2019 年 10 月 11 日 星期五

编译原理实验一　实验报告

171860689 蔡跃龙 857570220@qq.com
171860673 林昊 1184264181@qq.com

实验内容

本次实验任务是编写一个程序对使用 C 语言书写的源代码进行词法和语法分析，并打印分析结果。使用词法分析工具 GNU Flex 和语法分析工具 GNU Bison，并使用 C 语言来完成。如果源代码不存在语法错误，则输出语法树作为分析结果，否则，输出错误提示。要求完成附加功能 1.2——识别指数形式的浮点数。

程序实现功能以及过程

- 词法分析

在书的附录 A 的文法定义中有词法正则表达式的表示方法。在书写正则表达式的表示时，需要注意 Tokens 的书写顺序，否则会产生错误的匹配，同时为了实现指数形式浮点数的错误表示，定义了 ERR_FLOAT 的正则表达式。首先正确的浮点数分为非指数型和指数型浮点数，分别构造正则表达式，而错误的浮点数只考虑指数型浮点数，考虑以下情况——指数符号（e、E）前面基数部分没有小数点，或者后面没有指数部分，其余错误情况不在词法分析中考虑。另外，由于语法分析需要词法分析器在返回 Tokens 的时候同时返回其属性，所以设该属性的类型为联合体中包含我们自己定义的 treeNode*类型。同时注意为了能够满足最终打印 TYPE、INT、FLOAT、ID 的特殊要求，在词法分析获取此类词法单元时调用 sscanf 函数将 yytext 内容按格式存入 treeNode*类型中的 data 联合体当中。

- 语法分析

在 Bison 程序当中，首先把终结符 Tokens 和非终结符 Non-terminals 均定义为 struct treeNode* type_node 类型，然后根据指导书内容设置运算符优先级、结合性，

消除 if else 语句在文法中可能存在的二义性。在书写文法的产生式部分，我们根据词法分析中通过 yylval 返回的 Tokens 及其属性，对当前产生式头的非终结符调用 InitNode 函数创建新结点，然后调用 InsertNode 函数将已经生成的产生式体各结点作为该新结点的子节点插入，最终在 Program 结点处生成一个完整的语法分析树，把树的根节点赋值给全局变量 root。

- 语法树的建立

语法树的建立核心是多叉树的建立，由于 Bison 的语法分析器是自底向上进行分析的，因此语法树的建立也是自底向上的。

语法树结点的定义中，包含标识语法单元的名称、子节点数、行数、union 类型的表示 int/float/str 的 data，节点的右兄弟指针，节点的第一个子节点。

为了实现语法树的建立，定义了两个函数，InitNode 和 InsertNode。InitNode 实现节点的内存分配，和成员变量的初始化。错误的节点内存分配会导致出现 Segmentaion fault，或者导致语法单元和词法单元的交互产生错误。InsertNode 函数的参数是不确定的，因为每个产生式的右边语法单元数并不相同，所以多叉树的节点的子节点个数不确定。子节点不确定的解决方案有两种，一种是利用指针数组，构造存储子节点的数组，另一种是，用一个指针指向第一个子节点，然后子节点之间用指针相连，即链表结构。本次实验采用第二种方案。InsertNode 函数的具体实现使用了 C 语言提供的变长参数 va_list，可以简化语法分析的调用该函数的方法。使用了 va_list 解决变长参数后，问题就简化成了链表的建立，比较简单，不再赘述。

- 语法树的输出

语法树的输出比较简单，利用 DFS 遍历语法树，利用 t_no 表示缩进数量。由于词法单元和语法单元输出要求有所不同，因此需要利用 data 输出不同类型的词法单元的内容。

- 错误处理

对于词法错误，我们对不符合正则表达式的词法单元直接输出类型为 A 的错误；对于语法错误，我们为语法分析规则添加含有 error 的产生式，并在发现每一个 error 的地方调用 PrintSynErr 函数打印类型为 B 的错误。此处，我们添加 error 的原则是尽量在';', '{', '}', '(', ')', '[', ']'之前添加 error，以满足指导书上的要求。对发现的每一个词法错误或语法错误，我们会维护两个全局变量 lexErrNum 和 synErrNum，当且仅当这两个全局变量均为 0 时才会调用 PrintDFS 函数打印语法分析树，否则就不会打印语法分析树。

程序编译方法

- 程序编译

命令行进入 Lab/Code 目录，依次键入 make clean、make、make test 命令并回车即可生成 parser 处理程序并测试相应测试文件。另外，如果进入 Makefile 文件进行编辑，可以测试不同的测试文件。

程序设计创新

- Debug

在程序的开发过程中，曾经利用 C 语言提供的 assert 函数检查代码逻辑，利用自己编写的 PrintInfol、PrintInfom 函数检查词法分析获得的词法单元的正确性，使用重写的 yyerror 函数等检查每一个出现语法错误的地方，提高了 Debug 效率。

- 脚本批量测试

在程序的测试阶段，曾经编写 run.sh 脚本文件利用循环批量读入 10 个书上提供和 31 个自己构造的 test.cmm 文件测试输出，节省了在命令行中反复输入重复命令的工作量，提高了工作效率。