实验三 词法分析实验报告

1 实验目的

1) 熟悉 C 语言的词法规则，了解编译器词法分析器的主要功能和实现技术，掌握典型词法分析器构造方法，设计并实现 C 语言词法分析器。
2) 了解 Flex 工作原理和基本思想，学习使用工具自动生成词法分析器。
3) 掌握编译器从前端到后端各个模块的工作原理，词法分析模块与其他模块之间的交互过程。

2 实验内容

根据 C 语言的词法规则，设计识别 C 语言所有单词类的词法分析器的确定有限状态自动机，并使用 C++，采用程序中心法设计并实现词法分析器。词法分析器的输入为 C 语言源程序，输出为属性字流。

3 实验步骤

以 C 语言作为源语言，构建 C 语言的词法分析器，对于给定的测试程序，输出属性字符流。词法分析器的构建按照 C 语言的词法规则进行。以 C11 为基准，对 C 语言的词法规则进行简要的描述。

3.1 状态机处理技巧

针对标识符、关键字、常量、符号定义了 40 中间态。针对 C 语言词法规则特点。实现中使用了以下技巧

（1） 对于关键字，先统一识别为标识符，再去查询该标识符是否属于关键字集合，将关键字的识别整合到标识符识别中。由于 C 语言的所有关键字，都满足标识符的生成规则，因此该操作合理。

（2） 对于整型常量，定义后缀识别的初始状态，分别识别十进制、八进制、十六进制的数值部分后，进入共享的后缀识别初始态。

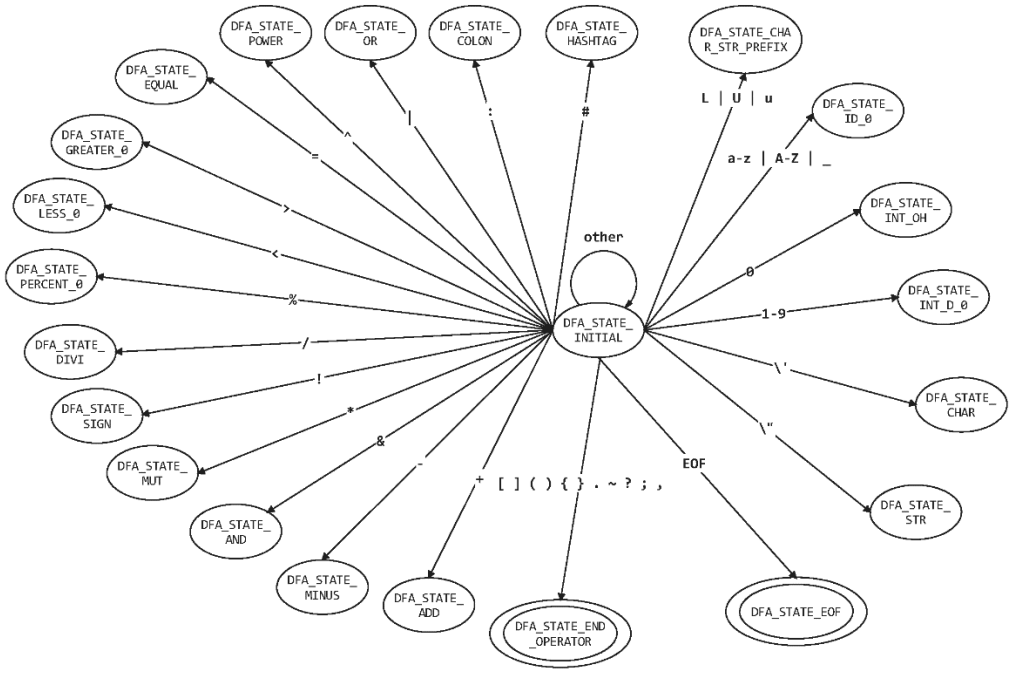
（3） 对于不作为其他运算符前缀的部分符号，包括 '[', ']', '(', ')', '{', '}', '.', '~', '?', ';', ','，可以直接查询识别，对于作为其他运算符前缀的运算符设置中间状态。据此，可把运算符分成以下几类

a) -, ->, --, -=
b) +, ++, +=
c) &, &&, &=
d) /, /=,
e) !, !=,
f) =, ==
g) ^, ^=
h) :, :>
i) %, %=, %:, %:%:, %>
j) <, <=, <<, <%, <<=, <:
k) >, >>, >=, >>=

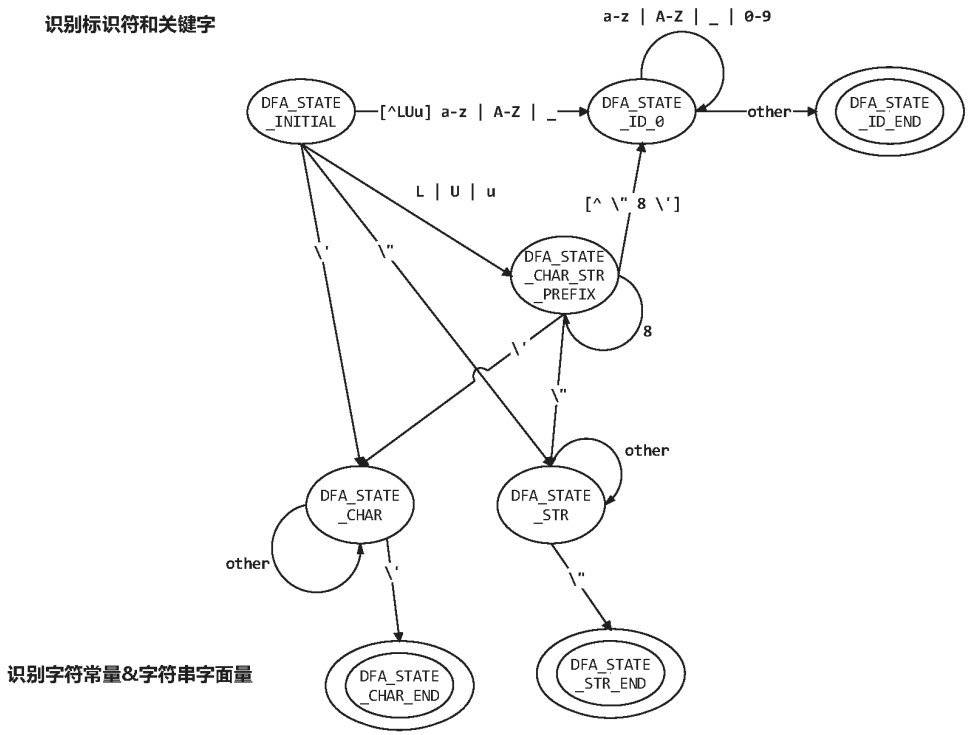
3.2 状态机 DFA

状态机 DFA 如下：

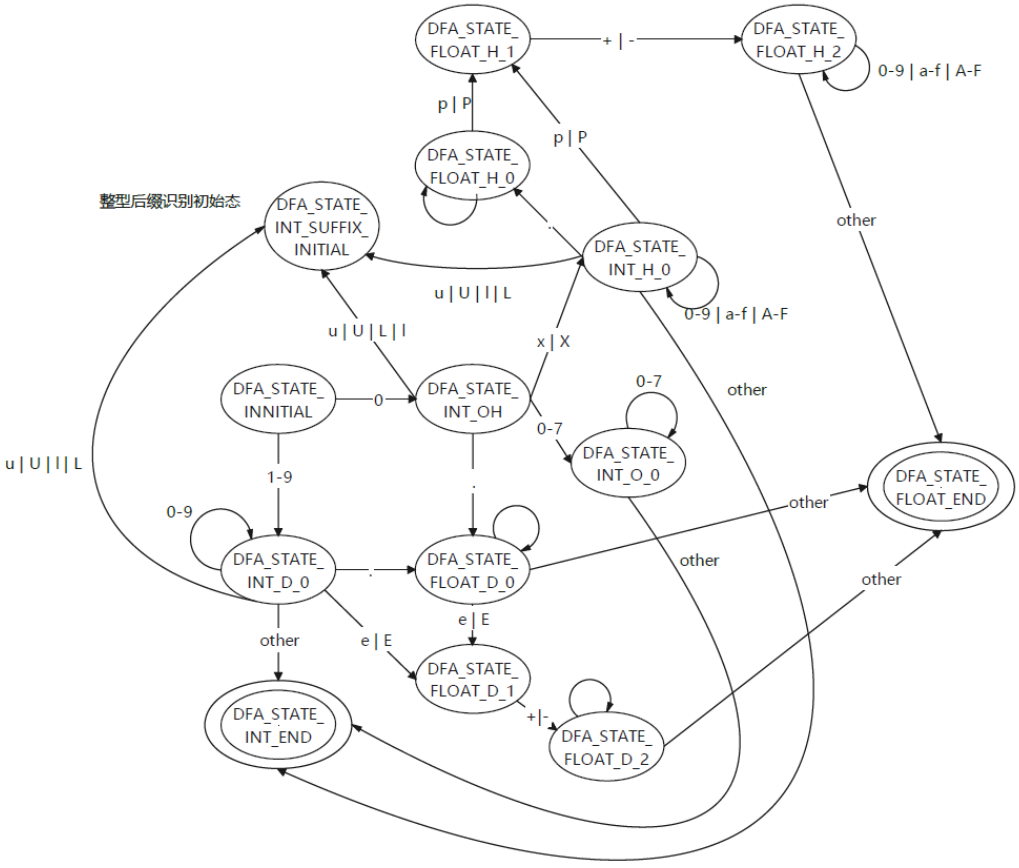
1. 初始状态的一步转化结果：



2. 标识符、关键字、字符常量、字符串字面量识别 DFA


识别标识符和关键字
a-z | A-Z | _ | 0-9
DFA_STATE_INITIAL
[^LUu] a-z | A-Z | _
DFA_STATE_ID_0
other
DFA_STATE_ID_END
L | U | u
[^ \" 8 \']
\'
\"
DFA_STATE_CHAR_STR_PREFIX
8
\'
\"
other
DFA_STATE_CHAR
DFA_STATE_STR
other
\'
\"
识别字符常量&字符串字面量
DFA_STATE_CHAR_END
DFA_STATE_STR_END

3. 整型常量 & 浮点型常量


DFA_STATE_FLOAT_H_1
+ | -
DFA_STATE_FLOAT_H_2
0-9 | a-f | A-F
p | P
p | P
DFA_STATE_FLOAT_H_0
整型后缀识别初始态
DFA_STATE_INT_SUFFIX_INITIAL
DFA_STATE_INT_H_0
other
u | U | l | L
0-9 | a-f | A-F
u | U | L | l
x | X
DFA_STATE_INNITIAL
0
DFA_STATE_INT_OH
0-7
other
0-7
DFA_STATE_INT_O_0
u | U | l | L
1-9
DFA_STATE_FLOAT_END
0-9
other
DFA_STATE_INT_D_0
.
DFA_STATE_FLOAT_D_0
other
other
e | E
other
e | E
DFA_STATE_FLOAT_D_1
DFA_STATE_INT_END
+|-
DFA_STATE_FLOAT_D_2

3.3 主要数据结构

1) vector<string> srcLines: 记录每行源程序
2) string strTokens ：记录输出的 Token 描述
3) bool keep: 标记是否保存当前读入的字符
4) bool end: 标记是否完成 scan
5) int row, col: 记录当前读入字符的行列位置
6) int pos: 记录当前读入字符在整个源程序中的位置
7) int iTokenNum: 记录当前 token 的编号
8) unordered_set<string> keywordSet: hash 集合存储关键字
9) unordered_set<char> nonPrefixOperator: hash 集合存储非前缀符号

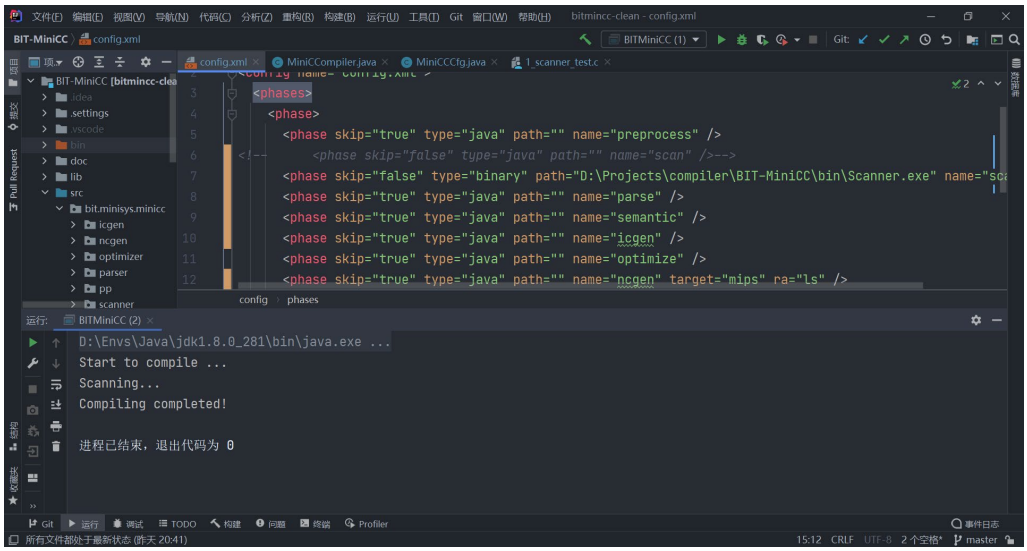
实现的 Scanner 类定义如下：

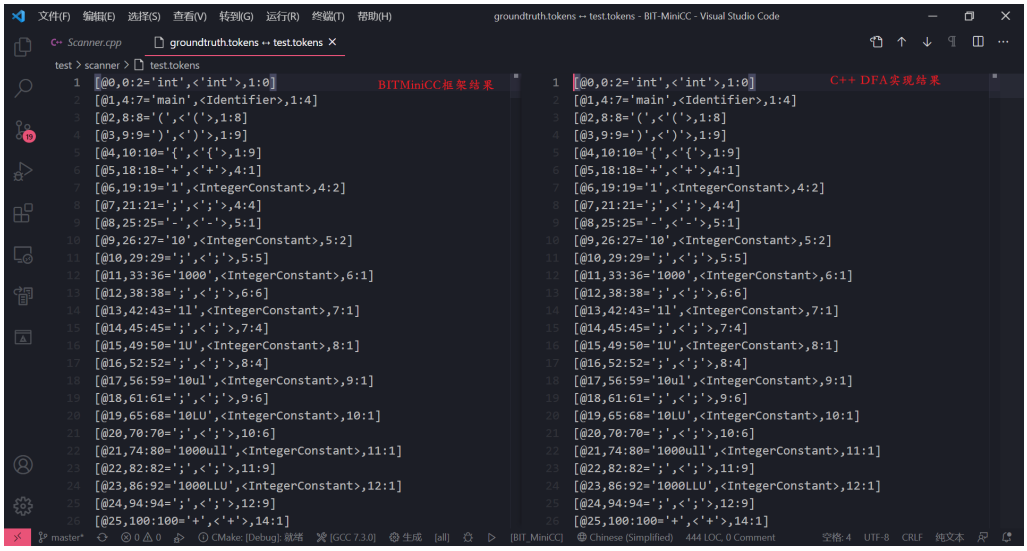
```
class Scanner
{
private:
    unordered_set<string> keywordSet;
    unordered_set<char> nonPrefixOperator;
    vector<string> srcLines;
    int row, col;
    int iTknNum;
    int pos;

    void readFile(string &inputFilePath);
    static void writeFile(const string &strTokens, const string &outputFilePath);
    char getNextChar();
    static string ch2String(char ch);
    static bool isDigit(const char &ch);
    static bool isHexDigit(const char &ch);
    static bool isAlpha(const char &ch);
    static bool isIntSuffix(const char &ch);
    bool isAlphaOrDigit(const char &ch) const;
    string genToken(const string &lexme, const string &type);
    string genToken2(const string &lexme, const string &type);
    string genToken(const string &lexme, const string &type, int col, int row, int pos);

public:
    Scanner();
    ~Scanner();
    void run(string &inputFilePath, string &outputFilePath);
};
```

4 运行结果





使用文本比对工具，比对框架 scanner 的结果和自己使用 C++实现的 DFA 结果相同。

5 实验心得与体会

1. 词法分析的算法思想较为简单，设计有限状态自动机，通过程序中心法实现该自动机。但是实现该过程遇到了一些困难，当可接受字符串中有较多的前缀时，需要添加很多的状态以进行区分，逻辑较为复杂。
2. 设计 DFA 对于不同 DFA 有共同的前缀（后缀）部分时，构造前缀（后缀）初始态，作为一个相对独立的自动机，进行识别。实现 DFA 的复用。