**编译系统实验报告**

**实验1 - 词法分析与语法分析**

|  |
| --- |
| 程序实现的功能及如何实现： |
| 程序主要实现的功能如下：   1. 能够读入程序文本文件，并识别出其中的词法错误与语法错误。 2. 对于程序中的词法错误（错误类型A）：出现C--词法中未定义的字符以及任何不符合C--词法单元定义的字符。在遇到此类错误时，输出报错提示：   Error type A at Line [行号]: [说明文字].   1. 对于程序中的语法错误（错误类型B），在遇到此类错误时，输出报错提示：   Error type B at Line [行号]: [说明文字].   1. 如果程序中没有任何词法或语法错误的输入文件，则构造出先序遍历语法树，打印每一个结点的信息。   具体的实现方法简要介绍如下：   1. 语法分析程序的入口点是yyparse()。当程序调用yyparse( )时,语法分析程序就试图分析输入流，如图所示，展示了读入程序文本的整个过程。 2. 语法分析树节点结构。   设计结构体ASTnode来构建语法分析树。在ASTnode中，储存着当前节点所对应的行数，token类型，节点类型（如果是int，float则保存具体数值）。词法分析过程中，在lab1.l文件中生成叶节点（终结符），语法分析过程中，在lab1.y文件中生成其他节点（非终结符），从而完成AST的构建。   1. AST的构建   首先通过C库宏va\_start()读取参数，然后根据当前父节点的子节点个数及存在情况进行如下不同的操作：若当前节点还有子节点，则通过va\_arg()检索下一个参数，作为第一个子节点，并通过函数setChildTag() 修改相应状态；若当前节点是终结符或者null，则将对应的值存入节点的union联合体中。由于代码篇幅较长，故在此处不作展示。   1. 先序遍历打印节点。   用递归的方法实现递归遍历，在遍历的过程中，输出当前节点的type（由于代码篇幅较长此处不作展示）。   1. 用正则表达式完成不同类型数据的表示。具体如图所示： |
| 如何编译程序： |
| 最终的代码文件如图所示：  其中：  lab1.c, lab1.h是包含AST结构体，函数定义以及部分错误处理的C文件。  lab1.tab.h, lab1.tab.c, lex.yy.c是中间过程生成的文件。  test1\_x.c是测试文件，对应着指导书中的示例1.2~1.4，1.7。  parser 是gcc编译后得到的可执行文件。  执行以下几条命令以编译程序：  bison -d lab1.y  flex lab1.l  gcc lab1.tab.c lab1.c lex.yy.c -lfl -ly -o parser  在本实验中，我将以上命令写入脚本，在Linux下使用时，切入lab1文件夹，直接执行 init.sh 便可完成程序编译。  完成之后，输入 ./parser textfile 便可以用程序对进行词法分析与语法分析。 |
| 程序亮点： |
| 1. 完成了部分附加功能：可以识别出以科学计数法形式表示的float的值。其正则表达式如下：   该部分的测试用例为test1\_7.c（对应指导书的测试用例1.7），运行结果如下： |