编译原理实验一报告

161220072 李颖

一、实验任务

使用词法分析工具 GNU Flex 和语法分析工具 GNU Bison，编写一个程序对使用 C--语言书写的源代码进行词法分析和语法分析，并打印分析结果。

二、已实现功能

在本实验中已经实现的功能如下：
必做功能：识别 c--源代码中的词法、语法错误，在没有以上两种错误的情况下输出语法树；
选做功能：（1）识别八进制数和十六进制数，
（2）识别指数形式的浮点数，
（3）识别两种风格的注释，
（4）*识别不合法的标识符。

三、提交文件及使用

提交的文件包括子文件 Code、Test 以及可执行文件 parser 和实验报告 report.pdf。既可以直接使用该可执行文件，也可以在 Code 目录下命令行输入 make 编译链接代码得到可执行文件 parser，最终需要以“./parser test1.cmm”的形式执行。

四、实验设计

1. 语法树设计

在本实验中，使用子女-兄弟链表构造树，每个结点包含它的第一个子女（firstChild）结点地址和它的下一个兄弟（nextSibling）结点地址。每个结点所储存的信息包括语法单元名称、所在行数、类型（是否终结符号）。对于终结符号 INT、FLOAT、ID 和 TYPE，结点还会储存它们在源代码中的词素。

仅在语法分析部分进行结点的创建和树的创建，结点所需词素从词法分析的 yylval 中获取，从而我们需要对 YYSTYPE 的类型进行重定义。通过利用 Bison 内置功能，将 YYSTYPE 设置如下：

```
%union {
  struct TreeNode* node;
  int type_int;
  float type_float;
  char* type_string;
}
```

语法单元 INT FLOAT 类型为 type_int 和 type_float，其他非终结符号类型为 node，而终结符号类型为 type_string。接着，在 Bison 的语义动作中添加创建结点和建树的过程，例如：

```
ExtDef : Specifier ExtDecList SEMI
```

```
{   TreeNode* n3 = createNode("SEMI", 0, @3.first_line);
    $$ = createNode("ExtDef", 1, @$.first_line);
    $$->firstChild = $1;
    $1->nextSibling = $2;
    $2->nextSibling = n3;
 }
```

对于建立好的语法树，采用深度优先递归前序遍历打印出来，值得注意的是，根据打印要求，firstChild 为 NULL 的非终结符号不用打印出来，因为其对应的语法单元产生了空串。

2. 八进制数和十六进制数识别

通过查阅 C--语法规范，我们可以写出合法的八进制和十六进制的正则表达式：

- ✓ Octal number: 0[0-7]+
- ✓ Hex number: 0[Xx][0-9a-fA-F]+

形如 09 和 0xG12 的是非法的数字，根据定义我们可以得出非法八进制为以 0 而非 0x(X)开头的数字串中出现了不是 0~7 的数字或字母，数字包含 8 和 9，字母包含 a~f 和 A~F。为什么不是所有的字母呢？因为既然 C--仅支持八、十、十六进制，那么数字中不会出现 g~z 或 G~Z。如果出现了这些字母呢？那我们将其视为不合法的标识符，下文会说。而不合法的十六进制是以 0x(X)开头的数字串中出现了字母 g~z 或 G~Z。

因而写出非法数字的正则表达式：

- ✓ Illegal Octal number: (0[0-7]+([89a-fA-F]+[0-7]*)+)|(0[89a-fA-F][0-9A-Fa-f]*)
- ✓ Illegal Hex number: 0[xX][0-9A-Fa-f]*([G-Zg-z]+[0-9A-Fa-f]*)+

3. 指数形式浮点数识别

根据 C--语言规划，写出浮点数的正则表达式：

- ✓ 非指数形式：([0-9]+\.[0-9]*)|(\.[0-9]+)
- ✓ 指数形式：(([0-9]+\.[0-9]*)|(\.[0-9]+))([eE][-+]?[0-9]+)

接下来，判定几种非法的浮点数（只列出仅有基数或指数错误的情况）：

- ✓ 基数非法：
  - 空：[eE][-+][0-9]+
  - 无小数点：[0-9]+[eE][-+]?[0-9]*
  - 多小数点：([0-9]*\.([0-9]*\.)+[0-9]*)([eE][-+]?[0-9]+)?
  - 只有小数点：\.[eE][-+]?[0-9]+
- ✓ 指数非法：
  - 空： (([0-9]+\.[0-9]*)|(\.[0-9]+))[eE]
  - 存在小数点：(([0-9]+\.[0-9]*)|(\.[0-9]+))[eE][-+]?[0-9]*\.([0-9]*\.)*[0-9]*

4. 两种注释识别

对于“//”形式的注释，忽略当前行的内容，这可以利用词法分析的 input 来做。而对于“/*...*/”形式的注释，既然不支持嵌套，所以在遇到一个“/*”，只要查找到第一个“*/”，忽略它们之间的所有内容，这里需要考虑找不到“*/”的情况，也即到了文本结尾，需要给出报错。

5. 不合法的标识符识别

上文已经提到了，对于以数字开头的包含数字、字母、下划线的串是不合法的标识符，具体可分为两类：

以 0 开头的非法标识符：包含 f~z 或 F~Z 或下划线：0[0-9a-fA-F]*([g-zG-Z_]+[0-9a-fA-F]*)+

以非 0 的数字开头，包含字母、数字或下划线：[1-9][0-9]*([a-zA-Z_]+[0-9]*)+

识别出不合法的标识符的好处有：

（1）给出更明确的报错信息

（2）减轻错误恢复的压力