

# C 语言语法分析程序的设计与实现

2017211305 班 2017211240 于海鑫

版本:  $\epsilon$

更新: *October 8, 2019*

本文档为编译原理课程实验“词法分析程序的设计与实现”的实验报告。

## 1 概述

### 1.1 实验内容

构建一个 C 语言的词法分析器，该分析器可以完成如下任务：

- 可以识别出用 C 语言编写的源程序中的每个单词符号，并按照记号的形式输出每个单词符号。
- 跳过源程序中的注释。
- 统计源程序的行数，各类单词的个数，字符总数，并输出统计结果。
- 检查程序中存在的词法错误，并报告错误的所在位置。
- 通过对错误进行恢复，实现只需一次扫描即可报告程序中存在的全部语法错误。

### 1.2 实验环境

操作系统: Windows 10 编程语言: C++ IDE: CLion

## 2 程序设计说明

在不使用 flex 等 Lexer Generator 生成代码时，我们实现词法分析程序主要就是根据语言的规范，手动写出对应的自动机。C 语言的完整 token 定义可以在其规范中的 6.4

节 Lexical elements 内找到。本次试验中使用的 C 规范为 ISO C N2176 草案，并根据自身状况对于标准内的定义进行了些许简化，主要变动如下：

- 不具体解析预处理器相关的内容，实际上在当前市面上主流的开源 C 编译器内，词法分析与预处理也是两个不同的阶段。其余的主流语言甚至没有 C 语言这种简单粗暴的宏机制，因此我们只将预处理相关的部分构成为一个单一的 PreprocessinDirective Token，而不是将其按照 ISO C 规范内要求的那样按照 token 拆分甚至进行替换。
- 处理 Identifier 时，不识别 Universal Character Name
- 将 FloatingConstant 与 IntegerConstant 合并为一类 Token，命名为 NumericConstant，这样我们就可以使用 C++ 的库函数较为方便的判断一组字符是否为 NumericConstant。
- 在处理字符串时，不会处理字符串内出现的换行行为，该部分与 C++ 中的 Raw String 类似，同时我们也不会将多个相邻的字符串合并为一个字符串。
- 在尝试构造 NumericConstant 时，我们会尽可能多的吞掉允许的字符，一旦处理失败，我们会生成一个 NumericConstantWithError Token，以此标注对应的错误。如此设计与实际错误发生的情况比较贴合，可以有效查找出因为错误插入字符而造成的错误。
- 在遇到不识别的字符时，生成一个 Unkown Token，以此表示一个错误，同时吞掉不识别的字符，以此使得词法分析可以继续。

## 2.1 模块划分

### 2.1.1 FileWrapper

该类是对文件的进一步抽象化，以此提供我们所需的记录字符的位置，统计字符总数，查看后续字符等功能。其内部实现为一个 ifstream 以及一个双端队列，类的定义如下：

```
1 class FileWrapper {
2     public:
3     explicit FileWrapper(std::string fileName);
4     char getNextChar();
5     char peekChar(size_t const offset);
6     void eatChars(size_t const num);
7     bool eof();
8     [[nodiscard]] size_t getCount() const;
9     [[nodiscard]] size_t getLineCount() const;
```

```

10     [[nodiscard]] std::string getName() const;
11     friend std::ostream& operator<<(std::ostream& os, const
        FileWrapper & fileWrapper);
12     TokenLocation getLocation();
13 private:
14     void read();
15
16     std::string fileName_;
17     std::ifstream sourceFile_;
18     std::deque<char> buffer_;
19     bool eof_;
20     size_t line_;
21     size_t column_;
22     size_t count_;
23 };

```

各个函数的定义如下：

**explicit FileWrapper(std::string fileName)** 构造函数，根据文件名打开对应的文件。

**char getNextChar()** 获取下一个字符，并更新对应的位置信息。

**char peekChar(size\_t const offset)** 查看后续的字符。

**void eatChars(size\_t const num)** 吞掉 n 个字符。

**bool eof()** 判断文件是否结束。

**[[nodiscard]] size\_t getCount() const** 获取当前字符计数。

**[[nodiscard]] size\_t getLineCount() const** 获取当前行数。

**[[nodiscard]] std::string getName() const** 获取文件名。

**std::ostream& operator<<(std::ostream & os, const FileWrapper & fileWrapper)** 输出调试信息。

**TokenLocation getLocation()** 获取当前字符的位置。

**void read()** 从输入流读取一个字符到队列内。

### 2.1.2 TokenLocation

用以表示 Token 起始字符的位置，定义如下：

```
1 class TokenLocation {
2     public:
3         TokenLocation();
4         TokenLocation(std::string file, size_t col, size_t line);
5         [[nodiscard]] size_t getColCount() const;
6         [[nodiscard]] std::string toString() const;
7     private:
8         std::string fileName;
9         size_t colCount;
10        size_t lineCount;
11    };
```

### 2.1.3 TokenType

用以表示 Token 的类型，定义如下：

```
1 enum class TokenType {
2     #define TOKEN_TYPE(X, Y) X,
3     #include <def/TokenType.def>
4     #undef TOKEN_TYPE
5 };
```

### 2.1.4 KeyWord

用以表示关键字的类型，定义如下：

```

1 enum class KeyWord {
2 #define KEYWORD(X, Y) X,
3 #include <def/Keyword.def>
4 #undef KEYWORD
5 };

```

## 2.2 Punctuator

用以表示符号的类型，定义如下：

```

1 enum class Punctuator {
2 #define PUNCTUATOR(X, Y) X,
3 #include <def/Punctuator.def>
4 #undef PUNCTUATOR
5 };

```

## 2.3 Token

用以保存 Token 的信息，定义如下：

```

1 class Token {
2     public:
3         Token(TokenType tokenType, TokenLocation tokenLocation,
4             KeyWord keyWord, Punctuator punctuator,
5             std::string literalValue);
6         Token(TokenType tokenType, const TokenLocation& tokenLocation,
7             KeyWord keyWord);
8         Token(TokenType tokenType, TokenLocation tokenLocation,
9             Punctuator punctuator);
10        Token(TokenType tokenType, TokenLocation tokenLocation, std::
11            string literalValue);
12        Token();
13        [[nodiscard]] TokenType getTokenType() const;
14        [[nodiscard]] TokenLocation getTokenLocation() const;
15        [[nodiscard]] KeyWord getKeyWord() const;
16        [[nodiscard]] Punctuator getPunctuator() const;
17        [[nodiscard]] std::string getLiteralValue() const;

```

```

14         [[nodiscard]] std::string toString() const;
15         // For debug.
16         friend std::ostream& operator<<(std::ostream& os, const Token
           & token);
17     private:
18         TokenType tokenType_;
19         TokenLocation tokenLocation_;
20         KeyWord keyWord_;
21         Punctuator punctuator_;
22         std::string literalValue_;
23     };

```

## 2.4 Lexer

词法分析器的主体部分，在这个类内完成 `Token` 的生成。类的定义如下：

```

1  class Lexer {
2  public:
3      explicit Lexer(const std::string& fileName);
4      [[nodiscard]] Token getToken() const;
5      Token getNextToken();
6      [[nodiscard]] size_t getCount() const;
7      [[nodiscard]] size_t getLineCount() const;
8      [[nodiscard]] std::string getSrcName() const;
9  private:
10     void skipLineComment();
11     void skipBlockComment();
12     Token getNextNumericToken();
13     Token getNextStringLiteralToken();
14     Token getNextCharacterConstantToken();
15     Token getNextIdentifierToken();
16     Token getNextPreprocessingDirectiveToken();
17     FileWrapper fileWrapper_;
18     TokenLocation currentLocation_;
19     Token token_;
20     std::string tokenBuffer_;
21 };

```

其中 getNextToken 为词法分析的核心函数，在这个函数内，我们根据首字符的不同情况，返回不同的 token，并将之保存到类内部的当前 token 内。

### 2.4.1 Counter

计数器，对各类别 Token 出现的个数进行计数并输出。定义如下：

```
1 class Counter {
2 public:
3     void update(const Token& token);
4     friend std::ostream& operator<<(std::ostream& os, const Counter &
        counter);
5 private:
6     #define TOKEN_TYPE(X, Y) size_t count_##X;
7     #include <def/TokenType.def>
8     #undef TOKEN_TYPE
9 };
```

### 2.4.2 main

主函数，以上述的各个模块为基础，实现试验要求。

## 3 测试

### 3.1 完全正确的程序一例

```
1 #include <stdio.h>
2
3 #define TEST(X, Y, format, ...) \
4 if(X != Y) { \
5     fprintf(stderr, format __VA_OPT__(,) __VA_ARGS__ ); \
6 }
7
8 // Main Function
9 int main(void) {
10     /* Here
```

```

11     * is
12     * a comment */
13     unsigned long long a = 1ULL;
14     double b = .24e+10f;
15     uint64_t c = a ++;
16     a >>= 2;
17     return 0;
18 }

```

输出如下:

```

1 PS D:\Projects\cLex\cmake-build-debug> .\cLex.exe .\right.c
2 Tokens:
3 Type: PreprocessingDirective
4 Location: .\right.c:1:1
5 Value: include <stdio.h>
6
7 Type: PreprocessingDirective
8 Location: .\right.c:3:1
9 Value: define TEST(X, Y, format, ...) \
10 if(X != Y) { \
11     fprintf(stderr, format __VA_OPT__(,) __VA_ARGS__ ); \
12 }
13
14 Type: Keyword
15 Location: .\right.c:9:1
16 Value: int
17
18 Type: Identifier
19 Location: .\right.c:9:5
20 Value: main
21
22 Type: Punctuator
23 Location: .\right.c:9:9
24 Value: (
25
26 Type: Keyword
27 Location: .\right.c:9:10
28 Value: void
29

```



```
30 Type: Punctuator
31 Location: .\right.c:9:14
32 Value: )
33
34 Type: Punctuator
35 Location: .\right.c:9:16
36 Value: {
37
38 Type: KeyWord
39 Location: .\right.c:13:5
40 Value: unsigned
41
42 Type: KeyWord
43 Location: .\right.c:13:14
44 Value: long
45
46 Type: KeyWord
47 Location: .\right.c:13:19
48 Value: long
49
50 Type: Identifier
51 Location: .\right.c:13:24
52 Value: a
53
54 Type: Punctuator
55 Location: .\right.c:13:26
56 Value: =
57
58 Type: NumericConstant
59 Location: .\right.c:13:28
60 Value: 1ULL
61
62 Type: Punctuator
63 Location: .\right.c:13:32
64 Value: ;
65
66 Type: KeyWord
67 Location: .\right.c:14:5
68 Value: double
```

```
69
70 Type: Identifier
71 Location: .\right.c:14:12
72 Value: b
73
74 Type: Punctuator
75 Location: .\right.c:14:14
76 Value: =
77
78 Type: NumericConstant
79 Location: .\right.c:14:16
80 Value: .24e+10f
81
82 Type: Punctuator
83 Location: .\right.c:14:24
84 Value: ;
85
86 Type: Identifier
87 Location: .\right.c:15:5
88 Value: uint64_t
89
90 Type: Identifier
91 Location: .\right.c:15:14
92 Value: c
93
94 Type: Punctuator
95 Location: .\right.c:15:16
96 Value: =
97
98 Type: Identifier
99 Location: .\right.c:15:18
100 Value: a
101
102 Type: Punctuator
103 Location: .\right.c:15:20
104 Value: ++
105
106 Type: Punctuator
107 Location: .\right.c:15:22
```

```
108 Value: ;
109
110 Type: Identifier
111 Location: .\right.c:16:5
112 Value: a
113
114 Type: Punctuator
115 Location: .\right.c:16:7
116 Value: >>=
117
118 Type: NumericConstant
119 Location: .\right.c:16:11
120 Value: 2
121
122 Type: Punctuator
123 Location: .\right.c:16:12
124 Value: ;
125
126 Type: KeyWord
127 Location: .\right.c:17:5
128 Value: return
129
130 Type: NumericConstant
131 Location: .\right.c:17:12
132 Value: 0
133
134 Type: Punctuator
135 Location: .\right.c:17:13
136 Value: ;
137
138 Type: Punctuator
139 Location: .\right.c:18:1
140 Value: }
141
142 File info:
143 File name:      .\right.c
144 Total chars:    319
145 Total lines:    18
146 KeyWord:        7
```

```

147 Identifier:      7
148 NumericConstant:      4
149 NumericConstant(Error Detected):      0
150 CharacterConstant:      0
151 StringLiteral:  0
152 Punctuator:      14
153 PreprocessingDirective: 2
154 EndOfFile:      0
155 Unknown:      0

```

## 3.2 出错程序一例

```

1  #include <stdio.h>
2
3  int main() {
4      int 2ch;
5      `
6      double d = 0x14ge+f;
7      return 1;
8  }

```

结果如下:

```

1  PS D:\Projects\cLex\cmake-build-debug> .\cLex.exe .\wrong.c
2  Tokens:
3  Type: PreprocessingDirective
4  Location: .\wrong.c:1:1
5  Value: include <stdio.h>
6
7  Type: KeyWord
8  Location: .\wrong.c:3:1
9  Value: int
10
11 Type: Identifier
12 Location: .\wrong.c:3:5
13 Value: main
14
15 Type: Punctuator

```

```
16 Location: .\wrong.c:3:9
17 Value: (
18
19 Type: Punctuator
20 Location: .\wrong.c:3:10
21 Value: )
22
23 Type: Punctuator
24 Location: .\wrong.c:3:12
25 Value: {
26
27 Type: KeyWord
28 Location: .\wrong.c:4:5
29 Value: int
30
31 Type: NumericConstant(Error Detected)
32 Location: .\wrong.c:4:9
33 Value: 2c
34
35 Type: Identifier
36 Location: .\wrong.c:4:11
37 Value: h
38
39 Type: Punctuator
40 Location: .\wrong.c:4:12
41 Value: ;
42
43 Type: Unknown
44 Location: .\wrong.c:5:5
45 Value: `
46
47 Type: KeyWord
48 Location: .\wrong.c:6:5
49 Value: double
50
51 Type: Identifier
52 Location: .\wrong.c:6:12
53 Value: d
54
```

```
55 Type: Punctuator
56 Location: .\wrong.c:6:14
57 Value: =
58
59 Type: NumericConstant
60 Location: .\wrong.c:6:16
61 Value: 0x14
62
63 Type: Identifier
64 Location: .\wrong.c:6:20
65 Value: ge
66
67 Type: Punctuator
68 Location: .\wrong.c:6:22
69 Value: +
70
71 Type: Identifier
72 Location: .\wrong.c:6:23
73 Value: f
74
75 Type: Punctuator
76 Location: .\wrong.c:6:24
77 Value: ;
78
79 Type: KeyWord
80 Location: .\wrong.c:7:5
81 Value: return
82
83 Type: NumericConstant
84 Location: .\wrong.c:7:12
85 Value: 1
86
87 Type: Punctuator
88 Location: .\wrong.c:7:13
89 Value: ;
90
91 Type: Punctuator
92 Location: .\wrong.c:8:1
93 Value: }
```

```
94
95 File info:
96 File name:      .\wrong.c
97 Total chars:    92
98 Total lines:    8
99 KeyWord:        4
100 Identifier:     5
101 NumericConstant:      2
102 NumericConstant(Error Detected):      1
103 CharacterConstant:    0
104 StringLiteral: 0
105 Punctuator: 9
106 PreprocessingDirective: 1
107 EndOfFile: 0
108 Unknown: 1
```