# X语言词法分析器

1 实验目的及意义

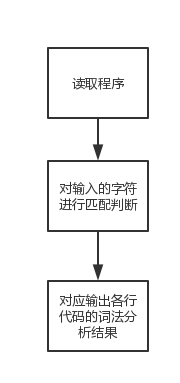
通过构造编译器，可以帮助同学们了解词法分析器和语法分析器的工作原理，加深对词法分析和语法分析的理解，掌握词法分析和语法分析的方法。在用编程语言实现的过程中，可以使同学们将理论与实际应用结合起来，强化编程和软件设计能力，在此过程中也可以帮助同学们复习并更深的理解编译原理，数据结构，编程语言等知识。使同学们受到软件设计等开发过程的全面训练，从而提高学生实践能力。

2 词法结构

（说明该语言的词法结构及相应的正则表达式）

**词法分析:** 词法分析的实现主要利用有穷自动机（DFA）理论。有穷自动机（DFA）可用作描述在输入串中识别模式的过程，因此也能用作构造扫描程序。通过有穷自动机（DFA）理论能够容易的设计出词法分析器。

**词法程序主流程图：**



**图 词法分析**

## 词法分析阶段 ：

## （2）C-语言的词法说明

① 语言的关键字：

**else if int return void while**

所有的关键字都是保**（2）C-语言的词法说明**

① 语言的关键字：

**else if int return void while**

所有的关键字都是保留字，并且必须是小写。

②专用符号：

**+ - \* / < <= > >= == != = ; , ( ) [ ] { } /\* \*/**

③其他标记是**ID**和**NUM**，通过下列正则表达式定义：

ID = letter letter\*

NUM = digit digit\*

letter = a|..|z|A|..|Z

digit = 0|..|9

注：ID表示标识符，NUM表示数字，letter表示一个字母，digit表示一个数字。

小写和大写字母是有区别的。

④ 空格由空白、换行符和制表符组成。空格通常被忽略。

⑤ 注释用通常的c语言符号**/ \* . . . \* /**围起来。注释可以放在任何空白出现的位置(即注释不能放在标记内)上，且可以超过一行。注释不能嵌套。留字，并且必须是小写。

②专用符号：

**+ - \* / < <= > >= == != = ; , ( ) [ ] { } /\* \*/**

③其他标记是**ID**和**NUM**，通过下列正则表达式定义：

ID = letter letter\*

NUM = digit digit\*

letter = a|..|z|A|..|Z

digit = 0|..|9

注：ID表示标识符，NUM表示数字，letter表示一个字母，digit表示一个数字。

小写和大写字母是有区别的。

④ 空格由空白、换行符和制表符组成。空格通常被忽略。⑤ 注释用通常的c语言符号**/ \* . . . \* /**围起来。注释可以放在任何空白出现的位置(即注释不能放在标记内)上，且可以超过一行。注释不能嵌套。

**词法分析子流程图**



3 Token定义及DFA设计

3.1Token的定义和类型

3.1.1Token定义:

//定义的Token结构体，包括类型、对应的串、所在代码的行号

struct Token

{

TokenType tokenType;

string tokenString;

int lineNo;

};

3.1.2Token类型（TokenType）:

程序中，将Token的类型分为31种，分别对应于else、if、int、return、void、while、+、-、\*、/、<、<=、>、>=、==、!=、=、;、,、(、)、[、]、{、}、/\*、\*/、num、id、错误、文件结束，TokenType的定义和种别码分别如下所示：

//定义的Token的类型(31种)，分别对应于else、if、int、return、void、while、+、-、\*、/、<、<=、>、>=、==、!=、=、;、,、(、)、[、]、{、}、/\*、\*/、num、id、错误、结束

typedef enum

{

ELSE = 1,IF,INT,RETURN,VOID,WHILE,

PLUS,MINUS,TIMES,OVER,LT,LEQ,GT,GEQ,EQ,NEQ,ASSIGN,SEMI,COMMA,LPAREN,RPAREN,LMBRACKET, RMBRACKET,LBBRACKET,RBBRACKET,LCOMMENT,RCOMMENT,

NUM,ID,ERROR,ENDFILE

} TokenType;

3.1.3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 单词符号 | 种别码 | Value | 单词符号 | 种别码 | Value |
| else | ELSE | 1 | = | ASSIGN | 17 |
| if | IF | 2 | ; | SEMI | 18 |
| int | INT | 3 | , | COMMA | 19 |
| return | RETURN | 4 | ( | LPAREN | 20 |
| void | VOID | 5 | ) | RPAREN | 21 |
| while | WHILE | 6 | [ | LMBRACKET | 22 |
| + | PLUS | 7 | ] | RMBRACKET | 23 |
| - | MINUS | 8 | { | LBBRACKET | 24 |
| \* | TIMES | 9 | } | RBBRACKET | 25 |
| / | OVER | 10 | /\* | LCOMMENT | 26 |
| < | LT | 11 | \*/ | RCOMMENT | 27 |
| <= | LEQ | 12 | num | NUM | 28 |
| > | GT | 13 | id | ID | 29 |
| >= | GEQ | 14 | 错误 | ERROR | 30 |
| == | EQ | 15 | 结束 | ENDFILE | 31 |
| != | NEQ | 16 |  |  |  |

3.2 DFA设计

由于词法分析程序分为两个步骤处理：删除注释和词法分析获取Token。所以对应有两个DFA，程序分别根据这两个DFA进行编写，现根据DFA分析两程序deleteComments()和scan()如下：

**3.2.1删除注释的DFA描述：**

删除注释的DFA如下所示，一共分为5个状态，在开始状态1时，如果输入的字符为/,则进入状态2，此时有可能进入注释状态，如果在状态2时，输入的字符为\*，则进入注释状态，状态将转到3，如果在状态3时，输入的字符为\*，则有可能结束注释状态，此时状态将转到状态4，如果在状态4时输入的字符为/，则注释状态结束，状态转移到结束状态。



图 删除注释的DFA

3.2.2 C—的DFA分析

* 词法分析的DFA如下所示，一共分为5个状态：START、INNUM、INID、INDBSYM、DONE。状态START表示开始状态，状态INNUM表示数字类型（NUM）Token的状态，状态INID表示字符串类型Token的状态（如关键字和一般的标示符），状态INDBSYM表示双目运算符型Token的状态（如<=、>=、!=、==），状态DONE表示接收在开始状态START时
* 如果输入的字符为空白符，如空格换行等，则仍在START状态
* 如果输入的字符为digit，则进入状态INNUM，即可能是数字类型（NUM）Token的状态
* 如果输入的字符为letter，则进入状态INID，即可能是字符串类型Token的状态
* 如果输入的字符为>、<、!、=，则进入状态INDBSYM，即可能是双目运算符型Token的状态
* 如果输入的字符为是除以上之外的，则进入状态DONE，这次输入的字符可能是单目运算符、错误等
* 在状态INNUM时
* 如果输入的字符为digit，则仍停留在INNUM状态
* 如果输入的为其他的字符，则转到DONE状态
* 在状态INID时
* 如果输入的字符为letter，则仍停留在INID状态
* 如果输入的为其他的字符，则转到DONE状态
* 在状态INDBSYM时
* 如果输入的字符为=，将双目运算标志赋值为true后转到DONE状态
* 如果输入的为其他的字符，则直接转到DONE状态
* 在状态DONE时

接受状态，根据分析过程中获取的字符串确定Token的类型，并生成和保存相应的Token

状态。



图 C—的DFA

4 代码结构分析

4.1删除注释的DFA及代码实现：

删除注释的功能通过Scanner 类的成员函数deleteComments()实现，功能是将源代码中的注释过滤掉，将其余代码输出到sourceFile.txt文件中。deleteComments()函数按照上面的DFA编写，在函数中，state变量表示状态，共分为5个状态：

 在状态1时，如果输入的字符为/,则进入状态2，否则仍处在状态1，且输出输入的字符

 在状态2时，（此时可能进入注释状态），如果输入的字符为\*，则进入注释状态，状态将转到3，否则跳转到状态1，并输出/和输入的字符

 在状态3时，（此时可能结束注释状态），如果输入的字符为\*，此时状态将转到状态4，否则继续在状态3

 在状态4时，如果输入的字符为/，则注释状态结束，状态转移到结束状态

相应的DFA代码：

代码如下所示：

void Scanner :: deleteComments()

{

ofstream fout\_Sourse("sourseFile.txt");

int state = 1;

char ch;

while(state<6)

{

ch = getNextChar();

if('\0'==ch)//文件结束

break;

if(1==state)//DFA中的状态

{

if('/'==ch)

state = 2;

else

{

state = 1;

fout\_Sourse<<ch;

}

}

else if(2==state)//DFA中的状态

{

if('\*'==ch)

{

state = 3;

commentFlag = false;

}

else

{

state = 1;

fout\_Sourse<<"/"<<ch;

}

}

else if(3==state)//DFA中的状态

{

if('\*'==ch)

state = 4;

else

{

state = 3;

}

}

else if(4==state)//DFA中的状态

{

if('\*'==ch)

state = 4;

else if('/'==ch)

state = 5;

else

{

state = 3;

}

}

if(5==state)//结束状态，处理

{

commentFlag = true;

state = 1;

}

}

if(!commentFlag)

{

cout<<"注释错误，没有结束符！"<<endl;

scanSuccess = false;

}

}

4.2词法分析及代码实现

对应的DFA的代码分析：

词法分析的功能通过Scanner 类的成员函数scan()实现，功能是根据过滤掉注释的源代码获取Token并保存，获取到的Token通过printToken()函数输出到Token.txt文件中。scan()函数按照上面的DFA编写，在函数中，doubleSym变量表示双目运算符标志，当doubleSym为true时表示INDBSYM状态接收到的字符为'='，str变量是在词法分析过程中用来存储分析的Token的字符串，如INT型Token在分析过程中得到" int"的字符串，变量ch为当前输入的字符。分析到最后，在状态DONE中，根据分析过程中获取的str确定Token的类型，并生成和保存相应的Token。程序一共分为5个状态：START、INNUM、INID、INDBSYM、DONE。函数在各状态下的处理分析如下：

 在开始状态START时

 如果输入的字符为空白符，如空格换行等，则仍在START状态

 如果输入的字符为digit，则进入状态INNUM，即可能是数字类型（NUM）Token的状态，同时将ch添加到str中去（str += ch）

 如果输入的字符为letter，则进入状态INID，即可能是字符串类型Token的状态，同时将ch添加到str中去（str += ch）

 如果输入的字符为>、<、!、=，则进入状态INDBSYM，即可能是双目运算符型Token的状态，同时将ch添加到str中去（str += ch）

 如果输入的字符为是除以上之外的，则进入状态DONE，同时将ch添加到str中去（str += ch），这次输入的字符可能是单目运算符、错误等

 在状态INNUM时

 如果输入的字符为digit，则仍停留在INNUM状态，同时将ch添加到str中去（str += ch）

 如果输入的为其他的字符，则转到DONE状态

 在状态INID时

 如果输入的字符为letter，则仍停留在INID状态，同时将ch添加到str中去（str += ch）

 如果输入的为其他的字符，则转到DONE状态

 在状态INDBSYM时

 如果输入的字符为=，将双目运算标志赋值为true后转到DONE状态，同时将ch添加到str中去（str += ch），并将doubleSym 赋值为true（doubleSym = true），表示ch为‘=’

 如果输入的为其他的字符，则直接转到DONE状态

 在状态DONE时

接受状态，根据分析过程中获取的字符串确定Token的类型，并生成和保存相应的Token，同时将str设为空，（以便分析下一个Token）。

在DFA中，[other]表示ch没有被添加到str中后的转移，所以当状态是通过条件[other]跳转到DONE状态的情况时，则需将分析的字符回退回去（backToLastChar()）。

相应的DFA代码：

代码如下：

void Scanner :: scan()

{

bool doubleSym = false;

getSourseStringFromFile("sourseFile.txt");

int state = START;

lineCount = 0;

char ch;

//每一次循环都执行一次DFA，并获取到一个Token

while(state<6)

{

ch = getNextChar();

if('\0'==ch)//到达文件末尾，文件结束

{

Token t;

t.lineNo = lineCount;

t.tokenString = "";

t.tokenType = ENDFILE;

tokenList.push\_back(t);

break;

}

if(START==state)//初始状态和空格

{

state = charType(ch);

if(state!=START)

str += ch;

}

else if(INNUM==state)//digit

{

state = charType(ch);

if(state!=INNUM)

state = DONE;

else

str += ch;

}

else if(INID==state)//letter

{

state = charType(ch);

if(state!=INID)

state = DONE;

else

str += ch;

}

else if(INDBSYM==state)//除了<>=!之外的各种符号

{

if('='==ch)

{

str += ch;

doubleSym = true;

}

else

doubleSym = false;

state = DONE;

}

if(DONE==state)//接收状态

{

int tp = 0;

if('\n'==ch)

tp = 1;

//根据str生成Token

Token t;

t.lineNo = lineCount-tp;

t.tokenString = str;

t.tokenType = returnTokenType(str);

tokenList.push\_back(t);

if(ERROR==t.tokenType)

scanSuccess = false;

//针对[other]的情形，回退一个字符

int lastState = charType(str[str.length()-1]);

if(lastState==INNUM || lastState==INID || (lastState==INDBSYM && doubleSym==false))

backToLastChar();

str = "";

state = START;

if(doubleSym==true)

doubleSym = false;

}

}

printToken();//输出Token到文件Token.txt中

}

（1. 说明词法分析的整体代码结构 2. 说明DFA到代码的转换）

5 实验结果与分析

5.1 词法分析测试用例：

**实验组正确用例**：**（source.txt）**

/\* A program to perform Eucild's

Algorithm to compute gcd. \*/

int gcd (int u, int v)

{

if (v==0) return u;

else return gcd(v,u-u/v\*v);

/\* u-u/v\*v== u mod v \*/

}

void main(void)

{

int x; int y;

x=input(); y=input();

output(gcd(x,y));

}

>=;<=;==;!=;

**对照组错误用例：(source1.txt)**

/\* A program to perform Eucild's

Algorithm to compute gcd. \*/

int gcd (int u, int v)

{ ? returk; re9;

! if (v==0) return u;

" else return gcd(v,u-u/v\*v);

/\* u mod v \*/

}

**词法分析结果输出：**

**实验组正确结果：**

**1: /\* A program to perform Eucild's**

**2: Algorithm to compute gcd. \*/**

**3: int gcd (int u, int v)**

**3:reserved word: int**

**3:ID, name= gcd**

**3:(**

**3:reserved word: int**

**3:ID, name= u**

**3:,**

**3:reserved word: int**

**3:ID, name= v**

**3:)**

**4: {**

**4:{**

**5: if (v==0) return u;**

**5:reserved word: if**

**5:(**

**5:ID, name= v**

**5:==**

**5:NUM, val= 0**

**5:)**

**5:reserved word: return**

**5:ID, name= u**

**5:;**

**6: else return gcd(v,u-u/v\*v);**

**6:reserved word: else**

**6:reserved word: return**

**6:ID, name= gcd**

**6:(**

**6:ID, name= v**

**6:,**

**6:ID, name= u**

**6:-**

**6:ID, name= u**

**6:/**

**6:ID, name= v**

**6:\***

**6:ID, name= v**

**6:)**

**6:;**

**7: /\* u-u/v\*v== u mod v \*/**

**8:**

**9: }**

**9:}**

**10: void main(void)**

**10:reserved word: void**

**10:ID, name= main**

**10:(**

**10:reserved word: void**

**10:)**

**11: {**

**11:{**

**12: int x; int y;**

**12:reserved word: int**

**12:ID, name= x**

**12:;**

**12:reserved word: int**

**12:ID, name= y**

**12:;**

**13: x=input(); y=input();**

**13:ID, name= x**

**13:=**

**13:ID, name= input**

**13:(**

**13:)**

**13:;**

**13:ID, name= y**

**13:=**

**13:ID, name= input**

**13:(**

**13:)**

**13:;**

**14: output(gcd(x,y));**

**14:ID, name= output**

**14:(**

**14:ID, name= gcd**

**14:(**

**14:ID, name= x**

**14:,**

**14:ID, name= y**

**14:)**

**14:)**

**14:;**

**15: }**

**15:}**

**16: >=;<=;==;!=;!**

**16:>=**

**16:;**

**16:<=**

**16:;**

**16:==**

**16:;**

**16:!=**

**16:;**

**17:EOF**

**对照组错误结果：**

**1: /\* A program to perform Eucild's**

**2: Algorithm to compute gcd. \*/**

**3: int gcd (int u, int v)**

**3:reserved word: int**

**3:ID, name= gcd**

**3:(**

**3:reserved word: int**

**3:ID, name= u**

**3:,**

**3:reserved word: int**

**3:ID, name= v**

**3:)**

**4: { ? returk; re9;**

**4:{**

**4:ERROR: ?**

**4:ID, name= returk**

**4:;**

**4:ID, name= re**

**4:NUM, val= 9**

**4:;**

**5: ! if (v==0) return u;**

**5:ERROR: !**

**5:reserved word: if**

**5:(**

**5:ID, name= v**

**5:==**

**5:NUM, val= 0**

**5:)**

**5:reserved word: return**

**5:ID, name= u**

**5:;**

**6: " else return gcd(v,u-u/v\*v);**

**6:ERROR: "**

**6:reserved word: else**

**6:reserved word: return**

**6:ID, name= gcd**

**6:(**

**6:ID, name= v**

**6:,**

**6:ID, name= u**

**6:-**

**6:ID, name= u**

**6:/**

**6:ID, name= v**

**6:\***

**6:ID, name= v**

**6:)**

**6:;**

**7: /\* u mod v \*/**

**8: }**

**8:}**

**9:EOF**

（给出具体测试用例和测试结果，并进行分析。不能使用console截屏，建议采用表格形式。）

6 小结

收获:

* + 1. 熟悉C语言的结构、指针、文件方面的使用。
    2. 学会用startUml构造状态图，及用switch-case结构实现状态转换实现词法分析。
    3. 学会用递归下降方法实现EBNF文法规则，进而实现语法分析。
    4. 熟悉构造编译器的步骤及程序实现的框架及部分方法。通过实现Cminus的词法分析器和语法分析器，掌握了编译原理的理论知识，通过动手实现练习了正则表达式的设计，从正则表达式到NFA再到DFA的转换，理解了BNF文法和文法的二义性的消除。在词法分析过程中掌握了基本的调试功能。 通过语法分析，理解了递归下降分析和LL(1)文法。

不足:

1. 程序语法实现部分只用了递归下降分析一种方法。
2. 此程序框架与教材《编译原理及实践》附录B Tiny编译器的程序框架相似。
3. 递归下降分析方法手工实现应经验少程序逻辑错误多，调试费时久，不能保证程序的高健壮性。

程序的实现参考了书本中的tiny的源码实现，对其中部分不完善和不适用的部分进行了改进，但仍然出现了很多隐藏的bug，所幸的是程序能正常实现功能。该程序考虑的还不够全面，在运行的时候可能出现一些未知的没有考虑到的错误。

（描述收获与体会，总结经验和不足）