《编译原理与技术》课程实验报告

词法分析程序设计与实现

黎昱彤

学号：2022211414

班级：2022211312

计算机科学与技术

目录

1 背景 1

1.1 实验介绍 1

1.2 实验任务 1

2 程序设计说明 1

2.1 实验原理 1

2.2 程序设计 1

3 测试报告 2

3.1 基本要求 2

3.2 扩展要求 5

3.3 测试用例通过情况 7

4 实验总结 7

1 背景

1.1 实验介绍

《编译原理与技术（第 2 版）》第 3 章 程序设计 1

1.2 实验任务

设计并实现一个 C 语言的词法分析程序。 要求：

- 识别单词符号并以记号形式输出，并标出该单词符号所在行数；
- 能够识别并跳过注释；
- 能够检查到错误的词法；
- 能够统计行数、各个单词符号的类别数，以及词法错误数。

2 程序设计说明

2.1 实验原理

词法分析阶段是编译过程的第一步，其主要任务是从左至右逐个字符地对源程序进行扫描，按照源语言的词法规则识别出单词符号并存入符号表中，产生用于语法分析的记号序列。

2.2 程序设计

按照书中步骤（1）给出词法规则；（2）构造状态转换图；（3）根据状态转换图构造词法分析程序。我在设计程序时，用 while 循环读入文件的字符，基于图 3-10 词法分析程序，使用 switch 分支结构和一个 state 全局变量实现了自动机，直到最终完成状态转换图为图 1（注：后续补充的一些状态没有完全遵从转换规则，几乎只是借用 state 分块，注释部分在状态机之外）

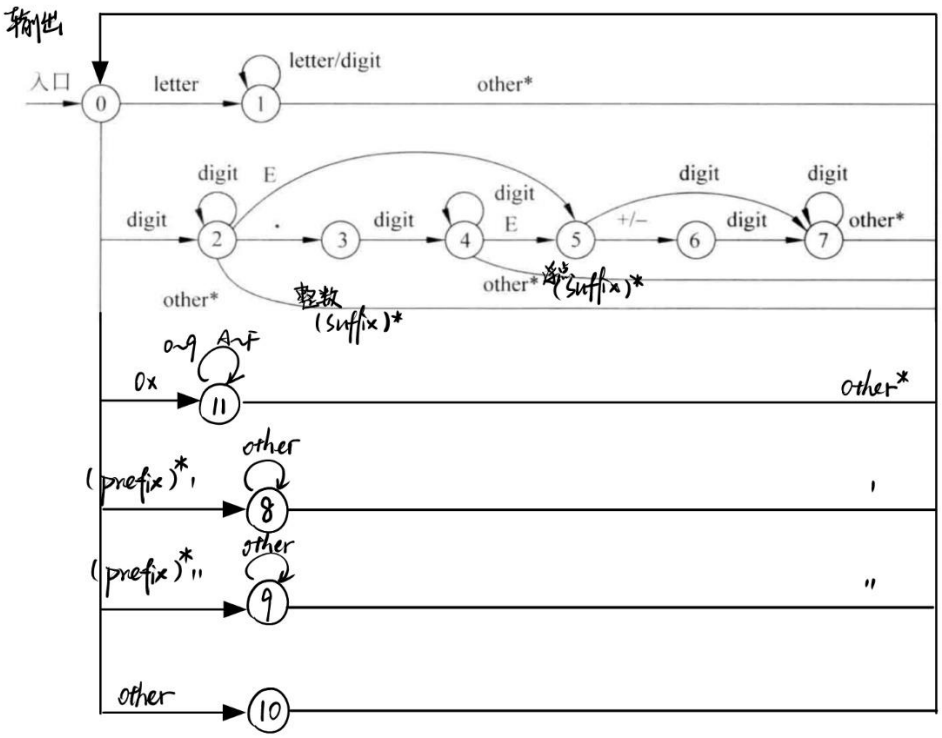

图 1 状态转换图

3 测试报告

3.1 基本要求

以下为程序测试过程中具有代表性的部分：

1. EOF

这确实是一开始就发现的bug但是尝试了很多次才发现真正问题在哪里。毕竟test1就过不了，还以为是volatile出什么问题了，暂时放下看后面的错才注意到是遇到EOF了，挣扎了一会读文件，最终还是以循环外统一处理解决了，但是考虑每种结束状态的输出有些繁琐。

```
if(!word.empty()){// 到 EOF 未结束的
      if(isKeyword(word)){
```

```
        cout << line_cnt << " <KEYWORD," << word << ">" << endl;
        KW_cnt++;
      }
      else if(state==2 || state==4 || state==7 || state==11 || state==12){
        cout << line_cnt << " <NUMBER," << word << ">" << endl;
        NUM_cnt++;
      }
      else{
        cout << line_cnt << " <IDENTIFIER," << word << ">" << endl;
        ID_cnt++;
      }
      word.clear();
    }
```

2. 行数不对

这一点也是尝试了几次，发现出问题的行数都变成了正确的2倍，因为一开始是简单地遇到‘\n’就行数计数器自增。处理办法：只在state0自增，注释和未闭合引号单独处理。

3. 运算符

state10是印象最深刻的，因为我想节省状态，OPERATOR又有很多重叠的前缀，所以采用了一种先吞再吐的方法。

```
case 10:// 运算符
            word += c;
            if(word.size()==3){
              string str0 = word.substr(0, 1);
              string str1 = word.substr(0, 2);
              if(isOperator(word)){// 3
                cout << line_cnt << " <OPERATOR," << word << ">" << endl;
                OP_cnt++;
                word.clear();
```

```
            state = 0;
          }
          else if(isOperator(str1)){// 2
            cout << line_cnt << " <OPERATOR," << str1 << ">" << endl;
            OP_cnt++;
            file.unget();
            word.clear();
            state = 0;
          }
          else if(isOperator(str0)){// 1
            cout << line_cnt << " <OPERATOR," << str0 << ">" << endl;
            OP_cnt++;
            file.unget();
            file.unget();
            word.clear();
            state = 0;
          }
        }
        else{
          if(isOperator(word)){
            if(word.size()==2 && file.peek()!=EOF)
              continue;
            cout << line_cnt << " <OPERATOR," << word << ">" << endl;
            OP_cnt++;
            word.clear();
            state = 0;
          }
        }
      break;
```

4. 1e非法/@在注释和引号中合法

   没看测试用例确实是没想到。

3.2 扩展要求

1. 八进制/十六进制

此外添加了对非法的检测。

图2 test21

2. 带后缀的十进制数

一开始想的很简单，但是最后通过22还是尝试了很多遍。我写的时候一直在纠结lll这种怎么办，最后还是回归了状态机解决了（虽然我因为目前状态太多了就是不想再加而又一次用了先吞再吐这种方法）

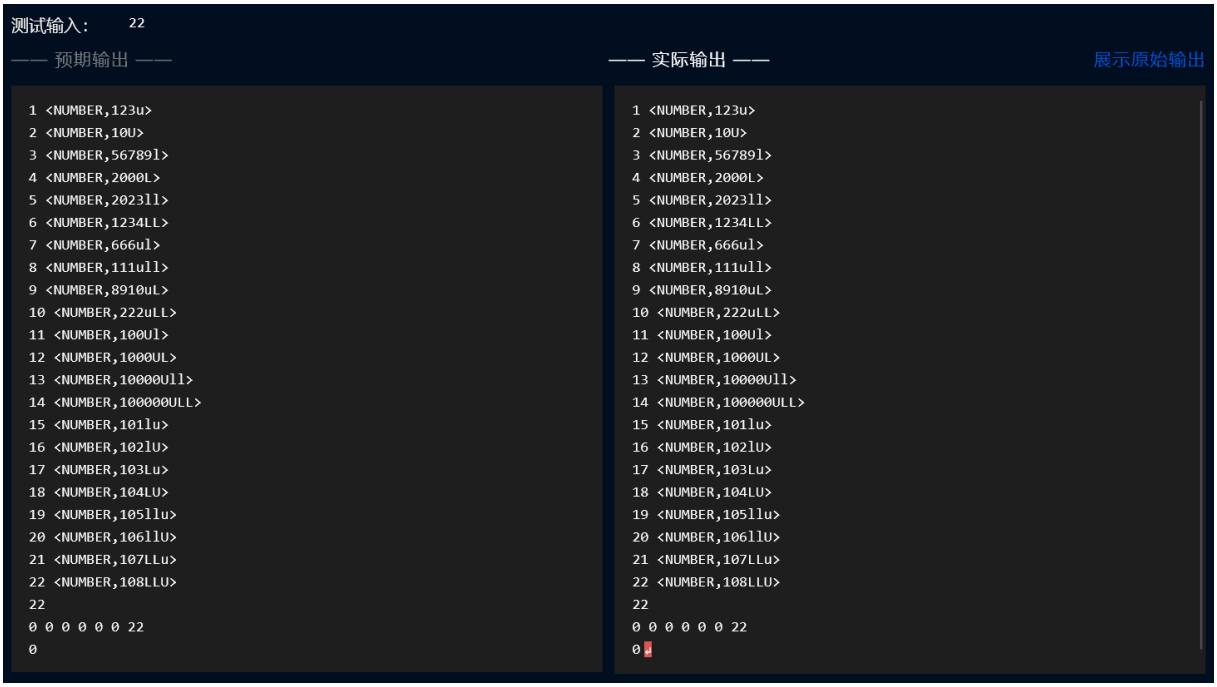

图3 test22

3. 完整的浮点型常量

2解决后3就不是什么难事了，在做基础要求时已经处理了.1为合法数值的情况。

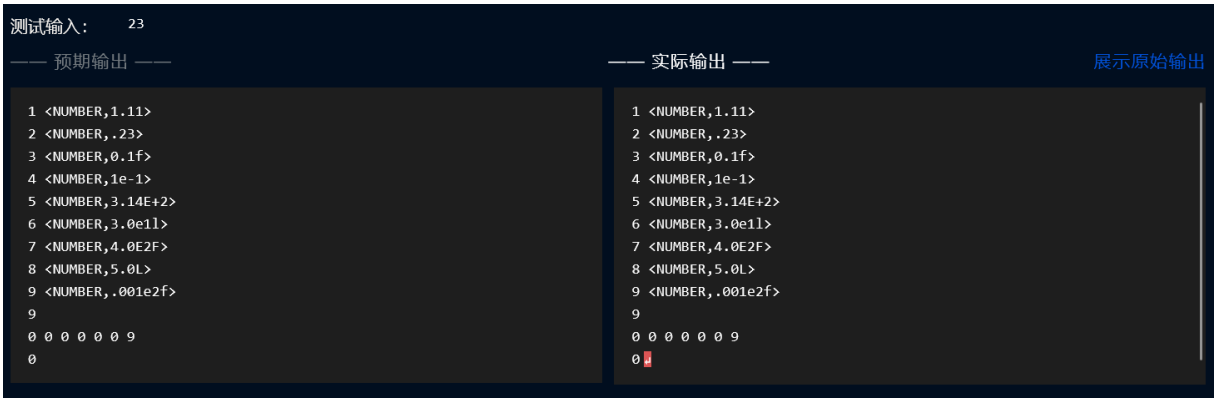

图4 test23

4. 带前缀的字符串常量

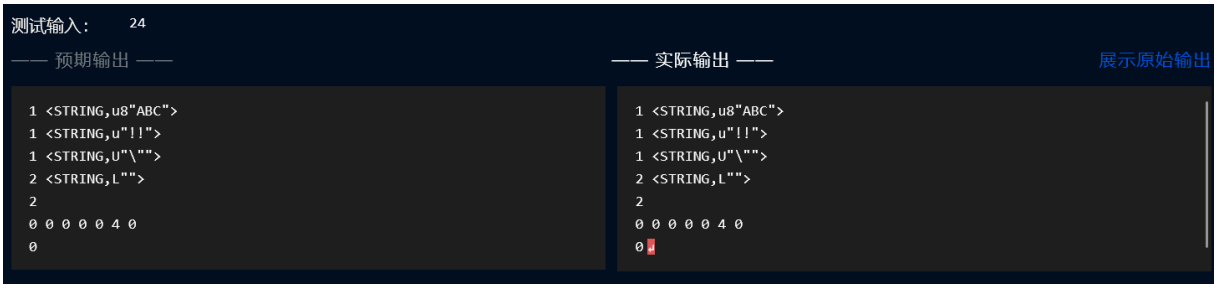

图5 test24

5. 完整的字符常量

4和5共用了前缀，是同时完成的。一开始想合并在state1（读入了一个letter）然后直接对比前缀，但是发现还是会把前缀和后面引号部分分开，最后还是在state0判断进行状态转移了。

图6 test25

3.3 测试用例通过情况

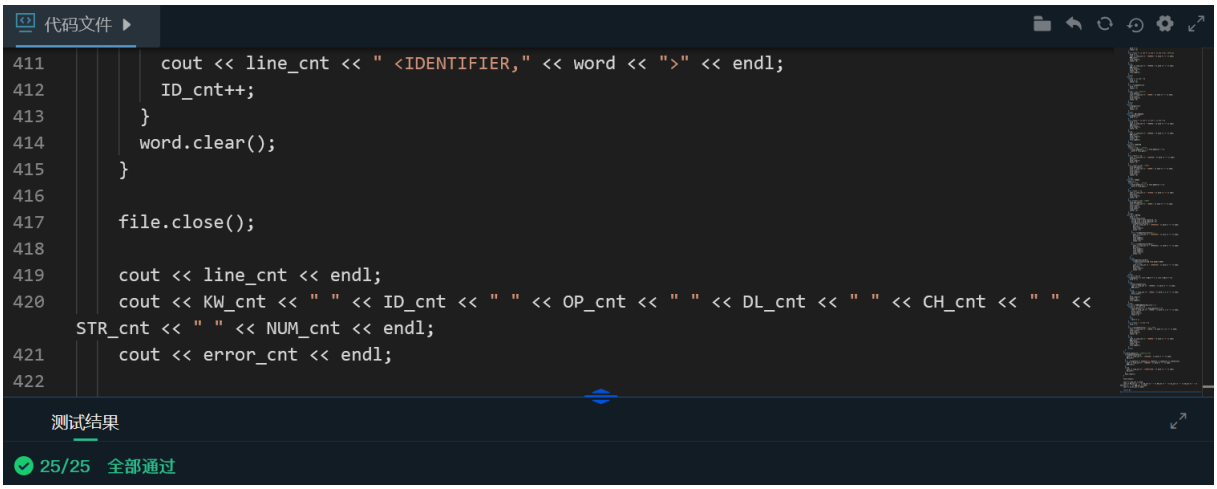

图 7 头歌提供的用例全部测评通过

4 实验总结

本次实验实现了词法分析程序，过程中复习了基础知识，调试过程中发现了合理的自己没考虑完全的漏洞，收获颇丰。一点反思：还有很大完善空间，比如没报ERROR的有不合法的后缀，最后处理EOF未结束的部分是后改的看起来有些丑陋。