parser-stage 实验报告

计 93 王哲凡 2019011200

实验内容

1. `p_relational`：
   - 类似于 `p_equality`，等价 EBNF 为：

     ```
     relational: additive { '<' additive | '>' additive | '<=' additive | '>=' additive}
     ```

   - 对应修改即可。
2. `p_logical_and`：
   - 类似于 `p_logical_or`，等价 EBNF 为：

     ```
     logical_and: equality { '&&' equality }
     ```

   - 对应修改即可。
3. `p_assignment`：
   - 首先通过 `lookahead` 匹配 `Assign`。
   - 通过 `p_expression` 解析得到赋值表达式的右侧。
   - 借此二者构造 `Assignment` 结点并返回。
4. `p_expression`：
   - 直接返回 `p_assignment` 调用结果即可。
5. `p_statement`：
   - 前面已经处理了非 `if/return` 的情况。
   - 如果 `self.next` 对应 `If`，则返回 `p_if` 调用结果。
   - 如果 `self.next` 对应 `Return`，则返回 `p_return` 调用结果。
   - 否则根据 `next_token` 报错 `DecafSyntaxError`。
6. `p_declaration`：
   - 前面已经解析了 `type` 和 `Identifier`。
   - 后续判断如果存在 `Assignment` 即定义包含初始化：
     - 首先通过 `lookahead` 匹配 `Assign`。
     - 调用 `p_expression` 解析得到初始化表达式，并赋值给 `decl.init_expr`。
7. `p_block_item`：
   - 如果 `self.next` 对应 `p_statement` 的 First 集合，则调用 `p_statement` 并返回。
   - 如果 `self.next` 对应 `p_declaration` 的 First 集合，则调用 `p_declaration` 并返回。
8. `p_if`：
   - 首先通过 `lookahead` 匹配 `If` 和 `LParen`。
   - 调用 `p_expression` 解析 `cond` 部分。
   - 通过 `lookahead` 匹配 `RParen`。
   - 调用 `p_statement` 解析 `body`（此时还没有块语句）。
   - 根据 `cond` 和 `body` 构建 `If` 结点。

- 如果 `self.next` 对应 `Else` 则匹配 `Else`，并将新构建的 `If` 结点的 `otherwise` 赋值为 `p_statement` 的调用结果。
- 返回 `If` 结点。

9. `p_return`：
   - 首先通过 `lookahead` 匹配 `Return`。
   - 调用 `p_expression` 解析返回表达式。
   - 通过 `lookahead` 匹配 `Semi`。
   - 根据解析的返回表达式构建 `Return` 结点并返回。
10. `p_type`：
    - 通过 `lookahead` 匹配 `Int`。
    - 构建 `TInt` 结点并返回。

思考题

第一题

可转换为：

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

第二题

举例：考虑以下错误代码：

```
int main() {
    int a = 1;
    if (a == 0)
        a = a + 1)
    else
        a = a - 1;
    return 0;
}
```

在对于 `if` 的 `body` 的解析中，会根据解析进程进入到 `additive` 的解析中。

当匹配完成第一个 `+` 和 `1` 时，发现匹配失败，此时的 $EndSym$ 集合应该至少包含 `else`、`;`、`+`、`-` 等符号，$BeginSym$ 则包含 `-`、`!`、`~`、`(`、`Identifier`、`Integer`。

此时需要跳过不在 $S = EndSym \cup BeginSym$ 中的 `)`。

在遇到下一个符号即 `else` 时，停止跳过，继续解析。

之后框架应该可以完成 `if` 的 `otherwise` 部分解析并进而完成 `program` 的完成解析。

第三题

我认为框架中的 TODO 部分提示对完成实验十分有帮助。

我认为可以考虑适当引入简单的错误恢复机制以助于同学在实验中更好理解。