编译原理Lab1实验报告

匡亚明学院
181240035 刘春旭
181240035@smail.nju.edu.cn

实验目标和结果

利用 flex 和 bison 完成对 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。完成了所有基本要求以及附加要求，对学长学姐测试资料中的样例都可以通过；定义了一部分细节化的报错信息。

> 编译方式和原 Makefile 无区别，并在README中有说明。

代码框架

以下将 Code 中的三份代码分开进行细节说明：

- lexical.l：
  1. 定义了语法树中的节点结构 struct Node：

```
struct Node{
    char *token_name; //用于储存节点名称
    struct Node *first_child; //用于多叉树的先序遍历，第一个孩子
    struct Node *next_sib; //用于多叉树的先序遍历，右边的兄弟
    int first_line; // 用于储存节点的第一次出现位置
    int is_token; // 1代表是terminal, 0代表是non-terminal
    union{ // 用于转换INT和FLOAT以及储存ID，使用sscanf转换后储存在这里
      unsigned int int_value; // 4 Byte
      float float_value; // 4 Byte
      char *token_text; // 4 Byte-8Byte
    };
  };
```

2. 处理INT（包括八进制和十六进制）和FLOAT时：
    - 在 token_text 中保留一份原始输入，并用 sscanf() 转换成一个对应的数值，储存在结构体的匿名联合中。打印时打印的是匿名联合中的值。
    - 对于FLOAT，打印时 printf() 打印出的小数默认保留6位，所以过小的小数位数不会显示。
    - ⭐对于八进制和十六进制， 同第一条 ，打印时打印的是转换成十进制后的整数值。
    - ⭐定义了错误的八进制和十六进制格式，可以识别常见的八进制和十六进制数的错误。
3. 处理注释时：
    - 单行注释同书上的匹配方式：使用 input() 函数。
    - ⭐ ~~多行注释目前使用正则表达式，匹配开始和结束的 /*...*/ ，受启发于 这个问答。~~ 目前利用 状态转换 实现嵌套注释的报错。切换的主要原因是实践表示用 input() 读入的EOF字符串无法被识别出来，导致无法实现对未闭合的注释的处理（直接忽略后续所有内容）。当然也可以写一个更复杂的正则表达式来吃掉后续所有内容，但是感觉可能还是状态更简洁一些。
    - ⭐对未闭合的多行注释 [*][/] ，用正则表达式匹配，错误类型为 语法错误 （Error type B），若只有前半边 "/*" ，则通过状态定义。
4. 进行 yylex() 时，遇到 lexical.l 中定义的符号时的动作是创建一个树节点，将指针存储在 yylval 中，它会在 syntax.y 中被利用。

- syntax.y
    1. 定义了添加节点的函数 insert_node(char *node_name, struct YYLTYPE node_loc, int num_nodes, ...)
        - 使用 va_list 实现变长参数列表。注意 va_arg() 只进行指针的加减操作，不进行类型正确性的判断。（其中一次 segmentation fault 的根本原因）
    2. 定义了先序遍历的函数 void print_tree(struct Node* cur, int depth)
        - 省略具体打印代码。先序遍历过程为：

```
if(cur->first_child!=NULL)  print_tree(cur->first_child, depth+1);
if(cur->next_sib!=NULL)   print_tree(cur->next_sib, depth);
```

    3. 错误恢复：本实验最痛苦的部分，来回修改了十几天。

       定义了自己的 print_error() 函数，并且禁止 yyerror 打印任何内容，这样就可以输出自定义化的错误信息，也有助于debug.

⭐总结出的经验：

1. error要尽可能放在上下文语义明确的位置，尽可能让它能被可识别出的语法单元包围。否则可能会“意外地”出现在其他语法单元的FOLLOW集中，导致移入规约出现问题。最佳示例：`Exp : ID LP error RP {print_error("Wrong Exp");}`
2. 如果无法做到上下文语义明确，至少一边的语义要明确，让它能尽量恢复。
3. ⭐出现不能恢复的问题怎么办：至少要打印出这一行的行号和错误信息。所以在`yyerror()`中定义一个计数器`prod_err`，在自己定义的`print_error()`中定义一个计数器`handled_err`，每次恢复成功都会调用`print_error()`来打印错误信息，并且将这两个计数器同步，最后如果`yyparse()`提前返回，并且这两个计数器的个数不等，那说明目前遇到了不能恢复的错误，此时进行打印。

- `main.c`

1. 在`lexcial.l`和`syntax.y`中都分别定义了记录错误个数的计数器，只有两个计数器的和为0时才打印树的信息。
2. 在`syntax.y`中定义了一个`yyparse()`的wrapper function，在`main.c`里调用，可以使用`make debug`命令得到带bison状态机调试输出的可执行文件。
3. 之前定义过的`#YYABORT`，它可以让`yyparse()`在出错后立刻返回，在调试过程中帮到了忙，但本次实验结果中不需要。

测试结果

非常感谢学长学姐们的测试代码，帮助我找到了很多bug。最终的结果是基本全部可以通过。这里的“基本”是指除了两个仅由注释组成的文件中，我的打印结果是`Program(1)`，但是给出的标准答案是`Program(-1)`（-1代表文件最后一行），没搞懂为什么，所以就此作罢。