实验一　词法分析与语法分析

一、程序功能及其实现

1. 功能概述

本程序实现了一个简单的词法分析器和语法分析器，能够识别并解析 C - - 代码。除了基本的语法树打印、词法错误和语法错误识别外，程序还实现了选做的全部内容，包括：

- 识别八进制数和十六进制数；
- 识别指数形式的浮点数；
- 识别“//”和“/*…*/”形式的注释。

此外，为了支持报多行错误，程序还实现了错误恢复机制，能够在识别到错误后继续分析。经测试，程序在 test1 ∼ test10 上运行的输出与预期结果一致。

2. 实现细节

1. **语法树**的结点定义为一个结构体，包含结点名称、行号、类型和子结点指针等。由于语法树为多叉树，所以采用了儿子-兄弟表示法，方便程序编写。

2. **词法分析**中，程序使用正则表达式识别各种类型的 token，并调用 `tree.c` 定义的 `newLeafNode` 函数生成叶子结点，将 token 的相关信息存入结点中。当词法分析器遇到未定义的字符时，会报 A 类错误错并跳过该字符（如 ∼ 或 @）。

3. **语法分析**中，程序基于给定的 C - - 语法规则，调用 `tree.c` 中定义的 `newInternalNode` 函数构建语法树。为了解决二义性问题，程序显式地指定了一些非终结符的优先级和结合性。

二、程序编译方式

1. 实验环境

- 操作系统：Ubuntu 20.04.6 LTS
- 编译器：gcc version 9.4.0
- Flex：2.6.4
- Bison：3.5.1

2. 文件结构

```
lab1-code/（根目录）
├── build.sh（编译脚本）
├── lexical.l（词法分析器源代码）
├── main.c（主程序源代码）
├── nodeType.h（语法树结点类型定义）
├── syntax.y（语法分析器源代码）
├── tree.c（语法树操作函数）
├── tree.h（语法树头文件）
├── utils.c（工具函数）
└── utils.h（工具函数头文件）
```

3. 编译方式

在根目录下执行 `./build.sh` 即可编译程序。编译完成后，会生成一个名为 `parser` 的可执行文件。可以通过 `./parser path/to/test.cmm` 运行程序，其中 `test.cmm` 是待分析的 C - - 源代码文件。

三、程序创新与亮点

1. 实现了对八进制数、十六进制数和指数形式的浮点数的识别

八进制、十六进制数和指数形式的浮点数的识别与计算均在词法分析阶段完成。在构造结点时，程序会将八进制、十进制和十六进制的字符串转换为十进制整数，存入结点的

ival 字段中。对于指数形式的浮点数字符串，程序会将其转换为 float 类型，存入结点的 fval 字段中。相关转换程序在 utils.c 中实现。

当遇到不合法的八进制、十六进制数或浮点数时（如 float i = 1.05e;），一般有如下两种错误处理方式：

- 将此 token 视为不合词法规则的 token，报 A 类错误；
- 尽可能多地识别出 token 中的合法部分，并将剩余部分视为下一个 token，继续进行语法分析。如果后续分析出现错误，再报 B 类错误。

两种方式各有优劣，第一种方式更符合直观，但难以列出所有可能的错误情况，而且需要谨慎考虑，防止将其他错误或合法 token 误判为非法数值 token；第二种方式易于实现，且鲁棒性较好，不存在误判的可能，但可能导致错误的传播。得益于我在 syntax.y 中实现的错误恢复机制，程序能够在遇到错误后及时继续分析，因此我选择了第二种报错方式。

2. 实现了对单行注释和多行注释的识别

两种注释的识别也在词法分析阶段完成。这里有必要说明注释的识别方法。当遇到“//”时，程序调用 Flex 库函数 input 读取字符，直到遇到换行符为止，二者之间的字符均视为注释。当遇到“/*”时，程序不断读取字符，直到遇到“*/”，或读到文件末尾为止。

根据实验要求，C - - 不支持嵌套注释，因此本程序没有实现嵌套注释的识别。如果遇到嵌套注释，程序会将第二个“*/”视为两个运算符，从而报 B 类错误。如果要支持嵌套注释，可以用一个栈来维护注释的层级，每次遇到“/*”时入栈，遇到“*/”时出栈。

3. 实现了较为完善的错误恢复机制

为了实现从错误中恢复，以便继续分析，我在 syntax.y 中实现了错误恢复机制，这需要在原来语法规则的基础上添加一些错误处理规则。例如，我添加的一条生成式 Exp | Exp LB error RB 可以在缺少右方括号时继续分析，将后续的 token 视为 error，直到遇到下一个右方括号。

由于错误恢复机制的实现较为复杂，需要考虑多种情况，因此我实现的错误恢复机制大概率并不完美。为了尽可能实现合理的错误恢复，我自己参考、书写了许多测试用例，不断改进恢复机制。现在的程序在目前所有的测试用例上均能正确运行。