电 子 科 技 大 学

实 验 报 告

学生姓名：郭志猛　　学 号：2017080201005　指导教师：陈文宇

实验地点：　HOME　　实验时间：2020.6.1

一、实验室名称：HOME

二、实验项目名称：词法分析器的设计与实现

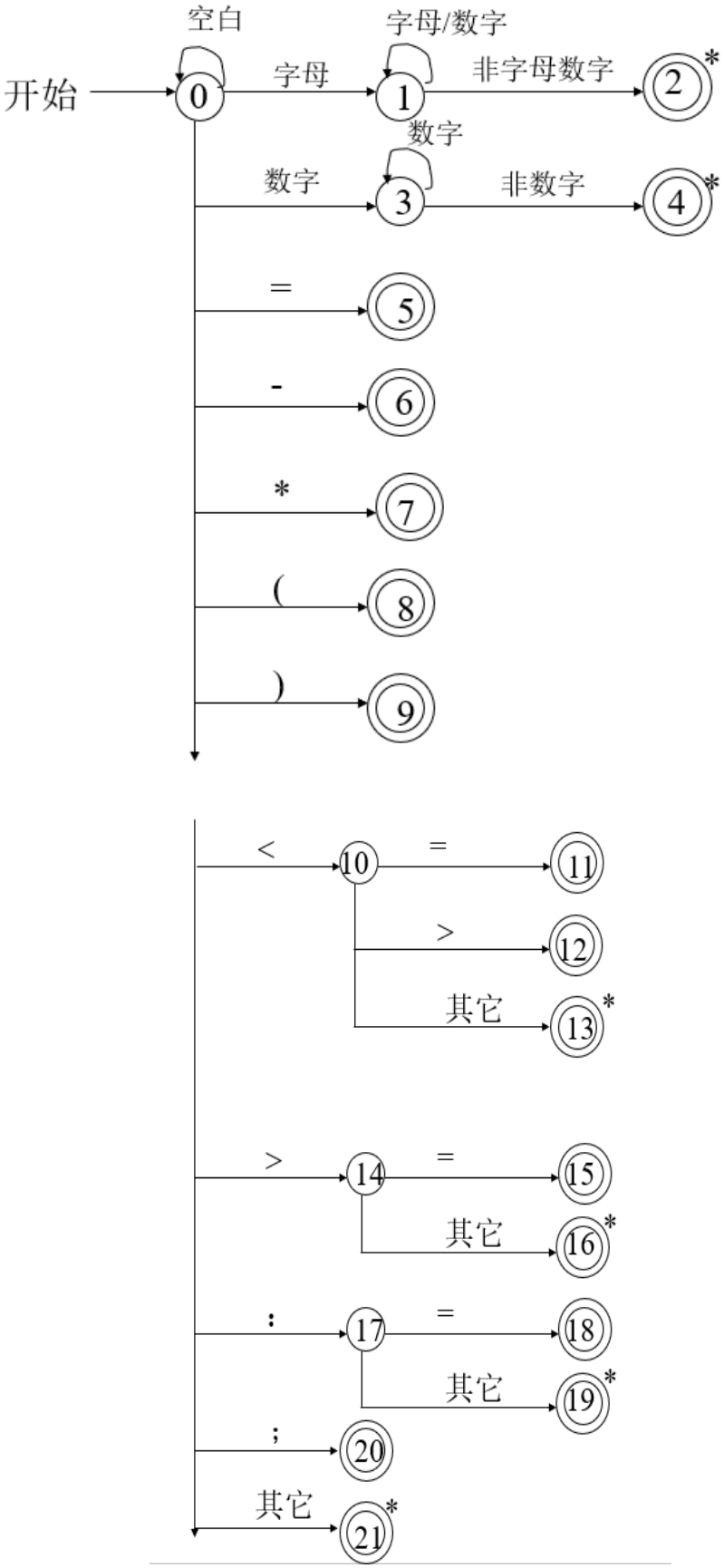
三、实验学时：4 学时

四、实验原理

词法分析的过程中，我们从左到右逐个地扫描源程序字符串。根据词法规则，利用状态转换图，将字符串与种别表比较识别出单词符号作为输出。并且，将过程中发现的词法错误输出。

单词符号	种别	单词符号	种别	单词符号	种别
begin	1	end	2	integer	3
if	4	then	5	else	6
function	7	read	8	write	9
标识符	10	常数	11	=	12
<>	13	<=	14	<	15
>=	16	>	17	-	18
*	19	:=	20	(	21
)	22	;	23		

图表 1：单词符号与种别对照表


空白
字母/数字
开始
0
字母
1
非字母数字
2
*
数字
数字
3
非数字
4
*
=
5
-
6
*
7
(
8
)
9
<
10
=
11
>
12
其它
13
*
>
14
=
15
其它
16
*
:
17
=
18
其它
19
*
;
20
其它
21
*

图表 2：状态转换图

五、实验目的

为了掌握词法分析的方法，熟悉编程实现词法分析器的过程。并且能够利用分析器扫描源程序的代码，对其进行词法分析，输出二元式，并找出其中的词法错误。

六、实验内容

变成实现词法分析器，扫描源程序代码，输出二元式，并定位找出代码中的词法错误。

七、实验器材（设备、元器件）

1. 操作系统：Windows 10
2. 开发工具：VS code
3. PC 机
4. MinGW 编译器
5. C++11 标准

八、实验步骤

(1) 编写代码实现要用到的函数

getchar()：读入字符到变量中；

getnbc()：读入非空白字符；

concat()：连接字符串；

letter()：判断是否是字母；

digit()：判断是否是数字；

retract()：回退字符；

reserve()：对于保留字，给出对应的种别；

symbol()：处理标识符；

constant()：将常熟存入常数表；

return_tuple()：返回二元式；

error()：进行出错处理。

(2) 根据状态转换图，编写词法分析器

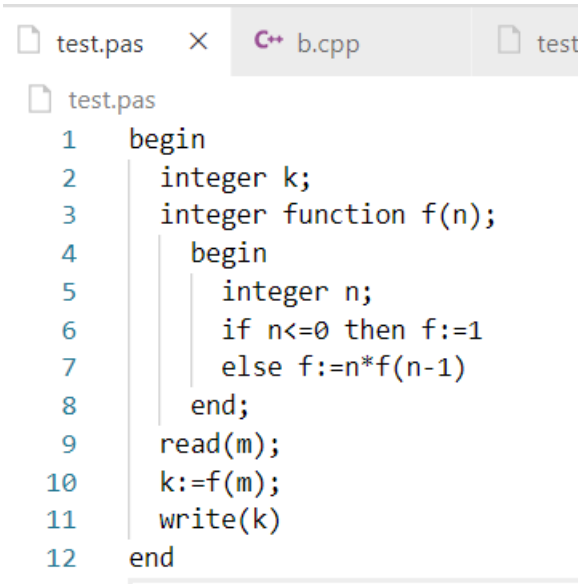
根据状态装换图，编写 LexAnalyze()函数，每次读入第一个字符后，针对读入字符进行各种情况的匹配，进行状态转换，读出单词。将结果以二元式的形式输出到文件中。

(3) 调试代码，得到输出，进行总结

初步完成的代码有错误，不断调试，并逐渐增加报错的功能，直到得到正确的输出。

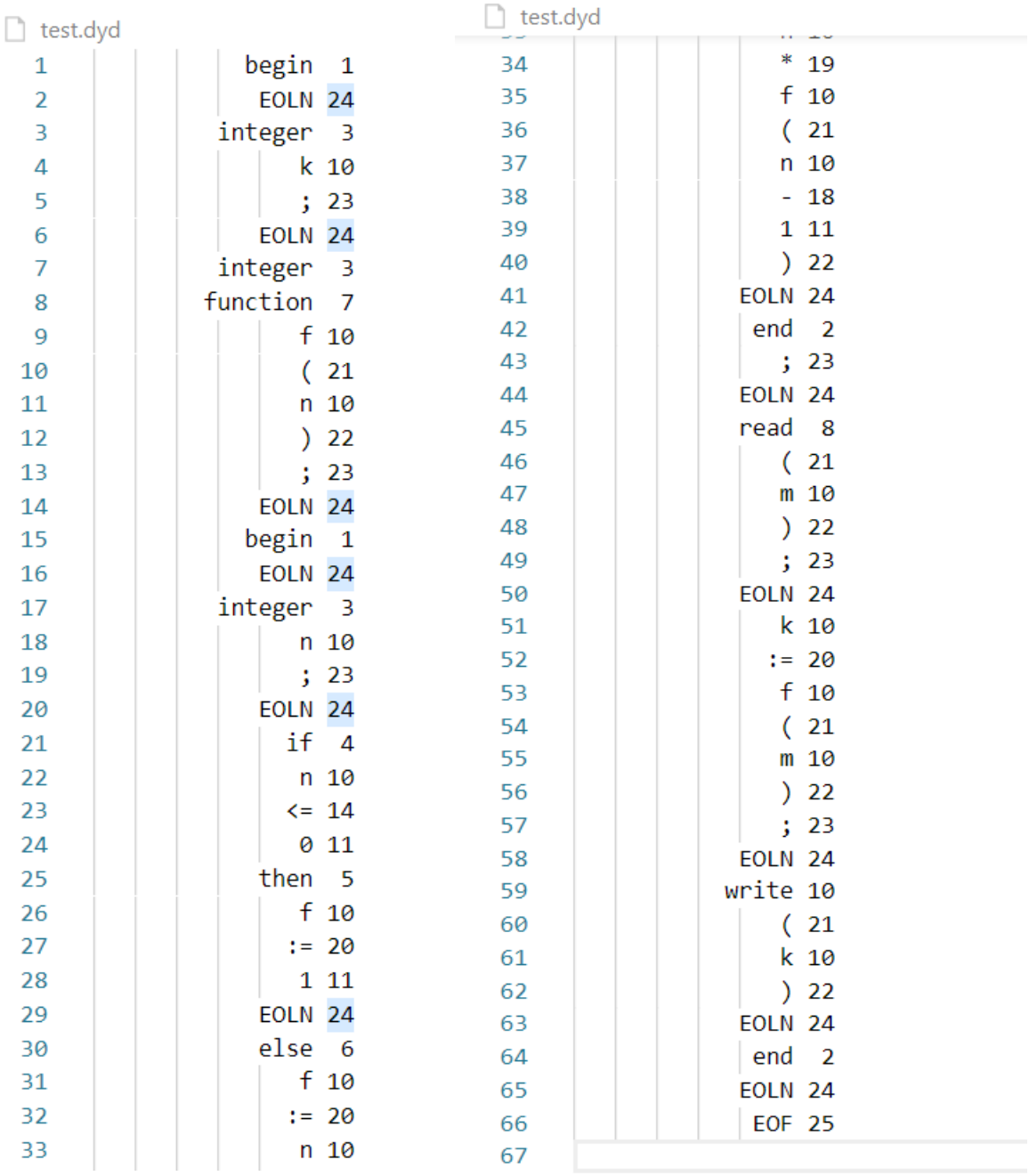
九、实验数据及结果分析

首先给出测试输入文件：



图表 3：测试输入

输出二元式为：



图表 4：输出的二元式

注意到，给出的测试文件没有词法错误，所以报错的文件并没有输出。

为了显示对错误的处理，我们重新给出含有三个错误的输入文件。

error.pas

```
1  begin
2    integer k;
3    integer function f(n);
4      begin:
5        integer n1234567890abcdef;
6        if %<=0 then f:=1
7        else f:=n*f(n-1)
8      end;
9    read(m);
10   k:=f(m);
11   write(k)
12 end
```

图表 5：含有错误的输入

并给出输出：

error.dyd

```
1   begin  1
2    EOLN 24
3  integer  3
4      k 10
5      ; 23
6    EOLN 24
7  integer  3
8  function  7
9      f 10
10     ( 21
11     n 10
12     ) 22
13     ; 23
14   EOLN 24
15   begin  1
16   EOLN 24
17 integer  3
18   EOLN 24
19     if  4
20     <= 14
21      0 11
22   then  5
23     f 10
24     := 20
25       12
26      1 11
27   EOLN 24
28   else  6
29     f 10
30     := 20
31       12
32     n 10
33     * 19
34       f 10
35       ( 21
36       n 10
37       - 18
38       1 11
39       ) 22
40   EOLN 24
41    end  2
42     ; 23
43   EOLN 24
44   read  8
45     ( 21
46     m 10
47     ) 22
48     ; 23
49   EOLN 24
50     k 10
51    := 20
52       12
53     f 10
54     ( 21
55     m 10
56     ) 22
57     ; 23
58   EOLN 24
59  write 10
60     ( 21
61     k 10
62     ) 22
63   EOLN 24
64    end  2
65   EOLN 24
66    EOF 25
```

图表 6：错误文件的输出

并给出报错提示信息：

error.err

```
1    ***LINE:4 :不匹配
2    ***LINE:5 标识符长度溢出
3    ***LINE:6 非法字符
```

图表 7：报错提示信息

十、实验结论

本次实验利用状态转换图编写词法分析程序，利用种别表对识别出单词进行判断，最后输出二元式文件。同时，在有词法错误的时候能够报告出错误位置，并给出相应二元式文件。

十一、总结及心得体会

1.要先完成一个最小可行方案，再附加功能，否则一开始想得太完美难以实现；

2.注意在 switch 语句时及时 break；

3.最开始很难下手，可以多看看书再开始；

4.本次实验很有趣，不算太难，但很能体会做成一件东西的乐趣。

十二、对本实验过程及方法、手段的改进建议

本次实验调试过程中，遇到了很多文件读取方面的 bug。我觉得编写这样程序用 python 比较好，便于忽略不必要的细节，抓住核心思想，因为对于文件读取、输出 python 都有很容易的实现。

报告评分：

指导教师签字：

电 子 科 技 大 学

实　验　报　告

学生姓名：郭志猛　　学 号：2017080201005　指导教师：陈文宇

实验地点：　HOME　　实验时间：2020.6.1

一、实验室名称：HOME

二、实验项目名称：递归下降分析器的设计与实现

三、实验学时：4 学时

四、实验原理

原始文法：

<程序>→<分程序>
<分程序>→begin <说明语句表>；<执行语句表> end
<说明语句表>→<说明语句>｜<说明语句表> ；<说明语句>
<说明语句>→<变量说明>｜<函数说明>
<变量说明>→integer <变量>
<变量>→<标识符>
<标识符>→<字母>｜<标识符><字母>｜ <标识符><数字>
<字母>→a｜b｜c｜d｜e｜f｜g｜h｜i｜j｜k｜l｜m｜n｜o ｜p｜q ｜
r｜s｜t｜u｜v｜w｜x｜y｜z
<数字>→0｜1｜2｜3｜4｜5｜6｜7｜8｜9
<函数说明>→integer function <标识符>（<参数>）；<函数体>
<参数>→<变量>
<函数体>→begin <说明语句表>；<执行语句表> end
<执行语句表>→<执行语句>｜<执行语句表>；<执行语句>
<执行语句>→<读语句>｜<写语句>｜<赋值语句>｜<条件语句>
<读语句>→read(<变量>)
<写语句>→write(<变量>)
<赋值语句>→<变量>:=<算术表达式>
<算术表达式>→<算术表达式>-<项>｜<项>
<项>→<项>*<因子>｜<因子>

<因子>→<变量>｜<常数>｜<函数调用>
<常数>→<无符号整数>
<无符号整数>→<数字>｜<无符号整数><数字>
<函数调用>→<标识符>(算术表达式）
<条件语句>→if<条件表达式>then<执行语句>else <执行语句>
<条件表达式>→<算术表达式><关系运算符><算术表达式>
<关系运算符> →<｜<=｜>｜>=｜=｜<>

符号表示：

```
A:程序          A->B
B:分程序        B->begin C;M end
C:说明语句表    C->D|C;D
D:说明语句      D->E|J
E:变量说明      E->integer F
F:变量          F->G
G:标识符        G->H|GH|GI
H:字母          H->a|b|...|z
I:数字          I->0|1|...|9
J:函数说明      J->integer function G(K);L
K:参数          K->F
L:函数体        L->begin C;M end
M:执行语句表    M->N|M;N
N:执行语句      N->O|P|Q|X
O:读语句        O->read(F)
P:写语句        P->write(F)
Q:赋值语句      Q->F:=R
R:算术表达式    R->R-S|S
S:项            S->S*T|T
T:因子          T->F|U|W
U:常数          U->V
V:无符号整数    V->I|VI
W:函数调用      W->GR
X:条件语句      X->if Y then N else N
Y:条件表达式    Y->RZR
Z:关系运算符    Z-><|<+|>|>=|=|<>
```

消除左递归后的文法：

A:程序	A->B			
B:分程序	B->begin C;M end			
C:说明语句表	C->D|C;D	修改为	C->DC'	C'->;DC'|ε
D:说明语句	D->E|J			
E:变量说明	E->integer F			
F:变量	F->G			
G:标识符	G->H|GH|GI	修改为	G->HG'	G'->HG'|IG'|ε
H:字母	H->a|b|...|z			
I:数字	I->0|1|...|9			
J:函数说明	J->integer function G(K);L			
K:参数	K->F			
L:函数体	L->begin C;M end			
M:执行语句表	M->N|M;N	修改为	M->NM'	M'->;NM'|ε
N:执行语句	N->O|P|Q|X			
O:读语句	O->read(F)			
P:写语句	P->write(F)			
Q:赋值语句	Q->F:=R			
R:算术表达式	R->R-S|S	修改为	R->SR'	R'->-SR'|ε
S:项	S->S*T|T	修改为	S->TS'	S'->*TS'|ε
T:因子	T->F|U|W			
U:常数	U->V			
V:无符号整数	V->I|VI	修改为	V->IV'	V'->IV'|ε
W:函数调用	W->GR			
X:条件语句	X->if Y then N else N			
Y:条件表达式	Y->RZR			
Z:关系运算符	Z-><|<+|>|>=|=|<>			

并且我们发现，我们不需要对标识符重新识别，可以利用种别号简化。同时注意到说明语句推导出的变量说明和函数说明都是以”integer”开头，我们可以进一步简化。

五、实验目的

为了掌握递归下降分析的方法，熟悉编程实现语法分析器的过程。按照给定的文法规则对输入的符号串进行匹配，判断是否形成合法的句子，并对其中的错误给出提示。

六、实验内容

编程实现递归下降分析器，对输入符号串匹配，并输出变量表和过程表，且对过程中的错误给出相应提示。

七、实验器材（设备、元器件）

6. 操作系统：Windows 10
7. 开发工具：VS code
8. PC 机

9. MinGW 编译器

10. C++11 标准

八、实验步骤

1. 编写代码实现各条文法规则

首先消除左递归，然后对文法进行一定简化，再编写代码实现匹配。

2. 编写过程中要调用的模块

init()：打开要读取、修改的文件，并对变量初始化；
advance()：在二元式表中逐个读取二元式；
error()：对错误进行格式化输出；
varExist()：判断符号是否定义，帮助报错程序；
output()：打印输出存储的表格。

3. 调试代码，并增加对变量表、过程表的生成

在变量说明过程中，对变量信息进行存储；在函数说明时，对过程信息进行存储。

4. 实现对错误的处理，总结

主要是对缺少符号错、符号匹配错、符号无定义或重复定义的识别和报错。

九、实验数据及结果分析

首先给出测试输入文件 test.par ：

test.pas × b.cpp test

test.pas

```
1   begin
2     integer k;
3     integer function f(n);
4       begin
5         integer n;
6         if n<=0 then f:=1
7         else f:=n*f(n-1)
8       end;
9     read(m);
10    k:=f(m);
11    write(k)
12  end
```

图表 8：测试输入

利用词法分析器得出 test.dyd,下面我们给出过程名表和变量名表以及报错信息。

变量名表左起依次对应变量名，所属过程，分类，变量类型，变量层次，变量在变量表中位置。

test.var

```
1        k              main 0 integer 0 0
2        n                 f 0 integer 1 1
```

图表 9：变量名表

过程名表左起依次对应过程名，过程类型，过程层次，第一个变量在表中位置，最后一个变量在表中位置。

test.pro

```
1        main integer 0 0 0
2          f integer 1 1 1
```

图表 10：过程名表

test_syn.err

```
1    ***LINE 9: m符号无定义
2    ***LINE 10: m符号无定义
```

图表 11：报错信息

第二个测试文件：

test.pas

```
1   begin
2     integer k;
3     integer function f(n);
4       begin
5         integer n;
6         integer n;
7         if n<=0 then f:=1
8         else f:=n*f(n-1)
9       end;
10    read(m;
11    k:=f(m);
12    write(k)
```

报错信息：

test_syn.err

```
1    ***LINE 6: n符号重定义
2    ***LINE 10: m符号无定义
3    ***LINE 10: 读语句出错
4    ***LINE 11: m符号无定义
5    ***LINE 13: 分程序错误，缺少end
```

第三个测试文件：

test.pas

```
1   begin
2     integer k;
3     integer function f(n);
4       begin
5         integer n;
6         integer n;
7         1 n<=0 then f:=1
8         else f:=n*f(n-1)
9       end;
10    read(m);
11    k:=f(k);
12    write(k
```

报错信息：

test_syn.err

```
1   ***LINE 6: n符号重定义
2   ***LINE 7: 执行语句出错，匹配失败
3   ***LINE 7: 函数体出错，缺少end
4   ***LINE 7: 分程序错误，缺少;
```

我们可以看到，实验要求的缺少符号错、符号匹配错、符号无定义或重复定义错误都能够报出，并且过程名表和变量名表也能正确地给出。实验过程中，只要我们编写的各个文法规则能够正确地进行匹配，我们就可以在这个基础上添加报错、存表的功能，并继续完善。

十、实验结论

本次实验利用词法分析得到的二元式作为输入，采用递归下降分析法实现了语法分析器。整个过程建立在对于各个文法规则的匹配上，匹配时执行变量存表或者报错，就实现了所有的功能。

十一、总结及心得体会

对于产生式，不用完全从头分析，可以直接利用标识符、常数等已知信息进行匹配。同时，不要将产生式只当成字母，建立变量表时，想到肯定是变量说明时才会新建变量。要善于利用已知信息，灵活应对。

十二、对本实验过程及方法、手段的改进建议

要有自上而下的思想，先搭建大的框架，然后再逐步细化完善。大的框架确定可行之后，小的部分逐渐调试并优化即可。可以考虑用 python 实现，重点是思想，而 C 的文件读取等小的操作没有 python 简单。

报告评分：

指导教师签字：

实验一参考源代码如下：

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

char ch;
int flag = 0;
int line = 1;

//读入非空白字符
char getnbc(){
    char ch = getchar();
    while(ch == '\r' || ch == '\t' || ch == ' '){
        ch = getchar();
    }
    return ch;
}

//判断是否为字母
bool letter(char ch){
    if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <='Z'))
        return true;
    else
        return false;
}

//判断是否为数字
bool digit(char ch){
    if(ch >= '0' && ch <= '9')
        return true;
    else
        return false;
}

//回退，调用库函数 ungetc(),且要修改 ch 的值
void retract(char& ch){
    ungetc(ch, stdin);
```

```
    ch = NULL;
}

//处理保留字
int reserve(char* token){
    if(strcmp(token, "begin") == 0)
        return 1;
    if(strcmp(token, "end") == 0)
        return 2;
    if(strcmp(token, "integer") == 0)
        return 3;
    if(strcmp(token, "if") == 0)
        return 4;
    if(strcmp(token, "then") == 0)
        return 5;
    if(strcmp(token, "else") == 0)
        return 6;
    if(strcmp(token, "function") == 0)
        return 7;
    if(strcmp(token, "read") == 0)
        return 8;
    return 0;
}

//处理标识符的函数
int symbol(){
    return  10;
}

//处理常数的函数
int constant(){
    return 11;
}

//输出二元式的函数
void return_tuple(const char* token, int kind){
    printf("%16s %2d\n", token, kind);
}

//出错处理的函数
void error(int line, int errNum){
    const char* tip;
    switch (errNum){
        case 1:
```

```
            tip = "非法字符";
            break;
        case 2:
            tip = ":不匹配";
            break;
        case 3:
            tip = "标识符长度溢出";
    }
    fprintf(stderr, "***LINE:%d %s\n", line,tip);
}

//词法分析器
void LexAnalyze(){
    char token[17] = "";
    char ch;
    ch = getnbc();

    switch(ch){
        case '\n':
            return_tuple("EOLN", 24);
            line++;
            break;
        case EOF:
            return_tuple("EOF", 25);
            flag = 1;
            break;
        case 'a':
        case 'b':
        case 'c':
        case 'd':
        case 'e':
        case 'f':
        case 'g':
        case 'h':
        case 'i':
        case 'j':
        case 'k':
        case 'l':
        case 'm':
        case 'n':
        case 'o':
        case 'p':
        case 'q':
        case 'r':
```

```
case 's':
case 't':
case 'u':
case 'v':
case 'w':
case 'x':
case 'y':
case 'z':
    int count;
    count = 0;
    do{
        char s[2] = {ch}; //为了类型匹配
        strcat(token,s);
        ch = getchar();
        count++;
    }while(letter(ch) || digit(ch));
    if(count > 16){
        error(line, 3);
        break;
    }
    else{
        retract(ch);
        int num;
        num = reserve(token);
        if(num != 0)
            return_tuple(token, num);
        else{
            int val;
            val = symbol();
            return_tuple(token, val);
        }
        break;
    }
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
    while(digit(ch)){
```

```
            char s[2] = {ch};
            strcat(token,s);
            ch = getchar();
        }
        retract(ch);
        int val;
        val = constant();
        return_tuple(token, val);
        break;
    case '=':
        return_tuple(token, 12);
        break;
    case '<':
        ch = getchar();
        if (ch == '>')
            return_tuple("<>", 13);
        else if (ch == '=')
            return_tuple("<=", 14);
        else{
            retract(ch);
            return_tuple("<", 15);
        }
        break;
    case '>':
        ch = getchar();
        if (ch == '=')
            return_tuple(">=", 16);
        else{
            retract(ch);
            return_tuple(">", 17);
        }
        break;
    case '-':
        return_tuple("-",18);
        break;
    case '*':
        return_tuple("*",19);
        break;
    case ':':
        ch = getchar();
        if (ch == '='){
            return_tuple(":=",20);
            break;
        }
```

```
            else
                error(line,2);
                retract(ch);
            break;
        case '(':
            return_tuple("(",21);
            break;
        case ')':
            return_tuple(")",22);
            break;
        case ';':
            return_tuple(";",23);
            break;
        default:
            error(line, 1);
    }
}


int main(){
    // freopen("error.pas","r",stdin);
    // freopen("error.dyd","w",stdout);
    // freopen("error.err","w",stderr);
    freopen("test.pas","r",stdin);
    freopen("test.dyd","w",stdout);
    freopen("test.err","w",stderr);
    while(flag == 0){
        LexAnalyze();
    }
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
    return 0;
}
```

实验二参考源代码如下：

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

char sym[100][17];
int kind[100];
int token = 0;
int flag = 0;
int lineNum = 1;
int varNum = 0;
int proNum = 0;

FILE* inFile,*outFile,*errFile,*varFile,*proFile;

typedef enum{integer}types;

struct var{
    char vname[16];
    char vproc[16];
    bool vkind;
    types vtype;
    int vlev;
    int vadr;
};

struct pro{
    char pname[16];
    types ptype;
    int plev;
    int num;
    int fadr;
    int ladr;
    int parameter;
    bool parameterIsDefined;
};

var nowVar,vars[100];
pro nowPro,initPro,pros[100];

//初始化
void init(){
```

```
    //打开该打开的文件
    inFile = fopen("test.dyd","r");
    outFile = fopen("test.dys","w");
    errFile = fopen("test_syn.err","w");
    varFile = fopen("test.var","w");
    proFile = fopen("test.pro","w");
    //初始化当前过程名表
    strcpy(nowPro.pname,"main");
    nowPro.plev = 0;
    nowPro.num = 0;
    nowPro.parameter = -1;
    //将"test.dyd"的数据读入 sym 和 kind
    int count = 0;
    while(!feof(inFile)){
        char line[100];
        if(fgets(line,100,inFile)){
            char lineString[20] = "";
            strncpy(lineString,line,19);
            char* kindString = strrchr(lineString,' ');
            kind[count] = atoi(kindString + 1);
            char string[17] = "";
            strncpy(string,line,16);
            char* lastString = strrchr(string, ' ');
            strcpy(sym[count],lastString+1);
            count++;
        }
    }
}

//开始
void advance(){
    token++;
    if(strcmp(sym[token],"EOF")==0)
        flag = 1;
    if(strcmp(sym[token],"EOLN")==0){
        token++;
        lineNum++;
    }
}

//报错程序
void error(char const str[],int notice=0){
    freopen("test_syn.err","a+",stdout);
    if (notice==0){
```

```
        printf("***LINE %d: %s\n",lineNum,str);
    }
    else
        printf("***LINE %d: %s%s\n",lineNum,sym[token],str);
}

//判断符号无定义，辅助报错程序运行,读、写、赋值、因子时候判断
bool varExist(const char var[]){
    for(int i=0;i<varNum;i++){
        if(strcmp(vars[i].vname,var)==0){
            return true;
        }
    }
    if(strcmp(nowPro.pname,var)==0){
        return true;
    }
    return false;
}

//打印输出
void output(){
    for(int i=0;i<varNum;i++){
        int vkind = vars[i].vkind? 1: 0;
        const char* vtype = (vars[i].vtype == integer)?"integer":"";
        fprintf(varFile,"%16s %16s %d %s %d %d\n",vars[i].vname,vars[i]
.vproc,vkind,vtype,vars[i].vlev,vars[i].vadr);
    }
    for (int i=0;i<proNum+1;i++){
        const char* ptype=(pros[i].ptype == integer)?"integer":"";
        fprintf(proFile,"%16s %s %d %d %d\n",pros[i].pname,ptype,pros[i
].plev,pros[i].fadr,pros[i].ladr);
    }
    if (fseek(inFile,0,0)==0){
        while (!feof(inFile))
            fputc(fgetc(inFile),outFile);
    }
    fclose(inFile);
    fclose(outFile);
    fclose(errFile);
    fclose(varFile);
    fclose(proFile);
}
```

```
/*
A:程序          A->B
B:分程序        B->begin C;M end
C:说明语句表    C->D|C;D                    修改
为    C->DC'  C'->;DC'|ε
D:说明语句      D->E|J
E:变量说明      E->integer F
F:变量          F->G
G:标识符        G->H|GH|GI                  修改
为    G->HG'    G'->HG'|IG'|ε
H:字母          H->a|b|...|z
I:数字          I->0|1|...|9
J:函数说明      J->integer function G(K);L
K:参数          K->F
L:函数体        L->begin C;M end
M:执行语句表    M->N|M;N                    修改
为    M->NM'    M'->;NM'|ε
N:执行语句      N->O|P|Q|X
O:读语句        O->read(F)
P:写语句        P->write(F)
Q:赋值语句      Q->F:=R
R:算术表达式    R->R-S|S                    修改
为    R->SR'    R'->-SR'|ε
S:项            S->S*T|T                    修改
为    S->TS'    S'->*TS'|ε
T:因子          T->F|U|W
U:常数          U->V
V:无符号整数    V->I|VI                     修改为    V->IV'   V'->IV'|ε
W:函数调用      W->GR
X:条件语句      X->if Y then N else N
Y:条件表达式    Y->RZR
Z:关系运算符    Z-><|<+|>|>=|=|<>
*/

void A();
void B();
void C();
void C_();
void D();
void E();
void F();
void G();
void J();
void J_();
```

```
void K();
void L();
void M();
void M_();
void N();
void Q();
void R();
void R_();
void S();
void S_();
void T();
void W();
void X();
void Y();
void Z();


//A:程序           A->B
void A(){
    B();
}

// B:分程序         B->begin C;M end
void B(){
    if (strcmp(sym[token],"begin")==0){
        advance();
        C();
        if (strcmp(sym[token],";")==0){
            advance();
            M();
            if (strcmp(sym[token],"end")==0)
                advance();
            else
                error("分程序错误，缺少 end");
        }
        else
            error("分程序错误，缺少;");
    }
    else {
        error("分程序错误，缺少 begin");
    }
}

// C:说明语句表     C->D|C;D                    修改
为    C->DC'  C'->;DC'|ε
```

```
void C(){
    D();
    C_();
}

void C_(){
    if((strcmp(sym[token],";")==0) && ((strcmp(sym[token+1],"integer")==
=0)||((strcmp(sym[token+2],"integer")==0)&&(strcmp(sym[token+1],"EOLN")
==0)))){
        advance();
        D();
        C_();
    }
}

// D:说明语句      D->E|J
void D(){
    if((strcmp(sym[token+1],"function")==0)&&(strcmp(sym[token],"intege
r")==0)){
        J();
    }
    else if(strcmp(sym[token],"integer")==0){
        E();
    }
    else
        error("说明语句错误，缺少 integer");
}

// E:变量说明      E->integer F
void E(){
    if(strcmp(sym[token],"integer")==0){
        advance();
        strcpy(nowVar.vname,sym[token]);
        strcpy(nowVar.vproc,nowPro.pname);
        nowVar.vkind = (token==nowPro.parameter)?1:0;
        nowVar.vtype = integer;
        nowVar.vlev = nowPro.plev;
        nowVar.vadr = varNum;
        if (varExist(nowVar.vname)){
            error("符号重定义",2);
        }
        else{
            vars[varNum] = nowVar;
            if (nowPro.num == 0){
```

```
                nowPro.fadr = nowVar.vadr;
            }
            nowPro.ladr = nowVar.vadr;
            nowPro.num++;
            varNum++;
        }
        F();
    }
    else
        error("说明语句错误，缺少 integer");
}

// F:变量          F->G
void F(){
    G();
}

// G:标识符        G->H|GH|GI                  修改
为    G->HG'    G'->HG'|IG'|ε
void G(){
    if(kind[token]==10){
        if(!varExist(sym[token])){
            error("符号无定义",1);
        }
        advance();
    }
    else
        error("标识符出错");
}

// H:字母          H->a|b|...|z
// I:数字          I->0|1|...|9
// J:函数说明      J->integer function G(K);L
void J(){
    if(strcmp(sym[token],"integer")==0){
        advance();
        if(strcmp(sym[token],"function")==0){
            advance();
            if(kind[token]==10){
                pros[proNum] = nowPro;
                nowPro = initPro;
                proNum++;
                strcpy(nowPro.pname,sym[token]);
                nowPro.ptype = integer;
```

```
                nowPro.plev++;
                advance();
                if(strcmp(sym[token],"(")==0){
                    advance();
                    nowPro.parameter= token;
                    K();
                    if(strcmp(sym[token],")")==0){
                        advance();
                        if(strcmp(sym[token],";")==0){
                            advance();
                            L();
                            pros[proNum] = nowPro;
                        }
                        else
                            error("函数说明出错，缺少;");
                    }
                    else
                        error("函数说明出错，缺少)");
                }
                else
                    error("函数说明出错，缺少(");
            }
        }
        else
            error("函数说明出错，缺少 function");
    }
    else
        error("函数说明出错，缺少 integer");
}

// K:参数          K->F
void K(){
    if(kind[token]==10){
        advance();
    }
    else
        error("标识符出错");

}

// L:函数体        L->begin C;M end
void L(){
    if(strcmp(sym[token],"begin")==0){
        advance();
```

```
        C();
        if(strcmp(sym[token],";")==0){
            advance();
            M();
            if(strcmp(sym[token],"end")==0){
                advance();
            }
            else
                error("函数体出错，缺少 end");
        }
        else
            error("函数体出错，缺少;");
    }
    else
        error("函数体出错，缺少 begin");
}

// M:执行语句表    M->N|M;N                    修改
为    M->NM'    M'->;NM'|ε
void M(){
    N();
    M_();
}

void M_(){
    if(strcmp(sym[token],";")==0){
        advance();
        N();
        M_();
    }
}

// N:执行语句      N->O|P|Q|X
void N(){
    if(strcmp(sym[token],"read")==0){
        advance();
        if(strcmp(sym[token],"(")==0){
            advance();
            F();
            if(strcmp(sym[token],")")==0){
                advance();
            }
            else
                error("读语句出错");
```

```
        }
        else
            error("读语句出错");
    }
    else if(strcmp(sym[token],"write")==0){
        advance();
        if(strcmp(sym[token],"(")==0){
            advance();
            F();
            if(strcmp(sym[token],")")==0){
                advance();
            }
            else
                error("写语句出错");
        }
        else
            error("写语句出错");
    }
    else if(kind[token]==10){
        Q();
    }
    else if(strcmp(sym[token],"if")==0){
        X();
    }
    else
        error("执行语句出错，匹配失败");
}

// O:读语句         O->read(F)
// P:写语句         P->write(F)
// Q:赋值语句       Q->F:=R
void Q(){
    if(kind[token]==10){
        if(!varExist(sym[token])){
            error("符号无定义",1);
        }
        advance();
        if(strcmp(sym[token],":=")==0){
            advance();
            R();
        }
        else
            error("赋值语句出错，缺少:=");
    }
```

```
    else
            error("赋值语句出错");
}

// R:算术表达式    R->R-S|S                    修改
为    R->SR'    R'->-SR'|ε
void R(){
    S();
    R_();
}

void R_(){
    if(strcmp(sym[token],"-")==0){
        advance();
        S();
        R_();
    }
}

// S:项            S->S*T|T                    修改
为    S->TS'    S'->*TS'|ε
void S(){
    T();
    S_();
}

void S_(){
    if(strcmp(sym[token],"*")==0){
        advance();
        T();
        S_();
    }
}

// T:因子          T->F|U|W
void T(){
    if(kind[token]==10&&(strcmp(sym[token+1],"(")!=0)){
        if(!varExist(sym[token])){
            error("符号无定义",1);
        }
        advance();
    }
    else if(kind[token]==11){
        advance();
```

```
    }
    else{
        W();
    }
}

// U:常数          U->V
// V:无符号整数    V->I|VI                    修改
为     V->IV'   V'->IV'|ε
// W:函数调用      W->GR
void W(){
    G();
    if(strcmp(sym[token],"(")==0){
        advance();
        R();
        if(strcmp(sym[token],")")==0){
            advance();
        }
        else
            error("函数调用出错");
    }
    else
        error("函数调用出错");
}

// X:条件语句      X->if Y then N else N
void X(){
    if(strcmp(sym[token],"if")==0){
        advance();
        Y();
        if(strcmp(sym[token],"then")==0){
            advance();
            N();
            if(strcmp(sym[token],"else")==0){
                advance();
                N();
            }
            else
                error("条件语句出错，缺少 else");
        }
        else
            error("条件语句出错，缺少 then");
    }
    else
```

```
        error("条件语句出错，缺少 if");
}

// Y:条件表达式    Y->RZR
void Y(){
    R();
    Z();
    R();
}

// Z:关系运算符    Z-><|<+|>|>=|=|<>

void Z(){
    if(strcmp(sym[token],"<")==0){
        advance();
    }
    else if(strcmp(sym[token],"<=")==0){
        advance();
    }
    else if(strcmp(sym[token],">")==0){
        advance();
    }
    else if(strcmp(sym[token],">=")==0){
        advance();
    }
    else if(strcmp(sym[token],"=")==0){
        advance();
    }
    else if(strcmp(sym[token],"<>")==0){
        advance();
    }
    else
        error("无法找到运算符");
}


int main(){
    init ();
    A();
    output();
    return 0;
}
```