编译原理实验报告

刘驭壬
匡亚明学院
141242026
141242026@smail.nju.edu.cn

实验进度：

我完成了：
1.词法分析
2.语法分析
3.识别八进制十六进制数
4.识别科学技术法表示的浮点数
git可查，完整记录了实验的全过程。

运行：

进入src文件夹
make: 编译得到parser可执行文件（位于根目录）
make test: 测试根目录test文件夹上的测试用例
make clean: 删除中间文件和parser可执行文件
最后make的时候会报warning，不影响运行。

词法分析：

内容在lexical.l中，正则表达式等没有什么需要特别说明的，见代码。
该文件中定义了一个函数: void procToken(char *symbol);
该函数的作用是为每个识别的词法元素构造一个节点，该节点是语法树多叉树的叶结点。

语法分析：

内容在syntax.y中.
该文件中定义了一个函数：TreeNode *procTreeNode(TreeNode *p, char *symbol);
该函数的主要任务是为语法分析过程中的每个非终结符号构造一个节点，该节点是语法树多叉树的内部节点。

语法树：

1.语法树的数据结构(include/syntax_tree.h)：
(arity属性表示有多少个子节点，eg: arity = 5时，有5个子节点)

```
typedef struct TreeNode{
        char symbol[kMaxLen], text[kMaxLen];
        int lineno;
        int arity;
        union{
                int int_value;
                float float_value;
        };
        struct TreeNode *children[kMaxChildren];

}TreeNode;
```

2.语法树的相关函数：
函数位于src/syntax_tree.c中，主要有：
(1)TreeNode *createTreeNode(int arity, …);
(2)void printTree(TreeNode *p, int depth);
(3)void deleteTreeNode(TreeNode *p);

(1):负责新建一个节点，arity表示子节点数量，可变数量的参数分别表示每个子节点。我觉得可变的参数是我设计中比较重要的地方。
(2)负责递归打印语法树，当词法分析语法分析均不出错时被调用。
(3)负责递归删除整棵语法树。程序结束时被调用。

3.语法树的构造过程：
1.词法分析过程中的procToken函数会在识别一个词法单元后构造一个语法树的叶子节点，并传递给语法分析程序。
2.语法分析程序中的procTreeNode函数会在推导(自底向上或自顶向下)过程中为非终结符号构造语法树的中间节点。
3.如果词法分析语法分析均不出错，会构造并打印出语法树，否则会指出出错原因和位置。

出错恢复：

我觉的这部分的内容不能保证完全正确，因为error到底加在哪里，是我靠自己判断大概加的，主要位置是在所有右括号(]}))前加error，分号前加error。
对于出错恢复，主要做的事是：

```
yyerrok;
errorState = true;
$$ = procTreeNode(createTreeNode(0), "Stmt");
```

如果不加yyerrok，当bison发现一个错误后会被阻塞住，无法识别更多错误。

实验过程中遇到的坑：

在关于确定语法单元位置中。实验指导手册上面有这样的话，很有误导性，以为应该都加在.l中。

```
%locations
…
%{
  /* 此处省略#include部分 */
  int yycolumn = 1;
  #define YY_USER_ACTION \
    yylloc.first_line = yylloc.last_line = yylineno; \
    yylloc.first_column = yycolumn; \
    yylloc.last_column = yycolumn + yyleng - 1; \
    yycolumn += yyleng;
%}
```

实际上应该%locations在.y文件中，其他在.l文件中。