编译原理实验一报告

胡俊豪 181240020@smail.nju.edu.cn

1 功能

本实验完成了实验手册中所描述的所有功能，并通过了所有测试样例。

此外，需要额外说明的是：根据实验手册中的样例解释部分，我们将所有和浮点数或者十六进制、八进制数相关的错误，识别为 Type B error 而不是 Type A error。特此说明，以防止测试时被误伤。

1.1 正确程序的解析

只要输入程序是正确的 cmm 代码，我们的程序总能够按照实验要求，正确打印出文法树。比较需要说明的是，在输入文件中有词法文法错误的时候，会出现的状况，这些事项放在后几个小节说明。

1.2 注释相关的错误检查

我们通过正则表达式来对注释进行检测。其中，我们没有直接检测是否存在嵌套注释"/* */"，而是专注于匹配看到的第一个"/*" 和在这之后看到的第一个"*/"。如果存在嵌套注释，比如"/*/* */*/"，则"/*/* */" 会被匹配到，留下单个"*/"，这种情况会被接下来解释的机制匹配到，并报告错误。

如果遇到单个的"/*" 或"*/"，我们称这种情况为"dangling /*" 或"dangling */"。如果是"dangling /*"，则"/*" 之后的所有字符全部被匹配掉（一直到文件末尾），并报告错误。如果是"dangling */"，则只有"*/" 被匹配掉，并报告错误，后续的字符会被正常解析。

1.3 其他错误检查

我们发现，如果想充分利用 error 机制，就必须给 error “留足后路”，向前看符号必须明确，或者 error 所在的上下文必须明确。比如 LP Exp RP，或者 Specifier SEMI 类似这种产生式体部就被定义为 “上下文明确” 或者 “下文明确”。我们知道 Exp 后面必须跟一个有括号，Specifier 后面必须跟一个分号。于是我们可以放心大胆的把 Exp 和 Specifier 替换为 error。即把 LP error RP 和 error SEMI 加入产生式。

但这个时候我们发现，如果把所有“(上) 下文明确”的地方都加上 error，会导致移入/规约冲突或者规约/规约冲突，这可能是上层的 error 包含了下层的 error 导致的。所以我们并不是把每一个“(上) 下文明确”的地方都加上 error，而是在更靠近中间层，并且最有可能出错的地方，加上了 error。

2 如何编译

本实验的编写测试环境，文件夹结构，Makefile 内容均与手册上一致，只需要在 Code 文件夹下使用命令 make，就可以得到一个名为 parser 的可执行文件。输入"./parser testfile.cmm" 即可完成对文件的解析。

3 设计特色

3.1 树结构

我们利用经典的二叉树方式储存多叉树。

此外，在遇到终结符的时候，我们使用了 MAKE_NODE_RETURN 宏，如果今后想要增加树结构中存储的信息，我们不用更改每一条识别终结符的正则表达式后面的动作，只需要在这个宏里面做操作就可以了。比如我想在树的节点中新增加每一个终结符的列位置信息，我们只需要在 _node 数据结构里增加这一条信息，然后更改 make_node 函数就可以了。

在 insert_node 函数中，我们在最开始增加了节点是否为空指针的检查，无论外界怎么折磨 insert_node 函数，都不会出错。

3.2 正则定义

我们在 flex 文件第二部分中的所有正则表达式，均引用在 flex 第一部分中的正则定义，比如 {SEMI} {action}，{INT} {action}。也就是说，今后对正则表达式的修改均可以集中的第一部分，从而避免了第一部分和第二部分混杂的情况。

3.3 树的打印

我们申明了一个变量 is_print_tree，充当一个布尔变量。如果为 1，则打印文法树，如果为零则不打印。在 main.c 文件中，最初 is_print_tree 被设定为 1，在所有错误被检测到的地方被置为 0。在 main.c 文件的最后，我们根据这个变量决定是否最终打印树。

如果实验一结束，我们不再想打印树，可以直接在 main.c 的头部，将 is_print_tree 的初始值置为 0，这样无论如何也不会打印出树来。

4 不足与修改方案

在有限的时间里，我们只草草完成了基本功能，而没有对代码的可维护性做更高的要求。比如在 bison 文件里，每一条产生式后面都拖了一条长长的动作，冗余度极高，我们设法在下一个实验里，进行去冗余操作，使代码可读性，可扩展性更强。

5 致谢

感谢刘春旭和张思拓两位同学，课余饭后的讨论，使得在实验过程中累积的疑惑与不解得到逐一解决。

感谢张天昀学长的博客，在博客上我学到了很多，特别是加 error 的艺术。此外我们在张天昀学长的 error 基础上增加了一些 error，比如 ID LB INT RB 这个地方，我们额外引入了 ID LB error RB，使得在没有引入多余冲突的情况下，error 检查更加完备。