# **实验报告：词法分析程序scanner构建**

## **一、实验目的**

本实验旨在构建一个词法分析程序，该程序能够接收源语言程序作为输入，通过词法分析生成词法记号串，并将结果输出到文件中。词法分析程序需要实现以下功能：

1.单词设计：定义主文法中的所有词法单位。

2.输入预处理：能够删除源代码中的注解，并允许空白字符串作为分隔符。

3.分析框架：设计并实现一个名为scanner()的分析框架，该框架允许连续调用直到输入串被完全扫描，每次调用返回一个词法记号。

4.DFA设计：使用确定有限自动机（DFA）的设计结果来指导词法分析。

## **二、实验内容**

### **2.1 单词设计**

在单词设计中，我们根据源语言程序的主文法定义了各类词法单位，如关键字、标识符、常数、运算符、分隔符等，并为每种词法单位分配了唯一的记号。

### **2.2 输入预处理**

输入预处理模块负责删除源代码中的注解（如使用特定符号或字符块标记的注释），并将源代码中的空白字符（如空格、制表符、换行符等）消除忽略处理。

### **2.3 分析框架scanner()**

scanner()函数是词法分析程序的核心。它采用循环结构，不断从输入流中读取字符，并根据DFA的状态转移规则进行词法分析。每次循环结束时，scanner()返回一个词法记号，包括记号的类型和值（如果有的话）。当输入流被完全扫描时，scanner()结束执行。

### **2.4 DFA设计与超前搜索**

我们使用DFA来描述词法分析器的状态转移规则。DFA的设计基于源语言程序的词法规则，每个状态对应一个或多个可能的输入字符，以及根据输入字符进行的状态转移。为了提高性能，我们在DFA设计中引入了超前搜索功能，即在读取一个字符时，同时考虑该字符后面的字符序列，以便更快地确定当前词法记号的边界。

### **2.5 词法错误处理**

（可选部分）我们为词法分析程序设计了词法错误处理机制。当遇到不符合词法规则的输入时，程序会报告错误，并尝试从错误位置恢复分析。恢复策略可能包括跳过无效字符、插入缺失字符或删除多余字符等。

## **三、实验结果**

### **3.1 代码实现**

使用C++实现了上述词法分析程序，通过实现scanner类定义，实现了以上功能。代码包括单词定义、输入预处理模块、scanner()函数实现、DFA设计以及（可选的）词法错误处理机制。具体实现细节因编程语言而异，但总体结构保持一致。

实现主要功能的代码如下：

1. :保留字表：

*char* key\_word\_table[20][10]**=**{

    "include", *//80*

    "void", *// 81*

    "char", *// 82*

    "int", *// 83*

    "sizeof", *// 84*

    "const", *// 85*

    "return", *// 86*

    "continue", *// 87*

    "break", *// 88*

    "if", *// 89*

    "else", *// 90*

    "switch", *// 91*

    "case", *// 92*

    "default", *// 93*

    "for", *// 94*

    "do", *// 95*

    "while", *// 96*

    "scanf", *// 97*

    "printf", *// 98*

    "main" *//99*

};

1. ：所有符号加上保留字的表：

*const* *char* *table*[100][20] **=** {

    "", "", "", "", "", "", "", "", "", "",

    "","NUMBER", "DIVOP","ID","","","","","","","",

    "LBRACE", *// 21*

    "RBRACE", *// 22*

    "LBRACK", *// 23*

    "RBRACK", *// 24*

    "LPARENT", *// 25*

    "RPARENT", *// 26*

    "LOGRE", *// 27*

    "INPLUS", *// 28*

    "INMINUS", *// 29*

    "LOCRE", *// 30*

    "STAR", *// 31*

    "DIVOP", *// 32*

    "COMOP", *// 33*

    "PLUS", *// 34*

    "MINUS", *// 35*

    "RELG", *// 36*

    "RELL", *// 37*

    "RELGEQ", *// 38*

    "RELLEQ", *// 39*

    "EQUOP", *// 40*

    "UEQUOP", *// 41*

    "ANDAND", *// 42*

    "OROR", *// 43*

    "EQUAL", *// 44*

    "ASSIGNDIV", *// 45*

    "ASSIGNSTAR", *// 46*

    "ASSIGNCOM", *// 47*

    "ASSIGNPLUS", *// 48*

    "ASSIGNMINUS", *// 49*

    "COMMA", *// 50*

    "SHA", *// 51*

    "SEMI", *// 52*

    "COLON", *// 53*

    "AND", *// 54*

    "OR", *// 55*

    "QUOTA", *//56*

    "STRING", *//57*

    "POINT", *//58*

    "","",

    "","","","","","","","","","",

    "","","","", "","","","","",

    "INCLUDE", *//80*

    "VOID", *// 81*

    "CHAR", *// 82*

    "INT", *// 83*

    "SIZEOF", *// 84*

    "CONST", *// 85*

    "RETURN", *// 86*

    "CONTINUE", *// 87*

    "BREAK", *// 88*

    "IF", *// 89*

    "ELSE", *// 90*

    "SWITCH", *// 91*

    "CASE", *// 92*

    "DEFAULT", *// 93*

    "FOR", *// 94*

    "DO", *// 95*

    "WHILE", *// 96*

    "SCANF", *// 97*

    "PRINTF", *// 98*

    "MAIN" *//99*

    };

1. 初始化Initscanner函数：

*void* Initscanner(){

    ifstream file("in.c", ios**::***binary* **|** ios**::***ate*);

**if** (**!**file) {

        cerr **<<** "Unable to open file." **<<** endl;

**return**;

    }

    memset(*input*,0,**sizeof**(*input*));

*// 获取文件大小*

    streamsize size **=** file.tellg();

*// 重置文件位置到文件开头*

*char\_num***=**size;

    file.seekg(0, ios**::***beg*);

*// 读取文件到buffer*

    file.read(*input*, size);

*// 添加空字符到末尾（可选，如果你需要将buffer作为C字符串处理）*

*input*[size] **=** '\0';

*// 关闭文件*

    file.close();

    memset(*token*,0,**sizeof**(*token*));

*place***=**0;

*fg***=**0;

    cout**<<***input***<<**"and lenth is"**<<***char\_num***<<**endl;

**return**;

    }

代码分析：

实现功能：

((1))将文件内容读取到缓冲区input中。

((2))记录input长度到char\_num中。

((3))初始化token数组为0，记录读取到的位置数place为0，fg标识为0。

1. 实现分别4种类型的成员函数：
   1. ：处理保留字和ID的Isalpha函数：

*void* Isalpha()

    {

*int* m**=**0;

**if**((*input*[*place*]**<=**'z'**&&***input*[*place*]**>=**'a')**||**(*input*[*place*]**<=**'Z'**&&***input*[*place*]**>=**'A'))

    {

**while**((*input*[*place*]**<=**'z'**&&***input*[*place*]**>=**'a')**||**(*input*[*place*]**<=**'Z'**&&***input*[*place*]**>=**'A')**||**(*input*[*place*]**<=**'9'**&&***input*[*place*]**>=**'0'))

        {

*token*[m]**=***input*[*place*];

            m**++**;

*place***++**;

        }

*fg***=**13;

**for**(*int* i**=**0;i**<**20;i**++**)

**if**(strcmp(*token*,key\_word\_table[i])**==**0)

        {

*fg***=**i**+**80;

**break**;

        }

    }

    }

代码分析：处理字母开头直到不以字符或数字结尾。检查是否为保留字，若是则该token属于保留字，不是则为一个ID值。

* 1. ：处理数字的Isnumber函数

*void* Isnumber(){

*num***=**0;

**while**(*input*[*place*]**<=**'9'**&&***input*[*place*]**>=**'0')

    {

*num***=***num***\***10**+***input*[*place*]**-**'0';

*place***++**;

    }

*fg***=**11;

}

代码分析：处理以数字开头且全为数字的输入串。最后返回对应值。

* 1. ：处理与’/’符号有关的Isfan成员函数：

*void* Isfan(){

**if**(*input*[*place*]**==**'/')

    {

*place***++**;

**switch**(*input*[*place*])

        {

**case** '/':

**while**(*input*[*place*]**!=**'\n'**&&***input*[*place*]**!=**'\r')*place***++**;

*place***+=**2;

*fg***=-**1;

**break**;

**case** '\*':

**while**(true){

*place***++**;

**if**(*input*[*place*]**==**'\*'**&&***input*[*place***+**1]**==**'/')

                    {

*place***=***place***+**2;

*fg***=-**1;

**break**;

                    }

            }

**break**;

**case** '=':

*token*[0]**=**'/';

*token*[1]**=**'=';

*fg***=**45;

*place***++**;

**break**;

            default:

*token*[0]**=**'/';

*fg***=**12;

**break**;

        };

    }

    }

代码分析：由于需要忽略注释，所以对/符号单独进行处理。

在检测到第一个字符为’/’后

((1)):第二个字符为’/’：

进入单行注释模式，忽略中间所有字符，直到检测到’\r’’\n’时退出模式，fg置为-1，不做任何输出。

((2)):第二个字符为’\*’：

进入多行注释模式，忽略中间所有字符，直到检测到连续的’\*’’/’时推出模式，fg置位-1，不做任何输出。

((3))：第三个字符为’=’：

/=符号，除等运算，fg置为对应的45，按其他字符一样处理。

((4))：其他：

/符号，除法运算，fg置位对应的12，按其他字符一样处理。

* 1. ：处理其他字符的Isother函数：

*void* Isother(){

*int* m**=**0;

*char* ch**=***input*[*place*];

*//switch中，place是读到的最后一个字符*

**switch** (ch)

    {

**case** '<':

                    m**=**0;

*token*[m**++**]**=**ch;

*place***++**;

                    ch**=***input*[*place*];

**if**(ch**==**'=') *//产生<=*

                    {

*fg* **=** 39;

*token*[m]**=**ch;

                    }

**else** *//产生<*

                    {

*fg* **=** 37;

*place***--**;

                    }

**break**;

......

    };

*place***++**;*//place指向第一个还没有读的字符的位置*

}

代码分析：以首个字符为’<’为例：

检测到首个字符为’<’之后，进入对应的DFA：

该DFA接受四种输入：

1：’=’，构成”>=”符号，接收状态39。

2：其他符号，构成”>”符号，接收状态37。

到达接收状态后返回，此时token数组存储了该符号的完整内容。

1. ：向对应文件流里输出的Output成员函数：

*void* Output() {

FILE **\***file**=**freopen("out.txt","a",stdout);

**if**(*fg***==**11)

{

    cout**<<**"("**<<***table*[*fg*]**<<**","**<<***num***<<**")"**<<**endl;

}

**else** **if**(*fg***<**80**&&***fg***!=-**1)

{

    cout**<<**"("**<<***table*[*fg*]**<<**","**<<***token***<<**")"**<<**endl;

}

**else** **if**(*fg***==-**1)

{

}

**else**{

    string l**=**key\_word\_table[*fg***-**80];

    cout**<<**"("**<<**toUpper(l)**<<**",-)"**<<**endl;

}

}

根据不同的类，向out.txt种输出不同格式的内容。

1. 处理输入字符串的run\_scanner成员函数：

*void* run\_scanner(){

**while**(*place***<***char\_num*)

    {

**while**(*input*[*place*]**==**'\t'**||***input*[*place*]**==**' '**||***input*[*place*]**==**'\r'**||***input*[*place*]**==**'\n')*place***++**;

**if**((*input*[*place*]**>=**'a'**&&***input*[*place*]**<=**'z')**||**(*input*[*place*]**>=**'A'**&&***input*[*place*]**<=**'Z'))

    Isalpha();

**else** **if**(*input*[*place*]**<=**'9'**&&***input*[*place*]**>=**'0')

    Isnumber();

**else** **if**(*input*[*place*]**==**'/')

    Isfan();

**else**

    Isother();

    Output();

    memset(*token*,0,20);

    }

}

代码分析：

run\_scanner实现的功能

（1）过滤掉空格，制表符和换行符。

（2）每次根据起始字符的不同调用不同的字符处理函数。

（3）处理完成后，调用Output函数进行输出。

（4）最后对token进行初始化，以便下一个token符号的读取。

（5）重复以上步骤直到读入字符串为空。

### **3.2 测试用例**

我们设计了一组测试用例来验证词法分析程序的正确性。测试用例包括符合词法规则的输入序列、包含注解和空白字符的输入序列以及不符合词法规则的输入序列。通过运行这些测试用例，我们验证了词法分析程序能够正确处理各种情况，并生成正确的词法记号串。

输入文件内容：

//in.c:

#include<stdio.h>

*int* main()

{

*//this is a test file*

    printf("hello world");

*int* a**=**123,b**=**1;

*int* c**=**a**/**b;

*/\*this will not output*

*\*/*

**return** 0;

}

### **3.3 结果输出**

词法分析程序将生成的词法记号串输出到文件中。输出文件采用易于阅读的格式，每个词法记号占一行，包括记号的类型和值（如果有的话）。通过检查输出文件，我们可以验证词法分析程序的正确性。

输出结果：out.txt:

(SHA,#)

(INCLUDE,-)

(RELL,<)

(ID,stdio)

(POINT,.)

(ID,h)

(RELG,>)

(INT,-)

(MAIN,-)

(LPARENT,()

(RPARENT,))

(LBRACE,{)

(PRINTF,-)

(LPARENT,()

(STRING,hello world)

(RPARENT,))

(SEMI,;)

(INT,-)

(ID,a)

(EQUAL,=)

(NUMBER,123)

(COMMA,,)

(ID,b)

(EQUAL,=)

(NUMBER,1)

(SEMI,;)

(INT,-)

(ID,c)

(EQUAL,=)

(ID,a)

(DIVOP,/)

(ID,b)

(SEMI,;)

(RETURN,-)

(NUMBER,0)

(SEMI,;)

(RBRACE,})

## **四、总结与展望**

本实验成功构建了一个词法分析程序，该程序能够接收源语言程序作为输入，并生成正确的词法记号串。通过实现输入预处理、DFA设计等功能，我们提高了词法分析程序的性能和准确性。未来工作可以进一步优化词法分析程序，如引入更复杂的词法错误处理机制、支持多语言词法分析等。同时，我们还可以将词法分析程序与语法分析程序相结合，构建完整的编译器前端系统。