C 语言语法分析程序的设计与实现

2017211305 班 2017211240 于海鑫

版本：ϵ

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：

```
class FileWrapper {
    public:
    explicit FileWrapper(std::string fileName);
    char getNextChar();
    char peekChar(size_t const offset);
    void eatChars(size_t const num);
    bool eof();
    [[nodiscard]] size_t getCount() const;
    [[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：

```
class TokenLocation {
    public:
        TokenLocation();
        TokenLocation(std::string file, size_t col, size_t line);
        [[nodiscard]] size_t getColCount() const;
        [[nodiscard]] std::string toString() const;
    private:
        std::string fileName;
        size_t colCount;
        size_t lineCount;
    };
```

2.1.3 TokenType

用以表示 Token 的类型，定义如下：

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

2.1.4 KeyWord

用以表示关键字的类型，定义如下：

```
enum class KeyWord {
#define KEYWORD(X, Y) X,
#include <def/Keyword.def>
#undef KEYWORD
};
```

2.2 Punctuator

用以表示符号的类型，定义如下：

```
enum class Punctuator {
#define PUNCTUATOR(X, Y) X,
#include <def/Punctuator.def>
#undef PUNCTUATOR
};
```

2.3 Token

用以保存 Token 的信息，定义如下：

```
class Token {
    public:
        Token(TokenType tokenType, TokenLocation tokenLocation,
           KeyWord keyWord, Punctuator punctuator,
              std::string literalValue);
        Token(TokenType tokenType, const TokenLocation& tokenLocation
           , KeyWord keyWord);
        Token(TokenType tokenType, TokenLocation tokenLocation,
           Punctuator punctuator);
        Token(TokenType tokenType, TokenLocation tokenLocation, std::
           string literalValue);
        Token();
        [[nodiscard]] TokenType getTokenType() const;
        [[nodiscard]] TokenLocation getTokenLocation() const;
        [[nodiscard]] KeyWord getKeyWord() const;
        [[nodiscard]] Punctuator getPunctuator() const;
        [[nodiscard]] std::string getLiteralValue() const;
```

```
        [[nodiscard]] std::string toString() const;
        // For debug.
        friend std::ostream& operator<<(std::ostream& os, const Token
            & token);
    private:
        TokenType tokenType_;
        TokenLocation tokenLocation_;
        KeyWord keyWord_;
        Punctuator punctuator_;
        std::string  literalValue_;
    };
```

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：

```
class Lexer {
public:
    explicit Lexer(const std::string& fileName);
    [[nodiscard]] Token getToken() const;
    Token getNextToken();
    [[nodiscard]] size_t getCount() const;
    [[nodiscard]] size_t getLineCount() const;
    [[nodiscard]] std::string getSrcName() const;
private:
    void skipLineComment();
    void skipBlockComment();
    Token getNextNumericToken();
    Token getNextStringLiterialToken();
    Token getNextCharacterConstantToken();
    Token getNextIdentifierToken();
    Token getNextPreprocessingDirectiveToken();
    FileWrapper fileWrapper_;
    TokenLocation currentLocation_;
    Token token_;
    std::string tokenBuffer_;
};
```

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：

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

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

```
#include <stdio.h>

#define TEST(X, Y, format,  ...) \
if(X != Y) { \
    fprintf(stderr, format __VA_OPT__(,) __VA_ARGS__ ); \
}

// Main Function
int main(void) {
    /* Here
```

```
    * is
    * a comment */
    unsigned long long a = 1ULL;
    double b = .24e+10f;
    uint64_t c = a ++;
    a >>= 2;
    return 0;
}
```

输出如下：

```
PS D:\Projects\cLex\cmake-build-debug> .\cLex.exe .\right.c
Tokens:
Type: PreprocessingDirective
Location: .\right.c:1:1
Value: include <stdio.h>

Type: PreprocessingDirective
Location: .\right.c:3:1
Value: define TEST(X, Y, format,  ...) \
if(X != Y) { \
    fprintf(stderr, format __VA_OPT__(,) __VA_ARGS__ ); \
}

Type: KeyWord
Location: .\right.c:9:1
Value: int

Type: Identifier
Location: .\right.c:9:5
Value: main

Type: Punctuator
Location: .\right.c:9:9
Value: (

Type: KeyWord
Location: .\right.c:9:10
Value: void

```

```
Type: Punctuator
Location: .\right.c:9:14
Value: )

Type: Punctuator
Location: .\right.c:9:16
Value: {

Type: KeyWord
Location: .\right.c:13:5
Value: unsigned

Type: KeyWord
Location: .\right.c:13:14
Value: long

Type: KeyWord
Location: .\right.c:13:19
Value: long

Type: Identifier
Location: .\right.c:13:24
Value: a

Type: Punctuator
Location: .\right.c:13:26
Value: =

Type: NumericConstant
Location: .\right.c:13:28
Value: 1ULL

Type: Punctuator
Location: .\right.c:13:32
Value: ;

Type: KeyWord
Location: .\right.c:14:5
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
```

```
Value: ;

Type: Identifier
Location: .\right.c:16:5
Value: a

Type: Punctuator
Location: .\right.c:16:7
Value: >>=

Type: NumericConstant
Location: .\right.c:16:11
Value: 2

Type: Punctuator
Location: .\right.c:16:12
Value: ;

Type: KeyWord
Location: .\right.c:17:5
Value: return

Type: NumericConstant
Location: .\right.c:17:12
Value: 0

Type: Punctuator
Location: .\right.c:17:13
Value: ;

Type: Punctuator
Location: .\right.c:18:1
Value: }

File info:
File name:      .\right.c
Total chars:    319
Total lines:    18
KeyWord:        7
```

```
Identifier:     7
NumericConstant:        4
NumericConstant(Error Detected):        0
CharacterConstant:      0
StringLiteral:  0
Punctuator:     14
PreprocessingDirective: 2
EndOfFile:      0
Unknown:        0
```

3.2 出错的程序一例

```c
#include <stdio.h>

int main() {
    int 2ch;
    `
    double d = 0x14ge+f;
    return 1;
}
```

结果如下：

```
    PS D:\Projects\cLex\cmake-build-debug> .\cLex.exe .\wrong.c
    Tokens:
    Type: PreprocessingDirective
    Location: .\wrong.c:1:1
    Value: include <stdio.h>

    Type: KeyWord
    Location: .\wrong.c:3:1
    Value: int

    Type: Identifier
    Location: .\wrong.c:3:5
    Value: main

    Type: Punctuator
```

```
    Location: .\wrong.c:3:9
    Value: (

    Type: Punctuator
    Location: .\wrong.c:3:10
    Value: )

    Type: Punctuator
    Location: .\wrong.c:3:12
    Value: {

    Type: KeyWord
    Location: .\wrong.c:4:5
    Value: int

    Type: NumericConstant(Error Detected)
    Location: .\wrong.c:4:9
    Value: 2c

    Type: Identifier
    Location: .\wrong.c:4:11
    Value: h

    Type: Punctuator
    Location: .\wrong.c:4:12
    Value: ;

    Type: Unknown
    Location: .\wrong.c:5:5
    Value: `

    Type: KeyWord
    Location: .\wrong.c:6:5
    Value: double

    Type: Identifier
    Location: .\wrong.c:6:12
    Value: d

```

```
    Type: Punctuator
    Location: .\wrong.c:6:14
    Value: =

    Type: NumericConstant
    Location: .\wrong.c:6:16
    Value: 0x14

    Type: Identifier
    Location: .\wrong.c:6:20
    Value: ge

    Type: Punctuator
    Location: .\wrong.c:6:22
    Value: +

    Type: Identifier
    Location: .\wrong.c:6:23
    Value: f

    Type: Punctuator
    Location: .\wrong.c:6:24
    Value: ;

    Type: KeyWord
    Location: .\wrong.c:7:5
    Value: return

    Type: NumericConstant
    Location: .\wrong.c:7:12
    Value: 1

    Type: Punctuator
    Location: .\wrong.c:7:13
    Value: ;

    Type: Punctuator
    Location: .\wrong.c:8:1
    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
```