实验一 词法分析扫描器的设计实现-手动

> 李钰
>
> 19335112

> 2022/3/15

实验一　词法分析扫描器的设计实现-手动

一、 手动设计词法分析器的要求

词法规则

功能

二、 实验内容

（一） 前期准备—明确可识别单词范围

标识符表

字符串表

常数表

关键字表

界符表

（二） 实现思路

（三） 代码实现

三、 实验结果

四、 遇到的困难

一、 手动设计词法分析器的要求

词法规则

了解所选择编程语言单词符号及其种别值

功能

输入一个C语言源程序文件demo.c

输出一个文件tokens.txt，该文件包括每一个单词及其种类枚举值，每行一个单词

二、 实验内容

（一） 前期准备—明确可识别单词范围

标识符表

所有标识符都记作40

序号	标识符
40	a
40	b
40	c

字符串表

序号	字符串
41	smaller
41	bigger

常数表

序号	常数值
50	0
50	1
50	2

关键字表

序号	关键字
1	int
2	long
3	short
4	float
5	double
6	char
7	unsigned
8	signed
9	const
10	void
11	volatile
12	enum
13	struct
14	union
15	if
16	else
17	goto
18	switch
19	case
20	do
21	while
22	for
23	continue
24	break
25	return
26	defalut
27	typedef
28	auto
29	register
30	extern
31	static
32	sizeof

序号	关键字
33	begin
34	then
35	end
36	cout
37	main
38	endl
39	END

界符表

序号	界符
-1	ERROR
0	#
51	:=
52	=
53	+
54	-
55	*
56	/
57	(
58	)
59	[
60	]
61	{
62	}
63	,
64	:
65	;
66	>
67	<
68	>=
69	<=
70	==
71	!=
72	"
73	<<
74	!
999	/* */
1000	\0

（二） 实现思路

大致思路是读取源代码文件，逐个字符逐个字符的分析，分析之前要消除空格、换行的空白符。

A. 如果首字符是字母：继续向下扫描，直到扫描到了既不是字母也不是数字的时候停止

- 将刚刚扫描得到的单词和关键字表中的单词比对，如果匹配成功，则判定刚刚扫描得到的单词是**关键字**，并记录它对应的标号。
- 若在关键字表中没有匹配到，那我们判断它的上一个`word`是否为双引号，即上一个token元素的`type`值是否为72，并且还要满足下一个字符也是双引号，这时我们判定其为**字符串**。
- 否则判断它为**标识符**。

B. 如果首字符是数字，则一直扫描直到从缓冲区中获取的字符不是数字，这是我们判定刚刚扫描得到的是一个**常数**

C. 如果是其他情况，那么我们判断它们是否为**界符**。这里举一个例子说明。

- 例如：一开始扫描得到了`<`这个字符，但由于不能确定是否为`<<`或者`<=`我们继续向下扫描，如果是相应的符号，则对应保存他们的序号，如果是其他字母或数字，则进行回退。
- 如果对比表中所有的界符都没有匹配项，那我们记它为ERROR。

（三） 代码实现

1. 结构体 Token：用于记录Token序列中`word`对应的序号和值。

```
typedef struct //词的结构
{
    int type; //单词种类
    char* content; //单词的内容
} Token;
```

2. 全局变量

```
char input[255];            //输入缓冲区
char output[255] = "";      //扫描获得的结果
int input_index;            //index，指向扫描缓冲区下一个读取的位置，读完++
int output_index;           //指针属性，指向结果的元素位置
char temp;                  //临时读取的一个字符
int pre_type;               //表示上一个word的type序号

//关键字表
char* key_table[] = {"int", "long", "short", "float", "double", "char", "unsigned",
"signed", "const", "void", "volatile", "enum", "struct", "union",
                "if", "else", "goto", "switch", "case", "do", "while", "for",
"continue", "break", "return", "default", "typedef",
                "auto", "register", "extern", "static", "sizeof", "begin", "then",
"end", "cout", "main", "endl", END};
```

3. 功能函数

```
//在输入缓冲区中向后获取一个字符并返回
```

```
char get_one_char(){
    temp = input[input_index];
    input_index++;
    return (temp);
}

//将读到的字符连成一个字符串
void connection(){
    output[output_index] = temp;
    output_index++;
    output[output_index] = '\0';
}

//搜索关键字
int search_keys(){
    int num = 0;
    while(strcmp(key_table[num], END)){
        if(!strcmp(key_table[num], output)){
            return num + 1;
        }
        num++;
    }

    //如果前一个和后一个word都是双引号，判断它为字符串
    if(pre_type == 72){
        int move = input_index;
        char next_ch = input[move];
        //消除中间的空白符
        while(input[move] == ' ' || temp == '\n' || temp == '\t'){
            next_ch = input[move];
            move++;
        }
        if(next_ch == '\"'){
            return 41;
        }
    }

    //否则判断它为标识符
    return 40;
}
```

4. 扫描函数—核心

返回是一个Token结构体，存放了该次扫描得到的`word`的序号和值。其大致过程是逐个字符逐个字符的进行扫描，利用`if-else`语句进行情况的判断。对于首字符是字母或下划线的情况，之后由字母、数字和下划线混合组成的`word`则让其进入`search_keys`函数中，判断它是关键字、标识符亦或是字符串。对于首字符是数字的`word`来说一直扫描到非数字字符出现，将前面的一串设置为常数，并存储其对应的序号。对于各个界符，利用`switch-case`的语句来分情况判断。以上情况都不符合，则返回ERROR。

```
//词法扫描
Token *do_scan()
{
    Token* myword = new Token;
    myword->type = -2;
    myword->content = "";
```

```
output_index = 0;
get_one_char();

while(temp == ' ' || temp == '\n' || temp == '\t'){
    temp = input[input_index];
    input_index++;
}

if (isalpha(temp)) {
    //如int
    while (isalpha(temp) || isdigit(temp) || temp == '_'){
        connection(); //连接
        get_one_char();
    }
    input_index--;                  //回退一个字符
    myword->type = search_keys();   //关键字、字符串、标识符
    myword->content = output;
    return (myword);

}else if (isdigit(temp)){ //判断读取到的单词首字符是数字

    while (isdigit(temp)){
        connection();
        get_one_char();
    }
    input_index--;
    //数字单词种别码统一为50，单词自身的值为数字本身
    myword->type = 50;
    myword->content = output;
    return (myword);
}else
    switch (temp){

    case '=':
        get_one_char(); //首字符为=,再读取下一个字符判断
        if (temp == '='){
            myword->type = 70;
            myword->content = "==";
            return (myword);
        }
        input_index--; //读取到的下个字符不是=，则要回退，直接输出=
        myword->type = 52;
        myword->content = "=";
        return (myword);
        break;
    case '+':
        myword->type = 53;
        myword->content = "+";
        return (myword);
        break;
    case '-':
        myword->type = 54;
        myword->content = "-";
        return (myword);
        break;
    case '/':           //读取到该符号之后，要判断下一个字符是什么符号，判断是否为注释
        get_one_char();     //首字符为/,再读取下一个字符判断
```

```
            if (temp == '*'){ // 说明读取到的是注释

                get_one_char();
                while (temp != '*'){
                    get_one_char(); //注释没结束之前一直读取注释，但不输出
                    if (temp == '*'){
                        get_one_char();
                        if (temp == '/'){ //注释结束
                            myword->type = 999;
                            myword->content = "annotation";
                            return (myword);
                            break;
                        }
                    }
                }
            }
            else{
                input_index--; //读取到的下个字符不是*，即不是注释，则要回退，直接输出/

                myword->type = 56;
                myword->content = "/";
                return (myword);
                break;
            }
        case '*':
            myword->type = 55;
            myword->content = "*";
            return (myword);
            break;
        case '(':
            myword->type = 57;
            myword->content = "(";
            return (myword);
            break;
        case ')':
            myword->type = 58;
            myword->content = ")";
            return (myword);
            break;
        case '[':
            myword->type = 59;
            myword->content = "[";
            return (myword);
            break;
        case ']':
            myword->type = 60;
            myword->content = "]";
            return (myword);
            break;
        case '{':
            myword->type = 61;
            myword->content = "{";
            return (myword);
            break;
        case '}':
            myword->type = 62;
            myword->content = "}";
            return (myword);
```

```
            break;
        case ',':
            myword->type = 63;
            myword->content = ",";
            return (myword);
            break;
        case ':':
            get_one_char();
            if (temp == '='){
                myword->type = 51;
                myword->content = ":=";
                return (myword);
                break;
            }
            else{
                input_index--;
                myword->type = 64;
                myword->content = ":";
                return (myword);
                break;
            }
        case ';':
            myword->type = 65;
            myword->content = ";";
            return (myword);
            break;
        case '>':
            get_one_char();
            if (temp == '='){
                myword->type = 68;
                myword->content = ">=";
                return (myword);
                break;
            }
            input_index--;
            myword->type = 66;
            myword->content = ">";
            return (myword);
            break;
        case '<':
            get_one_char();
            if (temp == '='){
                myword->type = 69;
                myword->content = "<=";
                return (myword);
                break;
            }
            else if (temp == '<'){
                myword->type = 73;
                myword->content = "<<";
                return (myword);
                break;
            }
            else{
                input_index--;
                myword->type = 67;
                myword->content = "<";
                return (myword);
```

```
            }
        case '!':
            get_one_char();
            if (temp == '='){
                myword->type = 71;
                myword->content = "!=";
                return (myword);
                break;
            }
            input_index--;
            myword->type = 74;
            myword->content = "NOT";
            return (myword);
            break;
        case '\"':
            myword->type = 72;
            myword->content = "\"";
            return (myword);
            break;
        case '\0':
            myword->type = 1000;
            myword->content = "OVER";
            return (myword);
            break;
        case '#':
            myword->type = 0;
            myword->content = "#";
            return (myword);
            break;
        default:
            myword->type = -1;
            myword->content = "ERROR";
            return (myword);
            break;
        }
}
```

三、 实验结果

- 测试源程序—"data.txt"

  该程序测试了常见的关键字、标识符、字符串、以及常数、界符等内容。

```
int main()
{
    int a = 1, b = 2;
    double c;
    int d;
    c = (double)b / a;
    /*test annotation*/
    if(c > a){
         cout << "bigger" << endl;
    }else{
        cout << "smaller" << endl;
    }
```

```
    return 0;
}
```

- 测试结果——"res.txt"

```
[  1    int  ]
[  37   main  ]
[  57   (  ]
[  58   )  ]
[  61   {  ]
[  1    int  ]
[  40   a  ]
[  52   =  ]
[  50   1  ]
[  63   ,  ]
[  40   b  ]
[  52   =  ]
[  50   2  ]
[  65   ;  ]
[  5    double  ]
[  40   c  ]
[  65   ;  ]
[  1    int  ]
[  40   d  ]
[  65   ;  ]
[  40   c  ]
[  52   =  ]
[  57   (  ]
[  5    double  ]
[  58   )  ]
[  40   b  ]
[  56   /  ]
[  40   a  ]
[  65   ;  ]
[  15   if  ]
[  57   (  ]
[  40   c  ]
[  66   >  ]
[  40   a  ]
[  58   )  ]
[  61   {  ]
[  36   cout  ]
[  73   <<  ]
[  72   "  ]
[  41   bigger  ]
[  72   "  ]
[  73   <<  ]
[  38   endl  ]
[  65   ;  ]
[  62   }  ]
[  16   else  ]
[  61   {  ]
[  36   cout  ]
[  73   <<  ]
[  72   "  ]
[  41   smaller  ]
[  72   "  ]
```

```
[  73   <<  ]
[  38   endl  ]
[  65   ;  ]
[  62   }  ]
[  25   return  ]
[  50   0  ]
[  65   ;  ]
[  62   }  ]
```

四、 遇到的困难

- 最一开始持续不断地读入文件到`input`缓冲区中给我带来了一点小困难，分析`gets()`,`getline()`以及`scanf`的详细用法后得到了解决。并学习到：在`scanf`中`%[^#]`的形式意思为一直读到遇到`#`才结束。
- 在写扫描函数时，额外需要注意对于多于一个字符长度的界符判断，向后读取了不符合的字符时一定要记得回退，即把指针`input_index`减一，不然下次扫描时会漏掉了一个字符。
- 在关于字符串和标识符的判断上我也遇到了问题，后来想到在C/C++程序中字符串必须是在双引号内的，于是设置了记录前一个`word`种类的变量`pre_type`用于判断其前一位是否为双引号，以及顺次向后扫描一位看是否也为双引号，以此来判断是否是字符串。
- 思考不细致容易漏掉一些情况，例如一开始我漏掉了标识符中可能包含下划线的情况。