HW5 报告

胥昊天

git graph

具体实现详见代码仓库

```
(base) keats@OMEN-Yanxu:~/FudanCompilerH2025/HW5$ git log --graph
* commit 30b23dfbd8e2af7d925e0602e409aac4c499852c (HEAD -> main, origin/main)
| Author: xht03 <1620318777@qq.com>
| Date:   Thu Apr 17 20:22:51 2025 +0800
|
|     HW5 done.
|
* commit 931c1429bd871afca77efebe29cd9b61e6955d24
| Author: xht03 <1620318777@qq.com>
| Date:   Thu Apr 17 18:57:16 2025 +0800
|
|     finish all for debug.
|
* commit ee043c32c56431cefc132afc7c4bb3a9f15b6e7e
| Author: xht03 <1620318777@qq.com>
| Date:   Wed Apr 16 21:08:11 2025 +0800
|
|     prepare to upgrade
|
* commit 29bc62630cb238f7cb32994a530fb6ce2bd08d43
| Author: xht03 <1620318777@qq.com>
| Date:   Wed Apr 16 20:06:00 2025 +0800
|
|     backup for VarDecl
|
* commit 11b70ef85a419cd60531947017198b0850fce509
| Author: xht03 <1620318777@qq.com>
| Date:   Thu Apr 10 23:27:51 2025 +0800
|
|     seemingly completed without bug.
|
* commit 561c881021fd1284f05f00df4b76503ac5a6f386
| Author: xht03 <1620318777@qq.com>
| Date:   Thu Apr 10 23:11:00 2025 +0800
|
|     If not support nested If.
```

Figure 1: alt text

思考题

Q1.1: `treep.hh` 中 tigerirp 的 class 作用

- `tree::ExpStm`：将 `tree::Exp` 转换为 `tree::Stm`，忽略表达式的返回值
- `tree::Label`：作为程序跳转的目标，类似于 goto 标签
- `tree::Jump`：无条件跳转到指定的标签位置
- `tree::CJump`：条件跳转，根据比较结果决定跳转到 true 标签还是 false 标签
- `tree::Seq`：顺序执行多个语句，将多个 `tree::Stm` 组合成一个语句序列
- `tree::Move`：赋值操作，将右侧表达式的值赋给左侧
- `tree::TempExp`：将临时变量 `tree::Temp` 转换为表达式 `tree::Exp`
- `tree::Const`：将整数常量转换为表达式 `tree::Exp`
- `tree::BinOp`：表示二元运算操作，如加减乘除等
- `tree::Mem`：内存访问操作，用于读取或写入内存位置
- `tree::Call`：函数调用表达式，包含函数名和参数列表
- `tree::ESeq`：先执行一个语句，然后求值一个表达式并返回其结果

Q1.2: Tiger IR+ 相对 Tiger IR 的扩展内容

Tiger IR+ 相较于原始 Tiger IR 增加了以下内容：

1. **面向对象支持**：增加了与类和对象操作相关的指令，支持类的实例化、方法调用和字段访问
2. **数组操作扩展**：添加了更全面的数组操作支持，包括数组初始化、访问和长度获取
3. **统一对象记录 (UOR)**：为支持多态和类层次结构，引入了统一对象记录结构
4. **方法重命名机制**：为处理类方法，添加了方法重命名的功能

这些扩展是必要的，因为 FDMJ 语言与 Tiger 语言有本质区别：FDMJ 以语句 (stm) 为主体，而 Tiger 以表达式 (exp) 为主体。同时，FDMJ 支持面向对象特性，需要相应的 IR 指令来表示类和对象的操作。

Q2: 不带 class 时的翻译方法

- **算术运算**：通过 `tree::BinOp` 实现，根据不同操作符选择对应的操作类型
- **比较运算**：使用 `tree::CJump` 实现，生成条件跳转代码，结合临时变量存储结果
- **逻辑运算**：
  - 对于 AND：使用短路求值，先计算左操作数，若为 false 则直接返回 false

  - 对于 OR：同样使用短路求值，先计算左操作数，若为 true 则直接返回 true

- **赋值**：使用 `tree::Move` 将右侧表达式结果赋值给左侧变量
- **条件语句**：
  - if 语句：使用 `tree::CJump` 实现条件判断，生成 true 和 false 分支的代码
  - while 语句：生成循环开始标签、条件判断、循环体和循环结束标签，使用 `tree::Jump` 回到循环开始
- **数组操作**：
  - 初始化：使用内存分配函数，在-1 位置存储数组长度，然后对各元素位置赋值
  - 访问：计算内存偏移量 (index+1)，通过 `tree::Mem` 访问指定位置
  - 长度获取：通过访问数组-1 位置获得长度信息

Q3: 带 class 时的翻译方法

- **方法重命名**：采用”类名 _ 方法名”的形式重命名方法，确保不同类的同名方法不冲突
- **参数列表处理**：
  - main 方法：只包含声明的参数
  - 类方法：在参数列表前添加隐含的 `this` 参数，指向当前对象实例
- **this 处理**：将 `this` 作为方法的第一个参数传入，通过它访问对象的字段和方法
- **Unified Object Record**：
  - 为每个类创建统一对象记录结构，记录类的所有字段和方法
  - 在对象实例化时，分配内存并按 UOR 结构初始化对象
  - 字段按偏移量存储，方法表存储在对象头部
- **多态实现**：
  - 通过虚方法表实现动态绑定
  - 子类继承父类的 UOR 结构，并可以重写方法表中的方法指针
- **类操作翻译**：
  - 实例化：分配内存，初始化字段和方法表
  - 字段访问：通过 `this` 指针加上字段偏移量计算地址，使用 `tree::Mem` 访问

- 方法调用：通过 `this` 指针获取方法表，查找方法地址，使用 `tree::Call` 调用方法