编译原理实验报告

词法分析程序的设计与实现

姓名：胡敏臻

班级：2019211307

学号：2019211424

日期：2021/10/9

1 概述

1.1 实验内容

设计并实现 C 语言的词法分析程序,要求实现如下功能。

（1）可以识别出用 C 语言编写的源程序中的每个单词符号,并以记号的形式输出每个单词符号。

（2）可以识别并跳过源程序中的注释。

（3）可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。

（4）检查源程序中存在的词法错误,并报告错误所在的位置。

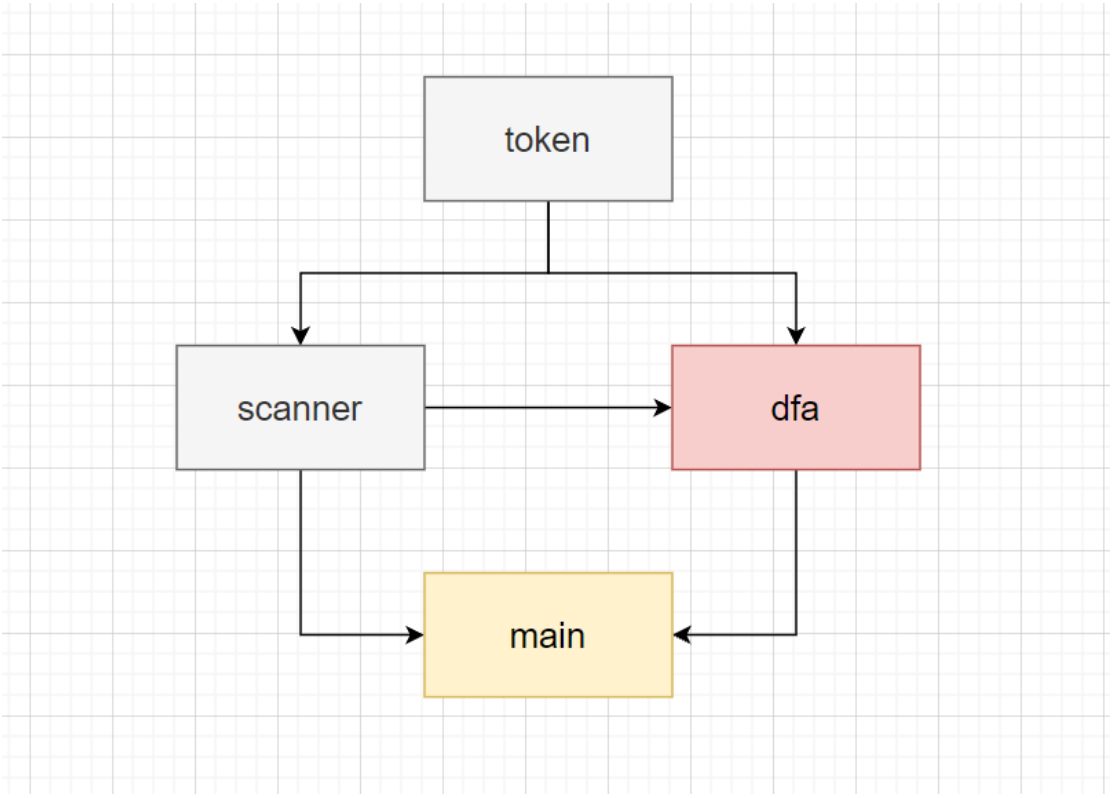
（5)对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行,对源程序进行一次扫描,即可检查并报告源程序中存在的所有词法错误。

1.2 实验环境

Visual Studio 2019

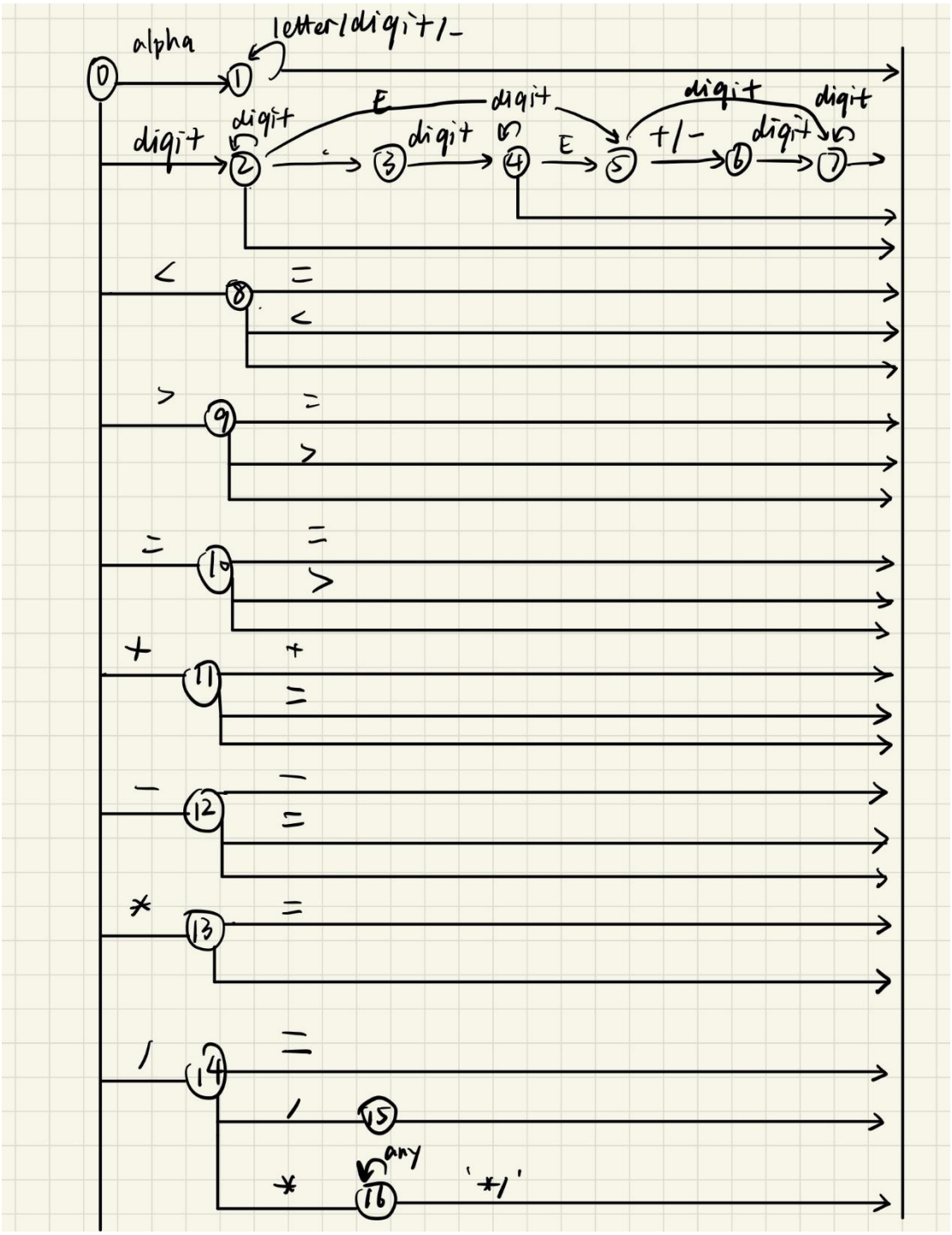
2 总体设计

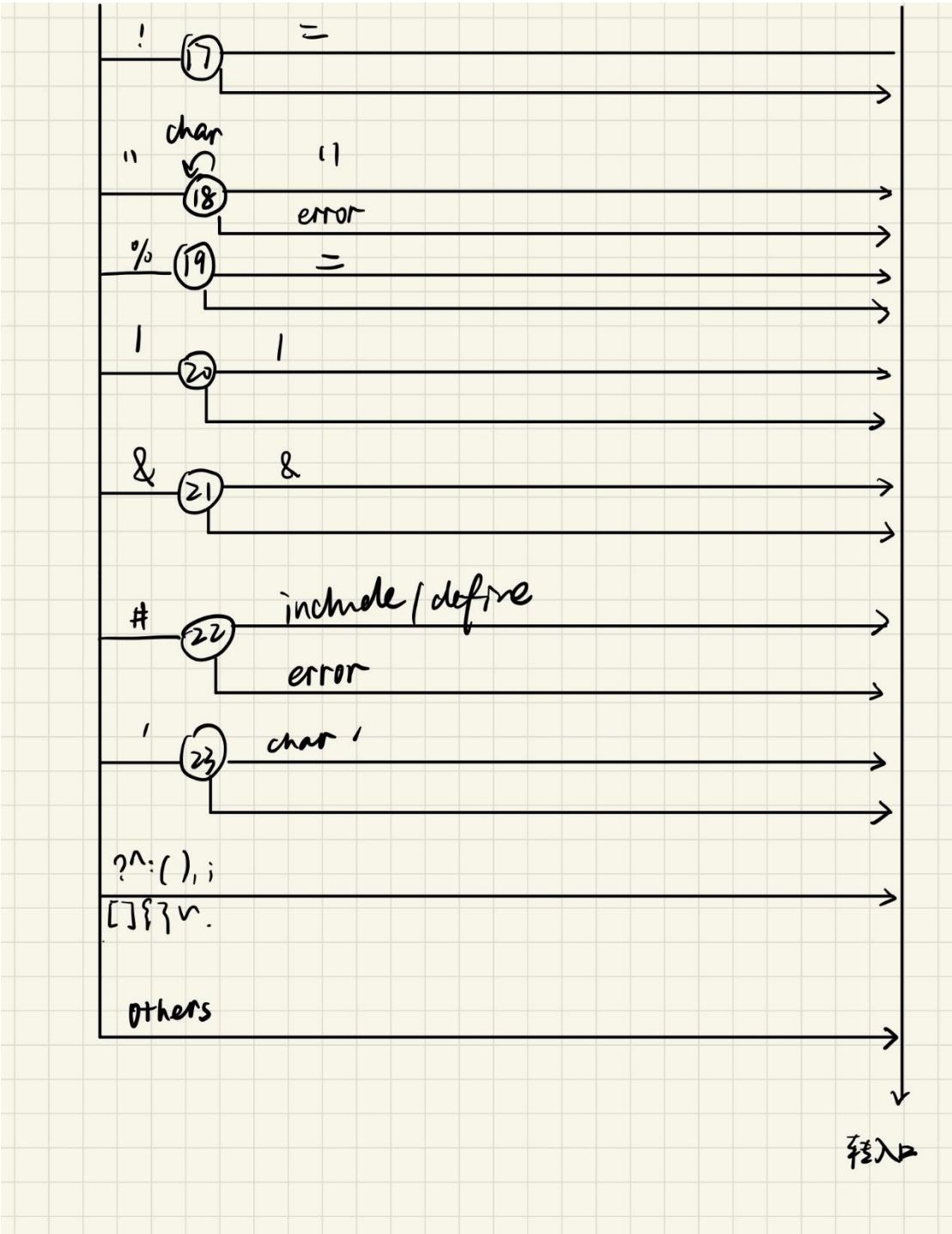
2.1 模块设计



token:记录 token 类型和种类
scanner:从缓冲区读取字符
dfa:根据 dfa 来进行词法分析
main:总运行

2.2 dfa 转换图


alpha
letter/digit/_
digit
digit
E
digit
digit
digit
digit
+/-
digit
E
<
=
<
>
=
>
=
=
+
+
=
-
-
=
*
=
/
=
/
*
any
'*/'
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16



3 实现说明

1、建一个 token 类，定义说明 token 存储的类型，以及需要定义 token 的种类，在识别出该类型的时候创立一个 token 实例，并以指针形式进行存储。

```
class token
{
public:
    tokenType type;
```

```
    int symbolPos = -1;
    std::string value = "";
    int line1, row1;
}
```

关于 tokenType 的设定，里面包括关键字、分隔符、以及自己设定的种类与错误类型等各种，具体设定如下；

```
enum class tokenType {
    //识别有误
    UNKNOWN = 0,
    //保留关键字
    AUTO, BREAK, CASE, CHAR, CONST, CONTINUE,
    DEFAULT, DO, DOUBLE, ELSE, ENUM, EXTERN, FLOAT, FOR, GOTO,
    IF, INT, LONG, REGISTER, RETURN, SHORT, SIGNED, STATIC,
    SIZEOF, STRUCT, SWITCH, TYPEDEF, UNIONUNSIGNED, VOID, VOLATILE, WHILE,
    //运算符
    ASSIGN,         //=
    ADD,            //=
    INC,            //++
    ADD_ASSIGN,     //+=
    SUB,            //-
    DEC,            //--
    SUB_ASSIGN,     //-=
    MUL,            //*
    MUL_ASSIGN,     //*=
    DIV,            // /
    DIV_ASSIGN,     // /=
    MOD,            //%
    MOD_ASSIGN,     //%=
    BITWISE_AND,    //&
    BITWISE_OR,     // |
    BITWISE_XOR,    // ^
    BITWISE_NOT,    // ~
    SHL,            // <<
    SHR,            // >>
    AND,            // &&
    OR,             // ||
    NOT,            // !
    LESS,           // <
    LESS_EQUAL,     // <=
    EQUAL,          // ==
    INEUQUAL,       // !=
    GREATER,        // >
    GREATER_EQUAL,// >=
```

```
    QUESTION,     //?
    COMMA,        //,
    COLON,        //:
    SEMICOLON,    //;
    DOT,          //.
    ARROW,        //=>
    L_BRACE,      //{
    R_BRACE,      //}
    L_SQUARE,     //[
    R_SQUARE,     //]
    L_PAREN,      //(
    R_PAREN,      //)
    WELL,         //#

    ID, //标识符
    NUM, //数字
    ACHAR, //字符
    STRING_BLOCK, //字符块
    PREACTION, //预处理
    TAB, //制表符

    ERROR_EXP, //注释错误
    EEEOR_NUM,  //数字表达错误
    ERROE_CHAR, //单引号字符错误
    ERROR_STRING, //字符串引号错误
    EEROR_PREACTION, //预处理错误

};
```

2、在 scanner 类中，实现字符串的读取，设置上下两个缓冲区进行读取，防止文件过大，分两个半区进行读写，当向前指针读到字符串末尾时，换一个缓冲区进行存储，当文件读到末尾时，在字符串后赋值 EOF。

在 scanner 类中，还需要记录此时读到的行和列，用于输出。在读取字符的时候需要完成读取空白字符，回退字符，读取字符，缓冲区读取等操作。需要注意的是：在回退字符的时候，之前记录的行和列也需要进行相应的修改。并且因为在 main 中与 dfa 中需要用到的是同一个类 scan，为了方便代码操作，直接将 scan 类 extern 了，这样就可以保证在不同的文件中访问的都是同一个实例 scan。

```
class scanner
{
private:
    char buffer_1[buffer_len]; //上半缓冲区
    char buffer_2[buffer_len]; //下半缓冲区
```

```
    char* lexemebegin = buffer_1;
    char* forwardp = buffer_1;

public:
    std::ifstream myfile;

private:
    std::string temps; //存放读进来的字符串
    int flag = 0; //flag为0表示正在上半缓冲区，flag为1表示正在下半缓冲区
    char c;//读入的字符串
public:
    int line=0,row=0;
    int t_row=0;
public:
    void init();
    void get_nbc();
    char get_char();
    void read_half(char buffer[]);
    void retract();
    char get_c();
    //bool is_end();
};
extern scanner scan;
```

3、在 dfa 类中，其中的 dfa_run()函数是程序语法分析的主要部分，关于 dfa 的状态转换图已经 2.2 中进行展示，此 dfa 的转换图在书上的基础上进行扩展，是其能识别出更多种类的 token。

并且在该类中定义了符号表来记录被识别的标识符，符号表的实现为字符串数组，在记录 token 的时候仅需要记录在符号表中的索引即可。同时为了统计各类字符串的数量，还建立了数组以记录每个种类出现的数量。同时为了找到 tokenType 的具体对应，还存储了字符数组以方便在打印的时候进行字符串数组对应。

```
#define type_number 84
class dfa
{
private:
    int state = 0; //记录此时处于自动机的哪个状态
    //int line = 0, row = 0;//记录此时的行数和列数
    std::string temps; //记录此时临时字符串
    std::vector<std::string> my_table; //记录标识符的符号表
    int counter[type_number]; //计数，记录每种种类出现过几次
    int indentry=0;
    char C;
    token* token_ptr = nullptr;
```

```
    std::vector<token*> all_token;
    std::vector<std::string>
keywords{ "auto","break","case","char","const","continue",
    "default","do","double","else","enum","extern","float","for","goto",
    "if","int","long ","register","return","short","signed","static",
    "sizeof","struct","switch","typedef","unionunsigned","void","volatile","while
" };
    std::vector<std::string> string_keywords{
    "UNKNOWN ",  "AUTO", "BREAK", "CASE", "CHAR", "CONST", "CONTINUE","DEFAULT",
"DO", "DOUBLE",
    "ELSE", "ENUM", "EXTERN", "FLOAT", "FOR", "GOTO","IF", "INT", "LONG",
"REGISTER",
    "RETURN", "SHORT", "SIGNED", "STATIC","SIZEOF", "STRUCT", "SWITCH", "TYPEDEF",
"UNIONUNSIGNED", "VOID",
    "VOLATILE", "WHILE","ASSIGN", "ADD",  "INC", "ADD_ASSIGN","SUB", "DEC",
    "SUB_ASSIGN","MUL",
    "MUL_ASSIGN", "DIV","DIV_ASSIGN","MOD","MOD_ASSIGN","BITWISE_AND","BITWISE_OR",
"BITWISE_XOR","BITWISE_NOT", "SHL",
    "SHR",   "AND",   "OR","NOT",  "LESS",  "LESS_EQUAL","EQUAL", "INEUQUAL",
"GREATER","GREATER_EQUAL,","QUESTION",
    "COMMA","COLON", "SEMICOLON", "DOT","ARROW", "L_BRACE"," R_BRACE","L_SQUARE",
"R_SQUARE","L_PAREN",
    "R_PAREN", "WELL",
    "ID", //标识符
    "NUM", //数字
    "ACHAR", //字符
    "STRING_BLOCK", //字符块
    "PREACTION", //预处理
    "TAB",//制表符
    " ERROR_EXP ", //注释错误
    "EEEOR_NUM",  //数字表达错误
    "ERROE_CHAR", //单引号字符错误
    "ERROR_STRING", //字符串引号错误
    "EEROR_PREACTION", //预处理错误
    };
public:
    void dfa_run();
    int judge_case0(char x);
    int table_insert();
    int is_keywords();
    void print_token();
    void print_static();
    //void print_error();
};
```

4、在 main 中需要读入我们需要识别的文件名，将文件名赋值给 scan 中的变量，之后运行 dfa_run()，执行完成后调用打印函数输出记号和统计结果。

```
int main() {
    //双缓冲区读入文件

    std::string filename;
    std::cout << "情输入语法分析的文件路径" << std::endl;
    std::cin >> filename;

    scan.myfile.open(filename, std::ios::in);
    scan.init();
    dfa lexicial;
    lexicial.dfa_run();
    lexicial.print_static();
    lexicial.print_token();

    scan.myfile.close();

    return 0;
}
```

4 关于错误情况处理

4.1 错误数字输入

当数字输入以“.”为结尾或者以“E”结尾或者以“+/-”结尾时，此种表达方式是错误的，是不合规范的，在 case3,case5,case6 中都会判断出此种错误。

```
        case 6:
            C = scan.get_char();
            if (isdigit(C)) {
                temps += C;
                state = 7;
            }
            else {
                state = 0;
                scan.retract();
                token_ptr = new token(tokenType::EEEOR_NUM, temps, scan.line,
scan.row);
                all_token.push_back(token_ptr);
                counter[(int)tokenType::EEEOR_NUM]++;
            }
            break;
```

4.2 块注释未闭合

当在程序中只出现 “/*”而没有出现“*/”闭包的时候，程序在最后识别的时候可以实现出这种错误，在代码中用 flag_exp 表示块注释是否出错。

```
    if (flag_exp == 0) {
        token_ptr = new token(tokenType::ERROR_EXP, temps, scan.line, scan.row);
        all_token.push_back(token_ptr);
        counter[(int)tokenType::ERROR_EXP]++;
    }
```

4.3 错误字符串输入

当出现一串字符中只有一个“”时，并且当整一行读完之后还没有出现第二个“”之后则说明此时的字符串输入错误。

```
            if (C != '"') {
                if (C == '\n') {
                    state = 0;
                    token_ptr = new token(tokenType::ERROR_STRING, temps,
scan.line, scan.row);
                    all_token.push_back(token_ptr);
                    counter[(int)tokenType::ERROR_STRING]++;
                    break;
                }
                state = 18;
            }
```

4.4 错误单个字符输入

单个字符的输入说明在“‘’”中只能读入一个字符而不能出现两个字符，因此直接读两个字符判断第二个符号是否是单引号。

```
            if (C == '\'') {
                token_ptr = new token(tokenType::ACHAR, temps, scan.line,
scan.row);
                all_token.push_back(token_ptr);
                counter[(int)tokenType::ACHAR]++;
            }
            else {
                token_ptr = new token(tokenType::ERROE_CHAR, temps, scan.line,
scan.row);
                all_token.push_back(token_ptr);
                counter[(int)tokenType::ERROE_CHAR]++;
            }
```

4.5 错误预处理

对于预处理的错误情况分析，因为我们知道“#”之后，一定跟的是 include 或者 define，因此若“#”之后不是这两个字符，我们就判断为预处理错误。

```
if (C == 'i') {
                for (i = 1; i < 7; i++) {
                    C = scan.get_char();
                    if (C == a1[i]) {
                        temps += C;
                    }
                    else {
                        state = 0;
                        scan.retract();
                        token_ptr = new token(tokenType::EEROR_PREACTION, temps,
scan.line, scan.row);
                        all_token.push_back(token_ptr);
                        counter[(int)tokenType::EEROR_PREACTION]++;
                        break;
                    }
                }
                if (i == 7) {
                    token_ptr = new token(tokenType::PREACTION, temps, scan.line,
scan.row);
                    all_token.push_back(token_ptr);
                    counter[(int)tokenType::PREACTION]++;
                }
            }
            else if (C == 'd') {
                for (i = 1; i < 6; i++) {
                    C = scan.get_char();
                    if (C == a2[i]) {
                        temps += C;
                    }
                    else {
                        state = 0;
                        scan.retract();
                        token_ptr = new token(tokenType::EEROR_PREACTION, temps,
scan.line, scan.row);
                        all_token.push_back(token_ptr);
                        counter[(int)tokenType::EEROR_PREACTION]++;
                        break;
                    }
                }
                if (i == 6) {
```

```
                    token_ptr = new token(tokenType::PREACTION, temps, scan.line,
scan.row);
                    all_token.push_back(token_ptr);
                    counter[(int)tokenType::PREACTION]++;
                }
            }
            else {
                state = 0;
                scan.retract();
                token_ptr = new token(tokenType::EEROR_PREACTION, temps, scan.line,
scan.row);
                all_token.push_back(token_ptr);
                counter[(int)tokenType::EEROR_PREACTION]++;
            }
```

4.6 无法识别的输入

对于一些利用 dfa 无法识别的标识符或者符号，以 UNKNOWN 的 token 种类进行存储。

```
        default:
            t_state = 0;
            token_ptr = new token(tokenType::UNKNOWN, x, scan.line, scan.row);
            all_token.push_back(token_ptr);
            counter[(int)tokenType::UNKNOWN]++;
            break;
        }
```

5 测试样例说明

5.1 测试样例 1

此样例为正确运行时的样例过程：

```
#include<stdio.h>

int main(){
    int a,b;
    a=2;
    /*
    b=3;*/
    b=4;
    //can you see me?
    printf("a+b=%d",a+b);
    return 0;
}
```

样例结果如下：

Microsoft Visual Studio 调试控制台

```
情输入语法分析的文件路径
test1.txt

程序语句行数 : 12
各类单词数据统计如下：
UNKNOWN  : 1
AUTO : 0
BREAK : 0
CASE : 0
CHAR : 0
CONST : 0
CONTINUE : 0
DEFAULT : 0
DO : 0
DOUBLE : 0
ELSE : 0
ENUM : 0
EXTERN : 0
FLOAT : 0
FOR : 0
GOTO : 0
IF : 0
INT : 2
LONG : 0
REGISTER : 0
RETURN : 1
SHORT : 0
SIGNED : 0
STATIC : 0
SIZEOF : 0
STRUCT : 0
SWITCH : 0
TYPEDEF : 0
UNIONUNSIGNED : 0
VOID : 0
VOLATILE : 0
WHILE : 0
ASSIGN : 2
ADD : 1
INC : 0
ADD_ASSIGN : 0
```

Microsoft Visual Studio 调试控制台

```
INC : 0
ADD_ASSIGN : 0
SUB : 0
DEC : 0
SUB_ASSIGN : 0
MUL : 0
MUL_ASSIGN : 0
DIV : 0
DIV_ASSIGN : 0
MOD : 0
MOD_ASSIGN : 0
BITWISE_AND : 0
BITWISE_OR : 0
BITWISE_XOR : 0
BITWISE_NOT : 0
SHL : 0
SHR : 0
AND : 0
OR : 0
NOT : 0
LESS : 1
LESS_EQUAL : 0
EQUAL : 0
INEUQUAL : 0
GREATER : 1
GREATER_EQUAL, : 0
QUESTION : 0
COMMA : 2
COLON : 0
SEMICOLON : 5
DOT : 1
ARROW : 0
L_BRACE : 1
 R_BRACE : 1
L_SQUARE : 0
R_SQUARE : 0
L_PAREN : 2
R_PAREN : 2
WELL : 0
ID : 10
NUM : 3
```

Microsoft Visual Studio 调试控制台

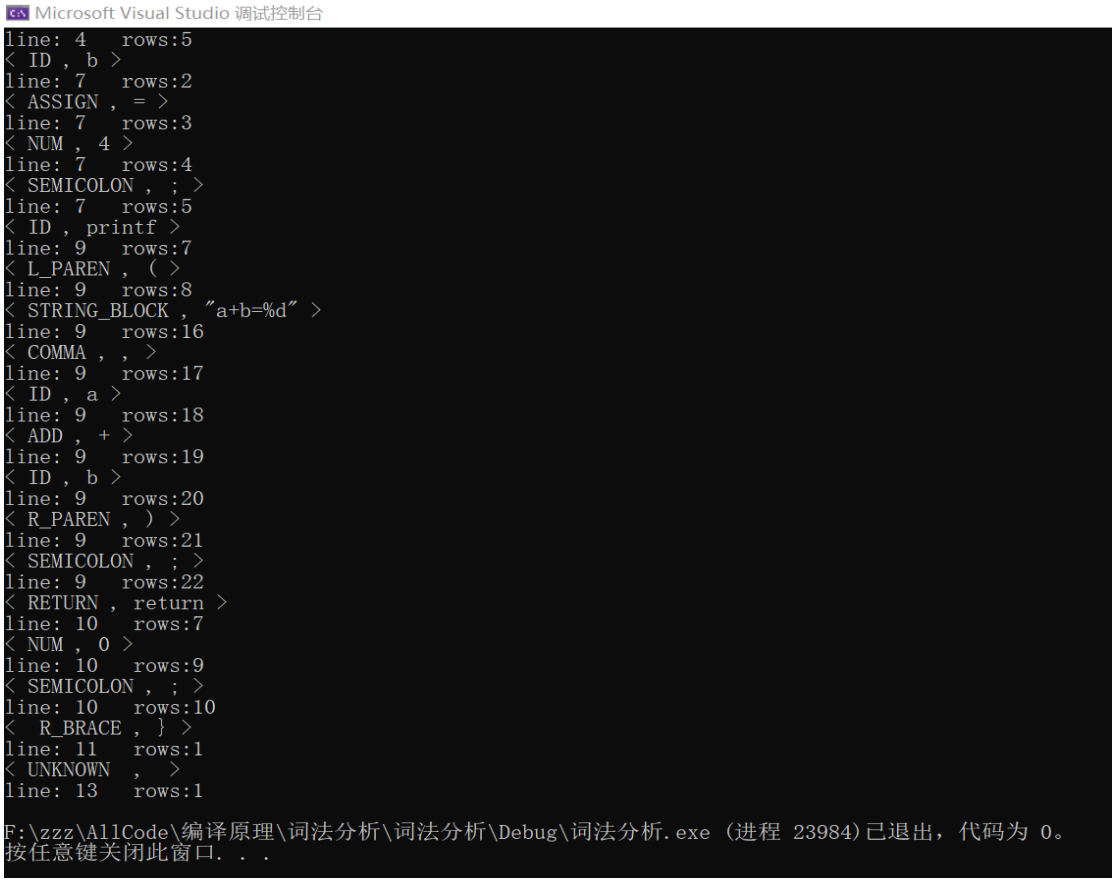
```
ID : 10
NUM : 3
ACHAR : 0
STRING_BLOCK : 1
PREACTION : 1
TAB : 0
ERROR_OPER : 0
EEEOR_NUM : 0
ERROE_CHAR : 0
ERROR_STRING : 0
EEROR_PREACTION : 0

字符总数为：38

词法分析结果如下，并以记号形式输出
< PREACTION , #include >
line: 0   rows:8
< LESS , < >
line: 0   rows:9
< ID , stdio >
line: 0   rows:14
< DOT , . >
line: 0   rows:15
< ID , h >
line: 0   rows:16
< GREATER , > >
line: 0   rows:17
< INT , int >
line: 2   rows:3
< ID , main >
line: 2   rows:8
< L_PAREN , ( >
line: 2   rows:9
< R_PAREN , ) >
line: 2   rows:10
< L_BRACE , { >
line: 2   rows:11
< INT , int >
line: 3   rows:4
< ID , a >
line: 3   rows:6
```

Microsoft Visual Studio 调试控制台

```
< ID , a >
line: 3   rows:6
< COMMA , , >
line: 3   rows:7
< ID , b >
line: 3   rows:8
< SEMICOLON , ; >
line: 3   rows:9
< ID , a >
line: 4   rows:2
< ASSIGN , = >
line: 4   rows:3
< NUM , 2 >
line: 4   rows:4
< SEMICOLON , ; >
line: 4   rows:5
< ID , b >
line: 7   rows:2
< ASSIGN , = >
line: 7   rows:3
< NUM , 4 >
line: 7   rows:4
< SEMICOLON , ; >
line: 7   rows:5
< ID , printf >
line: 9   rows:7
< L_PAREN , ( >
line: 9   rows:8
< STRING_BLOCK , "a+b=%d" >
line: 9   rows:16
< COMMA , , >
line: 9   rows:17
< ID , a >
line: 9   rows:18
< ADD , + >
line: 9   rows:19
< ID , b >
line: 9   rows:20
< R_PAREN , ) >
line: 9   rows:21
< SEMICOLON , ; >
```



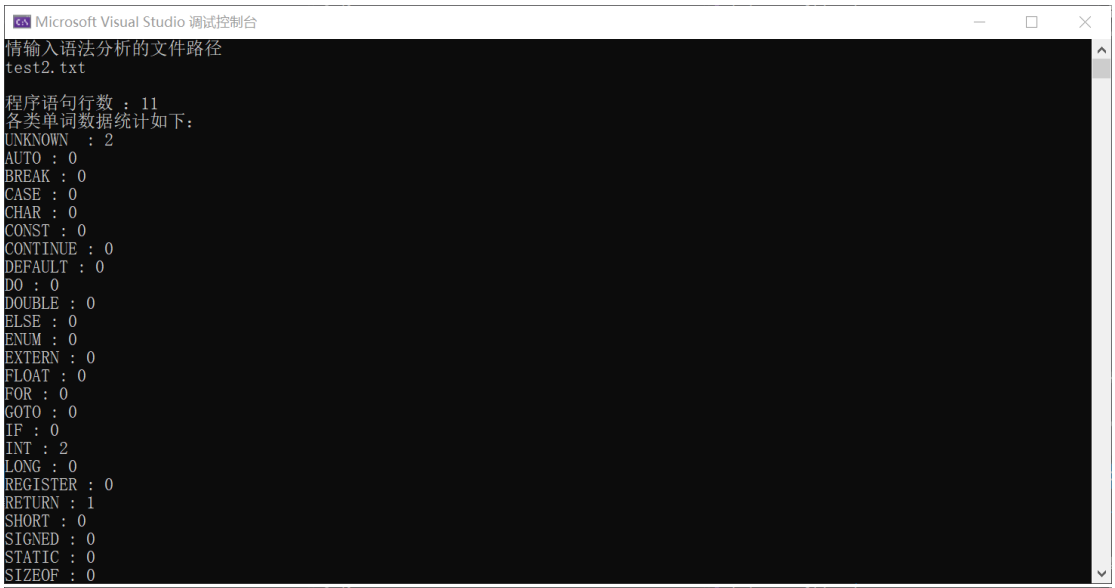
5.2 测试样例 2

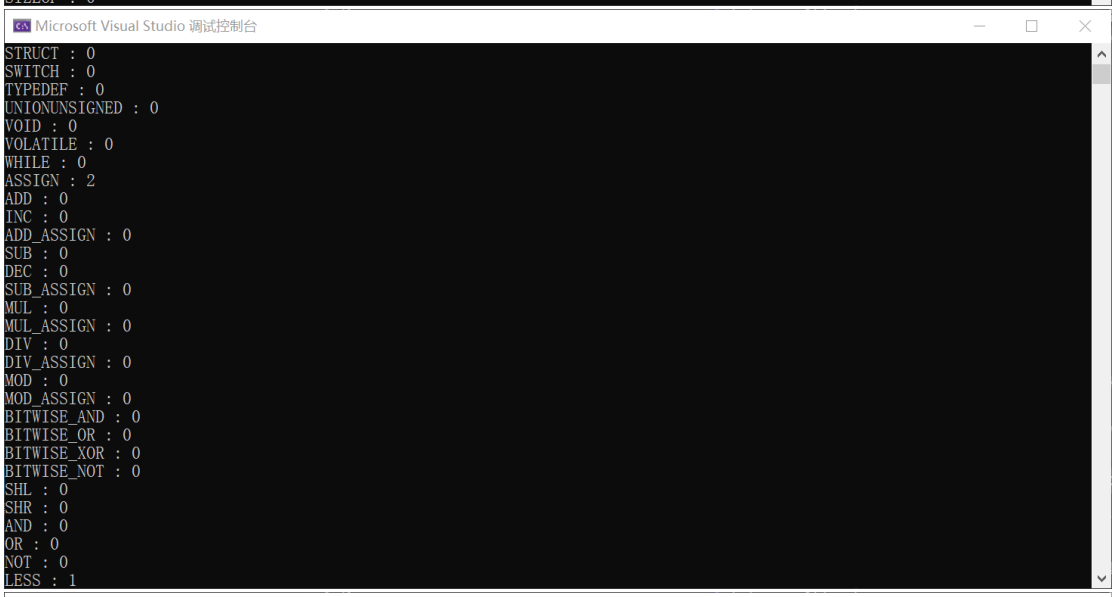
此样例为出现错误时的样例，出现的错误为识别一个错误字符与错误的字符串表示。

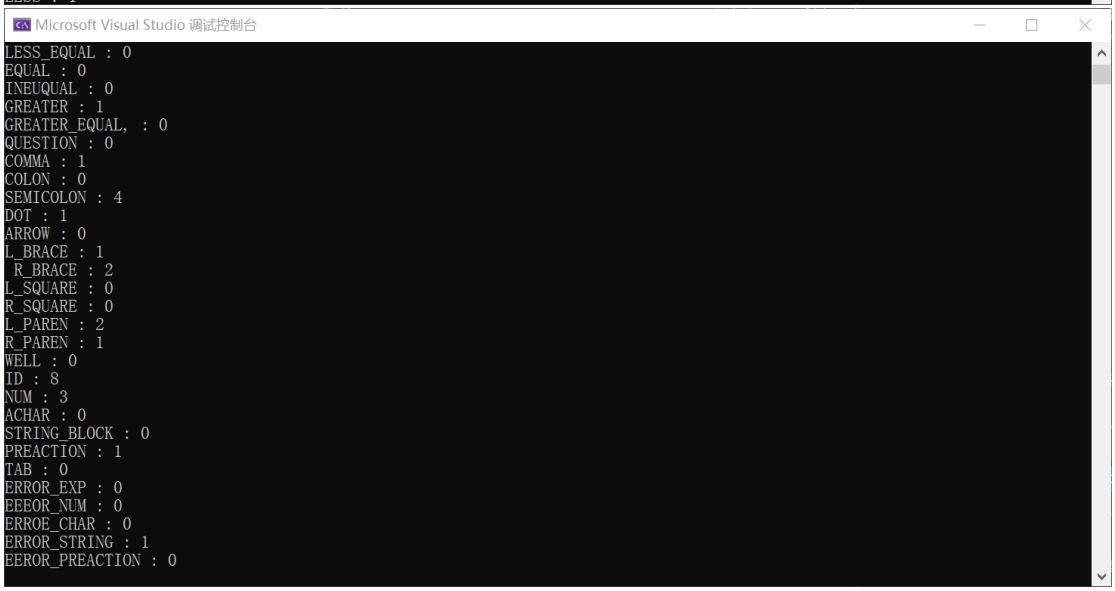
```
#include<stdio.h>

int main(){
    int a,b;
    a=2;
    /*
    b=3;*/
    b=4;
    @
    //can you see me?
    printf("a+b=%d,a+b);
    return 0;
}
```

运行的结果如下所示：


Microsoft Visual Studio 调试控制台
情输入语法分析的文件路径
test2.txt
程序语句行数 : 11
各类单词数据统计如下:
UNKNOWN : 2
AUTO : 0
BREAK : 0
CASE : 0
CHAR : 0
CONST : 0
CONTINUE : 0
DEFAULT : 0
DO : 0
DOUBLE : 0
ELSE : 0
ENUM : 0
EXTERN : 0
FLOAT : 0
FOR : 0
GOTO : 0
IF : 0
INT : 2
LONG : 0
REGISTER : 0
RETURN : 1
SHORT : 0
SIGNED : 0
STATIC : 0
SIZEOF : 0


Microsoft Visual Studio 调试控制台
STRUCT : 0
SWITCH : 0
TYPEDEF : 0
UNIONUNSIGNED : 0
VOID : 0
VOLATILE : 0
WHILE : 0
ASSIGN : 2
ADD : 0
INC : 0
ADD_ASSIGN : 0
SUB : 0
DEC : 0
SUB_ASSIGN : 0
MUL : 0
MUL_ASSIGN : 0
DIV : 0
DIV_ASSIGN : 0
MOD : 0
MOD_ASSIGN : 0
BITWISE_AND : 0
BITWISE_OR : 0
BITWISE_XOR : 0
BITWISE_NOT : 0
SHL : 0
SHR : 0
AND : 0
OR : 0
NOT : 0
LESS : 1


Microsoft Visual Studio 调试控制台
LESS_EQUAL : 0
EQUAL : 0
INEUQUAL : 0
GREATER : 1
GREATER_EQUAL, : 0
QUESTION : 0
COMMA : 1
COLON : 0
SEMICOLON : 4
DOT : 1
ARROW : 0
L_BRACE : 1
R_BRACE : 2
L_SQUARE : 0
R_SQUARE : 0
L_PAREN : 2
R_PAREN : 1
WELL : 0
ID : 8
NUM : 3
ACHAR : 0
STRING_BLOCK : 0
PREACTION : 1
TAB : 0
ERROR_EXP : 0
EEEOR_NUM : 0
ERROE_CHAR : 0
ERROR_STRING : 1
EEROR_PREACTION : 0

6 实验总结

本次实验让我们编写语法分析的过程，一开始很难将书本上的知识与自己动手写代码联合起来，对于怎么安排不同模块也没有什么思路，但是经过大量的研读课本和 ppt 之后，对语法分析越来越了解，对代码如何写也渐渐分析。

碍于本人能力所限，在很多细节的地方并没有处理好，比如对于非法标识符的读取没有想到一个好的方法，在标识符 1a 中会将其默认为数字和标识符两部分，对于这部分则可能需要在之后的语法分析的步骤再进行解决。

总而言之，本次作业让自己收获了很多，对于语法分析的具体实现有了更多更深层次的了解和熟悉，与刚学习时已经有了很大的不同。并且还锻炼了自己的代码能力。