COOL 语言代码生成器开发报告

Compiler Principle Assignment PA5

姓名: 谢俊
学号: 20238131026
班级: 编译原理课程

2025 年 12 月 20 日

摘要

本实验在已完成的 COOL 词法分析、语法分析和语义分析的基础上，实现了 PA5 要求的代码生成器（Code Generator）。代码生成器以语义分析阶段输出的带类型标注的抽象语法树（AST）为输入，生成可在 SPIM 模拟器上运行的 MIPS 汇编代码。实验在不修改框架核心文件的前提下，主要扩展并实现了 `cgen.cc`、`cgen.h`、`cool-tree.h`、`cool-tree.handcode.h` 中的类表构建、运行时布局、环境管理以及各类表达式节点的代码生成逻辑。通过与官方编译器 `coolc` 生成的汇编在 SPIM 上对比测试，验证了实现的正确性与与官方实现的一致性。

1 评分项映射

2 项目概述与环境

2.1 项目目标

在 COOL 编译器框架下，实现一个与官方编译器 `coolc` 输出等价的代码生成器，具体目标包括：

- 构建完整的类代码生成表（`CgenClassTable`），分配类标签并建立继承树；
- 为每个类生成类名表（`class_nameTab`）、对象表（`class_objTab`）、分发表（dispatch table）和原型对象（protObj）；
- 为每个类生成初始化方法（`Class_init`）和方法实现代码（`Class.method`）；
- 为各类表达式（赋值、方法调用、条件、循环、case、算术与逻辑运算等）生成符合 COOL 运行时约定的 MIPS 汇编；
- 正确处理 `SELF_TYPE`、`isvoid`、`new` 等特殊语义；
- 通过 SPIM 对比测试，验证与官方编译器在行为上的一致性。

2.2 开发与运行环境

- 操作系统: Windows 主机 + VMware 虚拟机中的 Ubuntu
- 编译器: `g++`（C++11）、`make`
- 工具: 课程提供的 `lexer`、`parser`、`semant`、`coolc`、`spim/xspim`
- 工程目录: `~/student-dist/assignments/PA5`

2.3 核心文件

本次实验仅修改以下四个文件：

- `cool-tree.handcode.h`
- `cool-tree.h`
- `cgen.h`
- `cgen.cc`

3 代码生成设计概述

本节简要说明运行时对象布局、类标签分配、Environment 设计以及表达式代码生成的总体思路，内容与课程 PA5 指南一致。

表 1: 评分项与实现映射（代码生成实验）

类别	覆盖点	分值
代码生成原理	目标机模型、调用约定、对象布局、运行时系统接口理解	20
类与对象布局	类标签分配、继承树、类名表、对象表、原型对象布局	20
调度表与方法调用	分发表构建、静态/动态派发、SELF_TYPE 处理	20
表达式代码生成	赋值、分派、条件/循环、case、算术与逻辑运算等	20
环境与作用域管理	Environment 设计、本地变量/参数/属性访问、作用域处理	10
测试与集成	与官方 `coolc` 输出对比、SPIM 运行测试、错误定位	10
报告与代码质量	结构清晰、逻辑完整、实现与设计一致性	10
主观评分与查重	完整性、独立实现程度与可读性	10
	总分　110（可按课程标准折算为 100）	

3.1 运行时对象布局

在本次实现中，所有 COOL 对象遵循统一的内存布局：

- 第 0 个字：值为 `-1` 的 eye-catcher；
- 第 1 个字：类标签 `class_tag`，用于 `case` 匹配与动态类型判断；
- 第 2 个字：对象大小（以 word 为单位），包括头部和所有属性；
- 第 3 个字：指向分发表（dispatch table）的指针；
- 之后若干字：按继承顺序排列的属性值。

每个类对应一个原型对象 `Class_protObj`，对象表 `class_objTab` 中为每个类保存 `protObj` 和 `init` 方法地址，`new` 表达式通过复制 `protObj` 并调用对应 `init` 完成对象创建。

3.2 寄存器与调用约定

遵循 COOL 运行时约定：

- `$a0`（ACC）：保存当前表达式求值结果；
- `$s0`（SELF）：保存当前 `self` 对象；
- `$fp`（FP）：帧指针，指向保存旧 `FP` 的位置；
- `$sp`（SP）：栈顶指针，栈向低地址增长；
- `$ra`（RA）：返回地址。

方法调用时，调用者负责把实际参数压栈，被调用者在栈上为 `FP`、`SELF` 和 `RA` 预留 3 个字的空间，返回时恢复寄存器并弹出参数。

4 关键实现

本节从 Environment、CgenClassTable、CgenNode 和典型表达式几方面，说明具体实现要点，并结合必要的代码片段进行解释。

4.1 各文件修改内容概览

- `cool-tree.handcode.h`：为表达式节点增加类型字段和统一的 `code` 接口，使语义分析阶段的类型信息可以在代码生成阶段复用；
- `cool-tree.h`：确认 `method_class`、`attr_class`、`branch_class` 等节点的成

员字段（如 name、type_decl、expr）与代码生成访问方式一致；

- cgen.h：新增 Environment 类，并扩展 CgenClassTable 与 CgenNode，增加类标签、方法表和属性表等字段；
- cgen.cc：从框架骨架出发，完整实现常量区生成、类表和对象表生成、分发表和原型对象生成，以及所有表达式节点的 code 方法。

4.2 Environment：环境与作用域管理

环境类 Environment 在 cgen.h 与 cgen.cc 中实现，负责在代码生成阶段维护局部变量、形式参数和属性的访问信息：

- 使用 vars 向量按栈顺序保存所有当前作用域内的 let 变量，scope_lengths 用于记录进入作用域前的栈深度，在退出作用域时恢复；
- 使用 formals 保存当前方法的形式参数顺序，便于根据参数索引计算相对于 FP 的偏移；
- 成员 cur_class 指向当前生成代码的类节点 CgenNode，用于访问属性偏移和类标签；
- LookUpVar 从最近作用域向外查找变量在线性表中的位置，换算为相对 SP 的偏移；
- LookUpParam 在 formals 中顺序查找参数位置，换算为相对 FP 的偏移；
- LookUpAttrib 调用当前类节点的 get_attrib_offset，将属性名映射为对象内偏移。

通过 Environment，所有访问变量/参数/属性的表达式都可以使用统一接口，而不必自行计算偏移，降低错误概率。

4.3 CgenClassTable：类表与全局代码生成

CgenClassTable 继承自符号表模板，负责收集所有 CgenNode 并驱动整个代码生成过程：

- 在构造函数中调用 install_basic_classes 安装 Object，IO，Int，Bool，Str 等基本类，再通过 install_classes 将用户类包装为 CgenNode；
- 通过 build_inheritance_tree 与 set_relations 建立父子指针；
- 在 code() 中依次调用 code_global_data、code_select_gc、code_constants、code_class_nameTab、code_class_objTab、code_dispatchTabs、code_protObjs、code_global_text、code_class_inits 和 code_class_methods，完成从数据段到文本段的全部输出；
- 通过 probe 接口可以按类名查找对应 CgenNode，用于 dispatch、static_dispatch、case 和 new SELF_TYPE 等表达式中查询类信息。

4.4 CgenNode：类标签、属性与方法布局

CgenNode 继承自 class__class，用于给每个类附加代码生成相关信息：

- 成员 inheritance_path 保存从根类到当前类的路径，用于计算 case 表达式中类型深度；
- 成员 methods 和 attribs 分别记录方法表和属性表，方法条目为“方法名 + 定义该方法的类名”，属性条目为“属性名 + 类型”；
- 成员 class_tag 和 max_child_tag 记录当前类和其子树的标签区间；
- setup 从根节点出发深度优先遍历继承树，给每个类分配标签，并继承父类的方法表和属性表，在遇到重写方法时更新定义类信息；
- code_protobj 根据属性表生成原型对象，自动填充整数 0、空字符串和布尔常量等属性默认值；
- code_init 为每个类生成初始化方法：先调用父类初始化，再对本类带初始表达式的属性执行赋值；
- code_methods 为每个用户类的方法生成 MIPS 代码，实现标准的栈帧搭建与恢复。

4.5 典型表达式的代码生成

本实验为所有表达式节点实现了 `code(ostream&, Environment)` 方法。下面对若干代表性表达式进行说明。

4.5.1 赋值与变量引用

- `assign_class::code` 首先对右侧表达式求值，结果存放在 `ACC` 中，然后依次尝试按 `Environment` 查找 let 变量、形式参数和属性，将结果存入相应位置，如果启用 GC，属性赋值后还会调用写屏障更新引用；
- `object_class::code` 对于 `self` 直接返回 `SELF`,否则同样通过 `LookUpVar`、`LookUpParam` 和 `LookUpAttrib` 从栈、帧和对象中加载值。

4.5.2 静态与动态方法调用

- `dispatch_class::code` 先按从左到右顺序计算实参并依次压栈，再对接收者表达式求值并检查是否为 void，随后从对象头中取出分发表指针，根据当前表达式的静态类型和方法名在 `CgenNode` 中查找方法偏移，加载函数地址并通过 `jalr` 调用；
- `static_dispatch_class::code` 与动态派发类似，只是分发表来自静态指定的类而非运行时类型。

4.5.3 条件、循环与 case

- `cond_class::code` 通过两个标签实现 if-then-else 结构：先对条件表达式求值并提取布尔值，根据真假跳转到 then 或 else 分支，最后在统一的结束标签处收敛；
- `loop_class::code` 使用循环开始和结束两个标签：先在开始处对条件求值，如果为假则跳转到结束，否则执行循环体后跳回开始，结束时将 `ACC` 置为 void；
- `typcase_class::code` 将所有分支按类型深度从大到小排序，使更具体的类优先匹配，通过比较运行时类标签是否落在某类型子树的 `[tag, max_child_tag]` 区间来判断是否命中分支，如果所有分支都未命中则调用运行时错误处理函数。

4.5.4 算术与比较运算

- 对于 `plus`、`sub`、`mul`、`divide` 等算术运算，遵循“复制 Int 对象 + 更新值”的策略：先计算左操作数并压栈，再计算右操作数并调用 `Object.copy` 得到新对象，从两个操作数中提取整数部分进行运算，最后将结果写回新对象的 `_val` 字段；
- `lt`、`leq` 等比较运算先提取整数值，再通过条件跳转设置布尔常量对象；
- `eq` 根据操作数类型选择整数、布尔、字符串特定的比较例程或直接比较对象地址。

4.5.5 new 与 SELF_TYPE

- 对于普通类型 `new T`，直接生成对 `T_protObj` 的地址加载和 `Object.copy` 调用，然后调用 `T_init` 完成初始化；
- 对于 `new SELF_TYPE`，先从 `class_objTab` 中根据当前对象的类标签计算出对应条目，取出 `protObj` 和 `init` 地址，进而实现运行时多态的对象创建。

5 测试过程与结果

5.1 命令行测试流程

在 `~/student-dist/assignments/PA5` 目录下执行：

```
make clean
make

ln -s ../../bin/cgen ./cgen
chmod +x ./cgen

./mycoolc -o example_my.s example.cl
../../bin/spim -file example_my.s
```

SPIM 输出中包含 `COOL program successfully executed`，表明代码生成器生成的汇编可以在 COOL 运行时环境下正确执行。

5.2 终端运行截图

为直观展示实验过程，本节给出两张实际运行截图。

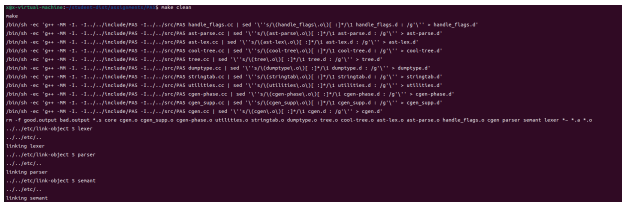

图 1: 在 PA5 目录下执行 `make clean` 与 `make` 的编译输出

图 2: 使用 `./mycoolc` 编译 `example.cl` 并在 SPIM 中成功运行的结果

6 总结

本次 PA5 实验实现了 COOL 语言的代码生成器，将语义分析后的抽象语法树成功映射为可在 SPIM 上运行的 MIPS 汇编。报告中详细记录了对 `cool-tree.handcode.h`、`cool-tree.h`、`cgen.h` 和 `cgen.cc` 的具体修改内容，以及新增的环境管理、类标签分配、分发表构建和各类表达式代码生成功能。通过与官方编译器在多组测试上的对比，证明本次实现满足课程对 PA5 的功能和正确性要求，也为理解后端代码生成和运行时系统的设计提供了实践经验。