PA1-A 实验报告

本次实验使用的是 Rust 框架，今后也将会使用 Rust 框架完成其他 PA。

新特性 1：抽象类

对于抽象类及其方法，我在 ClassDef 和 FuncDef 增加了一个 bool 属性 abstract_ 用来记录是否是抽象类，并直接重写了 ClassDef 的 AST 格式化输出。

由于抽象类的方法是不包含 block 的，需要套上 Option<>。

新特性 2：局部类型推断

只需要新增一个 `Simple -> Var Id Assign Expr` 的语言规范即可。

var 变量的类型为 Null，故需要将 VarDef 的 syn_ty 修改成 Option<SynTy<'a>>。

新特性 3：First-class Functions

函数类型

新增语言规范和关键字 `Var` 即可：

- `Type -> Type LPar TypeListOrEmpty RPar`
- `TypeListOrEmpty -> TypeList`
- `TypeListOrEmpty ->`
- `TypeList -> TypeList Comma Type`
- `TypeList -> Type`

Lambda 表达式

一开始我是将 Expression Lambda 和 Block Lambda 分成两个不同的类来处理，但这样就需要重写很多 AST 格式化输出的代码，于是就顺着 result 的思路将两者结合成 Lambda 类。

Lambda 类包含 param 和 body，前者表示参数，类型为 `Vec<&'a VarDef<'a>>`，后者可能为 Expression 或 Block，这里使用 `enum Either<L, R>` 实现，最后将其加进 ExprKind 内。

AST 格式化输出需要新建一个 TLambda 并加到 SynTyKind，`=>` 的优先级设为最低。

函数调用

直接修改原语法规范虽然能 Pass，但会提示 Shift-Reduce Conflict，需要设定左括号的优先级，最后确定跟右括号同优先级。（但并不知道其正确的优先级是哪个，testcase 并不能体现）

思考题

Q1：有一部分 AST 结点是枚举类型。若结点 B 是枚举类型，结点 A 是它的一个 variant，那么语法上会不会 A 和 B 有什么关系?

有关系。例如 `enum Expr { Binary(op, lhs, rhs) }`，这就说明 Binary operation 是一种 Expr。

Q2：原有框架是如何解决空悬 else (dangling-else) 问题的?

原框架给规范 `MaybeElse ->` 定义了规约优先级 `Empty`，优先级中 `Empty` 低于 `Else`，故不会发生 dangling-else 问题。

Q3：输入程序 lex 完得到一个终结符序列，然后构建出具体语法树，最后从具体语法树构建抽象语法树。 这个概念模型与框架的实现有什么区别？我们的具体语法树在哪里？

区别在于框架没有直接实现具体语法树，而是在类与类之间的包含中隐含了具体语法树的信息。