编译原理实验报告一

161220085 刘心悦

一、 实验简述

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分析和语法分析，并打印分析结果（语法树）。

本次实验涉及到 3 个文件：lexical.l, syntax.y,和 main.c。依次在控制台输入以下指令即可对测试文件 test.cmm 进行分析:

```
flex lexical.l
bison -d syntax.y
gcc main.c syntax.tab.c -lfl -ly -o parser
./parser test.cmm
```

二、 数据结构

在词法和语法分析过程中，数据以多叉树（用二叉树表示的多叉树）的形式保存在程序中，树中每一个节点 Node 保存一个语法单元。在此数据结构的基础上，实现了 createNode, addChild, addSibling, print 四种基本方法，分别用于新建一个树节点、将一个节点添加为另一节点的子节点、将一个节点添加为另一节点的兄弟节点以及打印语法树。

此部分代码实现在 lexical.l 文件的用户自定义代码部分中。

三、 词法识别

根据正则表达式识别到一个字符串后，会调用相应的 createNode 函数创建一个 Node 节点存放该词法单元的信息，然后将 createNode 函数返回的节点指针通过 yylval 传给 Bison。createNode 函数有多个变体，用于特殊节点的创建。最普通的 createNode 函数有三个参数，分别是词法单元名称 name、行号 lineno 和节点类型 type。type=0 表示终结符号节点，type=1 表示非终结符号节点，type=2 表示下一步推出空串的非终结符号节点。在词法分析过程中创建的节点 type 均为 0。

四、 语法识别

使用产生式进行推导时，先调用 createNode 函数创建一个 type 为 1（若推出空串则 type 为 2）的 Node 节点存放产生式头的信息，然后调用 addChild 和 addSibling 函数将产生式体中的符号对应的节点加为产生式头的子节点。

由于非终结符号的行号不能直接由 yylineno 得到，所以在调用 createNode 函数时行号参数均设为-1。通过观察可知，非终结节点的行号实际上等于其第一个子节点的行号，于是我们可以在 addChild 函数中将非终结节点的行号赋为子节点的行号。这样一来每个非终结符号都得到了正确的行号。

当规约到开始符号 Program 时，如果到此为止都没有出现错误，就调用 print 方法打印语法树。这一部分写在 Program 产生式的语义规则中。

此外，我们还需要解决语法的二义性冲突问题。不过这相当简单，只需要显式地指

定一系列运算符的优先级和结合性即可解决问题。

五、 错误处理

在词法分析过程中，若出现不能被已定义的正则表达式识别的字符串，则会打印 TYPE A 类型错误和行号。在语法分析过程中，我们通过对 yyerror 函数进行重写，在出错时打印 TYPE B 类型错误和行号。

此外，我们还可以通过在产生式中的放置 error 符号来进行语法层面的错误恢复。error 的具体放置位置如下：

```
ExtDefList : Specifier error SEMI
   | error SEMI
FunDec : error LP VarList RP
   | ID LP error RP
   | error LP RP
CompSt : LC DefList error RC
Stmt : error SEMI
   | RETURN error SEMI
   | IF LP error RP Stmt
   | IF LP error RP Stmt ELSE Stmt
   | WHILE LP error RP Stmt
```

如上添加 error 后可以恢复大部分语法错误，发现错误后程序会不断丢弃内容直到找到分号或括号为止。

六、 选做部分

a) 八进制

八进制在计算机上的表示是首位为 0，数字部分为 0~7。所以八进制整数的正则表达式可以写成：0[0-7]+。C 语言中没有将八进制字符串转换为 INT 型的函数，不过实现这个功能也并不麻烦，循环读入数字累加乘 8 即可。具体实现如下：

```
0[0-7]+      {
      int value = 0;
      yytext++;
      while(*yytext != '\0'){
            value = value*8 + *yytext -'0';
            yytext++;
      }
      yylval.n = createINTNode("INT", value, yylineno);
      return INT;
}
```

b) 十六进制

十六进制在计算机上的表示是开头为 0x，数字部分为 0~9A-Fa-f。所以十六进制整数的正则表达式可以写成：0x[0-9A-Fa-f]+。从字符串转换成 INT 的过程和八

进制中的过程类似，具体如下：

```
0x[0-9A-Fa-f]+      {
      int value = 0;
      yytext += 2;
      while(*yytext != '\0'){
            if (*yytext >= '0' && *yytext <= '9')
                  value = value*16 + *yytext -'0';
            else if (*yytext >= 'a' && *yytext <= 'f')
                  value = value*16 + *yytext -'a' + 10;
            else if (*yytext >= 'A' && *yytext <= 'F')
                  value = value*16 + *yytext -'A' + 10;
            yytext++;
      }
      yylval.n = createINTNode("INT", value, yylineno);
      return INT;
}
```

c) 指数形式浮点数

浮点数可以分为两部分：前面的基数部分和后面的指数部分。基数部分有 4 种形式：(1)数字 (2)数字. (3).数字 (4)数字.数字。前 2 种形式可以用{digit}+\.?表达（digit 指[0-9]），后 2 种形式可以用{digit}*\.{digit}+表达，指数部分可以用((e|E)[+-]?{digit}+)?表达。所以浮点数的正则表达式可以写成：

```
(({digit}+\.?)|({digit}*\.{digit}+))((e|E)[+-]?{digit}+)?
```

识别出浮点数字符串后，调用 C 语言自带的 strof 函数将字符串转换成 FLOAT 即可。

d) 注释

注释分 2 种：//…类型注释和/*…*/类型注释。其正则表达式分别为：

```
"//".*
"/*"(.|\n)*"*/"
```

识别出注释后不采取任何操作。