# 编译原理实验一: 词法分析与语法分析

1190200708 熊峰

2022年3月23日

## 1 程序的功能及实现

## 1.1 程序的功能

使用 flex 和 bison 实现 C-语言的此法和语法分析,并构造分析树。

## 1.2 程序的实现

所设置的节点的结构体如下:

```
typedef struct treeNode{
    // 词法单位的行
    int lineNo;
    // 词法单位的类型
    NodeType nodeType;
    // 词法单位的名称
    char* name;
    // 词法单位若为终结符则保存他的内容
    char* value;
    // 非终结符的孩子节点
    struct treeNode* childNode;
    // 非终结符的下一个兄弟节点
    struct treeNode* brotherNode;
}Node;
```

#### 1.2.1 节点相关操作

- 1) 创建节点,并根据 value 的值区分是否为终结符。
- 2) 插入节点,首先检查父节点是否存在子节点,若不存在子节点,则将 第二个参数的节点插入为父节点的子节点,若其存在子节点,则找到 父节点的子节点,并不断指向兄弟节点,直到其为空,将第二个参数 的节点插入为最后一个兄弟节点的兄弟节点。

- 3) 删除语法树,释放内存。
- 4) 根据层数打印语法树,确定节点类型,只有为 TOKEN\_INT,TOKEN\_FLOAT,TOKEN ID,TOKEN TYPE, 的时候才会打印他的值。

### 1.2.2 词法分析

- 1) 根据书后产生式,书写正则表达式,并使用别名命名,方便进一步的正则表达式的识别与匹配。
- 2) 完成正则表达式匹配后的动作,即需要建立叶子节点。
- 3) 根据示例中的错误进一步完善词法分析的功能,识别出更多的词法错误。

#### 1.2.3 语法分析

- 1) 首先在定义部分定义终结符与非终结符,并设置左右优先级。
- 2) 在规则段根据书后相关产生式书写相关规则,并根据规则中使用的符号构建语法树。

若产生式为 ExtDefList -> ExtDef ExtDefList 则需要创建 ExtDefList 的节点,并将 \$1 插入到 ExtDefList 的子节点中,接下来将 \$2 插入 \$1 的兄弟节点。

3) 在第三段中,声明了错误恢复的方法,由于 yacc 的错误恢复的特性,实现的方法可以指出语法错误的位置及原因。

#### 1.3 优点

#### 1.3.1 语法树的构建

通过先构建父节点再构建叶节点的方式,使整体脉络更加清晰。如代码 所示:

```
 \begin{array}{l} p = CreateNode(@\$.first\_line\;,\;NON\_TOKEN,\;\;"ExtDefList"\;,\;NULL);\\ InsertNode(p,\;\$1);\\ InsertNode(p,\;\$2);\\ \$\$ = p; \end{array}
```

### 1.3.2 打印语法分析树

引入枚举类型,为每一个节点分配类型,最终打印语法树时候,可以判断他的类型,只有在 int、float、type、id 时候才会打印。

2 如何编译程序 3

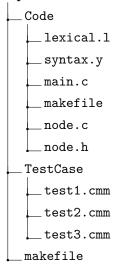
## 1.3.3 增强词法分析

根据书后示例,进一步完善了词法分析过程,使程序能够识别十六进制及八进制的词法错误。

## 2 如何编译程序

实验的根目录下有 makefile 文件。 实验的内容的目录结构如下:

## Experiment1



其中在 Experiment 根目录下的./makefile 会与./Code/makefile 嵌套调用。

本次实验中,所有指令在根目录执行即可,指令如下:

- 1) 编译./Code 下的所有代码,并生成可执行文件./Code/parser.
  - \$make compile
- 2) 清除./Code 下所有编译产物。

\$make clean

3) 测试示例。

\$make nessaryTest

位于实验目录的根目录执行上述目录,即可完成测试。