《词法分析程序的设计与实现》
实验报告

姓名：于孟孟

学号：2022211260

班级：2022211307

2024 年 10 月 17 日

目 录

1 需求分析

1.1 任务描述

设计并实现一个 C 语言词法分析程序。识别单词并以记号形式输出，并标出该单词符号所在行数；能够识别并跳过注释；能够检查到错误的语法；能够统计行数、各个单词符号的类别数，以及词法错误数。

使用 C/C++实现。

1.2 输入描述

从文件流等读入文件，给定待读取文件的文件名，利用代码中的字符串变量(它的值就是文件名)，完成输入数据的读取。

文件内容为一段 C 语言程序源代码，或者为每行一个单词符号的文本。

文件换行格式为 LF，即换行只需要考虑\n 一个字符。

1.3 输出描述

从文件流等读入文件，给定待读取文件的文件名，利用代码中的字符串变量(它的值就是文件名)，完成输入数据的读取。

文件内容为一段 C 语言程序源代码，或者为每行一个单词符号的文本。

文件换行格式为 LF，即换行只需要考虑\n 一个字符。

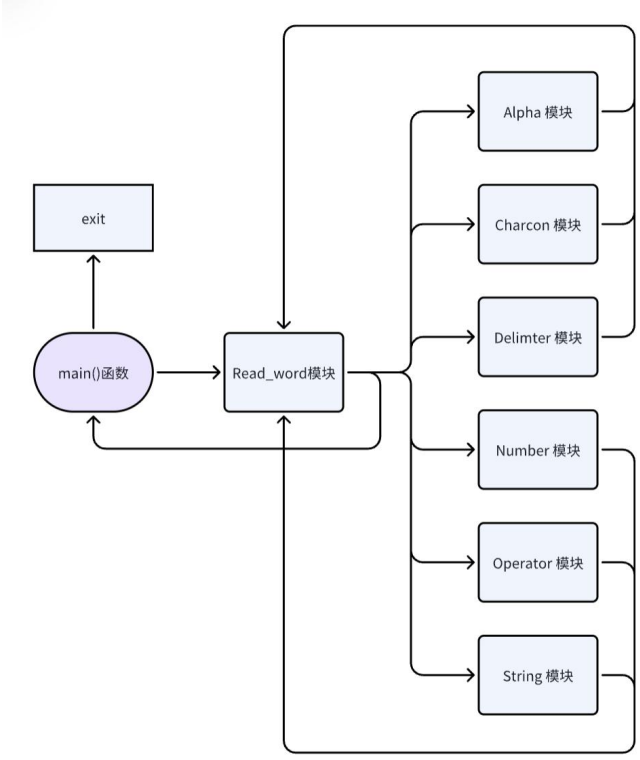
2 概要设计

注：提交代码包括两个版本。一种所有代码都集中在一个.cpp 文件中，用于提交头歌平台；另一种包括多个.cpp 与.h 文件，能更清晰的看到模块划分与不同模块功能的实现。报告为了更清晰的展示，依照多文件代码来写，但报告中所写功能在单文件代码中都有相同实现。

2.1 模块划分

分为 Read_word, Alpha, Charcon, Delimiter, Number, Operator 和 String 模块，这些模块各自定义了 class，在多文件版本的代码中各自拥有一个.h 和一个.cpp 文件。

各模块之间的关系如图所示

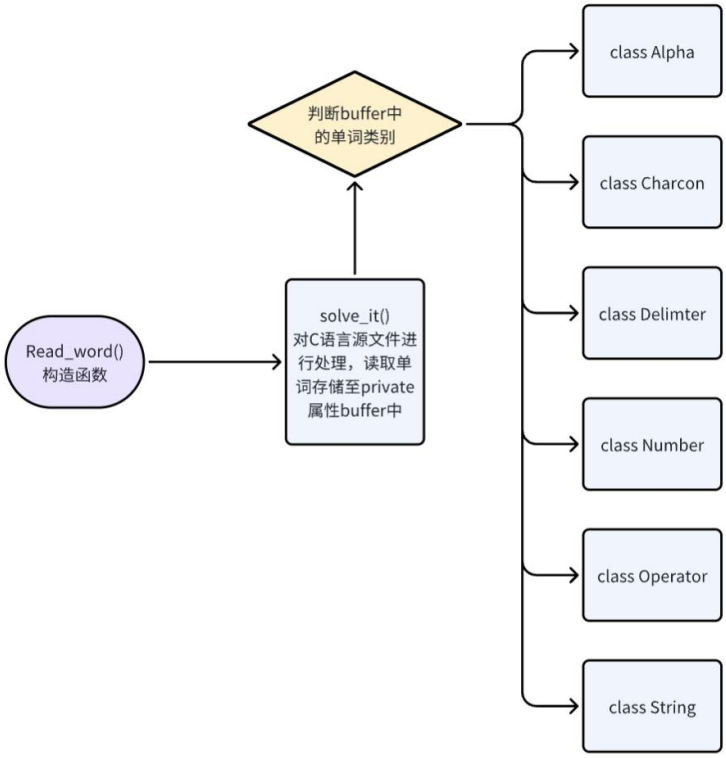


2.2 各模块说明

2.2.1 Read_word 模块(term.cpp 与 term.h 文件)

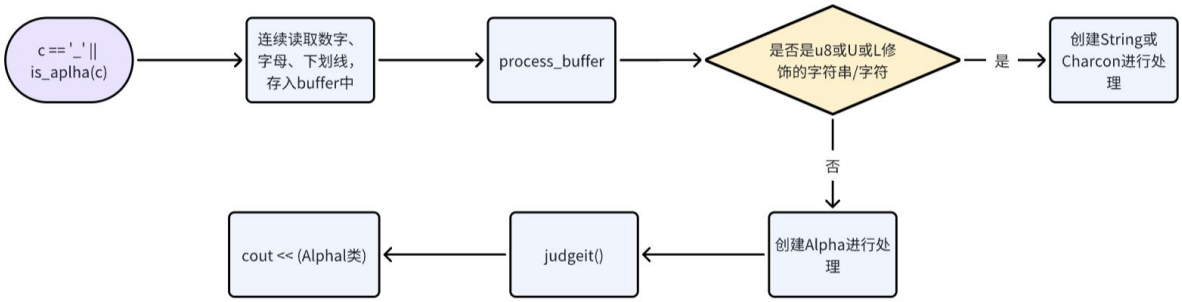
在此模块完成打开测试文件、从文件中选取词并调用其他模块处理词的功能。

模块流程图如下：



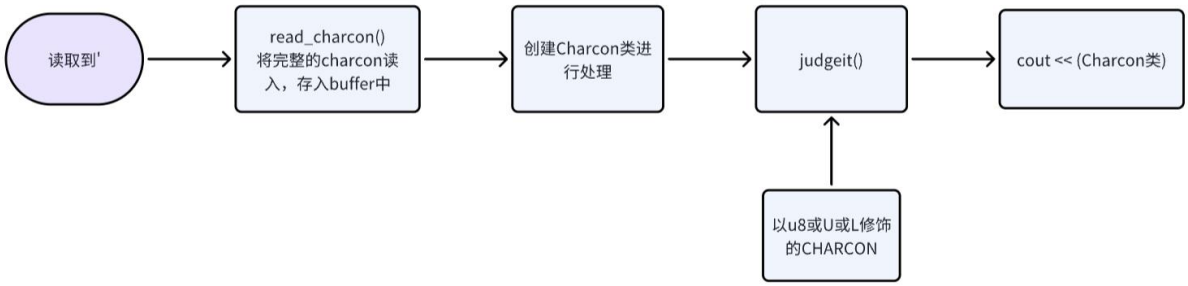
2.2.2 Alpha 模块(Alpha.cpp 和 Alpha.h)

处理以字母或下划线开头的 word 的流程图



2.2.3Charcon 模块(Charcon.h 和 Charcon.cpp)

处理 Charcon 的流程图



2.2.4 Delimiter(Delimiter.cpp 和 Delimiter.h)

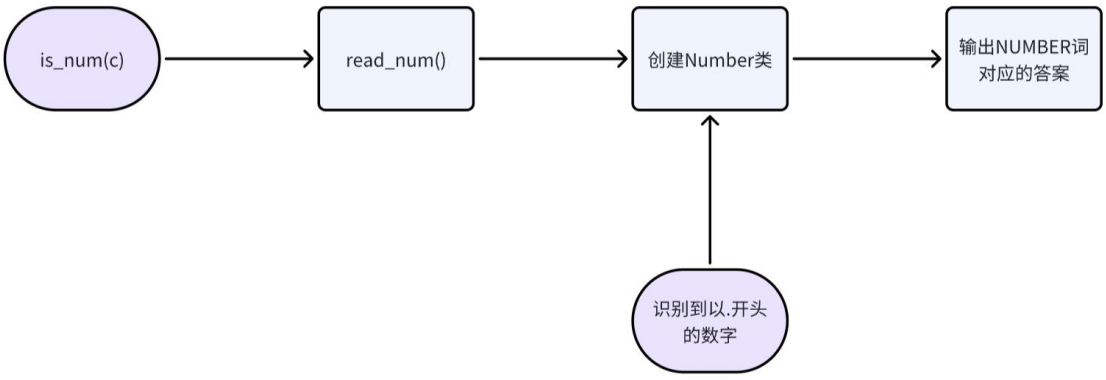
(1) 识别到 delimiter

(2) 创建 Delimiter 类

(3) 输出对 Delimiter 识别的答案

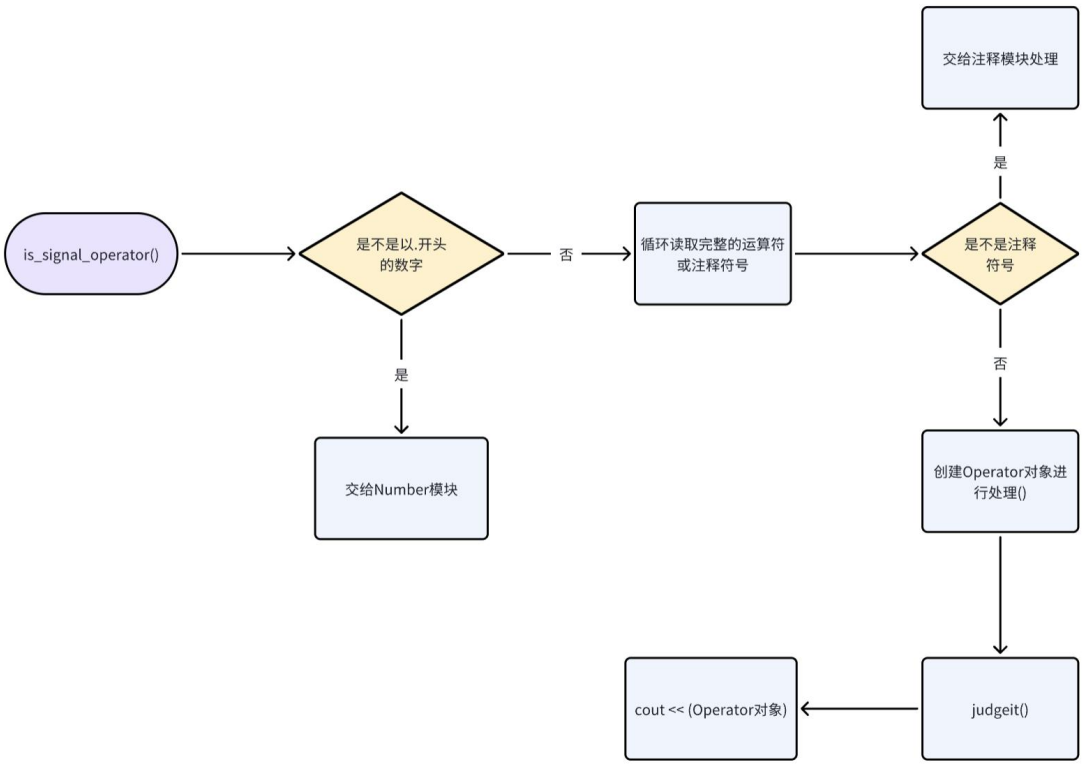
2.2.5 Number(Number.cpp 和 Number.h)

处理 Number 的流程图如下



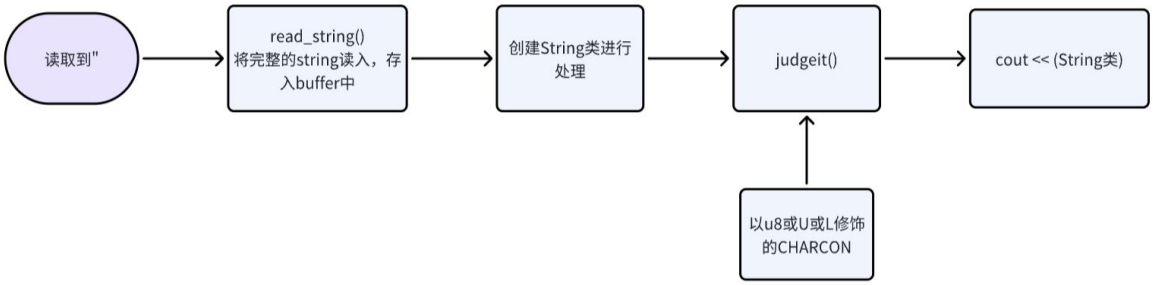
2.2.6 Operator(Operator.cpp 和 Operator.h)

处理 Operator 的流程图



2.2.7 String(String.cpp 和 String.h)

处理 String 的流程图



2.2.8 注释

(1) 当读取到//或/*时，直接调用 skipit 函数，跳过注释

(2) skipit()

① 对于//格式的注释，读取所有字符，直到遇到\n 或文件结束

② 对于/*格式的注释

1) 特别识别/**/的情况

2) 读取所有字符，直到遇到*/为止

3) 注意过程中遇到的换行符

③ buffer.clear()

2.3 数据结构

(1) Read_word 类

① 类的定义如图 1

② 私有成员

1) buffer 用来暂存读取到的单词，等待处理(vector)

2) file 用来读取文件

3) c 用来存储当前读到的字符

(2) Alpha 类

① 类的定义如图 2

② 私有成员

1) read_alpha 用于将 buffer 中暂存的单词转化成 string 并存储

2) is_keyword 和 is_delimter 用来存储判断结果

3) line 记录当前行

(3) Charcon 类

① 类的定义如图 3

② 私有成员

1) read_charcon 将 buffer 中暂存的单词转化为 string 并存储

2) is_charcon 用来存储判断结果

3) line 用来记录当前行

(4) Operator 类

① 类的定义如图 4

② 私有成员

1) operator_word 将 buffer 中暂存的单词转化为 string 并存储

2) is_operator 用来存储判断结果

3) line 用来记录当前行

(5) Number 类

① 类的定义如图 5

② 私有成员

1) number_word 将 buffer 中暂存的单词转化为 string 并存储

2) is_number 用来存储判断结果

3) line 用来记录当前行

(6) Delimter 类

① 类的定义如图 6

② 私有成员

1) c 用来存储当前读到的 Delimiter

2) line 用来记录当前行

(7) String 类

① 类的定义如图 7

② 私有成员

1) read_string 用来将 buffer(vector)中存储的单词转化为 string 并存储

2) is_string 用来记录对是否是 string 判断结果

3) line 用来存储当前行

(8) 全局变量

用来统计不同种类单词的个数

① int line_num = 1;

② int KEYWORD_num = 0;

③ int OPERATOR_num = 0;

④ int DELIMITER_num = 0;

⑤ int CHARCON_num = 0;

⑥ int STRING_num = 0;

⑦ int NUMBER_num = 0;

⑧ int all_num = 0;

⑨ int ERROR_num = 0;

```
class Read_word
{
public:
    函数接口

private:
    vector<char> buffer;
    ifstream file;
    char c;
};
```

```
class Alpha
{
public:
    函数接口

private:
    string read_alpha; // 存储读到的单词
    bool is_keyword;
    bool is_identifier;
    int line;
};
```

```
class Charcon
{
public:
    函数接口

private:
    string read_charcon;
    bool is_charcon;
    int line;
};
```

```
class Operator
{
public:
    函数入口

private:
    string operator_word;
    int line;
    bool is_operator;
};
```

```
class Number
{
public:
    函数接口

private:
    string number_word;
    bool is_number;
    int line;
};
```

```
class Delimter
{
public:
    函数接口

private:
    int line;
    char c;
};
```

```
class String

{

public:

    String(vector<char> &buffer);

    void judgeit();

    friend ostream &operator<<(ostream &Out, String &a_string);


private:

    string read_string; //  存储识别到的单词

    bool is_string;

    int line;

};
```

3 详细设计

```
int main(int argc, char *argv[])

{

    string filename = argv[1]; //  本程序识别的 C 语言程序的文件名

    Read_word buffer(filename);

    buffer.solve_it();

    return 0;

}
```

(1) 程序入口

① main()函数

② 创建 Read_word 类，并调用其成员函数 solve.it()解决对 C 语言源程序词法分析的问题

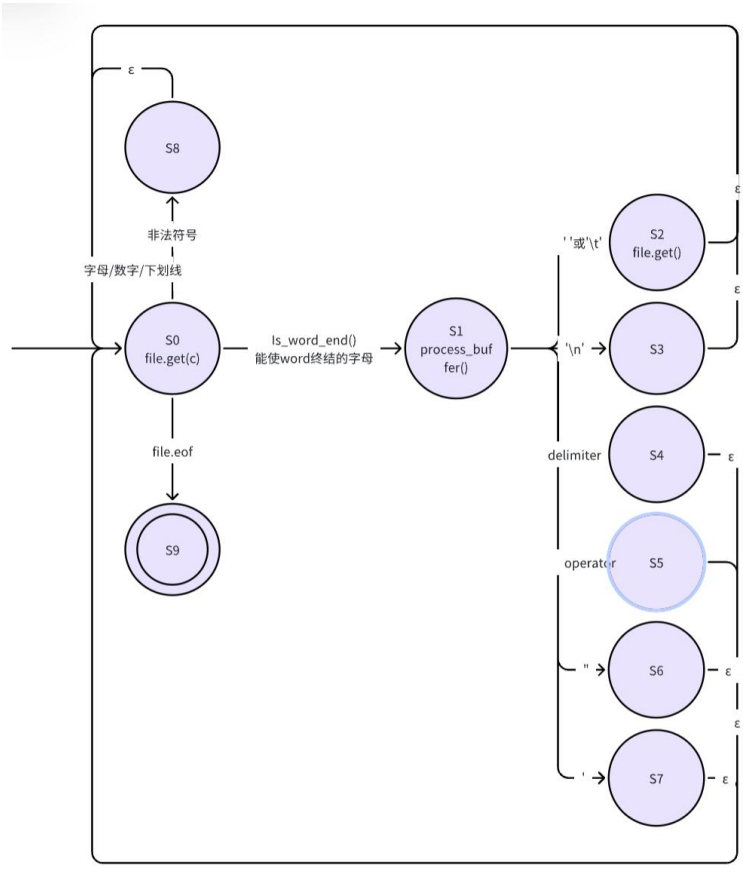
(2) Read_word

① 从文件中取词

1) 创建 Read_word 对象，读取单个字符，存入 buffer 中，组成字符串

② 判断 buffer 中的单词类别

状态转换图如下



③ 对不同类别的字符进行处理

1) ‘ ’ 和 ‘\t’ 不做任何处理，略过

2) ‘\n’使总行数加 1

3) delimiter

a. 处理 buffer 中的单词

b. 创建 Delimiter 类，识别 DELIMITER 并打印提示信息

4) operator

a. 处理特殊情况

a) 以’.’开头’的数字

b) 以 ”//” 或 ”/*开头的注释

c) 不是特殊情况，创建 Operator 类，识别 OPERATOR 并打印提示信息

5) ‘

a. 创建 CHARCON 类，识别 CHARCON 并打印提示信息

6) ‘_’或字母

a. 存储至 buffer 并继续

b. 一直存储，直至读到能分隔两个单词的字符时进行处理，创建 Alpha 类并判断其合法性

7) 数字

a. 完整读取一个数字，存到 buffer 中

b. 创建 Number 类判断其合法性并打印提示信息

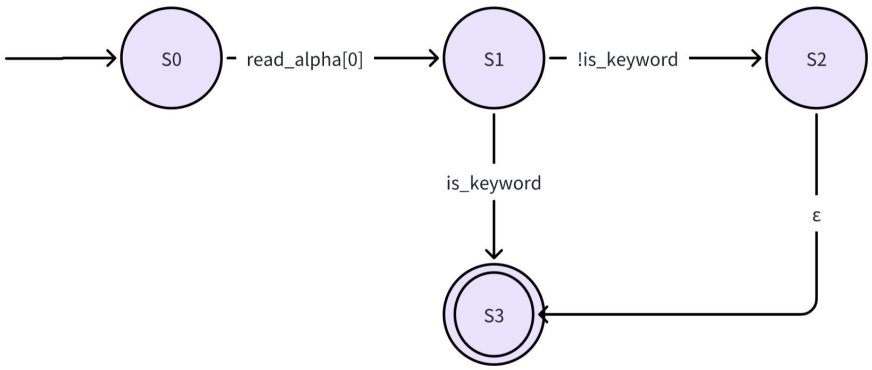
(3) Alpha

① a.judgeit()

1) 先判断是不是 KEYWORD，如果是，则 is_keyword 置 1

2) 如果不是 KEYWORD，再判断是不是 IDENTIFIER，如果是，则 is_identifier 置 1

3) 状态转换图如下



② a.operator<<

1) 根据 is_keyword 和 is_identifier 的情况输出答案

(4) Charcon

① read_charcon()

1) 一直读取字符，直到遇到’

2) 完整的读取整个转义字符

3) 若在’之前遇到回车，则出错

② judgeit()

1) 只需判断最后一个字符是不是’即可

2) 如果是 CHARCON，则 is_charcon 置 1

③ operator<<

1) 根据 is_charcon 的值，输出对这个词识别的答案

(5) Delimter

① 识别到 delimiter

② 创建 Delimiter 类

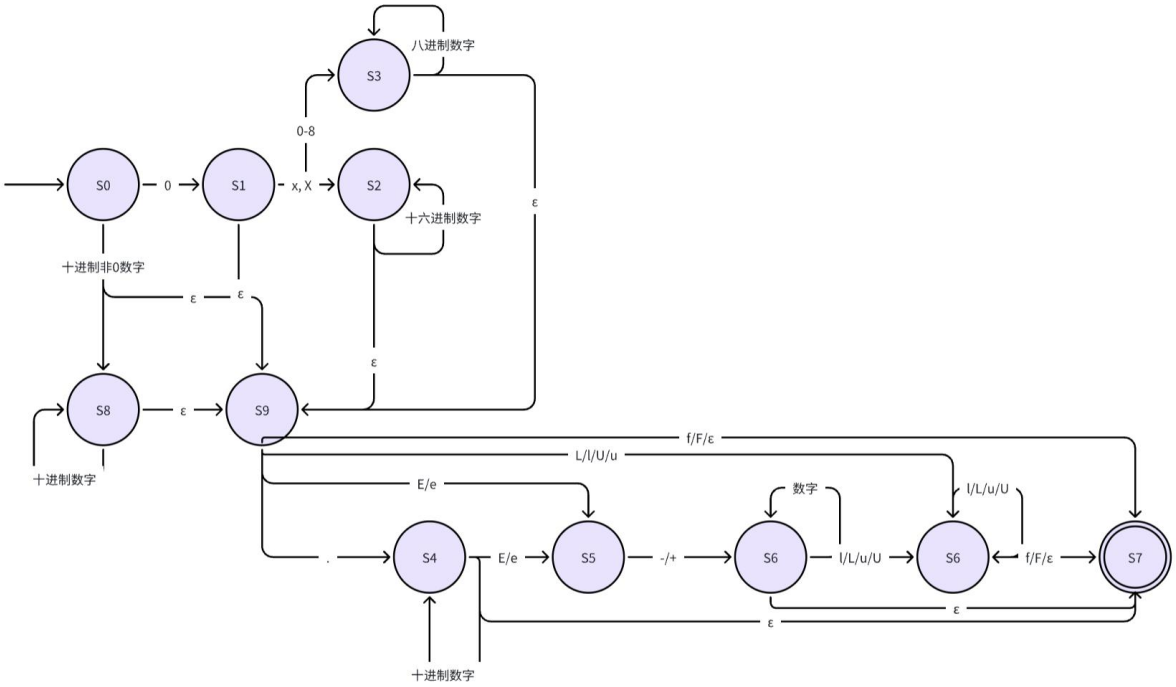
③ 输出对 Delimiter 识别的答案

(6) Number

① read_num()

1) 识别 NUMBER，存入 buffer 中

2) 状态转换图如下



3) operator<<

a. 根据创建 NUMBER 时传入的参数 is_number，判断数字是否有效，赋值给 is_number

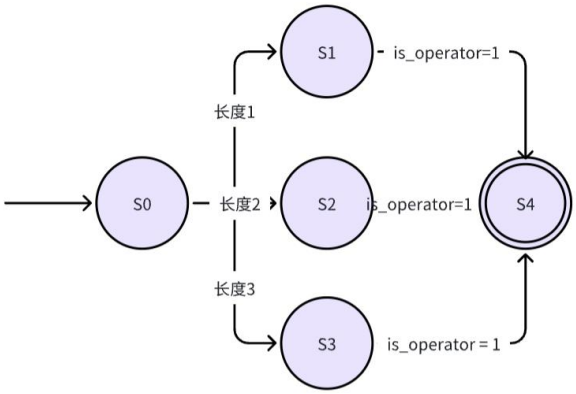
b. 根据 is_number，打印这个词的答案

(7) Operator

① judgeit()

1) 判断是不是合法 operator，并给 is_operator 赋值

2) 状态转换图如下



3) operator<<

a. 根据 is_operator 的值，打印识别 OPERATOR 词的答案

(8) String

① read_string()

1) 一直读取字符，直到遇到”

2) 完整的读取整个转义字符

3) 若在”之前遇到回车，则出错

② judgeit()

1) 只需判断最后一个字符是不是”即可

2) 如果是 String，则 is_string 置 1

③ operator<<

1) 根据 is_string 的值，输出对这个词识别的答案

(9) 注释

① 当读取到//或/*时，直接调用 skipit 函数，跳过注释

② skipit()

1) 对于//格式的注释，读取所有字符，直到遇到\n 或文件结束

2) 对于/*格式的注释

a. 特别识别/**/的情况

b. 读取所有字符，直到遇到*/为止

3) 注意过程中遇到的换行符

a. buffer.clear()

4 程序测试

(1) 头歌平台提交结果

① 25/25 全部通过

② 提交的是单文件版本的代码

(2) 测试样例

① test18.txt

1) 测试文件输入

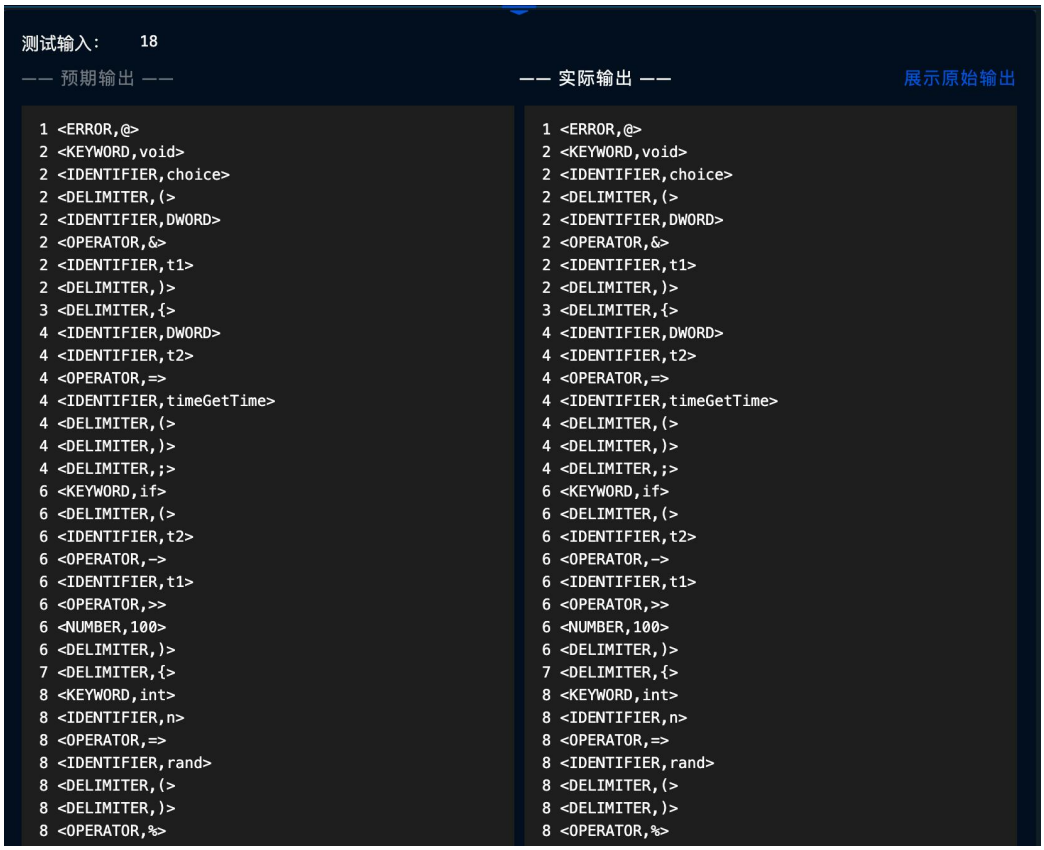
```
@
void choice(DWORD& t1)
{
    DWORD t2 = timeGetTime();

    if (t2 - t1 > 100)   //100ms产生一个烟花弹
    {
        int n = rand() % 20;     //0-19
        if (n < NUM && jet[n].isshoot == false && fire[n].show == false)
        {
            //重置烟花弹
            jet[n].x = rand() % (WND_WIDTH - 20);      You, 2周前 • update
            jet[n].y = rand() % 100 + 400;   //450-549
            jet[n].hx = jet[n].x;
            jet[n].hy = rand() % 400;        //0-399
            jet[n].height = jet[n].y - jet[n].hy;
            jet[n].isshoot = true;

            //n
            putimage(jet[n].x, jet[n].y, &jet[n].img[jet[n].n])@;
        }
        t1 = t2;
    }
}
```

2) 输出

测试输入： 18

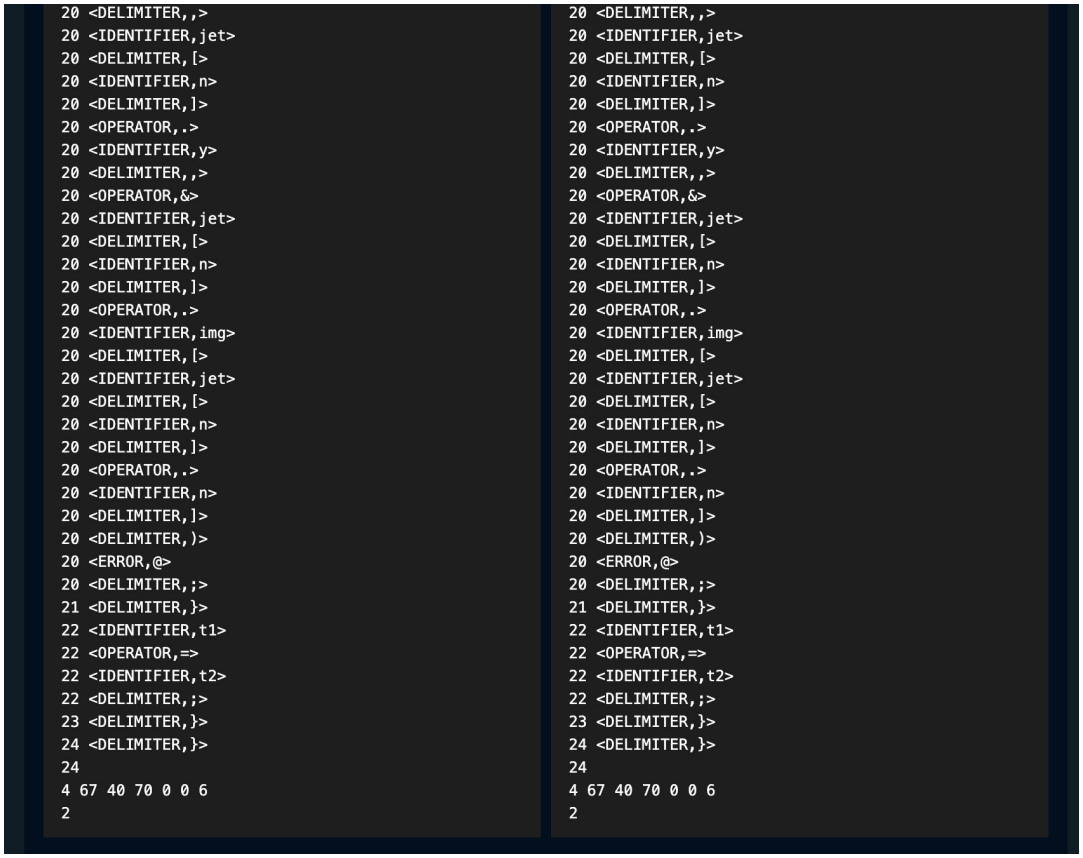


```
—— 预期输出 ——
1 <ERROR,@>
2 <KEYWORD,void>
2 <IDENTIFIER,choice>
2 <DELIMITER,(>
2 <IDENTIFIER,DWORD>
2 <OPERATOR,&>
2 <IDENTIFIER,t1>
2 <DELIMITER,)>
3 <DELIMITER,{>
4 <IDENTIFIER,DWORD>
4 <IDENTIFIER,t2>
4 <OPERATOR,=>
4 <IDENTIFIER,timeGetTime>
4 <DELIMITER,(>
4 <DELIMITER,)>
4 <DELIMITER,;>
6 <KEYWORD,if>
6 <DELIMITER,(>
6 <IDENTIFIER,t2>
6 <OPERATOR,->
6 <IDENTIFIER,t1>
6 <OPERATOR,>>
6 <NUMBER,100>
6 <DELIMITER,)>
7 <DELIMITER,{>
8 <KEYWORD,int>
8 <IDENTIFIER,n>
8 <OPERATOR,=>
8 <IDENTIFIER,rand>
8 <DELIMITER,(>
8 <DELIMITER,)>
8 <OPERATOR,%>
```

```
—— 实际输出 ——
1 <ERROR,@>
2 <KEYWORD,void>
2 <IDENTIFIER,choice>
2 <DELIMITER,(>
2 <IDENTIFIER,DWORD>
2 <OPERATOR,&>
2 <IDENTIFIER,t1>
2 <DELIMITER,)>
3 <DELIMITER,{>
4 <IDENTIFIER,DWORD>
4 <IDENTIFIER,t2>
4 <OPERATOR,=>
4 <IDENTIFIER,timeGetTime>
4 <DELIMITER,(>
4 <DELIMITER,)>
4 <DELIMITER,;>
6 <KEYWORD,if>
6 <DELIMITER,(>
6 <IDENTIFIER,t2>
6 <OPERATOR,->
6 <IDENTIFIER,t1>
6 <OPERATOR,>>
6 <NUMBER,100>
6 <DELIMITER,)>
7 <DELIMITER,{>
8 <KEYWORD,int>
8 <IDENTIFIER,n>
8 <OPERATOR,=>
8 <IDENTIFIER,rand>
8 <DELIMITER,(>
8 <DELIMITER,)>
8 <OPERATOR,%>
```

展示原始输出

……



```
20 <DELIMITER,,>
20 <IDENTIFIER,jet>
20 <DELIMITER,[>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <OPERATOR,.>
20 <IDENTIFIER,y>
20 <DELIMITER,,>
20 <OPERATOR,&>
20 <IDENTIFIER,jet>
20 <DELIMITER,[>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <OPERATOR,.>
20 <IDENTIFIER,img>
20 <DELIMITER,[>
20 <IDENTIFIER,jet>
20 <DELIMITER,[>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <OPERATOR,.>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <DELIMITER,)>
20 <ERROR,@>
20 <DELIMITER,;>
21 <DELIMITER,}>
22 <IDENTIFIER,t1>
22 <OPERATOR,=>
22 <IDENTIFIER,t2>
22 <DELIMITER,;>
23 <DELIMITER,}>
24 <DELIMITER,}>
24
4 67 40 70 0 0 6
2
```

```
20 <DELIMITER,,>
20 <IDENTIFIER,jet>
20 <DELIMITER,[>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <OPERATOR,.>
20 <IDENTIFIER,y>
20 <DELIMITER,,>
20 <OPERATOR,&>
20 <IDENTIFIER,jet>
20 <DELIMITER,[>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <OPERATOR,.>
20 <IDENTIFIER,img>
20 <DELIMITER,[>
20 <IDENTIFIER,jet>
20 <DELIMITER,[>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <OPERATOR,.>
20 <IDENTIFIER,n>
20 <DELIMITER,]>
20 <DELIMITER,)>
20 <ERROR,@>
20 <DELIMITER,;>
21 <DELIMITER,}>
22 <IDENTIFIER,t1>
22 <OPERATOR,=>
22 <IDENTIFIER,t2>
22 <DELIMITER,;>
23 <DELIMITER,}>
24 <DELIMITER,}>
24
4 67 40 70 0 0 6
2
```

② test20.txt

1) 测试文件输入

```
You, 2周前 | 1 author (You)
int main() {
    int 123a = 21;
    int b = 10;
    int c;
    char 1e='y';
    char msg[] = "hello!";
    double 00d;
    c = a + b;
    printf("a+b的值是%d", c);
    return 0;
}//#
```

2) 输出

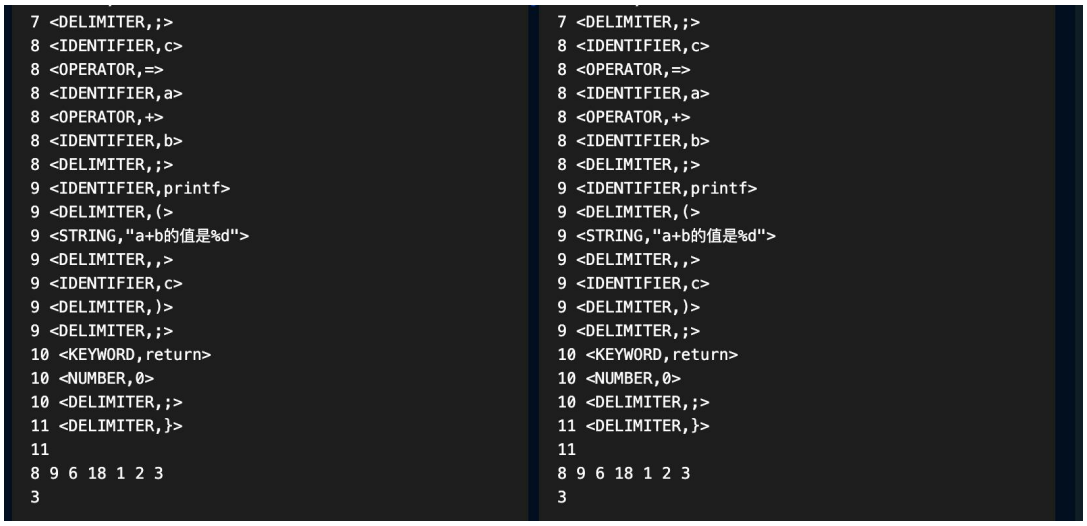
测试输入： 20

—— 预期输出 ——

```
1 <KEYWORD,int>
1 <IDENTIFIER,main>
1 <DELIMITER,(>
1 <DELIMITER,)>
1 <DELIMITER,{>
2 <KEYWORD,int>
2 <ERROR,123a>
2 <OPERATOR,=>
2 <NUMBER,21>
2 <DELIMITER,;>
3 <KEYWORD,int>
3 <IDENTIFIER,b>
3 <OPERATOR,=>
3 <NUMBER,10>
3 <DELIMITER,;>
4 <KEYWORD,int>
4 <IDENTIFIER,c>
4 <DELIMITER,;>
5 <KEYWORD,char>
5 <ERROR,1e>
5 <OPERATOR,=>
5 <CHARCON,'y'>
5 <DELIMITER,;>
6 <KEYWORD,char>
6 <IDENTIFIER,msg>
6 <DELIMITER,[>
6 <DELIMITER,]>
6 <OPERATOR,=>
6 <STRING,"hello!">
6 <DELIMITER,;>
7 <KEYWORD,double>
7 <ERROR,00d>
7 <DELIMITER,;>
```

—— 实际输出 ——　展示原始输出

```
1 <KEYWORD,int>
1 <IDENTIFIER,main>
1 <DELIMITER,(>
1 <DELIMITER,)>
1 <DELIMITER,{>
2 <KEYWORD,int>
2 <ERROR,123a>
2 <OPERATOR,=>
2 <NUMBER,21>
2 <DELIMITER,;>
3 <KEYWORD,int>
3 <IDENTIFIER,b>
3 <OPERATOR,=>
3 <NUMBER,10>
3 <DELIMITER,;>
4 <KEYWORD,int>
4 <IDENTIFIER,c>
4 <DELIMITER,;>
5 <KEYWORD,char>
5 <ERROR,1e>
5 <OPERATOR,=>
5 <CHARCON,'y'>
5 <DELIMITER,;>
6 <KEYWORD,char>
6 <IDENTIFIER,msg>
6 <DELIMITER,[>
6 <DELIMITER,]>
6 <OPERATOR,=>
6 <STRING,"hello!">
6 <DELIMITER,;>
7 <KEYWORD,double>
7 <ERROR,00d>
7 <DELIMITER,;>
```



5 总结

在本次实验中，我用 C++语言设计并实现了 C 语言的词法分析程序。

在程序的设计过程中，我复习了词法分析的相关知识，根据文法绘制了不同类别的词的识别的状态转移图，便于程序实现。

在编程实现的过程中，我首先完成的是综合在一个文件里的代码。针对不同的测试样例逐步实现并测试词法分析程序。这种分步、分模块的方式简化了任务实现的难度，便于查错。

在最后的测试阶段，我使用 20 以后的测试样例进行综合测试，发现了许多没有注意过的边界条件，重新修改程序，最终测试样例全部通过。

在测试样例全部通过后，我又复习了 C++面向对象编程，为每个模块设计了一个 C++ class，使结构更清晰，便于撰写实验报告。

此次实验非常有效的锻炼了我的编程能力与对编译原理知识的理解。