同濟大學
TONGJI UNIVERSITY

一个简单的类 C 语言词法分析器的设计与实现

姓　　名：周雨杨

学　　号：1953793

所在院系：电子与信息工程学院

学科专业：计算机科学与技术

指导教师：高秀芬老师

二○二一年十月

目录

第 1 章　实验背景

本章主要介绍类 C 语言词法分析器的背景。

1.1　实验要求

词法分析（lexical analysis）是计算机科学中将字符序列转换为标记（token）序列的过程。进行词法分析的程序或者函数叫作词法分析器（lexical analyzer，简称 lexer），也叫扫描器（scanner）。词法分析器一般以函数的形式存在，供语法分析器调用。

标记是一个字串，是构成源代码的最小单位。从输入字符流中生成标记的过程叫作标记化（tokenization），在这个过程中，词法分析器还会对标记进行分类。词法分析器通常不会关心标记之间的关系（属于语法分析的范畴）。

实验具体要求如下：

- 对给定的类 C 语言的单词集，试编写一个词法分析器，输入为源程序字符串，输出为单词的机内表示序列 (< 种别, 属性值 >)。
- 词法分析器设计为子程序以供主程序或后续实验的语法分析程序调用。
- 在完成以上基本要求情况下，对程序功能扩充（如：增加单词数量、出错处理、预处理程序等）。

1.2　类 C 词法规则

此类 C 语言是 C 的一个子集，包括了如下内容：

表 1.1　正则表达式定义的类 C 词法规则

种别	定义
关键字	int|void|if|else|while|return
标识符	字母 (字母 | 数字)*
数值	数字 (数字)*
赋值号	=
算符	+|-|*|/|=|==|>|>=|<|<=|!=
界符	;
分隔符	,
注释号	/* */ | //
左括号	(

续下页

续表 1.1　正则表达式定义的类 C 词法规则

种别	定义
右括号	)
左大括号	{
右大括号	}
字母	|a|...|z|A|...|Z|
数字	0|1|2|3|4|5|6|7|8|9

词法分析器则需要根据上述词法规则扫描源程序，将源程序拆解成标记序列输出，一个例子如下：

```
int  program(int a,int b,int c)
{
        int i;
        int j;
        i=0;
        if(a>(b+c))
        {
                j=a+(b*c+1);
        }
        else
        {
                j=a;
        }
        while(i<=100)
        {
                i=j*2;
        }
        return i;     }
```

```
<1, - >
<2, 1 >
<3, - >
<1, - >
<2, 2 >
… …
```

或

```
<$int, - >
<$ID, 1 >
<$(, - >
 <$int, - >
<$ID, 2 >
… …
```

第 2 章　运行和开发环境

2.1　开发环境

- **CPU：**Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
- **操作系统：** 5.13.19-2-MANJARO　*x*86_64　GNU/Linux
- **编译器：** g++(GCC) 11.1.0
- **调试器：** GNU gdb (GDB) 11.1
- **编辑器：** Visual Studio Code(核心代码)　Qt Creator 5.0.2(GUI)
- **界面布局工具：** QmlDesigner 5.0.2

2.2　运行环境

- g++(GCC) 11.1.0
- cmake version 3.21.3
- Qt version 5.9.7

第 3 章 系统设计

3.1 词法扩充

在本人的类 C 词法分析器实验中，扩充了较多基础功能，支持的语法更为丰富。

在关键字方面，本人主要增加了 float,double, char 等类型关键字，以支持后续对浮点数和字符串类型的处理。另外还增加了 main 等 C 语言中常用的关键字。

在出错处理方面，本人的类 C 词法分析器能够识别未定义类型的数据，并检测到字符串和浮点类型的错误情况（如只有一个引号，小数点后没有数字等等）。

在预处理程序方面，本人的词法分析器能够实现对注释的自动清除。

另外，本人增加了较多运算符，并为它们的类别重新命名，做出更细致的划分，以利于后续的语法分析步骤。符号具体分为以下几类

- **赋值运算符 (assign symbol)：** =| +=| -=| *=| /=
- **算数算符 (algebra calculator)：** +| -| *| /| ++| –
- **比较算符 (compare calculator)：** >| <| >=| <=| ==| !=
- **逻辑算符 (logic calculator)：** &&||||!
- **位运算 (bit calculator)：** |&||| << | >>
- **分界符 (delimiter symbol)：** ;
- **分隔符 (divider symbol)：** |
- **括号 (bracket symbol)：** (|)| {| }|[|]
- **其他符号 (other symbol)：** #|.

按照如上运算符分类规则，以及新增的字符串、浮点数、未定义类型 3 个新的种别，可以得到扩充后的词法规则如下：

表 3.1 扩充后的类 C 词法规则

种别	定义
关键字	double|float|char|main|int|void|if|else|while|return
标识符	字母 (字母 | 数字)*
整数	数字 (数字)*
浮点数	整数. 整数
字符串	“字符*”
赋值号	=| +=| -=| *=| /=
算数算符	+| -| *| /| ++| –
比较算符	>| <| >=| <=| ==| !=

续下页

续表 3.1　扩充后的类 C 词法规则

种别	定义
逻辑算符	&&||||!
位运算	|&||| << | >>
分界符	;
分隔符	,
括号	(|)| {| }|[|]
其他符号	#|.
字母	|a|...|z|A|...|Z|
数字	0|1|2|3|4|5|6|7|8|9
符号	以转义符或打印形式表示的 ascii 字符

3.2　模块划分

整个应用由 **GUI** 和**处理器**构成。用户在 GUI 中输入一段程序或打开源文件，其内容送至 UI 背后的处理器，也即词法分析器。词法分析器则返回一个 Token 组成的数组，再次显示到 GUI 中。

处理器由**预处理器**和**词法分析器**组成。在预处理器中，我们首先去除源程序中的注释，然后将多个空格或回车的情况统一视作一个空格，作为源程序的一种分割方式。然后，在词法分析器中，在之前预处理好的每个分割方式内部，依次扫描字符串，根据当前的状态和下一个扫描的字符内容确定是否分割。分割后，再确定每个部分是什么类型，得到 Token。

3.3　词法分析器流程图

词法分析器的本质是一个扫描器，也是一个状态机。词法分析器依次扫描字符，若字符属于合法输入，则读入缓存，并根据状态机的现有状态和输入，确定它的下一个状态；否则，认为此处发生了分割，判断缓存中的数据所属类型，并清空缓存、重置状态为初始值，然后再次读入当前 input。当读到空格或结尾时，当前片段读取完毕，判断缓存中数据所属类型，并清空缓存、重置状态为初始值，然后读入下一个字符或结束（读到结尾时）。

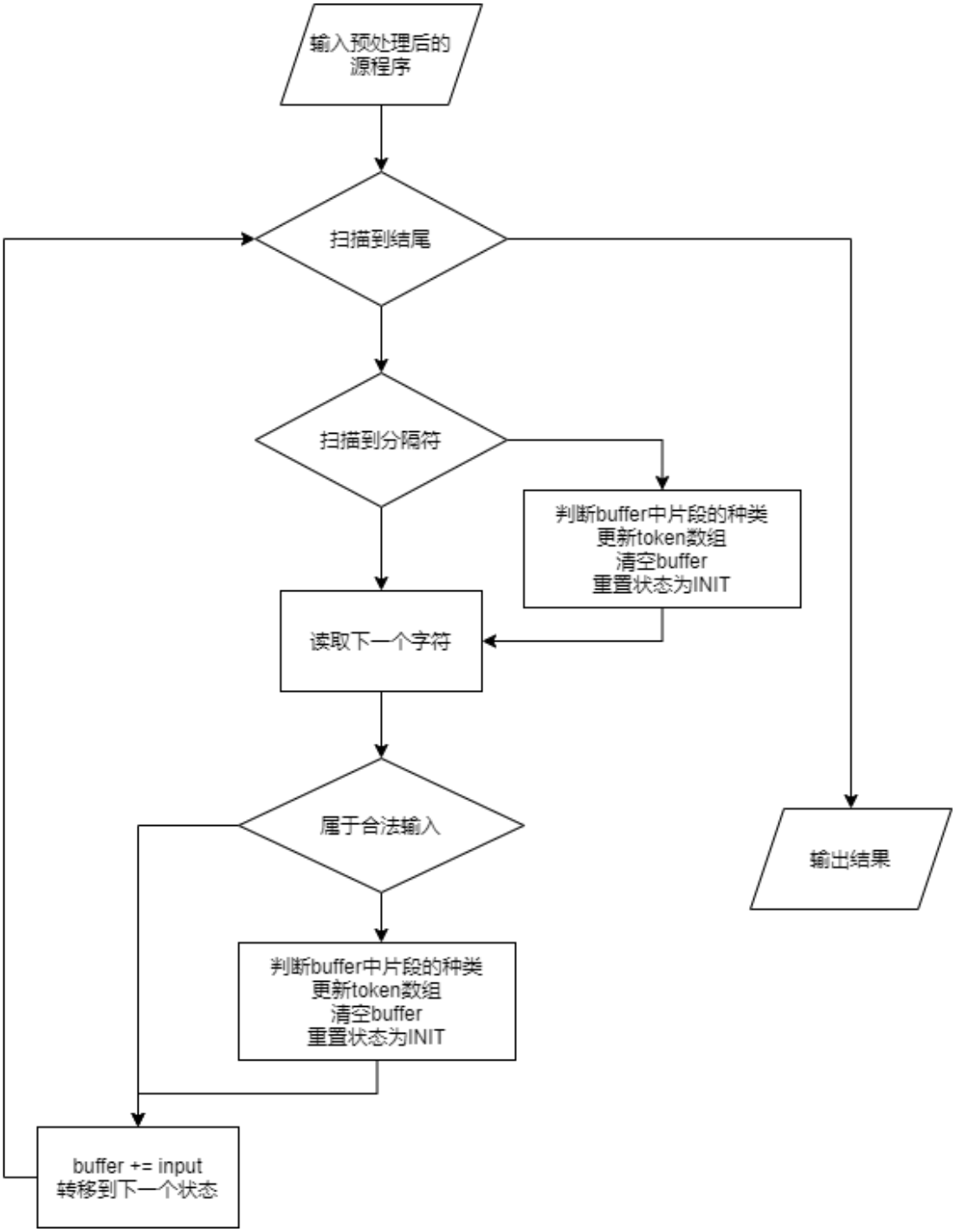

图 3.1 词法分析器流程图

3.4　状态转换图

对于初始状态，根据输入类型可能有 NUM, LETTER, SYMBOL, OPEN STRING 四个状态。在 NUM 状态中，可能会产生整数或浮点数；在 LETTER 状态中，可能会产生关键字和表示符；在符号状态中可能会产生定义的各类符号以及未定义的类型。在 OPEN STRING 状态中则可能产生 STRING 以及未定义的类型。

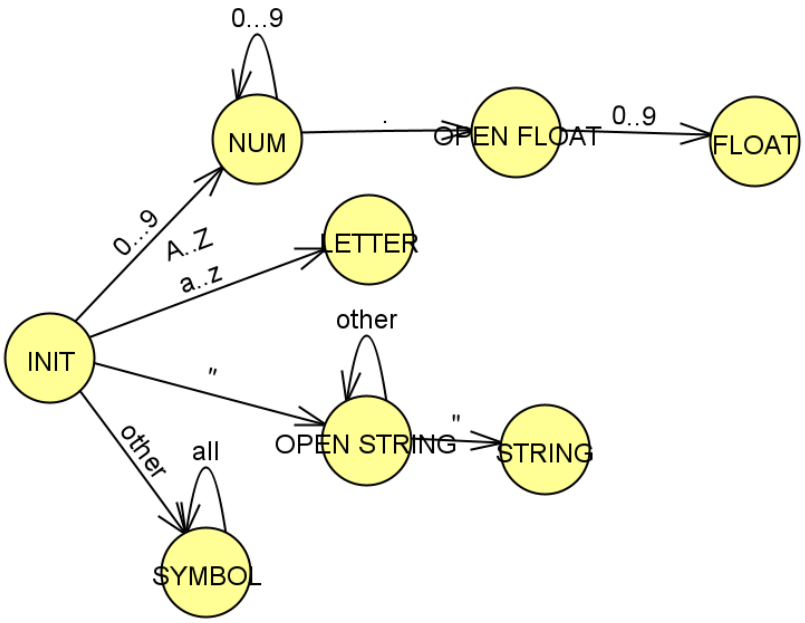

图 3.2　状态转换图

第 4 章　源代码结构说明

项目文件树如图所示,

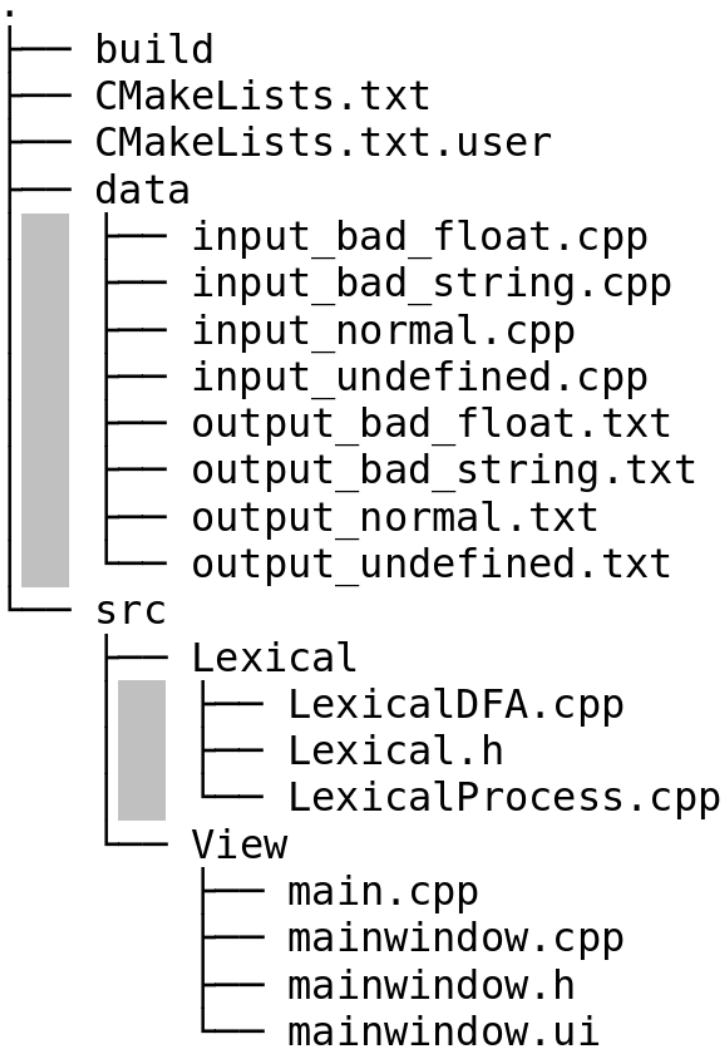

图 4.1　项目文件树

项目目录中有以下内容:

- **build 目录:**CMake 生成的中间文件和可执行文件
- **data 目录:** 测试用例文件及对应的输出文件
- **src 目录:** 全部的源代码文件
- **CMakeLists.txt** 项目编译环境控制文件。

src 目录中有以下内容:

- **Lexical 子目录：**包含词法分析器及预处理器的源代码
- **View 子目录：**包含 GUI 相关的源代码

将来，语法分析器的各个模块也将在此目录中。

Lexical 目录中有以下文件：

- **Lexical.h:** 词法分析器头文件，包含了内部常量和类型的定义以及对外的函数接口。其他模块可以通过 inclue 这个头文件来调用词法分析器。
- **LexicalDFA.cpp:** 词法分析器状态机的定义与状态转换的实现。
- **LexicalProcess.cpp:** 词法分析器整体处理逻辑的实现。

View 目录中有以下文件：

- **main.cpp:** 应用程序入口函数。
- **mainwindow.cpp:**GUI 各个组件的前端逻辑与接口。
- **mainwindow.h**GUI 各个组件和方法的定义。
- **mainwindow.ui**GUI 布局文件

第 5 章 测试用例

首先是一个综合了大部分 token 的普通情况。

input_normal.cpp

```
/*
 * Created Date: 2021-10-12 22:02:30
 * Author: yuyangz
 */

// input test file with all symbol defined.

int main()
{
    char s[] = "abc";
    double d = 1.2;
    int xVariable=1, y = 2;
    int z = 0;
    if (xVariable < y){
        z = y/xVariable;
    }
    else {
        z = xVariable*y;
    }
    while (z != 0) {
        z = z + xVariable;
    }

    return 0;
}

//End of file
```

另外是一个包含未定义符号的情况。

input_undefined.cpp

```
/*
 * Created Date: 2021-10-23 12:27:49
 * Author: yuyangz
 */

// input test file with some symbol undefined.

int main()
{
    int xVariable^% =1, y = 2;
    int z = 0;
    if (xVariable < y){
        z = y/xVariable;
    }
    else {
        z = xVariable*y;
    }
    while (z != 0) {
        z = z + xVariable;
    }

    return 0;
}

//End of file .
```

这个测试用例则包含字符串的错误情况。

input_bad_string.cpp

```
/*
 * Created Date: 2021-10-25 11:22:59
 * Author: yuyangz
 */


```

```
7
8  char s[] = '';
```

最后一个测试用例包括了浮点数的三种错误情况。

input_bad_float.txt

```
1  /*
2   * Created Date: 2021-10-25 11:23:05
3   * Author: yuyangz
4   */
5
6
7
8  float x = 1.23.;
9  float y = 1..23;
10 float z = .123;
```

第 6 章　运行结果

input_normal 测试用例的结果如下

图 6.1　input_normal.cpp 的结果

input_undefined 测试用例的结果如下

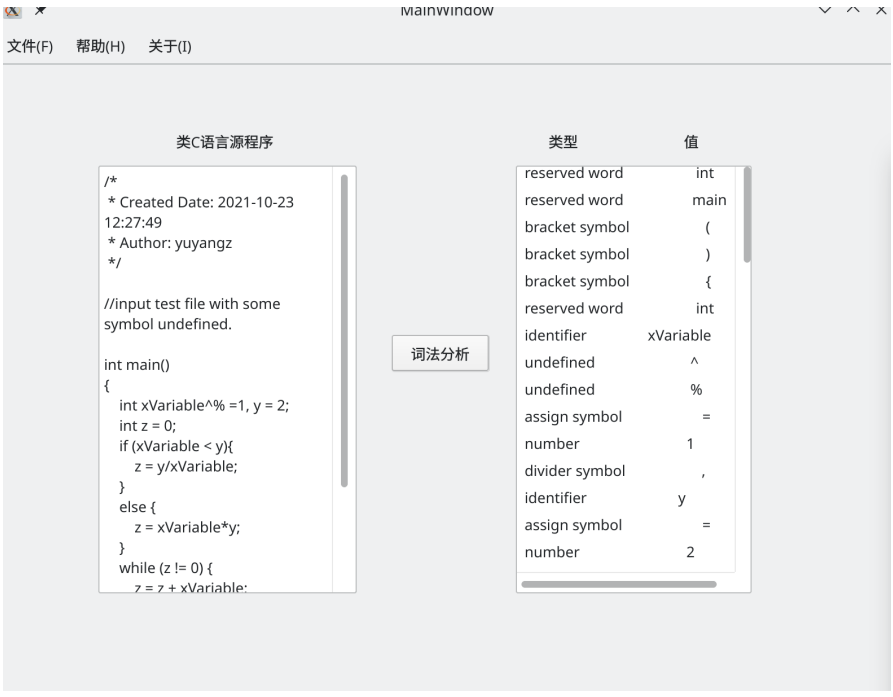

图 6.2　input_undefined.cpp 的结果

input_bad_string 测试用例的结果如下

图 6.3　input_bad_string.cpp 的结果

input_bad_float 测试用例的结果如下

图 6.4　input_bad_float.cpp 的结果