# MiniDecaf Parser Stage Report

梁业升 2019010547（计 03）

2022 年 11 月 27 日

## 1 实验内容

本次实验比较简单，根据文法实现相应的递归下降算法即可。

大部分的补全部分是基于 LL(1) 文法的递归下降，直接按照文法递归地调用对应的解析函数即可，如：

```
if (next_token.type == TokenType::RETURN) {
    return p_Return();
} else if (next_token.type == TokenType::IF) {
    return p_If();
} else if (next_token.type == TokenType::LPAREN) {
    lookahead(TokenType::LPAREN);
    ast::Expr *expr = p_Expression();
    lookahead(TokenType::RPAREN);
    lookahead(TokenType::SEMICOLON);
    return new ast::ExprStmt(expr, expr->getLocation());
} else if (next_token.type == TokenType::LBRACE) {
    return p_Block();
} else if (next_token.type == TokenType::SEMICOLON) {
    lookahead(TokenType::SEMICOLON);
    return new ast::EmptyStmt(next_token.loc);
}
```

而对于包含左递归的文法，根据提示，转换为 EBNF 文法，并使用 `while` 循环进行解析：

```
while (next_token.type == TokenType::OR) {
    Token Or = lookahead();
    ast::Expr *operand2 = p_LogicalAnd();
    node = new ast::OrExpr(node, operand2, Or.loc);
}
```

所有的待填空部分均为这两种情况之一，在代码中均有体现，在此不再赘述。

## 2 思考题

1. 原文法：

```
additive : additive '+' multiplicative
         | additive '-' multiplicative
         | multiplicative
```

不含左递归的文法：

```
additive : multiplicative A
A        : '+' multiplicative A
         | '-' multiplicative A
         | epsilon
```

2. 在递归解析某一条文法规则时，如果下一个 token 匹配错误，则继续消耗 token，直到遇到文法左侧非终结符的同步集合中的元素，则视为开始下一个文法规则的匹配。

例如：

```
int a = ;
int b = 0;
```

第一行对应的文法为：

```
DeclStmt : Type IDENTIFIER ASSIGN Expr SEMICOLON
```

`DeclStmt` 同步集合中包含关键字 `int`，因此遇到第二行的 `int` 时即视为错误已解决。