MiniDecaf Parser-Stage

> My-laniaKeA

实验内容

按照实验框架提示，实现以下函数：

```
p_Type p_StmtList p_Statement p_VarDecl p_Return p_If
p_Expression  p_Assignment p_LogicalAnd p_Relational
```

以不同返回类型为例，简要介绍实现思路：

ast::Type

首先，调用 `lookahead(TokenType t)` 匹配 token。其次，在 AST 中新建 Type 对应的节点 `IntType`。最后，返回该节点。

```
static ast::Type* p_Type(){
    Token int_token = lookahead(TokenType::INT);
    return new ast::IntType(int_token.loc);
}
```

ast::Expr, ast::Statement

通用的思路：根据产生式，遇到终结符则 lookahead，遇到非终结符则 parse，最后创建对应的 AST 节点。如果需要讨论多个产生式的情况，则通过 `next_node.type` 查看，不消耗终结符。

```
static ast::Expr* p_Assignment(){
  /*    assignment : Identifier '=' expression
                   | conditional              */
  ast::Expr* node = p_Conditional();
  if(next_token.type == TokenType::ASSIGN){
        // Match token
        Token assign = lookahead();
        // Parse
        ast::Expr* rhs = p_Expression();
        // Build and return
        ast::LvalueExpr* node_new = dynamic_cast<ast::LvalueExpr*> (node);
        return new ast::AssignExpr(node_new, rhs, assign.loc);
  }
```

ast::StmtList

大体思路与前述类似，区别在于：

- 用 First 和 Follow 集合判断合法性。
- parse 非终结符后，添加到 statement list 中。

思考题

1. 在框架里我们使用 EBNF 处理了 `additive` 的产生式。请使用课上学习的消除左递归、消除左公因子的方法，将其转换为不含左递归的 LL(1) 文法。（不考虑后续 multiplicative 的产生式）

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

消除左递归：

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

不需要消除左公因子。

2. 对于我们的程序框架，在自顶向下语法分析的过程中，如果出现一个语法错误，可以进行**错误恢复**以继续解析，从而继续解析程序中后续的语法单元。 请尝试举出一个出错程序的例子，结合我们的程序框架，描述你心目中的错误恢复机制对这个例子，怎样越过出错的位置继续解析。（注意目前框架里是没有错误恢复机制的。）

出错程序：

```
int main() {
    int x = 1;
    if (x) x = 2
    return x;
}
```

采用短语层恢复。

对于产生式

```
if        : 'if' '(' expression ')' statement ( 'else' statement )?
statement : 'if' | 'return' | ( expression )? ';' | '{' StmtList '}'
```

对于 `x = 2` 调用 `p_Statement()`：

```
static ast::Statement* p_Statement(){
  /*  statement : if | return | ( expression )? ';' | '{' StmtList '}' */
  if(isFirst[SymbolType::Expression][next_token.type]){
    ast::Expr* stmt_as_expr = p_Expression();
    lookahead(TokenType::SEMICOLON);
    return new ast::ExprStmt(stmt_as_expr,stmt_as_expr->getLocation());
  }else{
      // ...
  }
}
```

`lookahead(TokenType::SEMICOLON);` 会报错，因为此时 next token 是 `return`。此时跳过所有 $First(statement) \cup EndSym$ 之外的单词。由于 `retun` 在 $First(statement)$之中，所以解析程序从 `return x;` 继续分析。

3. 指出你认为的本阶段的实验框架/实验设计的可取之处、不足之处、或可改进的地方。

   `p_Assignment()` 的 hint 部分有些令人疑惑：

   ```
   /*
       * Hint: `AssignStmt` node need a `VarRef` node as its child,
         [not necessiarily `VarRef` (`Lvalue *left` in the original
   framework).]
       * but p_Conditional() returns a `LValueExpr` node here,
         [returns a `Expr` instead of `LvalueExpr`.]
       * hence some conversion is needed (`Expr` to `LvalueExpr` to `VarRef`)
       * consider dynamic_cast<> here
   **/
   ```

代码借鉴

无。