编译原理实验一报告

211220127顾嘉宇

实现功能

词法分析

词法分析借助工具GNU Flex，编写该词法单元对应的正则表达式和匹配的动作，来生成扫描器。

终结符判断

通过定义YYSTYPE将编译器中所有的节点属性更改为自定义的node*类型，而当词法中的单元匹配成功后，通过填写对应的type变量来实现不同类型对应的功能，以方便语法树的构建。

额外选做内容

我的额外任务是要求识别指数形式的浮点数，所以我在lexical.l中填写了EXP_FLOAT与其对应的正则表达式，使其能够成功匹配指数形式的浮点数而不报错。

语法分析

语法分析实现借助工具 GNU Bison，编写文法对应的生成式和对应的动作，来生成语法分析器。

语法树构建

本次实验中，我的语法树构建采用了多叉树的结构，每个树节点都带有一个二级指针型的孩子节点。当匹配了一个合法生成式后，将生成式匹配到的所有节点通过construct_children函数构建成一个二级指针型的变量，并将其变成左值对应节点的孩子节点。对于该函数，由于匹配到的节点是不固定的，所有通过va_list来对可变参数进行处理。最后采用先序遍历的方式即可正确打印语法树，且生成式中所有节点对应的TYPE均为NOT_TEM，即非终结符。

这种实现方式，大大简化了语法树构建的难度，便于实现封装良好的结点构造函数，减少构造结点时的代码冗余，逻辑也更加清晰。

错误恢复

错误恢复采用自顶向下的匹配方式进行考虑。对于 $FunDec$, $ParamDec$ 等涉及范围较小的则交给上层进行错误匹配，如果$ExtDef$等，以尽可能避免错误地移入其它合法语句块的情况。对于 $Stmt$ 这种存在语句块嵌套的文法，则尽可能细致地去匹配。

自顶向下的考虑方式，一方面不用过于思考较小的子文法的可能的错误类型；另一方面，在含有较多语句块的文法进行 $error$ 的匹配，从而使得错误恢复能较快地结束，尽可能少的影响较大的语句块。经过测试，大部分错误恢复都能匹配上，不过根据实验提交结果，仍然不是完全正确。

编译&测试方式

进入 `Code` 文件夹所在路径，执行 `make` 命令，即可获得 **parser** 可执行文件。

执行 `./parser test.cmm` 命令，即可对一个待测试的源文件进行词法和语法的分析。

实验感悟

本实验是编写一个编译器的开头，主要处理了语法和词法分析器，实验的大部分时间主要花在了阅读手册，语法树的构思以及错误恢复的修改和调试上。实验费时较长，也不算顺利，不过总体来说加深了我对词法分析中的有限状态机和语法分析中的LALR文法的理解。