编译原理实验报告一

词法分析器的实现

姓名：井劭杰

学号：71116234

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering

College of Software Engineering

Southeast University

二0一九年一月

一、实验目的

构造一个词法分析器，以一个 C++程序为输入，以一个 TOKEN 序列为输出。

二、实验内容

1、Input

Stream of characters
REs(The number of REs is decided by yourself)

2、Output

Sequence of tokens

3、Classes of words are defined by yourself

4、Error handling may be included

三、实验方法

1、总体思想

Programming based on FA
a)Define some REs by yourself
b)Convert REs into NFAs
c)Merge these NFAs into a single NFA
d)Convert the NFA into a DFA' with minimum states
e)Programming based on the DFA'

2、单词分类

（1） 保留字

"include" , "iostream" , "define" , "main" , "void" , "abstract" , "static" , "const" , "auto" , "double" , "int" , "float" , "struct" , "long" , "unsigned" , "char" , "if" , "else" , "switch" , "case" , "return" , "break" , "for" , "while" , "do" , "continue" , "print"

（2） 运算符

+ | - | * | / | ++ | -- | < | > | = | >= | <= | == | (|) | [|] | { | }

（3） 界符

, | ; | : | # | " | <<

（4） 数值常数

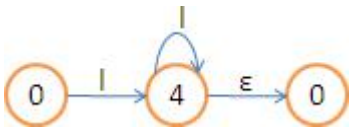
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

（5） 标识符

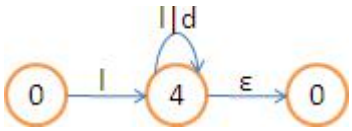
例如： a, b1, array, num

3、构造 NFA

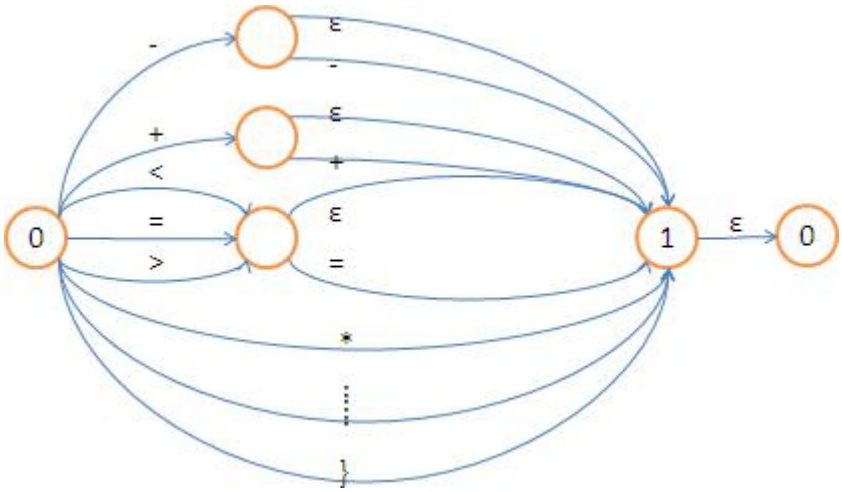
(1) 保留字：A..Za..z(A..Za..z)*



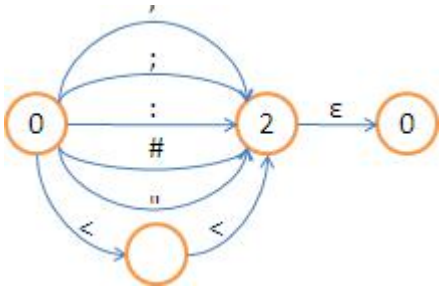
(2) 标识符：A..Za..z(A..Za..z | 0..9)*



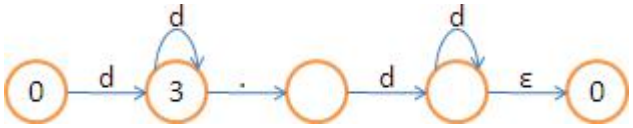
(3) 运算符：+ | - | * | / | ++ | -- | < | > | = | >= | <= | == | (|) | [|] | { | }



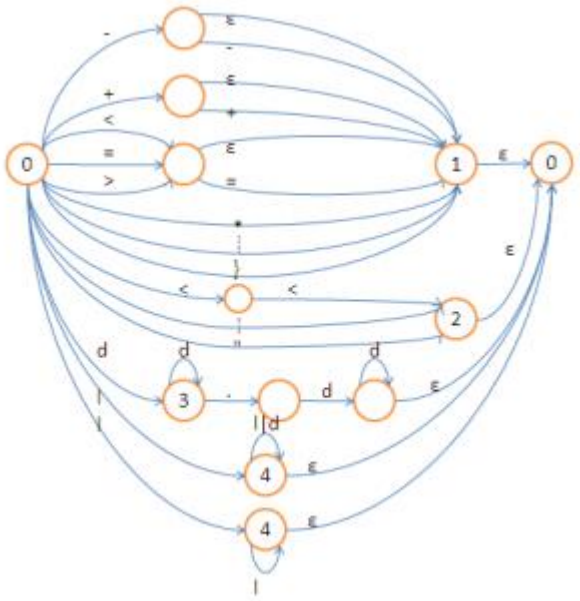
(4) 界符：, | ; | : | # | " | <<



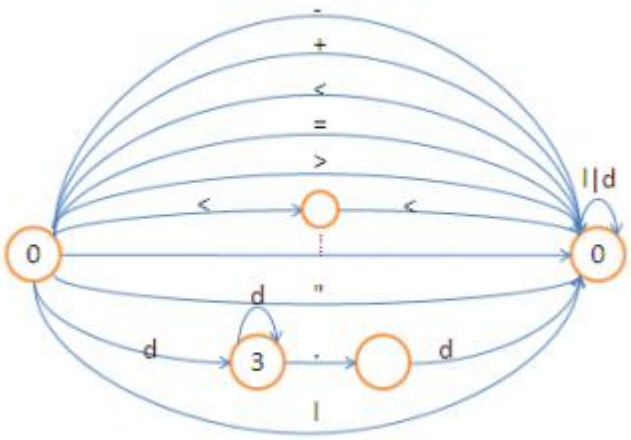
(5) 常数：(0..9)(0..9)* + (Ɛ | .(0..9)*)



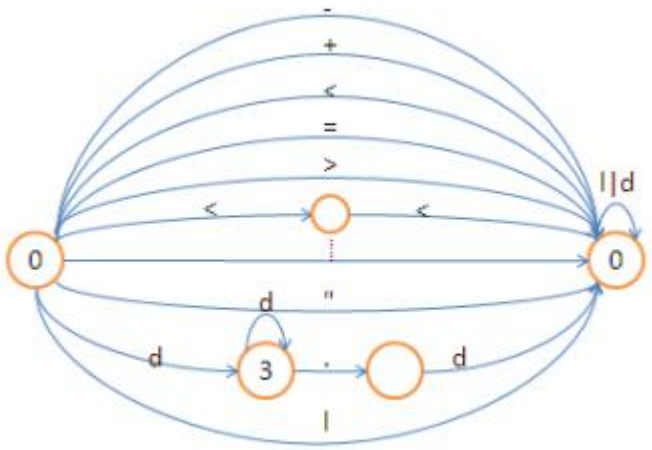
4、得到 NFA’



5、构造 DFA



6、得到 DFA’



四、实验代码

```
#include"stdafx.h"
#include<iostream>
using namespace std;
#define MAX 26

/*    Classification：
keyword----1
operator----2
delimiter----3
id----4
num----5
No-recognition----6
*/

char ch = ' ';
char token[100];//定义获取的字符

                        //定义 keyword
const char* keyWord[] = {
        "include","iostream","using","namespace","std""if","else","switch","case","break","for","while",
        "do","continue","true","false","const","auto","double","int","float","struct","long","char","main",
        "return","define","void","abstract","static","MAX","print","unsigned","short","class","system" };
```

```
//判断是否为关键字
bool isKey(char * token)
{
	for (int i = 0; i < MAX; i++)
	{
		if (strcmp(token, keyWord[i]) == 0)
			return true;
	}
	return false;
}

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

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


//词法分析
int Lexical_Analyze(FILE *input, FILE *output)
{
	while ((ch = fgetc(input)) != EOF) {

		//语句间的分隔符
		if (ch == ' ' || ch == '\n' || ch == '\t' || ch == '\r') {
			//ch = getc(input);
		}

		else if (isLetter(ch)) {
			char token[100] = { '\0' };
```

```
        int i = 0;

        while (isLetter(ch) || isDigit(ch)) {
                token[i] = ch;
                i++;
                ch = fgetc(input);
        }

        fseek(input, -1L, SEEK_CUR);

        if (isKey(token)) {
                fprintf(output, "%s\t\t%u\t%s\n", token, 1, "<keyword>");
        }
        else {
                fprintf(output, "%s\t\t%u\t%s\n", token, 4, "<id>");
        }
}

else if (isDigit(ch) || (ch == '.'))
{
        int i = 0;
        char token[100] = { '\0' };

        while (isDigit(ch) || (ch == '.' && isDigit(fgetc(input)))) {
                if (ch == '.')
                        fseek(input, -1L, SEEK_CUR);
                token[i] = ch;
                i++;
                ch = fgetc(input);
        }
        fseek(input, -1L, SEEK_CUR);
        //属于无符号数字
        fprintf(output, "%s\t\t%u\t%s\n", token, 5, "<num>");
}

else switch (ch) {
case '+': {
        ch = fgetc(input);
        if (ch == '+')
                fprintf(output, "%c%c\t\t%u\t%s\n", '+', ch, 2, "<operator>");
        else if (ch == '=')
                fprintf(output, "%c%c\t\t%u\t%s\n", '+', ch, 2, "<operator>");
        else {
                fprintf(output, "%c\t\t%u\t%s\n", '+', 2, "<operator>");
```

```
            fseek(input, -1L, SEEK_CUR);
        }
    }break;

    case '-': {
        ch = fgetc(input);
        if (ch == '-')
            fprintf(output, "%c%c\t\t%u\t%s\n", '-', ch, 2, "<operator>");
        else if (ch == '=')
            fprintf(output, "%c%c\t\t%u\t%s\n", '-', ch, 2, "<operator>");
        else {
            fprintf(output, "%c\t\t%u\t%s\n", '-', 2, "<operator>");
            fseek(input, -1L, SEEK_CUR);
        }
    }break;

    case '*': {
        ch = fgetc(input);
        if (ch == '=')
            fprintf(output, "%c%c\t\t%u\t%s\n", '*', ch, 2, "<operator>");
        else {
            fprintf(output, "%c\t\t%u\t%s\n", '*', 2, "<operator>");
            fseek(input, -1L, SEEK_CUR);
        }
    }break;

    case '/': {
        ch = fgetc(input);
        if (ch == '/') //注释符
            fprintf(output, "%c%c\t\t%u\t%s\n", '/', ch, 2, "<delimiter>");
        else if (ch == '=')
            fprintf(output, "%c\t\t%u\t%s\n", '/', ch, 2, "<operator>");
        else {
            fprintf(output, "%c\t\t%u\t%s\n", '/', 2, "<operator>");
            fseek(input, -1L, SEEK_CUR);
        }
    }break;

    case '(':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 2, "<operator>");
        break;
    case ')':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 2, "<operator>");
        break;
```

```
case '[':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 2, "<operator>");
        break;
case ']':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 2, "<operator>");
        break;
case '{':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 2, "<operator>");
        break;
case '}':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 2, "<operator>");
        break;
case '#':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 3, "<delimiter>");
        break;
case ',':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 3, "<delimiter>");
        break;
case '"':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 3, "<delimiter>");
        break;
case '\':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 3, "<delimiter>");
        break;
case ';':
        fprintf(output, "%c\t\t%u\t%s\n", ch, 3, "<delimiter>");
        break;

case '=': {
        ch = fgetc(input);
        if (ch == '=')
                fprintf(output, "%c%c\t\t%u\t%s\n", '=', ch, 2, "<operator>");
        else {
                fprintf(output, "%c\t\t%u\t%s\n", '=', 2, "<operator>");
                fseek(input, -1L, SEEK_CUR);
        }
}break;

case ':': {
        ch = fgetc(input);
        if (ch == '=')
                fprintf(output, "%c%c\t\t%u\t%s\n", ':', ch, 2, "<operator>");
        else {
                fprintf(output, "%c\t\t%u\t%s\n", ':', 2, "<operator>");
```

```
                fseek(input, -1L, SEEK_CUR);
            }
        }break;

        case '>': {
            ch = fgetc(input);
            if (ch == '>')
                fprintf(output, "%c%c\t\t%u\t%s\n", '>', ch, 3, "<delimiter>");
            if (ch == '=')
                fprintf(output, "%c%c\t\t%u\t%s\n", '>', ch, 2, "<operator>");
            else {
                fprintf(output, "%c\t\t%u\t%s\n", '>', 2, "<operator>");
                fseek(input, -1L, SEEK_CUR);
            }
        }break;

        case '<': {
            ch = fgetc(input);
            if (ch == '=')
                fprintf(output, "%c%c\t\t%u\t%s\n", '<', ch, 2, "<operator>");
            if (ch == '<')
                fprintf(output, "%c%c\t\t%u\t%s\n", '<', ch, 3, "<delimiter>");
            else {
                fprintf(output, "%c\t\t%u\t%s\n", '<', 2, "<operator>");
                fseek(input, -1L, SEEK_CUR);
            }
        }break;

            //无识别
        default:
            fprintf(output, "%c\t\t%u\t%s\n", ch, 6, "<No-recognition >");
        }
    }

    return 1;
}

int main() {
    char input[30];
    FILE *fin, *fout;

    fin = fopen("..\\bin\\in.txt", "r");
    if (fin == NULL) {
        cout << "The input file is not exist!" << endl;
```

```
        return 0;
    }

    fout = fopen("..\\bin\\out.txt", "w");
    if (fout == NULL) {
        cout << "The output file is not exist!" << endl;
        return 0;
    }

    if (Lexical_Analyze(fin, fout) == 1) {
        cout << "Lexical analyze succeeds!" << endl;
    }
    else
        cout << "Lexical analyze fails..." << endl;

    fclose(fin);
    fclose(fout);

    system("..\\bin\\out.txt");

    system("pause");
    return 0;
}
```

五、结果展示

输入文件：

in.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
int main() {
        int x = 100;
        int a[5] = {10,20,30,40,50};
        for(int i = 0; i < 5; i++ ){
                a[i] = (a[i] - x) * x;
                cout << a[i] << endl;
        }
        return 0;
}
```

输出文件：

out.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

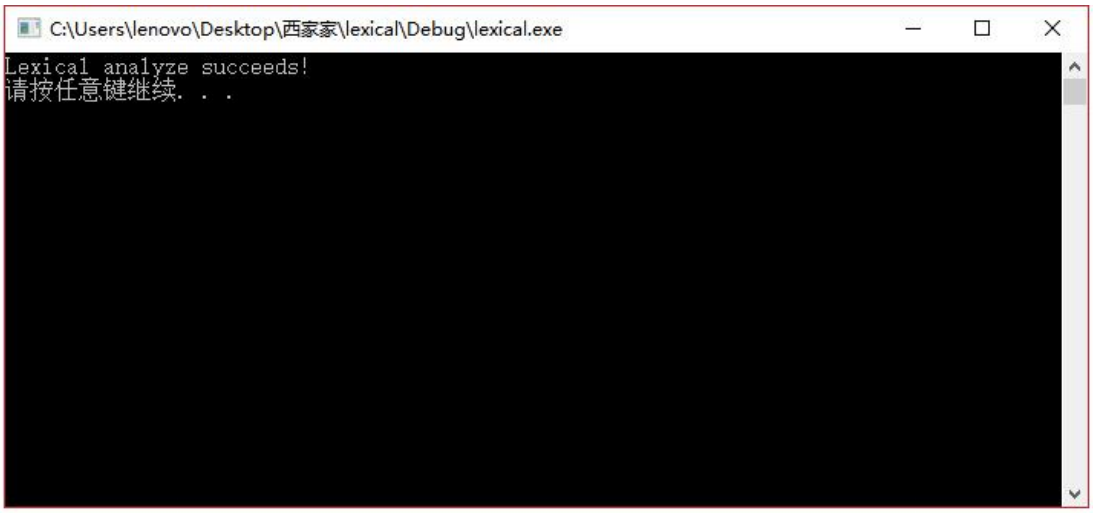
int	1	<keyword>
main	1	<keyword>
(	2	<operator>
)	2	<operator>
{	2	<operator>
int	1	<keyword>
x	4	<id>
=	2	<operator>
100	5	<num>
;	3	<delimiter>
int	1	<keyword>
a	4	<id>
[	2	<operator>
5	5	<num>
]	2	<operator>
=	2	<operator>
{	2	<operator>
10	5	<num>
,	3	<delimiter>
20	5	<num>
,	3	<delimiter>
30	5	<num>
,	3	<delimiter>
40	5	<num>
,	3	<delimiter>
50	5	<num>
}	2	<operator>
;	3	<delimiter>
for	1	<keyword>
(	2	<operator>
int	1	<keyword>
i	4	<id>
=	2	<operator>
0	5	<num>
;	3	<delimiter>
i	4	<id>
<	2	<operator>
5	5	<num>

out.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
i          4    <id>
<          2    <operator>
5          5    <num>
;          3    <delimiter>
i          4    <id>
++         2    <operator>
)          2    <operator>
{          2    <operator>
a          4    <id>
[          2    <operator>
i          4    <id>
]          2    <operator>
=          2    <operator>
(          2    <operator>
a          4    <id>
[          2    <operator>
i          4    <id>
]          2    <operator>
-          2    <operator>
x          4    <id>
)          2    <operator>
*          2    <operator>
x          4    <id>
;          3    <delimiter>
cout       4    <id>
<<         3    <delimiter>
a          4    <id>
[          2    <operator>
i          4    <id>
]          2    <operator>
<<         3    <delimiter>
endl       4    <id>
;          3    <delimiter>
}          2    <operator>
return     1    <keyword>
0          5    <num>
;          3    <delimiter>
}          2    <operator>
```

程序运行结果：



六、心得体会

本次试验的关键是根据正规表达式设计出对应的 NFA，之后的化简和编码就水到渠成了。通过本次实验，我更加深刻地理解了有限自动机在编译原理中的重要地位，也学习了如何在实际操作中使用有限自动机去解决问题，把理论和实践结合了起来。