以在读入  $a$  后再执行几个  $\epsilon$  转换，所以  $N$  在读入串  $xa$  后可以位于  $\epsilon\text{-closure}(move(T, a))$  中的任何状态上

## 最小化 DFA 的状态数

对同一个语言来说，可能存在多个识别此语言的 DFA。例如对正则表达式  $(a|b)^*abb$ ，下面两个 DFA 都能识别它：

这两个 DFA 不仅每个状态的名字不同，而且连状态数也不一样。实际上，任何正则语言都有一个唯一的数目最少的 DFA，本节就将介绍如何把一个 DFA 的状态数最小化。



在一个 DFA 中，如果有这样两个状态：

都不是终止状态；

对任意输入总是转移到同一个状态。

那么这两个状态是等价的。例如在上面提到的正则表达式  $(a|b)^*abb$  的左边的 DFA 中，状态 A 和 C 是等价的，因为它们都不是终止状态，并且对输入  $a$  都转移到状态 B，对输入  $b$  都转移到状态 C。

这就给出了一个最小化 DFA 状态数的思路——把等价的状态合并。因此我们有下面的算法：

输入：一个 DFA  $D_1$ ，其状态集合是  $S$ ，输入符号集合为  $\Sigma$ ，初始状态为  $s_0$ ，终止状态集合为  $F$ ；

输出：一个 DFA  $D_2$ ，它和  $D_1$  接受相同的语言，且状态数最少；

方法:

构造包含两个组  $F$  和  $S-F$  的初始划分  $\Pi$ , 这两个组分别是  $D1$  的终止状态组和非终止状态组;  
应用下图中的方法构造新的分划  $\Pi_{new}$ :

20171021\_img12

如果  $\Pi_{new} = \Pi$ , 令  $\Pi_{final} = \Pi$  并跳到步骤 4, 否则重复步骤 2;

在分划  $\Pi_{final}$  的每个组中选一个状态作为该组的代表, 这些代表构成了  $D2$  的状态集。 $D2$  的初始状态是包含  $D$  的初始状态组的代表,  $D2$  的终止状态是包含  $D$  的终止状态组的代表。令  $s$  是  $\Pi_{final}$  中某个组的代表, 在  $D1$  中从  $s$  经输入  $a$  到达状态  $t$ , 令  $r$  是  $t$  所在组的代表, 则在  $D2$  中有一个从  $s$  经输入  $a$  到达  $r$  的转换。

对于上面给出的正则表达式  $(a|b)^*abb$  左边的 DFA, 最小化其状态数的过程如下:

初始分划  $\Pi$  为  $\{A, B, C, D\}$  和  $\{E\}$  两个组, 分别是非终止状态组和终止状态组;

在  $\Pi$  中,  $\{E\}$  无法再分, 需要对  $\{A, B, C, D\}$  进行划分。对输入  $a$ ,  $A, B, C, D$  都转移到  $B$ ; 对输入  $b$ ,  $A$  和  $C$  转移到  $C$ ,  $B$  转移到  $D$ ,  $D$  转移到  $E$ , 由于  $A, B, C$  对输入  $b$  都转移到同一个组  $\{A, B, C, D\}$  中的状态, 而  $D$  对输入  $b$  转移到另一个组  $\{E\}$  中的状态, 因此把  $\{A, B, C, D\}$  划分成  $\{A, B, C\}$  和  $\{D\}$ 。此时  $\Pi_{new}$  为  $\{A, B, C\}$ 、 $\{D\}$  和  $\{E\}$ ;

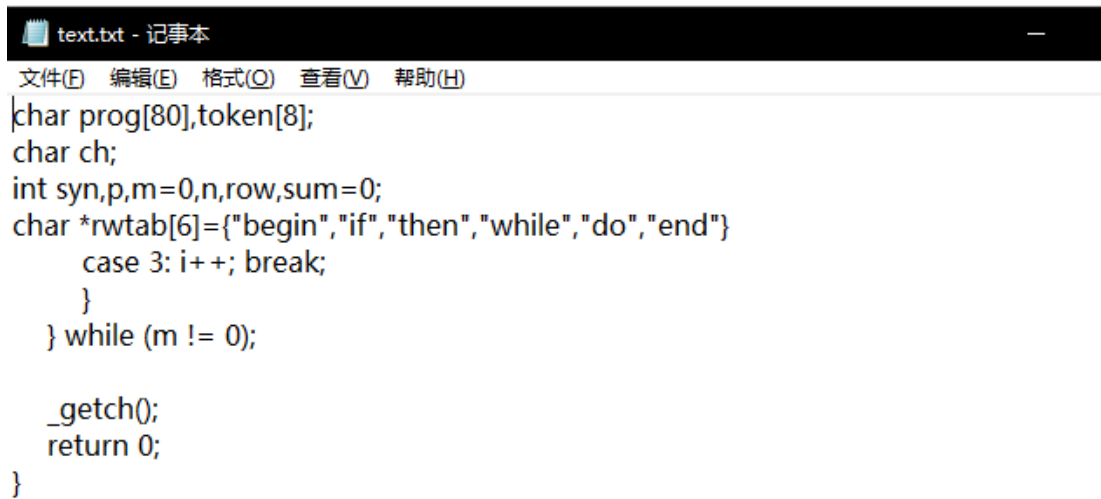
在  $\Pi_{new}$  中, 需要对  $\{A, B, C\}$  进行划分。对输入  $a$ ,  $A, B, C$  都转移到  $B$ ; 对输入  $b$ ,  $A$  和  $C$  转移到  $C$ ,  $B$  转移到  $D$ , 因此把  $\{A, B, C\}$  划分成  $\{A, C\}$  和  $\{B\}$ 。此时  $\Pi_{new}$  为  $\{A, C\}$ 、 $\{B\}$ 、 $\{D\}$  和  $\{E\}$ ;

在  $\Pi_{new}$  中, 此时所有组都不能再分, 因此此时的  $\Pi_{new}$  就是  $\Pi_{final}$ 。合并状态  $A$  和  $C$  后得到的状态数最少的 DFA 就是右边的 DFA。

## 实验记录:

```
C:\Users\57447\Desktop\学习\编译原理\必修\编译原理_实验报告模板\cp1.exe
char      (12, -) 保留字
prog      (12, 1) 标识符
[         (30, -) 界符
80        (13, 1) 数字
]         (31, -) 界符
,         (34, -) 界符
token     (12, 2) 标识符
[         (30, -) 界符
8         (13, 2) 数字
]         (31, -) 界符
;         (25, -) 界符
char      (12, -) 保留字
ch        (12, 3) 标识符
;         (25, -) 界符
int       (13, -) 保留字
syn       (12, 4) 标识符
,         (34, -) 界符
p         (12, 5) 标识符
,         (34, -) 界符
m         (12, 6) 标识符
=         (14, -) 运算符
0         (13, 3) 数字
,         (34, -) 界符
n         (12, 7) 标识符
,         (34, -) 界符
row       (12, 8) 标识符
,         (34, -) 界符
sum       (12, 9) 标识符
=         (14, -) 运算符
0         (13, 3) 数字
```

```
C:\Users\57447\Desktop\学习\编译原理\必修\编译原理_实验报告模板\cp1.
"         (33, -) 界符
,         (34, -) 界符
"         (33, -) 界符
then      (12, 12) 标识符
"         (33, -) 界符
,         (34, -) 界符
"         (33, -) 界符
while     (20, -) 保留字
"         (33, -) 界符
,         (34, -) 界符
"         (33, -) 界符
do        (19, -) 保留字
"         (33, -) 界符
,         (34, -) 界符
"         (33, -) 界符
end       (12, 13) 标识符
"         (33, -) 界符
}         (29, -) 界符
case      (12, 14) 标识符
3         (13, 4) 数字
:         (24, -) 界符
i         (12, 15) 标识符
+         (20, -) 运算符
+         (20, -) 运算符
;         (25, -) 界符
break     (18, -) 保留字
;         (25, -) 界符
}         (29, -) 界符
}         (29, -) 界符
while     (20, -) 保留字
```



```
char prog[80],token[8];
char ch;
int syn,p,m=0,n,row,sum=0;
char *rwtab[6]={"begin","if","then","while","do","end"}
    case 3: i++; break;
    }
} while (m != 0);

_getch();
return 0;
}
```

## 附录：源程序

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

int i, row = 0, line = 0;
char a[1000]; //程序
int number[1000][100]; //常数表
char mark[100][5]; //标识符表

//词法分析
int wordanalysis()
{
    if ((a[i] >= 'A' && a[i] <= 'Z') || (a[i] >= 'a' && a[i] <= 'z')) //分析标识符和保留字
    {
        char word[10];
        char pro[100][100] = { "PROGRAM", "BEGIN", "END", "VAR", "INTEGER", "WHILE", "IF",
"THEN", "ELSE", "DO", "PROCEDURE" ,
"char", "int", "if", "else", "var" , "return", "break", "do", "while", "for", "double", "float", "short"}; //保留字表

        int n = 0;
        word[n++] = a[i++];
        //若字符为 A~Z 或 0~9，则继续读取
        while ((a[i] >= 'A' && a[i] <= 'Z') || (a[i] >= '0' && a[i] <= '9') || (a[i] >= 'a' && a[i] <= 'z'))
        {
            word[n++] = a[i++];
        }
        word[n] = '\0';
        i--;

        //判断该标识符是否为保留字
        for (n = 0; n < 100; n++)
        {
            if (strcmp(word, pro[n]) == 0)
            {
                printf("%s\t(%d,-) 保留字\n", pro[n], n + 1);
                return 3;
            }
        }
    }
}
```

```

//判断标识符长度是否超出规定
if (strlen(word)>10)
{
    printf("%s\\tERROR\\n",word);
    return 0;
}

//判断该标识符是否存在标识符表中
int m = 0;
if (line != 0)
{
    int q = 0;
    while (q<line)
    {
        if (strcmp(word, mark[q++]) == 0)
        {
            printf("%s\\t(12,%d) 标识符\\n", word, q);
            return 3;
        }
    }
}

//将该标识符保存到标识符表中
strcpy(mark[line], word);

printf("%s\\t(12, %d) 标识符\\n", word, line + 1);
line++;
return 3;
}

else if (a[i] >= '0' && a[i] <= '9') //分析常数
{
    char x[100];
    int n = 0, sum;
    x[n++] = a[i++];
    //判断字符是否是 0~9
    while (a[i] >= '0' && a[i] <= '9')
    {
        x[n++] = a[i++];
    }
    x[n] = '\\0';
    i--;
}

```



```

int num = atoi(x); //将字符串转换成 int 型

//判断该常数是否存在于常数表中
if (row != 0)
{
    int y;
    for (y = 0; y < 1000; y++)
    {
        int w = number[y][0];
        sum = 0;
        int d;
        for (d = 1; d <= number[y][0]; d++)
        {
            w = w - 1;
            sum = sum + number[y][d] * pow(2, w);
        }
        if (num == sum)
        {
            printf("%d\t(13,%d)\n", num, y + 1);
            return 3;
        }
    }
}

int z = num, c = num;
int m = 0;
do //计算是几位二进制数
{
    z = z / 2;
    m++;
} while (z != 0);

for (n = m; n > 0; n--) //将二进制保存于常数表中
{
    number[row][n] = c % 2;
    c = c / 2;
}
number[row][0] = m;

int line = row;
printf("%d\t(13,%d)\n", num, line + 1);
row++;

return 3;
}

```

```

else                                     //分析符号
switch (a[i])
{
case ' ':
case '\n':
    return -1;
case '#': return 0;
case '=': printf("=\t(14,-)\n"); return 3;
case '<':
    i++;
    if (a[i] == '=')
    {
        printf("<=\t(16,-)\n");
        return 3;
    }
    else if (a[i] == '>')
    {
        printf("<>\t(19,-)\n");
        return 3;
    }
    else
    {
        i--;
        printf("<\t(15,-)\n");
        return 3;
    }
case '>':
    i++;
    if (a[i] == '=')
    {
        printf(">=\t(18,-)\n");
        return 3;
    }
    else
    {
        i--;
        printf(">\t(17,-)\n");
        return 3;
    }
case '+': printf("+\t(20,-)\n"); return 3;
case '-': printf("-\t(21,-)\n"); return 3;
case '*': printf("*\t(22,-)\n"); return 3;
case '/':
    i++;
    if(a[i]!='/') {

```

```

        i--;
        printf("\t(23,-)\n"); return 3;
    }

    else{

        while(1){
            if(a[i++]=="\n")
                return -1;
        }
        printf("//\t(35,-)\n");return 3;

    }

    case '.': printf("\t(24,-)\n"); return 3;
    case ';': printf("\t(25,-)\n"); return 3;
    case '(': printf("\t(26,-)\n"); return 3;
    case ')': printf("\t(27,-)\n"); return 3;
    case '{': printf("\t(28,-)\n"); return 3;
    case '}': printf("\t(29,-)\n"); return 3;
    case '[': printf("\t(30,-)\n"); return 3;
    case ']': printf("\t(31,-)\n"); return 3;
    case '|': printf("\t(32,-)\n"); return 3;
    case '"': printf("\t(33,-)\n"); return 3;
    case ',': printf("\t(34,-)\n"); return 3;
    case '\\': printf("\t(36,-)\n"); return 3;//单引号
    case '&':
        i++;
        if(a[i]!='&'){
            i--;
            printf("&\t(37,-)\n"); return 3;
        }
        else{
            printf("&&\t(38,-)\n");return 3;

        }
    case '\\': printf("\\\t(39,-)\n"); return 3;
}

}

int main()
{

    int l = 0;

```

```

int m;
i = 0;
FILE *fp;
fp=fopen("D:\\text.txt", "r");
if (fp == NULL)
{
    printf("Can't open file!\n");
    exit(0);
}

while (!feof(fp))
{
    a[l++] = fgetc(fp);
}
a[l] = '#';
do
{
    m = wordanalysis();

    switch (m)
    {
        case -1: i++; break;
        case 0: i++; break;
        case 3: i++; break;
    }
} while (m != 0);

_getch();
return 0;
}

```