编译原理实验报告一

161220070 李鑫烨

实现功能

实验一实现对 C–语言书写的源代码进行词法分析与语法分析，并构造语法分析树。实验主要思路在于

1. 借助 Flex 和正则表达式，将特定形式的字符串识别为对应词法单元，匹配失败则识别为词法错误。
2. 借助 Bison 和移入-归约语法分析技术，处理词法分析器中返回的词法单元，逐步归约为开始符号，完成语法分析。在移入-归约过程中，若状态机无法针对输入词法单元采取动作，则识别为语法错误并启动错误恢复过程。
3. 声明语法分析树节点类型，并以此定义词法单元。在文法规则附加动作中，使规则 $A := \alpha B \beta$ 中 B 父节点为 A，十分自然地得到语法分析树。

实验一实现了必做内容之外完成了以下选做内容:

- 识别八进制与十六进制数。程序识别符合词法定义的八进制数与十六进制数，否则识别为词法错误，错误格式八进制数或十六进制数，而不是将问题留给语法分析。lab1/Test 中 test5.cmm 和 test6.cmm 分别测试对于合法与非法的八进制数和十六进制数的识别。
- 识别指数形式的浮点数。相似地，程序识别符合词法定义的指数形式浮点数，否则识别为词法错误。lab1/Test 中 test7.cmm 和 test8.cmm 分别测试对于合法和非法指数形式浮点数的识别。
- 识别单行和多行注释，并识别未匹配的 /* 及之后内容为词法错误。对于未匹配的 */，程序将会识别为 * 和 /，提示为语法错误。lab1/Test 中 test9.cmm 和 test10.cmm 分别测试对于合法以及非法注释的识别。

编译方法

在 lab1/Code 中输入以下指令，

- make parser: 生成可执行文件 parser 并替换上级目录中 parser。
- make run: 已生成可执行文件 parser 条件下，测试 lab1/Test 中所有用例。
- make clean: 删除可执行文件及中间文件。

个性化内容

实验一较多地借助了 Flex&Bison，因此正式表达式识别字符串为词法单元，用移入-归约技术处理词法单元序列并构建语法分析树的过程大体相似。实验中对多行注释的正则表达式识别是我考虑较多的部分。

Flex 中正则表达式不支持后顾，因此识别多行注释需要较长的正则表达式或者添加一些附加状态信息。实验中采取了构造较先考虑较为正则表达式的方法，先考虑较简单的情形，多行注释/*...*/中不包含 * 号，正则表达式为

```
comment1  \/\*[^*]*\*\/
```

考虑较为复杂的情形，多行注释为/*...*...*...*/形式，正则表达式为

```
part1    \/\*[^*]*\*+
part2    [^*/][^*]*\*+
comment2  {part1}{part2}*\/
```

其中 part1 表示/* 以及不含 * 的字符串与 *，而 part2 表示不以/开始并以 * 结束的字符串，因此 part1 与 part2 连接，以及 part2 的闭包中都不会出现 */，因此匹配不会结束。注释当且仅当 part2 后接/时结束。

综合考虑发现，comment1 所表达语言包含于 comment2，最后正则表达式为

```
comment    \/\*[^*]*\*+([^*/][^*]*\*+)*\/
```