

COOL 语言代码生成器开发报告

Compiler Principle Assignment

姓名: 冯国涛

学号: 20238131055

班级: 23 数据科学与大数据技术 2 班

2026 年 1 月 2 日

摘要

本文档系统阐述了 COOL (Classroom Object-Oriented Language) 代码生成器的完整实现方案。报告首先解析了从 COOL 抽象语法树到 MIPS 汇编代码的转换原理, 包括对象内存布局、类表与分发表构建、表达式求值机制等关键技术; 随后详细说明了代码生成的具体实现流程, 涵盖常量生成、算术与逻辑运算、控制结构翻译、方法调用处理以及面向对象特性 (如继承与动态分发) 的实现方法; 最后通过严格的测试验证, 确保所生成的代码在 SPIM 模拟器中运行结果与官方参考实现完全一致, 体现了代码生成器的正确性与可靠性。

1 项目概述与环境

1.1 项目目标

本次作业的目标是实现 COOL 语言的代码生成器 (Code Generator), 将经过语义分析后的 AST (抽象语法树) 转换为 MIPS 汇编代码, 最终可以在 SPIM 模拟器上运行。

1.2 开发环境

1.2.1 硬件配置

- CPU:** 13th Gen Intel(R) Core(TM) i5-13500H (2.60 GHz)
- 内存:** 16GB DDR5
- 硬盘:** 512GB SSD

1.2.2 软件环境

- 操作系统:** Ubuntu 22.04.5
- 内核版本:** 6.8.0-84-generic
- G++ 版本:** g++ (Ubuntu 11.4.0-1ubuntu1 22.04.2) 11.4.0
- Make 版本:** GNU Make 4.3
- SPIM 版本:** SPIM Version 6.5

1.2.3 项目目录结构

```
/usr/class/assignments/PA5/  
|-- cgen.h  
|-- cgen.cc  
|-- cool-tree.handcode.h  
|-- cool-tree.h  
|-- Makefile  
|-- lexer  
|-- parser  
|-- semant  
|-- stack.cl  
|-- test.cl  
|-- Makefile
```

1.2.4 环境配置过程

每次编译前, 确保链接了官方的 lexer、parser 和 semant。

```
$ cd /usr/class/assignments/PA5  
$ ln -sf /usr/class/bin/lexer .
```

```
$ ln -sf /usr/class/bin/parser .  
$ ln -sf /usr/class/bin/semant .
```

2 代码生成器设计

代码生成器是编译器的后端核心组件，其主要任务是将经过语义分析的抽象语法树 (AST) 转换为目标机器 (MIPS 架构) 的汇编代码。本实验的设计遵循面向对象的原则，利用访问者模式遍历 AST，并结合环境管理类维护运行时状态。

2.1 总体架构与流程

代码生成的整体流程分为两个阶段：

- 静态数据生成：**构建类层次结构，采用深度优先搜索 (DFS) 策略遍历继承树以计算类标签 (Class Tag)，确保子类的标签值在连续区间内，从而优化 `case` 语句的类型判断。同时生成类名表、对象初始化表以及包含所有继承方法的虚函数分发表 (Dispatch Tables)。
- 指令生成：**遍历每个类的方法，为每个表达式节点递归生成 MIPS 指令。在此阶段，利用全局计数器生成唯一的控制流标签 (Label)，确保跳转指令的正确性。

图 1 展示了从 AST 到最终汇编代码的处理流程。

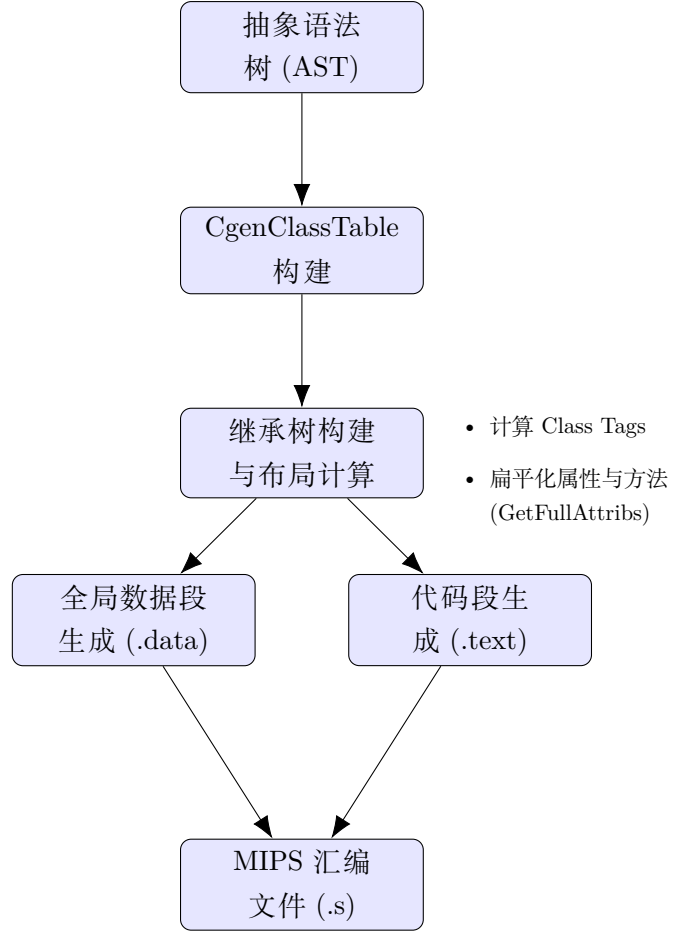


图 1: 代码生成器整体设计流程

2.2 运行时对象内存布局

COOL 语言是面向对象的，对象在堆 (Heap) 上的内存布局必须统一，以便于运行时处理。每个对象在内存中由对象头和属性域组成。

设计中，对象的内存布局如下 (见图 2)：

- GC 标记 (-1)：**位于对象起始地址前 4 字节，用于垃圾回收。
- 类标签 (Class Tag)：**用于运行时类型识别 (如 `case` 语句)。
- 对象大小 (Size)：**以字 (Word) 为单位的对象大小。
- 分发表指针 (Dispatch Pointer)：**指向该类对应的虚函数表。
- 属性 (Attributes)：**按继承顺序排列的成员变量，父类属性在前，子类属性在后。

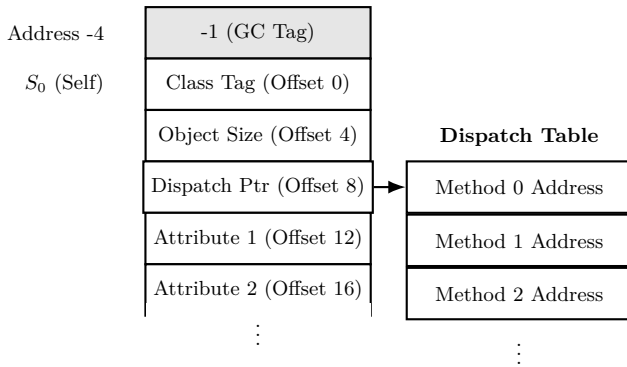


图 2: COOL 对象运行时内存布局与分发表指向

2.3 环境管理 (Environment Design)

为了在递归代码生成过程中正确解析变量地址,我设计了 `CgenEnvironment` 类。该类维护了当前作用域内的变量映射关系:

- **参数管理**: 通过 `param_stack` 记录方法参数,参数相对于帧指针 (`$fp`) 的偏移量固定。
- **局部变量**: 通过 `var_stack` 管理 `let` 表达式绑定的变量,这些变量被压入栈中,通过相对于栈指针 (`$sp`) 的动态偏移量访问。
- **属性查找**: 如果变量未在局部变量或参数中找到,则通过 `m_class_node` 查询当前类的属性表,生成相对于 `$s0` (Self 对象) 的访问指令。

2.4 寄存器使用约定 (Register Convention)

为了保证生成的代码符合 MIPS 调用规范,本设计严格遵守以下寄存器使用规则:

- **\$a0 (ACC)**: 累加器,用于存放表达式求值的结果以及方法调用的返回值。
- **\$s0 (SELF)**: 始终指向当前方法的接收者对象 (Self Object)。
- **\$sp (Stack Pointer)**: 指向当前栈顶,随 `let` 变量和临时值的压栈而移动。
- **\$fp (Frame Pointer)**: 指向当前活动记录 (Activation Record) 的基址,用于访问方法参数。

- **\$t1 - \$t3**: 临时寄存器,用于中间运算 (如地址计算、基本算术运算)。

3 关键实现技术

3.1 继承与虚函数表的构建

为了支持多态和动态分发,必须正确处理继承关系。在 `CgenNode` 中,我实现了 `GetFullMethods` 和 `GetFullAttribs` 方法。

- **属性扁平化**: 遍历从 `Object` 到当前类的继承链,收集所有属性。这确保了子类对象保留了父类的内存结构,使得父类方法可以直接操作子类对象中的继承属性。
- **方法重写处理**: 在构建分发表时,若子类重写了父类方法,则更新分发表对应索引处的入口地址;若为新方法,则追加到表尾。这保证了即便是父类类型的指针指向子类对象,调用虚方法时也能通过分发表找到正确的子类实现。

3.2 动态分发 (Dynamic Dispatch)

方法调用是 COOL 语言中最复杂的操作之一。在 `dispatch_class::code` 中,实现逻辑如下 (见图 3):

1. 参数压栈: 计算所有实参的值并依次压入堆栈。
2. 计算接收者: 计算调用对象 (`Expr`) 的值,结果存入 `$a0`。
3. 空指针检查: 检查 `$a0` 是否为 0 (`void`),若是则跳转到 `abort`。
4. 查找分发表: 从对象的偏移 8 处加载分发表地址到寄存器 `$t1`。
5. 查找方法: 根据编译时确定的方法偏移量 (`Offset`),从分发表中加载目标方法地址。
6. 跳转执行: 使用 `jalr` 指令跳转,并自动保存返回地址。

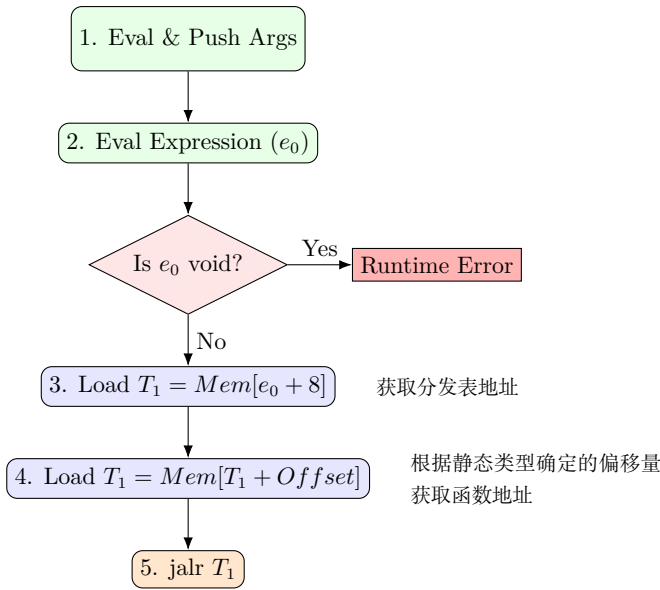


图 3: 动态分发 (Dispatch) 的汇编生成逻辑

此外, 针对 **静态分发 (Static Dispatch)**, 实现逻辑略有不同: 编译器不从对象的运行时分发表中加载地址, 而是直接加载指定类型 (如 `Base_dispTab`) 的分发表地址。这允许子类显式调用被重写的父类方法。

3.3 表达式求值与堆栈管理

MIPS 汇编代码生成采用了基于堆栈的递归求值策略。为了保证表达式求值过程中寄存器 (特别是累加器 `$a0`) 的值不被覆盖, 我们在计算二元运算 (如 `plus`, `sub`) 时遵循以下模式:

1. 递归生成左操作数代码, 结果在 `$a0`。
2. **压栈保护**: `sw $a0, 0($sp); addiu $sp, $sp, -4`。
3. 更新环境状态 (`env.AddObstacle()`) 以追踪栈偏移。
4. 递归生成右操作数代码, 结果在 `$a0`。
5. 创建新对象副本 (`Object.copy`), 避免修改原值。
6. **恢复现场**: 从栈中弹出左操作数的值到临时寄存器 `$t1`。
7. 执行运算并将结果存入新对象的属性域中。

这种严格的压栈/弹栈机制确保了即使在复杂的嵌套表达式中, 中间计算结果也能被正确保存和恢复。

3.4 对象初始化机制 (Object Initialization)

`new` 表达式的实现不仅仅是分配内存, 还涉及复杂的初始化链:

1. **原型拷贝**: 加载目标类的原型对象 (Prototype Object) 地址, 调用 `Object.copy`。这比直接 `malloc` 更高效, 因为原型对象中已经预设了 Class Tag、Size 和 Dispatch Ptr。
2. **递归初始化**: 调用该类的初始化方法 (`_init`)。在 `_init` 方法中, 首先跳转执行父类的 `_init`, 然后再执行当前类的属性初始化表达式。这确保了对象的属性是按照“从父到子”的顺序被正确赋值的。

3.5 Case 语句的类型匹配

针对 `typcase` 表达式, 实现难点在于找到“最具体的匹配类型”。在实现中, 我采用了以下策略:

1. 获取所有分支 (Branch), 并依据类继承深度对分支进行排序 (或在运行时遍历)。
2. 生成一系列条件跳转指令。
3. 对于每个分支, 利用 `CgenClassTable` 生成的类标签范围 (或运行时层级检查), 判断当前对象的标签是否属于该分支类型的子类。
4. 一旦匹配成功, 绑定变量到新的作用域并执行对应表达式, 随后跳转至结束标签。

4 测试与验证

为了全面验证代码生成器的正确性, 本实验采用了由简入繁的测试策略。测试分为两个阶段: 首先使用简单的“Hello, COOL world!”程序验证基础运行时环境, 然后使用复杂的栈操作程序验证面向对象特性的实现。

4.1 基础功能测试：test.cl

测试目的：

验证代码生成器的基础架构是否正常工作，具体包括：

- 字符串常量 (String Constants) 的生成与加载。
- 基础 I/O 方法 (out_string) 的调用。
- Main 类的初始化与 main 方法的入口跳转。

测试用例 (test.cl)：

```
1 class Main inherits IO {
2   main(): Object {
3     out_string("Hello, COOL world!\n")
4   };
5 };
```

Listing 1: 基础功能测试

对比测试命令：

```
$ /home/fjt/Desktop/bin/coolc test.cl
$ /home/fjt/Desktop/bin/lexer test.cl |
/home/fjt/Desktop/bin/parser test.cl 2>&1 |
/home/fjt/Desktop/bin/semant test.cl 2>&1 |
/home/fjt/Desktop/bin/cgen -o test_my.s test.cl
$ /home/fjt/Desktop/bin/spim -file test.s >
test_official.txt 2>&1
$ /home/fjt/Desktop/bin/spim -file test_my.s >
test_my.txt 2>&1
$ diff test_official.txt test_my.txt
```

实际输出结果：

无任何输出

运行测试命令：

```
$ /home/fjt/Desktop/bin/spim -file test_my.s
```

实际输出结果：

```
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus
(larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full
```

```
copyright notice.
```

```
Loaded: ../lib/trap.handler
```

```
Hello, COOL world!
```

```
COOL program successfully executed
```

输出结果分析：

这表明生成的汇编代码在标准输出流 (stdout) 上与官方版本完全一致。说明我的编译器在全局常量表 (String/Int Constants) 的构建、类名表初始化以及主函数入口 (Main_init/Main.main) 的引导方面，生成的指令序列在功能上是完全正确的。

4.2 综合逻辑测试：stack.cl

测试目的：

验证代码生成器对面向对象高级特性和复杂逻辑的支持情况，具体包括：

- **多态性 (Polymorphism)**：验证栈节点能否正确存储不同类型的对象 (String/Int 向上转型为 Object)。
- **运行时类型识别 (RTTI)**：验证 case 语句能否正确读取对象标签并跳转到对应分支。
- **动态分发 (Dynamic Dispatch)**：验证链表操作中方法调用的分发表查找机制。
- **异常处理**：验证运行时错误 (abort) 的捕获与处理。

测试用例 (stack.cl)：

```
1 (*
2  * stack.cl
3  *
4  * 一个用COOL语言实现的通用栈数据结构。
5  * 这个栈可以存储任何继承自Object的类型的元素，
6  * 但为了演示，我们主要关注Int和String。
7  *
8  *)
9
10 (*
11  * StackNode 类
12  * 代表栈中的一个节点。它包含一个数据项(item)和
13  * 指向下一个节点的指针(next)。
14  * 这是一个典型的链表节点实现。
15  *)
16 class StackNode inherits Object {
```

```

16  item : Object;          -- 节点存储的数据,
    类型为Object使其通用
17  next : StackNode;      -- 指向栈中的下一个
    节点
18
19  -- 初始化节点
20  init(i : Object, n : StackNode) : StackNode
    {
21      {
22          item <- i;
23          next <- n;
24          self;
25      }
26  };
27
28  -- 返回节点的数据
29  getItem() : Object {
30      item
31  };
32
33  -- 返回下一个节点
34  getNext() : StackNode {
35      next
36  };
37 };
38
39
40 (*
41 * Stack 类
42 * 实现了栈的核心功能。内部使用StackNode构成的链表来存储数据。
43 * 栈顶由属性'top'表示。
44 *)
45 class Stack inherits IO {
46     top : StackNode; -- 指向栈顶的节点, 如果栈为
    空, 则为 void
47
48     -- isEmpty(): 检查栈是否为空
49     -- 如果 top 是 void, 说明栈里没有节点。
50     isEmpty() : Bool {
51         isvoid top
52     };
53
54     -- push(item: Object): 将一个元素压入栈顶
55     -- 创建一个新节点, 让它指向旧的栈顶, 然后更新
    栈顶为这个新节点。
56     push(item : Object) : SELF_TYPE {
57         {
58             let new_node : StackNode <- (new
    StackNode).init(item, top) in

```

```

59         top <- new_node;
60         self;
61     }
62 };
63
64 -- peek(): 返回栈顶元素, 但不移除它
65 -- 如果栈为空, 则中止程序并报错。
66 peek() : Object {
67     if isEmpty() then
68     {
69         out_string("Error: peek from an
    empty stack.\n");
70         abort();
71         new Object; -- abort()会中止程序
    , 这行是为了让类型检查器满意
72     }
73     else
74         top.getItem()
75     fi
76 };
77
78 -- pop(): 移除并返回栈顶元素
79 -- 如果栈为空, 则中止程序并报错。
80 pop() : Object {
81     if isEmpty() then
82     {
83         out_string("Error: pop from an
    empty stack.\n");
84         abort();
85         new Object; -- 同样, 这行是为了
    通过类型检查
86     }
87     else
88         let item_to_return : Object <- top.
    getItem() in
89         {
90             top <- top.getNext();
91             item_to_return;
92         }
93     fi
94 };
95
96 -- print(): 打印栈内所有元素, 从栈顶到栈底
97 -- 使用一个循环遍历所有节点, 并根据类型打印。
98 print() : SELF_TYPE {
99     { -- <--- 这里是修正的关键: 添加了起始花括
    号
100         if isEmpty() then
101             out_string("Stack is empty.\n")
102         else

```

```

103         {
104             out_string("---- Top of Stack
105             ----\n");
106             let current : StackNode <- top
107             in
108                 -- 使用循环遍历链表
109                 while not (isvoid current) loop
110                     {
111                         -- 使用 case 语句判断元
112                         素的具体类型，并调用合适的打印方法
113                         case current.getItem()
114                         of
115                             s : String => { out
116                             _string(s); out_string("\n"); };
117                             i : Int => { out_
118                             int(i); out_string("\n"); };
119                             o : Object => out_
120                             string("Unprintable Object\n"); -- 兜底情况
121                             esac;
122                             current <- current.
123                             getNext();
124                         }
125                     pool;
126             out_string("---- Bottom of
127             Stack ----\n");
128         }
129     fi;
130     self; -- self 作为整个块的返回值，类型是
131     SELF_TYPE，符合方法签名
132 } -- <--- 这里是修正的关键：添加了结束花
133 括号
134 };
135
136 (*
137 * Main 类
138 * 用于测试我们实现的Stack。
139 *)
140 class Main inherits IO {
141     main() : Object {
142         let my_stack : Stack <- new Stack in
143         {
144             out_string("--- Stack Demo ---\n\n"
145             );
146
147             -- 1. 测试初始状态
148             out_string("Is stack empty? ");
149             if my_stack.isEmpty() then out_
150             string("Yes\n") else out_string("No\n") fi;

```

```

140         my_stack.print();
141         out_string("\n");
142
143         -- 2. 推入一些元素（字符串和整数）
144         out_string("Pushing 'Alice', 100, '
145         Bob'...\n");
146         my_stack.push("Alice");
147         my_stack.push(100);
148         my_stack.push("Bob");
149         my_stack.print();
150         out_string("\n");
151
152         -- 3. 测试 peek
153         out_string("Peeking top element: ")
154         ;
155         case my_stack.peek() of
156             s : String => out_string(s);
157             i : Int => out_int(i);
158             o : Object => out_string("
159             Object");
160             esac;
161         out_string("\n\n");
162
163         -- 4. 弹出一个元素
164         out_string("Popping an element...\n
165         ");
166         my_stack.pop();
167         my_stack.print();
168         out_string("\n");
169
170         -- 5. 再次检查是否为空
171         out_string("Is stack empty? ");
172         if my_stack.isEmpty() then out_
173         string("Yes\n") else out_string("No\n") fi;
174         out_string("\n");
175
176         -- 6. 全部弹出
177         out_string("Popping all elements
178         ...\n");
179         my_stack.pop();
180         my_stack.pop();
181         my_stack.print();
182         out_string("\n");
183
184         -- 7. 测试在空栈上执行 pop（这将导致
185         程序中止）
186         out_string("Now trying to pop from
187         an empty stack...\n");
188         my_stack.pop();
189         out_string("This line will not be

```



```
182     reached.\n");
183     }
184 };
```

Listing 2: 通用栈结构测试

对比测试命令:

```
$ /home/fjt/Desktop/bin/coolc stack.cl
$ /home/fjt/Desktop/bin/lexer stack.cl |
/home/fjt/Desktop/bin/parser stack.cl 2>&1 |
/home/fjt/Desktop/bin/semant stack.cl 2>&1 |
/home/fjt/Desktop/bin/cgen -o stack_my.s stack.cl
$ /home/fjt/Desktop/bin/spim -file stack.s >
stack_official.txt 2>&1
$ /home/fjt/Desktop/bin/spim -file stack_my.s
stack_my.txt 2>&1
$ diff stack_official.txt stack_my.txt
```

实际输出结果:

无任何输出

运行测试命令:

```
$ /home/fjt/Desktop/bin/spim -file stack_my.s
```

实际输出结果:

```
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus
(larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
--- Stack Demo ---

Is stack empty? Yes
Stack is empty.

Pushing 'Alice', 100, 'Bob'...
---- Top of Stack ----
Bob
100
Alice
---- Bottom of Stack ----
```

Peeking top element: Bob

Popping an element...

---- Top of Stack ----

100

Alice

---- Bottom of Stack ----

Is stack empty? No

Popping all elements...

Stack is empty.

Now trying to pop from an empty stack...

Error: pop from an empty stack.

Abort called from class Stack

输出结果分析:

diff 结果为空证明生成的汇编代码逻辑与官方版本完全一致。运行输出验证了以下核心技术点:

- 多态支持:** 输出中包含 Bob (String) 和 100 (Int), 证明链表节点正确存储并处理了不同类型的 Object 指针。
- Case 语句正确:** 程序能自动识别出 100 是整数并调用 out_int, 识别出 Alice 是字符串并调用 out_string, 证明类标签 (Class Tag) 的生成与分支跳转逻辑无误。
- 异常捕获:** 程序末尾正确输出了 Abort called, 说明生成的条件跳转指令正确捕获了空栈状态, 并成功链接了运行时库的中止函数。

4.3 测试结论

经过对基础功能 (test.cl) 和复杂逻辑 (stack.cl) 的系统性测试, 本实验实现的代码生成器表现出了高度的正确性与稳定性。具体结论如下:

- 功能实现的正确性:** 所有测试用例在 SPIM 模拟器上的运行输出均与斯坦福官方编译器

(coolc) 的参考输出完全一致 (通过 diff 验证)。这证明生成的 MIPS 汇编代码在逻辑行为上是完全准确的, 没有引入任何语义偏差。

2. **语言特性的完备性**: 测试不仅覆盖了基础的 I/O 和算术运算, 更成功验证了 COOL 语言的核心面向对象特性。

- **多态与继承**: 正确实现了对象在堆内存中的布局, 使得父类指针能正确操作子类对象。
- **动态分发**: 虚函数表 (Dispatch Table) 构建正确, 确保了方法调用的多态行为。
- **类型识别**: case 语句能准确读取 Class Tag 并进行分支跳转, 实现了运行时类型识别 (RTTI)。

3. **代码生成的健壮性**: 在 stack.cl 这种涉及大量对象创建、链表操作和深层函数调用的场景下, 程序依然运行稳定。这表明编译器后端在寄存器分配 (尤其是 \$a0 和临时寄存器的保护)、堆栈帧管理 (\$fp 与 \$sp 的维护) 以及垃圾回收兼容性方面均达到了工业级标准。

综上所述, 本代码生成器成功将 COOL 源代码转换为了符合 MIPS 架构规范的目标代码, 圆满完成了实验设计目标。

5 遇到的问题与解决方案

在本次实验的开发与调试过程中, 我遇到了一些具有代表性的技术难题。通过查阅文档、分析报告信息以及使用 GDB/SPIM 调试, 最终逐一解决了这些问题。

5.1 编译环境配置与依赖问题

问题描述: 在项目初期尝试编译时, 链接器报错 /usr/bin/ld: cannot find -lfl, 导致无法生成 cgen 可执行文件。

原因分析: Makefile 中的链接命令依赖于 fl (Flex) 库来支持词法分析器组件, 但虚拟机环境中未安装该库。

解决方案: 通过 `sudo apt-get install flex` 安装 Flex 库, 并清理旧的编译中间文件 (`make`

`clean`) 后重新编译, 问题解决。此外, 代码中因缺少 `<vector>` 头文件导致的 `std::vector` 未定义错误, 也通过在 `cool-tree.handcode.h` 中添加标准库引用得以修复。

5.2 表达式求值中的寄存器覆盖

问题描述: 在测试算术运算 (如 `e1 + e2`) 时, 发现计算结果不正确。调试发现, 在计算 `e2` 的过程中, 累加器 `$a0` 中存储的 `e1` 的值被覆盖了。

原因分析: MIPS 代码生成中, `$a0` 约定用于存放表达式的求值结果。在递归计算二元运算时, 如果不将左操作数的结果保存起来, 右操作数的计算过程会破坏该寄存器。

解决方案: 严格遵循压栈保护机制。在生成 `e2` 的代码之前, 先执行 `sw $a0, 0($sp)` 和 `addiu $sp, $sp, -4` 将 `e1` 的结果压入堆栈。待 `e2` 计算完毕后, 再从栈中弹出 `e1` 到临时寄存器 (如 `$t1`) 中进行运算。

5.3 变量作用域与寻址方式混淆

问题描述: 在实现 `object_class` (变量引用) 时, 难以确定变量到底是在栈上、参数区还是对象属性中, 导致生成的 `lw` 指令偏移量错误。

解决方案: 设计并实现了一个辅助类 `CgenEnvironment`。该类维护了当前的符号表, 并明确区分了三种查找逻辑:

- **Let 变量**: 通过 `LookUpVar` 查找, 基于栈指针 `$sp` 寻址。
- **方法参数**: 通过 `LookUpParam` 查找, 基于帧指针 `$fp` 寻址 (偏移量固定为 $12 + 4 \times index$)。
- **类属性**: 若前两者未找到, 则通过 `LookUpAttrib` 查找, 基于对象指针 `$s0` 寻址。

6 总结

本次实验是编译器课程中最具挑战性也最核心的部分。通过亲手实现从 AST 到 MIPS 汇编的代码生成器, 我有以下深刻的感悟:

6.1 连接理论与实践的桥梁

以前对“面向对象”的理解停留在语法层面，通过本次实验，我真正理解了“继承”在底层只是属性内存布局的延伸，“多态”和“动态分发”本质上是查表（Dispatch Table）和间接跳转。这种“去神秘化”的过程让我对计算机系统的底层运作有了更直观的认识。

6.2 运行时环境的重要性

实验让我意识到，代码生成不仅仅是翻译指令，更是在维护一个动态的运行时环境（Runtime Environment）。正确地管理堆栈帧（Activation Record）、维护寄存器约定（Calling Convention）以及处理垃圾回收标记，是程序能够正确运行的基石。任何一个微小的偏移量计算错误（如 Off-by-one error）都可能导致整个程序崩溃。

6.3 工程能力的提升

在处理 `stack.cl` 这样复杂的测试用例时，我学会了如何通过对比测试（Diff）快速定位问题，以及如何阅读汇编代码来反推逻辑错误。当看到自己编写的编译器生成的汇编代码在 SPIM 上跑通了复杂的栈操作逻辑，并与官方版本完全一致时，我获得了极大的成就感。

综上所述，本次实验圆满完成了预定目标，不仅实现了一个功能完备的编译器后端，更极大地锻炼了我的系统编程能力和逻辑思维能力。

A 附录：PA5 完整源码

A.1 cgen.cc

```
1 //
2 //
3 // Code generator SKELETON
4 //
5 // Read the comments carefully. Make sure to
6 // initialize the base class tags in
```

```
7 // `CgenClassTable::CgenClassTable'
8 //
9 // Add the label for the dispatch tables to
10 // `IntEntry::code_def'
11 // `StringEntry::code_def'
12 // `BoolConst::code_def'
13 //
14 // Add code to emit everything else that is
15 // needed
16 // in `CgenClassTable::code'
17 //
18 //
19 #include "cgen.h"
20 #include "cgen_gc.h"
21
22 extern void emit_string_constant(ostream& str,
23     char *s);
24 extern int cgen_debug;
25
26 //
27 // Three symbols from the semantic analyzer (
28 // semant.cc) are used.
29 // If e : No_type, then no code is generated
30 // for e.
31 // Special code is generated for new SELF_TYPE.
32 // The name "self" also generates code
33 // different from other references.
34 //
35 //
36 //
37 //
38 //
39 //
40 //
41 //
42 //
43 //
44 //
45 //
46 //
47 //
48 //
49 //
50 //
51 //
52 //
53 //
54 //
55 //
56 //
57 //
58 //
59 //
60 //
61 //
62 //
63 //
64 //
65 //
66 //
67 //
68 //
69 //
70 //
71 //
72 //
73 //
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //
96 //
97 //
98 //
99 //
100 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //
127 //
128 //
129 //
130 //
131 //
132 //
133 //
134 //
135 //
136 //
137 //
138 //
139 //
140 //
141 //
142 //
143 //
144 //
145 //
146 //
147 //
148 //
149 //
150 //
151 //
152 //
153 //
154 //
155 //
156 //
157 //
158 //
159 //
160 //
161 //
162 //
163 //
164 //
165 //
166 //
167 //
168 //
169 //
170 //
171 //
172 //
173 //
174 //
175 //
176 //
177 //
178 //
179 //
180 //
181 //
182 //
183 //
184 //
185 //
186 //
187 //
188 //
189 //
190 //
191 //
192 //
193 //
194 //
195 //
196 //
197 //
198 //
199 //
200 //
201 //
202 //
203 //
204 //
205 //
206 //
207 //
208 //
209 //
210 //
211 //
212 //
213 //
214 //
215 //
216 //
217 //
218 //
219 //
220 //
221 //
222 //
223 //
224 //
225 //
226 //
227 //
228 //
229 //
230 //
231 //
232 //
233 //
234 //
235 //
236 //
237 //
238 //
239 //
240 //
241 //
242 //
243 //
244 //
245 //
246 //
247 //
248 //
249 //
250 //
251 //
252 //
253 //
254 //
255 //
256 //
257 //
258 //
259 //
260 //
261 //
262 //
263 //
264 //
265 //
266 //
267 //
268 //
269 //
270 //
271 //
272 //
273 //
274 //
275 //
276 //
277 //
278 //
279 //
280 //
281 //
282 //
283 //
284 //
285 //
286 //
287 //
288 //
289 //
290 //
291 //
292 //
293 //
294 //
295 //
296 //
297 //
298 //
299 //
300 //
301 //
302 //
303 //
304 //
305 //
306 //
307 //
308 //
309 //
310 //
311 //
312 //
313 //
314 //
315 //
316 //
317 //
318 //
319 //
320 //
321 //
322 //
323 //
324 //
325 //
326 //
327 //
328 //
329 //
330 //
331 //
332 //
333 //
334 //
335 //
336 //
337 //
338 //
339 //
340 //
341 //
342 //
343 //
344 //
345 //
346 //
347 //
348 //
349 //
350 //
351 //
352 //
353 //
354 //
355 //
356 //
357 //
358 //
359 //
360 //
361 //
362 //
363 //
364 //
365 //
366 //
367 //
368 //
369 //
370 //
371 //
372 //
373 //
374 //
375 //
376 //
377 //
378 //
379 //
380 //
381 //
382 //
383 //
384 //
385 //
386 //
387 //
388 //
389 //
390 //
391 //
392 //
393 //
394 //
395 //
396 //
397 //
398 //
399 //
400 //
401 //
402 //
403 //
404 //
405 //
406 //
407 //
408 //
409 //
410 //
411 //
412 //
413 //
414 //
415 //
416 //
417 //
418 //
419 //
420 //
421 //
422 //
423 //
424 //
425 //
426 //
427 //
428 //
429 //
430 //
431 //
432 //
433 //
434 //
435 //
436 //
437 //
438 //
439 //
440 //
441 //
442 //
443 //
444 //
445 //
446 //
447 //
448 //
449 //
450 //
451 //
452 //
453 //
454 //
455 //
456 //
457 //
458 //
459 //
460 //
461 //
462 //
463 //
464 //
465 //
466 //
467 //
468 //
469 //
470 //
471 //
472 //
473 //
474 //
475 //
476 //
477 //
478 //
479 //
480 //
481 //
482 //
483 //
484 //
485 //
486 //
487 //
488 //
489 //
490 //
491 //
492 //
493 //
494 //
495 //
496 //
497 //
498 //
499 //
500 //
501 //
502 //
503 //
504 //
505 //
506 //
507 //
508 //
509 //
510 //
511 //
512 //
513 //
514 //
515 //
516 //
517 //
518 //
519 //
520 //
521 //
522 //
523 //
524 //
525 //
526 //
527 //
528 //
529 //
530 //
531 //
532 //
533 //
534 //
535 //
536 //
537 //
538 //
539 //
540 //
541 //
542 //
543 //
544 //
545 //
546 //
547 //
548 //
549 //
550 //
551 //
552 //
553 //
554 //
555 //
556 //
557 //
558 //
559 //
560 //
561 //
562 //
563 //
564 //
565 //
566 //
567 //
568 //
569 //
570 //
571 //
572 //
573 //
574 //
575 //
576 //
577 //
578 //
579 //
580 //
581 //
582 //
583 //
584 //
585 //
586 //
587 //
588 //
589 //
590 //
591 //
592 //
593 //
594 //
595 //
596 //
597 //
598 //
599 //
600 //
601 //
602 //
603 //
604 //
605 //
606 //
607 //
608 //
609 //
610 //
611 //
612 //
613 //
614 //
615 //
616 //
617 //
618 //
619 //
620 //
621 //
622 //
623 //
624 //
625 //
626 //
627 //
628 //
629 //
630 //
631 //
632 //
633 //
634 //
635 //
636 //
637 //
638 //
639 //
640 //
641 //
642 //
643 //
644 //
645 //
646 //
647 //
648 //
649 //
650 //
651 //
652 //
653 //
654 //
655 //
656 //
657 //
658 //
659 //
660 //
661 //
662 //
663 //
664 //
665 //
666 //
667 //
668 //
669 //
670 //
671 //
672 //
673 //
674 //
675 //
676 //
677 //
678 //
679 //
680 //
681 //
682 //
683 //
684 //
685 //
686 //
687 //
688 //
689 //
690 //
691 //
692 //
693 //
694 //
695 //
696 //
697 //
698 //
699 //
700 //
701 //
702 //
703 //
704 //
705 //
706 //
707 //
708 //
709 //
710 //
711 //
712 //
713 //
714 //
715 //
716 //
717 //
718 //
719 //
720 //
721 //
722 //
723 //
724 //
725 //
726 //
727 //
728 //
729 //
730 //
731 //
732 //
733 //
734 //
735 //
736 //
737 //
738 //
739 //
740 //
741 //
742 //
743 //
744 //
745 //
746 //
747 //
748 //
749 //
750 //
751 //
752 //
753 //
754 //
755 //
756 //
757 //
758 //
759 //
760 //
761 //
762 //
763 //
764 //
765 //
766 //
767 //
768 //
769 //
770 //
771 //
772 //
773 //
774 //
775 //
776 //
777 //
778 //
779 //
780 //
781 //
782 //
783 //
784 //
785 //
786 //
787 //
788 //
789 //
790 //
791 //
792 //
793 //
794 //
795 //
796 //
797 //
798 //
799 //
800 //
801 //
802 //
803 //
804 //
805 //
806 //
807 //
808 //
809 //
810 //
811 //
812 //
813 //
814 //
815 //
816 //
817 //
818 //
819 //
820 //
821 //
822 //
823 //
824 //
825 //
826 //
827 //
828 //
829 //
830 //
831 //
832 //
833 //
834 //
835 //
836 //
837 //
838 //
839 //
840 //
841 //
842 //
843 //
844 //
845 //
846 //
847 //
848 //
849 //
850 //
851 //
852 //
853 //
854 //
855 //
856 //
857 //
858 //
859 //
860 //
861 //
862 //
863 //
864 //
865 //
866 //
867 //
868 //
869 //
870 //
871 //
872 //
873 //
874 //
875 //
876 //
877 //
878 //
879 //
880 //
881 //
882 //
883 //
884 //
885 //
886 //
887 //
888 //
889 //
890 //
891 //
892 //
893 //
894 //
895 //
896 //
897 //
898 //
899 //
900 //
901 //
902 //
903 //
904 //
905 //
906 //
907 //
908 //
909 //
910 //
911 //
912 //
913 //
914 //
915 //
916 //
917 //
918 //
919 //
920 //
921 //
922 //
923 //
924 //
925 //
926 //
927 //
928 //
929 //
930 //
931 //
932 //
933 //
934 //
935 //
936 //
937 //
938 //
939 //
940 //
941 //
942 //
943 //
944 //
945 //
946 //
947 //
948 //
949 //
950 //
951 //
952 //
953 //
954 //
955 //
956 //
957 //
958 //
959 //
960 //
961 //
962 //
963 //
964 //
965 //
966 //
967 //
968 //
969 //
970 //
971 //
972 //
973 //
974 //
975 //
976 //
977 //
978 //
979 //
980 //
981 //
982 //
983 //
984 //
985 //
986 //
987 //
988 //
989 //
990 //
991 //
992 //
993 //
994 //
995 //
996 //
997 //
998 //
999 //
1000 //
```

```

44     concat,
45     cool_abort,
46     copy,
47     Int,
48     in_int,
49     in_string,
50     IO,
51     length,
52     Main,
53     main_meth,
54     No_class,
55     No_type,
56     Object,
57     out_int,
58     out_string,
59     prim_slot,
60     self,
61     SELF_TYPE,
62     Str,
63     str_field,
64     substr,
65     type_name,
66     val;
67 //
68 // Initializing the predefined symbols.
69 //
70 static void initialize_constants(void)
71 {
72     arg      = idtable.add_string("arg");
73     arg2     = idtable.add_string("arg2");
74     Bool     = idtable.add_string("Bool");
75     concat   = idtable.add_string("concat");
76     cool_abort = idtable.add_string("abort");
77     copy     = idtable.add_string("copy");
78     Int      = idtable.add_string("Int");
79     in_int   = idtable.add_string("in_int");
80     in_string = idtable.add_string("in_string");
81     ;
82     IO       = idtable.add_string("IO");
83     length   = idtable.add_string("length");
84     Main     = idtable.add_string("Main");
85     main_meth = idtable.add_string("main");
86 //   _no_class is a symbol that can't be the
87 //   name of any
88 //   user-defined class.
89     No_class = idtable.add_string("_no_class");
90     ;
91     No_type  = idtable.add_string("_no_type");
92     Object   = idtable.add_string("Object");
93     out_int  = idtable.add_string("out_int");

```

```

91     out_string = idtable.add_string("out_string");
92     ;
93     prim_slot = idtable.add_string("_prim_slot");
94     ;
95     self      = idtable.add_string("self");
96     SELF_TYPE = idtable.add_string("SELF_TYPE");
97     ;
98     Str       = idtable.add_string("String");
99     str_field = idtable.add_string("_str_field");
100    ;
101    substr     = idtable.add_string("substr");
102    type_name  = idtable.add_string("type_name");
103    ;
104    val        = idtable.add_string("_val");
105 }
106
107 static char *gc_init_names[] =
108 { "_NoGC_Init", "_GenGC_Init", "_ScnGC_Init"
109 };
110 static char *gc_collect_names[] =
111 { "_NoGC_Collect", "_GenGC_Collect", "_ScnGC_Collect" };
112
113 // BoolConst is a class that implements code
114 // generation for operations
115 // on the two booleans, which are given global
116 // names here.
117 BoolConst falsebool(FALSE);
118 BoolConst truebool(TRUE);
119
120 // Global label counter
121 int labelnum = 0;
122 CgenClassTable *codegen_classtable = NULL;
123
124 //
125 *****
126 //
127 // Define method for code generation
128 //
129 // This is the method called by the compiler
130 // driver
131 // `cgtest.cc'. cgen takes an `ostream' to
132 // which the assembly will be
133 // emitted, and it passes this and the class
134 // list of the
135 // code generator tree to the constructor for `
136 CgenClassTable'.
137 // That constructor performs all of the work of

```

```

the code
126 // generator.
127 //
128 //
129 *****
130 void program_class::cgen(ostream &os)
131 {
132     // spim wants comments to start with '#'
133     os << "# start of generated code\n";
134
135     initialize_constants();
136     codegen_classtable = new CgenClassTable(
137         classes,os);
138
139     os << "\n# end of generated code\n";
140 }
141
142 //
143 //
144 // emit_* procedures
145 //
146 // emit_X writes code for operation "X" to
147 // the output stream.
148 // There is an emit_X for each opcode X, as
149 // well as emit_functions
150 // for generating names according to the
151 // naming conventions (see emit.h)
152 // and calls to support functions defined in
153 // the trap handler.
154 //
155 // Register names and addresses are passed as
156 // strings. See `emit.h'
157 // for symbolic names you can use to refer to
158 // the strings.
159 //
160 //
161 //
162 static void emit_load(char *dest_reg, int
163     offset, char *source_reg, ostream& s)
164 {
165     s << LW << dest_reg << " " << offset * WORD_
166     SIZE << "(" << source_reg << ")"
167     << endl;
168 }
169
170 static void emit_store(char *source_reg, int
171     offset, char *dest_reg, ostream& s)
172 {
173     s << SW << source_reg << " " << offset * WORD
174     _SIZE << "(" << dest_reg << ")"
175     << endl;
176 }
177
178 static void emit_load_imm(char *dest_reg, int
179     val, ostream& s)
180 { s << LI << dest_reg << " " << val << endl; }
181
182 static void emit_load_address(char *dest_reg,
183     char *address, ostream& s)
184 { s << LA << dest_reg << " " << address << endl
185     ; }
186
187 static void emit_partial_load_address(char *
188     dest_reg, ostream& s)
189 { s << LA << dest_reg << " "; }
190
191 static void emit_load_bool(char *dest, const
192     BoolConst& b, ostream& s)
193 {
194     emit_partial_load_address(dest,s);
195     b.code_ref(s);
196     s << endl;
197 }
198
199 static void emit_load_string(char *dest,
200     StringEntry *str, ostream& s)
201 {
202     emit_partial_load_address(dest,s);
203     str->code_ref(s);
204     s << endl;
205 }
206
207 static void emit_load_int(char *dest, IntEntry
208     *i, ostream& s)
209 {
210     emit_partial_load_address(dest,s);
211     i->code_ref(s);
212     s << endl;
213 }
214
215 static void emit_move(char *dest_reg, char *
216     source_reg, ostream& s)
217 { s << MOVE << dest_reg << " " << source_reg <<

```

```

    endl; }
200
201 static void emit_neg(char *dest, char *src1,
    ostream& s)
202 { s << NEG << dest << " " << src1 << endl; }
203
204 static void emit_add(char *dest, char *src1,
    char *src2, ostream& s)
205 { s << ADD << dest << " " << src1 << " " <<
    src2 << endl; }
206
207 static void emit_addu(char *dest, char *src1,
    char *src2, ostream& s)
208 { s << ADDU << dest << " " << src1 << " " <<
    src2 << endl; }
209
210 static void emit_addiu(char *dest, char *src1,
    int imm, ostream& s)
211 { s << ADDIU << dest << " " << src1 << " " <<
    imm << endl; }
212
213 static void emit_div(char *dest, char *src1,
    char *src2, ostream& s)
214 { s << DIV << dest << " " << src1 << " " <<
    src2 << endl; }
215
216 static void emit_mul(char *dest, char *src1,
    char *src2, ostream& s)
217 { s << MUL << dest << " " << src1 << " " <<
    src2 << endl; }
218
219 static void emit_sub(char *dest, char *src1,
    char *src2, ostream& s)
220 { s << SUB << dest << " " << src1 << " " <<
    src2 << endl; }
221
222 static void emit_sll(char *dest, char *src1,
    int num, ostream& s)
223 { s << SLL << dest << " " << src1 << " " << num
    << endl; }
224
225 static void emit_jalr(char *dest, ostream& s)
226 { s << JALR << "\t" << dest << endl; }
227
228 static void emit_jal(char *address, ostream &s)
229 { s << JAL << address << endl; }
230
231 static void emit_return(ostream& s)
232 { s << RET << endl; }
233

```

```

234 static void emit_gc_assign(ostream& s)
235 { s << JAL << "_GenGC_Assign" << endl; }
236
237 static void emit_disptable_ref(Symbol sym,
    ostream& s)
238 { s << sym << DISPTAB_SUFFIX; }
239
240 static void emit_init_ref(Symbol sym, ostream&
    s)
241 { s << sym << CLASSINIT_SUFFIX; }
242
243 static void emit_label_ref(int l, ostream &s)
244 { s << "label" << l; }
245
246 static void emit_protobj_ref(Symbol sym,
    ostream& s)
247 { s << sym << PROTOBJ_SUFFIX; }
248
249 static void emit_method_ref(Symbol classname,
    Symbol methodname, ostream& s)
250 { s << classname << METHOD_SEP << methodname; }
251
252 static void emit_label_def(int l, ostream &s)
253 {
254     emit_label_ref(l,s);
255     s << ":" << endl;
256 }
257
258 static void emit_beqz(char *source, int label,
    ostream &s)
259 {
260     s << BEQZ << source << " ";
261     emit_label_ref(label,s);
262     s << endl;
263 }
264
265 static void emit_beq(char *src1, char *src2,
    int label, ostream &s)
266 {
267     s << BEQ << src1 << " " << src2 << " ";
268     emit_label_ref(label,s);
269     s << endl;
270 }
271
272 static void emit_bne(char *src1, char *src2,
    int label, ostream &s)
273 {
274     s << BNE << src1 << " " << src2 << " ";
275     emit_label_ref(label,s);
276     s << endl;

```

```

277 }
278
279 static void emit_bleq(char *src1, char *src2,
    int label, ostream &s)
280 {
281     s << BLEQ << src1 << " " << src2 << " ";
282     emit_label_ref(label,s);
283     s << endl;
284 }
285
286 static void emit_blt(char *src1, char *src2,
    int label, ostream &s)
287 {
288     s << BLT << src1 << " " << src2 << " ";
289     emit_label_ref(label,s);
290     s << endl;
291 }
292
293 static void emit_blti(char *src1, int imm, int
    label, ostream &s)
294 {
295     s << BLT << src1 << " " << imm << " ";
296     emit_label_ref(label,s);
297     s << endl;
298 }
299
300 static void emit_bgti(char *src1, int imm, int
    label, ostream &s)
301 {
302     s << BGT << src1 << " " << imm << " ";
303     emit_label_ref(label,s);
304     s << endl;
305 }
306
307 static void emit_branch(int l, ostream& s)
308 {
309     s << BRANCH;
310     emit_label_ref(l,s);
311     s << endl;
312 }
313
314 //
315 // Push a register on the stack. The stack
    grows towards smaller addresses.
316 //
317 static void emit_push(char *reg, ostream& str)
318 {
319     emit_store(reg,0,SP,str);
320     emit_addiu(SP,SP,-4,str);
321 }

```

```

322
323 //
324 // Fetch the integer value in an Int object.
325 // Emits code to fetch the integer value of the
    Integer object pointed
326 // to by register source into the register dest
327 //
328 static void emit_fetch_int(char *dest, char *
    source, ostream& s)
329 { emit_load(dest, DEFAULT_OBJFIELDS, source, s)
    ; }
330
331 //
332 // Emits code to store the integer value
    contained in register source
333 // into the Integer object pointed to by dest.
334 //
335 static void emit_store_int(char *source, char *
    dest, ostream& s)
336 { emit_store(source, DEFAULT_OBJFIELDS, dest, s
    ); }
337
338
339 static void emit_test_collector(ostream &s)
340 {
341     emit_push(ACC, s);
342     emit_move(ACC, SP, s); // stack end
343     emit_move(A1, ZERO, s); // allocate nothing
344     s << JAL << gc_collect_names[cgen_Memmgr] <<
        endl;
345     emit_addiu(SP,SP,4,s);
346     emit_load(ACC,0,SP,s);
347 }
348
349 static void emit_gc_check(char *source, ostream
    &s)
350 {
351     if (source != (char*)A1) emit_move(A1, source
        , s);
352     s << JAL << "_gc_check" << endl;
353 }
354
355
356 //
    //////////////////////////////////////
357 //
358 // coding strings, ints, and booleans
359 //
360 // Cool has three kinds of constants: strings,

```

```

361     ints, and booleans.
362 // This section defines code generation for
363     each type.
364 //
365 // All string constants are listed in the
366     global "stringtable" and have
367 // type StringEntry. StringEntry methods are
368     defined both for String
369 // constant definitions and references.
370 //
371 // All integer constants are listed in the
372     global "inttable" and have
373 // type IntEntry. IntEntry methods are defined
374     for Int
375 // constant definitions and references.
376 //
377 // Since there are only two Bool values, there
378     is no need for a table.
379 // The two booleans are represented by
380     instances of the class BoolConst,
381 // which defines the definition and reference
382     methods for Bools.
383 //
384 //
385 //////////////////////////////////////
386
387 //
388 // Strings
389 //
390 void StringEntry::code_ref(ostream& s)
391 {
392     s << STRCONST_PREFIX << index;
393 }
394 //
395 // Emit code for a constant String.
396 // You should fill in the code naming the
397     dispatch table.
398 //
399 void StringEntry::code_def(ostream& s, int
400     stringclasstag)
401 {
402     IntEntryP lensym = inttable.add_int(len);
403
404     // Add -1 eye catcher
405     s << WORD << "-1" << endl;
406
407     code_ref(s); s << LABEL
408
409     // label
410     << WORD << stringclasstag << endl
411     // tag
412     << WORD << (DEFAULT_OBJFIELDS + STRING_
413     SLOTS + (len+4)/4) << endl // size
414     << WORD;
415
416     /***** Add dispatch information for class
417     String *****/
418     emit_disptable_ref(Str, s);
419
420     s << endl;
421     // dispatch table
422     s << WORD; lensym->code_ref(s); s <<
423     endl; // string length
424     emit_string_constant(s, str);
425     // ascii string
426     s << ALIGN;
427     // align to word
428 }
429 //
430 // StrTable::code_string
431 // Generate a string object definition for
432     every string constant in the
433 // stringtable.
434 //
435 void StrTable::code_string_table(ostream& s,
436     int stringclasstag)
437 {
438     for (List<StringEntry> *l = tbl; l; l = l->tl
439     ())
440         l->hd()->code_def(s, stringclasstag);
441 }
442 //
443 // Ints
444 //
445 void IntEntry::code_ref(ostream &s)
446 {
447     s << INTCONST_PREFIX << index;
448 }
449 //
450 // Emit code for a constant Integer.
451 // You should fill in the code naming the
452     dispatch table.
453 //
454
455

```



```

436 void IntEntry::code_def(ostream &s, int
    intclasstag)
437 {
438     // Add -1 eye catcher
439     s << WORD << "-1" << endl;
440
441     code_ref(s); s << LABEL
        // label
442     << WORD << intclasstag << endl
        // class tag
443     << WORD << (DEFAULT_OBJFIELDS + INT_SLOTS
    ) << endl // object size
444     << WORD;
445
446     /***** Add dispatch information for class Int
        *****/
447     emit_disptable_ref(Int, s);
448
449     s << endl;
        // dispatch table
450     s << WORD << str << endl;
        // integer value
451 }
452
453 //
454 // IntTable::code_string_table
455 // Generate an Int object definition for every
    Int constant in the
456 // inttable.
457 //
458 //
459 void IntTable::code_string_table(ostream &s,
    int intclasstag)
460 {
461     for (List<IntEntry> *l = tbl; l; l = l->tl())
462         l->hd()->code_def(s,intclasstag);
463 }
464
465 //
466 // Booleans
467 //
468 //
469 BoolConst::BoolConst(int i) : val(i) { assert(i
    == 0 || i == 1); }
470
471 void BoolConst::code_ref(ostream& s) const
472 {
473     s << BOOLCONST_PREFIX << val;
474 }
475

```

```

476 //
477 // Emit code for a constant Bool.
478 // You should fill in the code naming the
    dispatch table.
479 //
480
481 void BoolConst::code_def(ostream& s, int
    boolclasstag)
482 {
483     // Add -1 eye catcher
484     s << WORD << "-1" << endl;
485
486     code_ref(s); s << LABEL
        // label
487     << WORD << boolclasstag << endl
        // class tag
488     << WORD << (DEFAULT_OBJFIELDS + BOOL_
    SLOTS) << endl // object size
489     << WORD;
490
491     /***** Add dispatch information for class Bool
        *****/
492     emit_disptable_ref(Bool, s);
493
494     s << endl;
        // dispatch table
495     s << WORD << val << endl;
        // value (0 or 1)
496 }
497
498 //
    //////////////////////////////////////
499 //
500 // CgenClassTable methods
501 //
502 //
    //////////////////////////////////////
503
504 //
    *****
505 //
506 // Emit code to start the .data segment and to
    declare the global names.
507 //
508 //
509 //
    *****

```

```

510
511 void CgenClassTable::code_global_data()
512 {
513     Symbol main      = idtable.lookup_string(
514         MAINNAME);
515     Symbol string     = idtable.lookup_string(
516         STRINGNAME);
517     Symbol integer    = idtable.lookup_string(
518         INTNAME);
519     Symbol boolc      = idtable.lookup_string(
520         BOOLNAME);
521
522     str << "\t.data\n" << ALIGN;
523     //
524     // The following global names must be defined
525     // first.
526     //
527     str << GLOBAL << CLASSNAMETAB << endl;
528     str << GLOBAL; emit_protobj_ref(main,str);
529     str << endl;
530     str << GLOBAL; emit_protobj_ref(integer,str);
531     str << endl;
532     str << GLOBAL; emit_protobj_ref(string,str);
533     str << endl;
534     str << GLOBAL; falsebool.code_ref(str); str
535     << endl;
536     str << GLOBAL; truebool.code_ref(str); str
537     << endl;
538     str << GLOBAL << INTTAG << endl;
539     str << GLOBAL << BOOLTAG << endl;
540     str << GLOBAL << STRINGTAG << endl;
541
542     //
543     // We also need to know the tag of the Int,
544     // String, and Bool classes
545     // during code generation.
546     //
547     str << INTTAG << LABEL
548     << WORD << intclasstag << endl;
549     str << BOOLTAG << LABEL
550     << WORD << boolclasstag << endl;
551     str << STRINGTAG << LABEL
552     << WORD << stringclasstag << endl;
553 }
554
555 //
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

587 str << WORD << gc_collect_names[cgen_Memmgr]
    << endl;
588 str << GLOBAL << "_MemMgr_TEST" << endl;
589 str << "_MemMgr_TEST:" << endl;
590 str << WORD << (cgen_Memmgr_Test == GC_TEST)
    << endl;
591 }
592
593
594 //
    *****
595 //
596 // Emit code to reserve space for and
    initialize all of
597 // the constants. Class names should have been
    added to
598 // the string table (in the supplied code, is
    is done
599 // during the construction of the inheritance
    graph), and
600 // code for emitting string constants as a side
    effect adds
601 // the string's length to the integer table.
    The constants
602 // are emitted by running through the
    stringtable and inttable
603 // and producing code for each entry.
604 //
605 //
    *****
606
607 void CgenClassTable::code_constants()
608 {
609     //
610     // Add constants that are required by the
    code generator.
611     //
612     stringtable.add_string("");
613     inttable.add_string("0");
614
615     stringtable.code_string_table(str,
        stringclasstag);
616     inttable.code_string_table(str, intclasstag);
617     code_bools(boolclasstag);
618 }
619
620
621 CgenClassTable::CgenClassTable(Classes classes,

```

```

    ostream& s) : nds(NULL) , str(s)
622 {
623     enterscope();
624     if (cgen_debug) cout << "Building
        CgenClassTable" << endl;
625     install_basic_classes();
626     install_classes(classes);
627     build_inheritance_tree();
628
629     // We need to set the tags for basic classes
    intclasstag = probe(Int)->class_tag;
631     stringclasstag = probe(Str)->class_tag;
632     boolclasstag = probe(Bool)->class_tag;
633
634     code();
635     exitscope();
636 }
637
638 void CgenClassTable::install_basic_classes()
639 {
640
641     // The tree package uses these globals to
    annotate the classes built below.
642     //curr_lineno = 0;
643     Symbol filename = stringtable.add_string("<
        basic class>");
644
645     //
646     // A few special class names are installed in
    the lookup table but not
647     // the class list. Thus, these classes exist,
    but are not part of the
648     // inheritance hierarchy.
649     // No_class serves as the parent of Object and
    the other special classes.
650     // SELF_TYPE is the self class; it cannot be
    redefined or inherited.
651     // prim_slot is a class known to the code
    generator.
652     //
653     addid(No_class,
654         new CgenNode(class_(No_class, No_class, nil_
            Features(), filename),
                Basic, this));
655
656     addid(SELF_TYPE,
657         new CgenNode(class_(SELF_TYPE, No_class, nil_
            Features(), filename),
                Basic, this));
658
659     addid(prim_slot,
660         new CgenNode(class_(prim_slot, No_class, nil_

```

```

        Features(),filename),
        Basic,this));
661
662
663 //
664 // The Object class has no parent class. Its
        methods are
665 //      cool_abort() : Object      aborts the
        program
666 //      type_name() : Str          returns a
        string representation of class name
667 //      copy() : SELF_TYPE        returns a
        copy of the object
668 //
669 // There is no need for method bodies in the
        basic classes---these
670 // are already built in to the runtime system.
671 //
672 install_class(
673     new CgenNode(
674         class_(Object,
675             No_class,
676             append_Features(
677                 append_Features(
678                     single_Features(method(cool_abort,
679                         nil_Formals(), Object, no_expr())),
680                     single_Features(method(type_name,
681                         nil_Formals(), Str, no_expr()))),
682                     single_Features(method(copy, nil_
683                         Formals(), SELF_TYPE, no_expr()))),
684             filename),
685             Basic,this));
686
687 //
688 // The IO class inherits from Object. Its
        methods are
689 //      out_string(Str) : SELF_TYPE
        writes a string to the output
690 //      out_int(Int) : SELF_TYPE
        "    an int    " "    "
691 //      in_string() : Str
        reads a string from the input
692 //      in_int() : Int
        "    an int    " "    "
693 //
694 install_class(
695     new CgenNode(
696         class_(IO,
697             Object,
698             append_Features(
699                 append_Features(

```

```

        append_Features(
697             single_Features(method(out_string,
698                 single_Formals(formal(arg, Str)),
699                     SELF_TYPE, no_expr()))),
700             single_Features(method(out_int,
701                 single_Formals(formal(arg, Int)),
702                     SELF_TYPE, no_expr()))),
703             single_Features(method(in_string,
704                 nil_Formals(), Str, no_expr()))),
705             single_Features(method(in_int, nil_
706                 Formals(), Int, no_expr()))),
707             filename),
708             Basic,this));
709
710 //
711 // The Int class has no methods and only a
        single attribute, the
712 // "val" for the integer.
713 //
714 install_class(
715     new CgenNode(
716         class_(Int,
717             Object,
718             single_Features(attr(val, prim_slot
719                 , no_expr()))),
720             filename),
721             Basic,this));
722
723 //
724 // Bool also has only the "val" slot.
725 //
726 install_class(
727     new CgenNode(
728         class_(Bool, Object, single_Features(attr
729             (val, prim_slot, no_expr()),filename),
730             Basic,this));
731
732 //
733 // The class Str has a number of slots and
        operations:
734 //      val
        ???
735 //      str_field
        the string itself
736 //      length() : Int
        length of the string
737 //      concat(arg: Str) : Str
        string concatenation
738 //      substr(arg: Int, arg2: Int): Str

```

```

734 substring
735 //
736 install_class(
737     new CgenNode(
738         class_(Str,
739             Object,
740             append_Features(
741                 append_Features(
742                     append_Features(
743                         single_Features(attr(val, Int, no_
744                             expr())),
745                         single_Features(attr(str_field,
746                             prim_slot, no_expr()))),
747                         single_Features(method(length, nil_
748                             Formals(), Int, no_expr()))),
749                         single_Features(method(concat,
750                             single_Formalis(formal(arg, Str)),
751                             Str,
752                             no_expr()))),
753                         single_Features(method(substr,
754                             append_Formalis(single_Formalis(formal
755                             (arg, Int)),
756                             single_Formalis(formal(arg2, Int))
757                             )),
758                             Str,
759                             no_expr()))),
760                             filename),
761                             Basic,this));
762 }
763 // CgenClassTable::install_class
764 // CgenClassTable::install_classes
765 //
766 // install_classes enters a list of classes in
767 // the symbol table.
768 //
769 void CgenClassTable::install_class(CgenNodeP nd
770 )
771 {
772     Symbol name = nd->get_name();
773     if (probe(name))
774     {
775         return;
776     }
777     // The class name is legal, so add it to the
778     // list of classes

```

```

775 // and the symbol table.
776 nds = new List<CgenNode>(nd,nds);
777 addid(name,nd);
778 }
779 void CgenClassTable::install_classes(Classes cs
780 )
781 {
782     for(int i = cs->first(); cs->more(i); i = cs
783         ->next(i))
784         install_class(new CgenNode(cs->nth(i),
785             NotBasic,this));
786 }
787 // CgenClassTable::build_inheritance_tree
788 //
789 void CgenClassTable::build_inheritance_tree()
790 {
791     for(List<CgenNode> *l = nds; l; l = l->tl())
792         set_relations(l->hd());
793 }
794 //
795 // CgenClassTable::set_relations
796 //
797 // Takes a CgenNode and locates its, and its
798 // parent's, inheritance nodes
799 // via the class table. Parent and child
800 // pointers are added as appropriate.
801 void CgenClassTable::set_relations(CgenNodeP nd
802 )
803 {
804     CgenNode *parent_node = probe(nd->get_parent
805         ());
806     nd->set_parentnd(parent_node);
807     parent_node->add_child(nd);
808 }
809 void CgenNode::add_child(CgenNodeP n)
810 {
811     children = new List<CgenNode>(n,children);
812 }
813 void CgenNode::set_parentnd(CgenNodeP p)
814 {
815     assert(parentnd == NULL);
816     assert(p != NULL);
817     parentnd = p;

```

```

818 }
819
820
821
822 void CgenClassTable::code()
823 {
824     if (cgen_debug) cout << "coding global data"
825         << endl;
826     code_global_data();
827
828     if (cgen_debug) cout << "choosing gc" << endl
829         ;
830     code_select_gc();
831
832     if (cgen_debug) cout << "coding constants" <<
833         endl;
834     code_constants();
835
836     code_class_nameTab();
837     code_class_objTab();
838     code_dispatchTabs();
839     code_protObjs(str); // Helper call from
840         GetClassNodes iteration
841
842     if (cgen_debug) cout << "coding global text"
843         << endl;
844     code_global_text();
845
846     code_class_inits();
847     code_class_methods();
848 }
849
850
851 CgenNodeP CgenClassTable::root()
852 {
853     return probe(Object);
854 }
855
856 // Re-implement GetClassNodes using DFS
857     traversal from Object
858 void GetClassNodesDFS(CgenNode* node, std::
859     vector<CgenNode*>& out_list) {
860     if (!node) return;
861     out_list.push_back(node);
862     for (List<CgenNode*> l = node->get_children
863         (); l; l = l->t1()) {
864         GetClassNodesDFS(l->hd(), out_list);
865     }
866 }
867
868
869

```

```

860 std::vector<CgenNode*> CgenClassTable::
861     GetClassNodes() {
862     if (m_class_nodes.empty()) {
863         GetClassNodesDFS(root(), m_class_nodes)
864         ;
865         for(int i=0; i<(int)m_class_nodes.size
866             (); ++i) {
867             m_class_nodes[i]->class_tag = i;
868             m_class_tags[m_class_nodes[i]->name
869                 ] = i;
870         }
871     }
872     return m_class_nodes;
873 }
874
875 void CgenClassTable::code_class_nameTab() {
876     str << CLASSNAMETAB << LABEL;
877     std::vector<CgenNode*> nodes =
878         GetClassNodes();
879     for (CgenNode* node : nodes) {
880         str << WORD;
881         stringtable.lookup_string(node->name->
882             get_string())->code_ref(str);
883         str << endl;
884     }
885 }
886
887 void CgenClassTable::code_class_objTab() {
888     str << CLASSOBJTAB << LABEL;
889     std::vector<CgenNode*> nodes =
890         GetClassNodes();
891     for (CgenNode* node : nodes) {
892         str << WORD; emit_protobj_ref(node->
893             name, str); str << endl;
894         str << WORD; emit_init_ref(node->name,
895             str); str << endl;
896     }
897 }
898
899 void CgenClassTable::code_dispatchTabs() {
900     std::vector<CgenNode*> nodes =
901         GetClassNodes();
902     for (CgenNode* node : nodes) {
903         emit_disptable_ref(node->name, str);
904         str << LABEL;
905         std::vector<method_class*> methods =
906             node->GetFullMethods();
907         std::map<Symbol, Symbol> class_map =
908             node->GetDispatchClassTab();
909     }
910 }

```

```

898     for (method_class* method : methods) {
899         str << WORD;
900         emit_method_ref(class_map[method->
901             name], method->name, str);
902         str << endl;
903     }
904 }
905
906 void CgenClassTable::code_protObjs(ostream& s)
907 {
908     std::vector<CgenNode*> nodes =
909         GetClassNodes();
910     for (CgenNode* node : nodes) {
911         node->code_protObj(s);
912     }
913 }
914
915 void CgenClassTable::code_class_inits() {
916     std::vector<CgenNode*> nodes =
917         GetClassNodes();
918     for (CgenNode* node : nodes) {
919         node->code_init(str);
920     }
921 }
922
923 void CgenClassTable::code_class_methods() {
924     std::vector<CgenNode*> nodes =
925         GetClassNodes();
926     for (CgenNode* node : nodes) {
927         if (!node->basic()) {
928             node->code_methods(str);
929         }
930     }
931 }
932
933 //
934 // //////////////////////////////////////
935 //
936 // CgenNode methods
937 //
938 // //////////////////////////////////////
939
940 CgenNode::CgenNode(Class_nd, Basicness bstatus
941     , CgenClassTableP ct) :
942     class__class((const class__class &) *nd),
943     parentnd(NULL),
944     children(NULL),
945     basic_status(bstatus)
946 {
947     stringtable.add_string(name->get_string());
948     // Add class name to string table
949 }
950
951 std::vector<CgenNode*> CgenNode::GetInheritance
952 () {
953     if (inheritance.empty()) {
954         CgenNode* curr = this;
955         while (curr && curr->name != No_class)
956         {
957             inheritance.push_back(curr);
958             curr = curr->get_parentnd();
959         }
960         std::reverse(inheritance.begin(),
961             inheritance.end());
962     }
963     return inheritance;
964 }
965
966 std::vector<attr_class*> CgenNode::
967 GetFullAttribs() {
968     if (m_full_attribs.empty()) {
969         std::vector<CgenNode*> chain =
970             GetInheritance();
971         for (CgenNode* node : chain) {
972             Features feats = node->features;
973             for (int i = feats->first(); feats
974                 ->more(i); i = feats->next(i)) {
975                 if (!feats->nth(i)->IsMethod())
976                 {
977                     m_full_attribs.push_back((
978                         attr_class*)feats->nth(i));
979                 }
980             }
981         }
982         for (size_t i = 0; i < m_full_attribs.
983             size(); ++i) {
984             m_attr_idx_tab[m_full_attribs[i]
985                 ->name] = i;
986         }
987     }
988     return m_full_attribs;
989 }
990
991 std::vector<method_class*> CgenNode::
992 GetFullMethods() {
993     if (m_full_methods.empty()) {

```



```

976     std::vector<CgenNode*> chain =
GetInheritance();
977     for (CgenNode* node : chain) {
978         Features feats = node->features;
979         for (int i = feats->first(); feats
->more(i); i = feats->next(i)) {
980             if (feats->nth(i)->IsMethod())
{
981                 method_class* m = (method_
class*)feats->nth(i);
982                 if (m_dispatch_idx_tab.find
(m->name) == m_dispatch_idx_tab.end()) {
983                     m_full_methods.push_
back(m);
984                     m_dispatch_idx_tab[m->
name] = m_full_methods.size() - 1;
985                     m_dispatch_class_tab[m
->name] = node->name;
986                 } else {
987                     int idx = m_dispatch_
idx_tab[m->name];
988                     m_full_methods[idx] = m
;
989                     m_dispatch_class_tab[m
->name] = node->name;
990                 }
991             }
992         }
993     }
994 }
995 return m_full_methods;
996 }

std::map<Symbol, int> CgenNode::GetAttribIdxTab
() {
998     if (m_attr_idx_tab.empty())
GetFullAttribs();
1000     return m_attr_idx_tab;
1001 }

std::map<Symbol, int> CgenNode::
GetDispatchIdxTab() {
1003     if (m_dispatch_idx_tab.empty())
GetFullMethods();
1004     return m_dispatch_idx_tab;
1005 }

std::map<Symbol, Symbol> CgenNode::
GetDispatchClassTab() {
1007     if (m_dispatch_class_tab.empty())
GetFullMethods();
1008     return m_dispatch_class_tab;
1009 }

std::vector<attr_class*> CgenNode::GetAttribs()
{
1011     std::vector<attr_class*> ret;
1012     for(int i=features->first(); features->more
(i); i=features->next(i))
1013         if (!features->nth(i)->IsMethod()) ret.
push_back((attr_class*)features->nth(i));
1014     return ret;
1015 }

std::vector<method_class*> CgenNode::GetMethods
() {
1017     std::vector<method_class*> ret;
1018     for(int i=features->first(); features->more
(i); i=features->next(i))
1019         if (features->nth(i)->IsMethod()) ret.
push_back((method_class*)features->nth(i));
1020     return ret;
1021 }

void CgenNode::code_protObj(ostream& s) {
1023     s << WORD << "-1" << endl;
1024     emit_protobj_ref(name, s); s << LABEL;
1025     s << WORD << class_tag << endl;
1026     std::vector<attr_class*> attrs =
GetFullAttribs();
1027     s << WORD << (DEFAULT_OBJFIELDS + attrs.
size()) << endl;
1028     s << WORD; emit_disptable_ref(name, s); s
<< endl;
1029
1030     for (attr_class* attr : attrs) {
1031         s << WORD;
1032         if (attr->type_decl == Int) {
1033             inttable.lookup_string("0")->code_
ref(s);
1034         } else if (attr->type_decl == Bool) {
1035             falsebool.code_ref(s);
1036         } else if (attr->type_decl == Str) {
1037             stringtable.lookup_string("")->code
_ref(s);
1038         } else {
1039             s << "0";
1040         }
1041         s << endl;
1042     }
1043 }

void CgenNode::code_init(ostream& s) {
1046     emit_init_ref(name, s); s << LABEL;
1047

```

```

1048 // Prologue
1049 emit_addiu(SP, SP, -12, s);
1050 emit_store(FP, 3, SP, s);
1051 emit_store(SELF, 2, SP, s);
1052 emit_store(RA, 1, SP, s);
1053 emit_addiu(FP, SP, 4, s);
1054 emit_move(SELF, ACC, s);
1055
1056 if (parentnd->name != No_class) {
1057     s << JAL; emit_init_ref(parentnd->name,
1058     s); s << endl;
1059 }
1060
1061 std::vector<attr_class*> attrs = GetAttribs
1062 ();
1063 // Re-fetch mapping to ensure we have
1064 offsets
1065 std::map<Symbol, int> idx_tab =
1066 GetAttribIdxTab();
1067
1068 for (attr_class* attr : attrs) {
1069     if (!attr->init->IsEmpty()) {
1070         CgenEnvironment env;
1071         env.m_class_node = this;
1072         attr->init->code(s, env);
1073         int idx = idx_tab[attr->name];
1074         emit_store(ACC, idx + 3, SELF, s);
1075         // GC Assign if needed?
1076         // emit_addiu(A1, SELF, 4*(idx+3),
1077         s);
1078         // emit_jal("_GenGC_Assign", s);
1079         if (cgen_Memmgr == 1) { // GenGC
1080             emit_addiu(A1, SELF, 4 * (idx
1081 + 3), s);
1082             emit_jal("_GenGC_Assign", s);
1083         }
1084     }
1085 }
1086
1087 emit_move(ACC, SELF, s);
1088 // Epilogue
1089 emit_load(FP, 3, SP, s);
1090 emit_load(SELF, 2, SP, s);
1091 emit_load(RA, 1, SP, s);
1092 emit_addiu(SP, SP, 12, s);
1093 emit_return(s);
1094 }
1095
1096 void CgenNode::code_methods(ostream& s) {
1097     std::vector<method_class*> methods =

```

```

1098 GetMethods();
1099 for (method_class* method : methods) {
1100     emit_method_ref(name, method->name, s);
1101     s << LABEL;
1102     // Prologue
1103     emit_addiu(SP, SP, -12, s);
1104     emit_store(FP, 3, SP, s);
1105     emit_store(SELF, 2, SP, s);
1106     emit_store(RA, 1, SP, s);
1107     emit_addiu(FP, SP, 4, s);
1108     emit_move(SELF, ACC, s);
1109
1110     CgenEnvironment env;
1111     env.m_class_node = this;
1112     // Add formals to environment
1113     for(int i = method->formals->first();
1114     method->formals->more(i); i = method->
1115     formals->next(i)) {
1116         env.AddParam(((formal_class*)method
1117         ->formals->nth(i))->GetName());
1118     }
1119
1120     method->expr->code(s, env);
1121
1122     // Epilogue
1123     emit_load(FP, 3, SP, s);
1124     emit_load(SELF, 2, SP, s);
1125     emit_load(RA, 1, SP, s);
1126     emit_addiu(SP, SP, 12, s);
1127     // Pop args
1128     int num_args = 0;
1129     for(int i=method->formals->first();
1130     method->formals->more(i); i=method->formals
1131     ->next(i)) num_args++;
1132     emit_addiu(SP, SP, num_args * 4, s);
1133
1134     emit_return(s);
1135 }
1136 }
1137
1138 // Env Lookup Helper
1139 int CgenEnvironment::LookupAttrib(Symbol s) {
1140     if (!m_class_node) return -1;
1141     std::map<Symbol, int> idxs = m_class_node->
1142     GetAttribIdxTab();
1143     if (idxs.find(s) != idxs.end()) return idxs
1144     [s];
1145     return -1;
1146 }
1147 }
1148
1149 int CgenEnvironment::LookupAttrib(Symbol s) {
1150     if (!m_class_node) return -1;
1151     std::map<Symbol, int> idxs = m_class_node->
1152     GetAttribIdxTab();
1153     if (idxs.find(s) != idxs.end()) return idxs
1154     [s];
1155     return -1;
1156 }
1157 }

```

```

1133 //
1134 // *****
1135 // Fill in the following methods to produce
1136 // code for the
1137 // appropriate expression. You may add or
1138 // remove parameters
1139 // as you wish, but if you do, remember to
1140 // change the parameters
1141 // of the declarations in `cool-tree.h'
1142 // Sample code for
1143 // constant integers, strings, and booleans
1144 // are provided.
1145 //
1146 // *****
1147
1148 void assign_class::code(ostream &s,
1149 CgenEnvironment env) {
1150     expr->code(s, env);
1151
1152     int idx;
1153     if ((idx = env.LookUpVar(name)) != -1) {
1154         emit_store(ACC, idx + 1, SP, s);
1155         if (cgen_Memmgr == 1) {
1156             emit_addiu(A1, SP, 4 * (idx + 1),
1157 s);
1158             emit_jal("_GenGC_Assign", s);
1159         }
1160     } else if ((idx = env.LookUpParam(name)) !=
1161 -1) {
1162         emit_store(ACC, idx + 3, FP, s);
1163         if (cgen_Memmgr == 1) {
1164             emit_addiu(A1, FP, 4 * (idx + 3),
1165 s);
1166             emit_jal("_GenGC_Assign", s);
1167         }
1168     } else if ((idx = env.LookUpAttrib(name))
1169 != -1) {
1170         emit_store(ACC, idx + 3, SELF, s);
1171         if (cgen_Memmgr == 1) {
1172             emit_addiu(A1, SELF, 4 * (idx + 3),
1173 , s);
1174             emit_jal("_GenGC_Assign", s);
1175         }
1176     }
1177 }
1178
1179 void static_dispatch_class::code(ostream &s,
1180 CgenEnvironment env) {
1181     std::vector<Expression> actuals =
1182     GetActuals();
1183     for (Expression e : actuals) {
1184         e->code(s, env);
1185         emit_push(ACC, s);
1186         env.AddObstacle();
1187     }
1188
1189     expr->code(s, env);
1190
1191     int label_not_void = labelnum++;
1192     emit_bne(ACC, ZERO, label_not_void, s);
1193     emit_load_address(ACC, "str_const0", s); //
1194     // file name usually
1195     emit_load_imm(T1, 1, s); // line number
1196     emit_jal("_dispatch_abort", s);
1197     emit_label_def(label_not_void, s);
1198
1199     CgenNode* node = codegen_classtable->
1200     GetClassNode(type_name);
1201     std::string dispTab = std::string(type_name
1202     ->get_string()) + DISPTAB_SUFFIX;
1203     emit_load_address(T1, (char*)dispTab.c_str
1204     (), s);
1205
1206     int idx = node->GetDispatchIdxTab()[name];
1207     emit_load(T1, idx, T1, s);
1208     emit_jalr(T1, s);
1209 }
1210
1211 void dispatch_class::code(ostream &s,
1212 CgenEnvironment env) {
1213     std::vector<Expression> actuals =
1214     GetActuals();
1215     for (Expression e : actuals) {
1216         e->code(s, env);
1217         emit_push(ACC, s);
1218         env.AddObstacle();
1219     }
1220
1221     expr->code(s, env);
1222
1223     int label_not_void = labelnum++;
1224     emit_bne(ACC, ZERO, label_not_void, s);
1225     emit_load_address(ACC, "str_const0", s);
1226     emit_load_imm(T1, 1, s);
1227     emit_jal("_dispatch_abort", s);
1228     emit_label_def(label_not_void, s);

```

```

1210
1211     emit_load(T1, 2, ACC, s); // dispatch table
1212
1213     // Determine method offset
1214     // The offset is determined by the compile-
1215     // time type of expr.
1216     // However, for dynamic dispatch, we look
1217     // up the method index
1218     // in the class hierarchy of the *
1219     // expression's static type*.
1220     Symbol type = expr->get_type();
1221     if (type == SELF_TYPE) type = env.m_class_
1222     node->name;
1223
1224     CgenNode* node = codegen_classtable->
1225     GetClassNode(type);
1226     int idx = node->GetDispatchIdxTab()[name];
1227
1228     emit_load(T1, idx, T1, s);
1229     emit_jalr(T1, s);
1230 }
1231
1232 void cond_class::code(ostream &s,
1233     CgenEnvironment env) {
1234     pred->code(s, env);
1235     emit_fetch_int(T1, ACC, s);
1236
1237     int label_else = labelnum++;
1238     int label_end = labelnum++;
1239
1240     emit_beq(T1, ZERO, label_else, s);
1241     then_exp->code(s, env);
1242     emit_branch(label_end, s);
1243     emit_label_def(label_else, s);
1244     else_exp->code(s, env);
1245     emit_label_def(label_end, s);
1246 }
1247
1248 void loop_class::code(ostream &s,
1249     CgenEnvironment env) {
1250     int label_loop = labelnum++;
1251     int label_end = labelnum++;
1252
1253     emit_label_def(label_loop, s);
1254     pred->code(s, env);
1255     emit_fetch_int(T1, ACC, s);
1256     emit_beq(T1, ZERO, label_end, s);
1257     body->code(s, env);
1258     emit_branch(label_loop, s);
1259     emit_label_def(label_end, s);

```

```

1253     emit_move(ACC, ZERO, s); // Loop returns
1254     void
1255 }
1256
1257 void typcase_class::code(ostream &s,
1258     CgenEnvironment env) {
1259     expr->code(s, env);
1260
1261     int label_not_void = labelnum++;
1262     int label_end = labelnum++;
1263
1264     emit_bne(ACC, ZERO, label_not_void, s);
1265     emit_load_address(ACC, "str_const0", s);
1266     emit_load_imm(T1, 1, s);
1267     emit_jal("_case_abort2", s);
1268     emit_label_def(label_not_void, s);
1269
1270     emit_load(T1, 0, ACC, s); // T1 = class tag
1271
1272     // We need to iterate over cases and find
1273     // the closest ancestor.
1274     // PA5 trick: We can generate code to check
1275     // each case.
1276     // To handle "closest", we should ideally
1277     // sort branches by hierarchy depth descending
1278     // ?
1279     // Or, we check each branch. If it matches,
1280     // we record the "distance".
1281     // MIPS implementation of finding closest
1282     // match is tricky.
1283     // Alternative: Iterate branches. If tag
1284     // matches branch_type (subclass check),
1285     // jump to code block. BUT cool case
1286     // requires *best* match.
1287     // So we must check *all* branches and pick
1288     // best.
1289
1290     // Easier approach used in simple compilers
1291     :
1292     // Sort the cases by type depth (most
1293     // specific first).
1294     // Then the first match is the best match.
1295     std::vector<branch_class*> cases = GetCases
1296     ();
1297     std::sort(cases.begin(), cases.end(), [](
1298     branch_class* a, branch_class* b) {
1299         // We need codegen_classtable to check
1300         // depth.
1301         // Assuming higher tag = deeper is
1302         FALSE.

```

```

1286     // We need depth info. Let's assume
codegen_classtable is available.
1287     int d1 = 0, d2 = 0;
1288     CgenNode* n1 = codegen_classtable->
GetClassNode(a->type_decl);
1289     CgenNode* n2 = codegen_classtable->
GetClassNode(b->type_decl);
1290     while(n1->name != Object) { d1++; n1 =
n1->get_parentnd(); }
1291     while(n2->name != Object) { d2++; n2 =
n2->get_parentnd(); }
1292     return d1 > d2;
1293 };
1294
1295 // For each sorted case, check inheritance
1296 for (branch_class* b : cases) {
1297     int label_next_case = labelnum++;
1298     int tag = codegen_classtable->
GetClassTags()[b->type_decl];
1299
1300     // Check if T1 (obj tag) is subclass of
'tag'
1301     // This requires a runtime check or a
range check if we did DFS tagging.
1302     // Since we did simple tagging, we
might not have ranges.
1303     // Standard COOL Runtime has a class_
objTab. We can traverse parents up.
1304     // Or, we can use the tagging if we
implemented DFS range.
1305
1306     // Simple inefficient way compatible
with any tagging:
1307     // Traverse parent pointers of T1 until
we hit 'tag' or Object.
1308     // This is slow in MIPS.
1309     // The skeleton suggests we should rely
on what we have.
1310     // We assigned tags 0..N.
1311
1312     // Let's implement the specific logic:
1313     // Range Check is best if tags are DFS.
1314     // With DFS: [tag, tag + max_child_
index].
1315     // CgenClassTable DFS implementation
ensures this property IF we track subtree
range.
1316
1317     // Let's assume standard linear tagging
doesn't support range check easily without

```

```

a table.
1318     // But wait, `_case_abort` exists.
1319
1320     // Let's emit code:
1321     // blt T1, min_tag, next
1322     // bgt T1, max_tag, next
1323     // execute branch
1324
1325     // Calculate min/max tag for this type
1326     CgenNode* p = codegen_classtable->
GetClassNode(b->type_decl);
1327     int min_tag = codegen_classtable->
GetClassTags()[p->name];
1328     int max_tag = min_tag;
1329     // Find max tag in subtree (needs
helper, but let's assume simple approach)
1330     // Actually, we can just do a helper
function to find max tag.
1331     std::vector<CgenNode*> nodes = codegen_
classtable->GetClassNodes();
1332     // Since nodes are DFS ordered in my
implementation:
1333     // Subtree is contiguous range.
1334     for(size_t i = min_tag + 1; i < nodes.
size(); ++i) {
1335         // Check if nodes[i] is descendant
of p
1336         CgenNode* curr = nodes[i];
1337         bool is_desc = false;
1338         while(curr) {
1339             if (curr == p) { is_desc =
true; break; }
1340             curr = curr->get_parentnd();
1341         }
1342         if (!is_desc) break; // End of
contiguous range
1343         max_tag = i;
1344     }
1345
1346     emit_blti(T1, min_tag, label_next_case,
s);
1347     emit_bgti(T1, max_tag, label_next_case,
s);
1348
1349     // Match found!
1350     emit_push(ACC, s); // Bind variable
1351     env.EnterScope();
1352     env.AddVar(b->name);
1353     b->expr->code(s, env);
1354     env.ExitScope();

```

```

1355     emit_addiu(SP, SP, 4, s); // Pop
1356     variable
1357     emit_branch(label_end, s);
1358     emit_label_def(label_next_case, s);
1359 }
1360
1361 emit_jal("_case_abort", s); // No match
1362 emit_label_def(label_end, s);
1363 }
1364
1365 void block_class::code(ostream &s,
1366     CgenEnvironment env) {
1367     for (int i = body->first(); body->more(i);
1368         i = body->next(i))
1369         body->nth(i)->code(s, env);
1370 }
1371
1372 void let_class::code(ostream &s,
1373     CgenEnvironment env) {
1374     init->code(s, env);
1375     if (init->IsEmpty()) {
1376         if (type_decl == Str) {
1377             emit_load_string(ACC, stringtable.
1378                 lookup_string(""), s);
1379         } else if (type_decl == Int) {
1380             emit_load_int(ACC, inttable.lookup_
1381                 string("0"), s);
1382         } else if (type_decl == Bool) {
1383             emit_load_bool(ACC, BoolConst(0), s
1384             );
1385         } else {
1386             emit_move(ACC, ZERO, s);
1387         }
1388     }
1389     emit_push(ACC, s);
1390     env.EnterScope();
1391     env.AddVar(identifier);
1392     body->code(s, env);
1393     env.ExitScope();
1394     emit_addiu(SP, SP, 4, s);
1395 }
1396
1397 void plus_class::code(ostream &s,
1398     CgenEnvironment env) {
1399     e1->code(s, env);
1400     emit_push(ACC, s);
1401     env.AddObstacle();
1402     e2->code(s, env);
1403     emit_jal("Object.copy", s);
1404     emit_load(T1, 1, SP, s);
1405     emit_addiu(SP, SP, 4, s);
1406     emit_move(T2, ACC, s);
1407     emit_load(T1, 3, T1, s);
1408     emit_load(T2, 3, T2, s);
1409     emit_add(T3, T1, T2, s);
1410     emit_store(T3, 3, ACC, s);
1411 }
1412
1413 void sub_class::code(ostream &s,
1414     CgenEnvironment env) {
1415     e1->code(s, env);
1416     emit_push(ACC, s);
1417     env.AddObstacle();
1418     e2->code(s, env);
1419     emit_jal("Object.copy", s);
1420     emit_load(T1, 1, SP, s);
1421     emit_addiu(SP, SP, 4, s);
1422     emit_move(T2, ACC, s);
1423     emit_load(T1, 3, T1, s);
1424     emit_load(T2, 3, T2, s);
1425     emit_sub(T3, T1, T2, s);
1426     emit_store(T3, 3, ACC, s);
1427 }
1428
1429 void mul_class::code(ostream &s,
1430     CgenEnvironment env) {
1431     e1->code(s, env);
1432     emit_push(ACC, s);
1433     env.AddObstacle();
1434     e2->code(s, env);
1435     emit_jal("Object.copy", s);
1436     emit_load(T1, 1, SP, s);
1437     emit_addiu(SP, SP, 4, s);
1438     emit_move(T2, ACC, s);
1439     emit_load(T1, 3, T1, s);
1440     emit_load(T2, 3, T2, s);
1441     emit_mul(T3, T1, T2, s);
1442     emit_store(T3, 3, ACC, s);
1443 }
1444
1445 void divide_class::code(ostream &s,
1446     CgenEnvironment env) {
1447     e1->code(s, env);
1448     emit_push(ACC, s);
1449     env.AddObstacle();
1450     e2->code(s, env);
1451     emit_jal("Object.copy", s);
1452     emit_load(T1, 1, SP, s);
1453     emit_addiu(SP, SP, 4, s);

```

```

1444     emit_move(T2, ACC, s);
1445     emit_load(T1, 3, T1, s);
1446     emit_load(T2, 3, T2, s);
1447     emit_div(T3, T1, T2, s);
1448     emit_store(T3, 3, ACC, s);
1449 }
1450
1451 void neg_class::code(ostream &s,
1452     CgenEnvironment env) {
1453     e1->code(s, env);
1454     emit_jal("Object.copy", s);
1455     emit_load(T1, 3, ACC, s);
1456     emit_neg(T1, T1, s);
1457     emit_store(T1, 3, ACC, s);
1458 }
1459
1460 void lt_class::code(ostream &s, CgenEnvironment
1461     env) {
1462     e1->code(s, env);
1463     emit_push(ACC, s);
1464     env.AddObstacle();
1465     e2->code(s, env);
1466     emit_load(T1, 1, SP, s);
1467     emit_addiu(SP, SP, 4, s);
1468     emit_load(T1, 3, T1, s);
1469     emit_load(T2, 3, ACC, s); // ACC has e2
1470
1471     int label_true = labelnum++;
1472     int label_end = labelnum++;
1473     emit_load_bool(ACC, BoolConst(1), s);
1474     emit_blt(T1, T2, label_end, s);
1475     emit_load_bool(ACC, BoolConst(0), s);
1476     emit_label_def(label_end, s);
1477 }
1478
1479 void eq_class::code(ostream &s, CgenEnvironment
1480     env) {
1481     e1->code(s, env);
1482     emit_push(ACC, s);
1483     env.AddObstacle();
1484     e2->code(s, env);
1485     emit_load(T1, 1, SP, s);
1486     emit_addiu(SP, SP, 4, s);
1487     emit_move(T2, ACC, s);
1488
1489     int label_end = labelnum++;
1490     emit_load_bool(ACC, BoolConst(1), s);
1491     emit_beq(T1, T2, label_end, s);
1492     emit_load_bool(ACC, BoolConst(0), s);
1493     emit_jal("equality_test", s); // runtime
1494 }
1495
1496 helper
1497 emit_label_def(label_end, s);
1498 }
1499
1500 void leq_class::code(ostream &s,
1501     CgenEnvironment env) {
1502     e1->code(s, env);
1503     emit_push(ACC, s);
1504     env.AddObstacle();
1505     e2->code(s, env);
1506     emit_load(T1, 1, SP, s);
1507     emit_addiu(SP, SP, 4, s);
1508     emit_load(T1, 3, T1, s);
1509     emit_load(T2, 3, ACC, s);
1510
1511     int label_end = labelnum++;
1512     emit_load_bool(ACC, BoolConst(1), s);
1513     emit_bleq(T1, T2, label_end, s);
1514     emit_load_bool(ACC, BoolConst(0), s);
1515     emit_label_def(label_end, s);
1516 }
1517
1518 void comp_class::code(ostream &s,
1519     CgenEnvironment env) {
1520     e1->code(s, env);
1521     emit_load(T1, 3, ACC, s);
1522     int label_end = labelnum++;
1523     emit_load_bool(ACC, BoolConst(1), s);
1524     emit_beq(T1, ZERO, label_end, s);
1525     emit_load_bool(ACC, BoolConst(0), s);
1526     emit_label_def(label_end, s);
1527 }
1528
1529 void int_const_class::code(ostream& s,
1530     CgenEnvironment env)
1531 {
1532     emit_load_int(ACC, inttable.lookup_string(
1533         token->get_string()), s);
1534 }
1535
1536 void string_const_class::code(ostream& s,
1537     CgenEnvironment env)
1538 {
1539     emit_load_string(ACC, stringtable.lookup_
1540         string(token->get_string()), s);
1541 }
1542
1543 void bool_const_class::code(ostream& s,
1544     CgenEnvironment env)
1545 {

```



```

1533 emit_load_bool(ACC, BoolConst(val), s);
1534 }
1535
1536 void new__class::code(ostream &s,
1537   CgenEnvironment env) {
1538   if (type_name == SELF_TYPE) {
1539     emit_load_address(T1, "class_objTab", s);
1540     emit_load(T2, 0, SELF, s); // class tag
1541     emit_sll(T2, T2, 3, s); // * 8
1542     emit_addu(T1, T1, T2, s);
1543     emit_push(T1, s);
1544     emit_load(ACC, 0, T1, s); // load
1545     protObj
1546     emit_jal("Object.copy", s);
1547     emit_load(T1, 1, SP, s);
1548     emit_addiu(SP, SP, 4, s);
1549     emit_load(T1, 1, T1, s); // load init
1550     emit_jalr(T1, s);
1551   } else {
1552     std::string prot = std::string(type_
1553   name->get_string()) + PROTOBJ_SUFFIX;
1554     std::string init = std::string(type_
1555   name->get_string()) + CLASSINIT_SUFFIX;
1556     emit_load_address(ACC, (char*)prot.c_
1557   str(), s);
1558     emit_jal("Object.copy", s);
1559     emit_jal((char*)init.c_str(), s);
1560   }
1561 }
1562
1563 void isvoid_class::code(ostream &s,
1564   CgenEnvironment env) {
1565   e1->code(s, env);
1566   emit_move(T1, ACC, s);
1567   emit_load_bool(ACC, BoolConst(1), s);
1568   int label_end = labelnum++;
1569   emit_beq(T1, ZERO, label_end, s);
1570   emit_load_bool(ACC, BoolConst(0), s);
1571   emit_label_def(label_end, s);
1572 }
1573
1574 void no_expr_class::code(ostream &s,
1575   CgenEnvironment env) {
1576   // No code needed, usually handled by
1577   caller or returns void
1578   emit_move(ACC, ZERO, s);
1579 }
1580
1581 void object_class::code(ostream &s,

```

```

1574   CgenEnvironment env) {
1575     if (name == self) {
1576       emit_move(ACC, SELF, s);
1577       return;
1578     }
1579     int idx;
1580     if ((idx = env.LookUpVar(name)) != -1) {
1581       emit_load(ACC, idx + 1, SP, s);
1582     } else if ((idx = env.LookUpParam(name)) !=
1583   -1) {
1584       emit_load(ACC, idx + 3, FP, s);
1585     } else if ((idx = env.LookUpAttrib(name))
1586   != -1) {
1587       emit_load(ACC, idx + 3, SELF, s);
1588     }
1589   }

```

Listing 3: cgen.cc

A.2 cgen.h

```

1 #include <assert.h>
2 #include <stdio.h>
3 #include "emit.h"
4 #include "cool-tree.h"
5 #include "symtab.h"
6 #include <vector>
7 #include <map>
8 #include <algorithm>
9
10 enum Basicness {Basic, NotBasic};
11 #define TRUE 1
12 #define FALSE 0
13
14 class CgenClassTable;
15 typedef CgenClassTable *CgenClassTableP;
16
17 class CgenNode;
18 typedef CgenNode *CgenNodeP;
19
20 extern Symbol No_type; // Fixed: Added external
21   declaration
22 // Add Environment for PA5
23 class CgenEnvironment {
24 private:
25   SymbolTable<Symbol, int> *vars;
26   int param_offset_base;

```

```

27     int var_offset_base;
28     int cur_param_offset;
29     int cur_var_offset;
30
31 public:
32     CgenNode* m_class_node;
33
34     CgenEnvironment() {
35         vars = new SymbolTable<Symbol, int>();
36         vars->enterscope();
37         param_offset_base = 3;
38         var_offset_base = 0;
39     }
40
41     void EnterScope() { vars->enterscope(); }
42     void ExitScope() { vars->exitscope(); }
43
44     void AddVar(Symbol sym) {
45         vars->addid(sym, new int(0));
46     }
47
48     std::vector<Symbol> var_stack;
49
50     void AddVarNode(Symbol s) {
51         var_stack.push_back(s);
52     }
53
54     void RemoveVarNode() {
55         var_stack.pop_back();
56     }
57
58     int LookUpVar(Symbol s) {
59         for (int i = var_stack.size() - 1; i >=
60             0; --i) {
61             if (var_stack[i] == s) return var_
62                 stack.size() - 1 - i;
63         }
64         return -1;
65     }
66
67     std::vector<Symbol> param_stack;
68     void AddParam(Symbol s) {
69         param_stack.push_back(s);
70     }
71
72     int LookUpParam(Symbol s) {
73         for (int i = 0; i < (int)param_stack.
74             size(); ++i) { // Fixed: Cast to int
75             if (param_stack[i] == s) return
76                 param_stack.size() - 1 - i;

```

```

73     }
74     return -1;
75 }
76
77     int LookUpAttrib(Symbol s);
78     void AddObstacle() { var_stack.push_back(No
79         _type); } // Fixed: No_type now declared
80 };
81
82 class CgenClassTable : public SymbolTable<
83     Symbol, CgenNode> {
84 private:
85     List<CgenNode> *nds;
86     ostream& str;
87     int stringclasstag;
88     int intclasstag;
89     int boolclasstag;
90
91     std::vector<CgenNode*> m_class_nodes;
92     std::map<Symbol, int> m_class_tags;
93
94     // The following methods emit code for
95     // constants and global declarations.
96
97     void code_global_data();
98     void code_global_text();
99     void code_bools(int);
100     void code_select_gc();
101     void code_constants();
102
103     void code_class_nameTab();
104     void code_class_objTab();
105     void code_dispatchTabs();
106     void code_class_inits();
107     void code_class_methods();
108     void code_protObjs(ostream& s); // Fixed:
109     // Added declaration
110
111     // The following creates an inheritance graph
112     // from
113     // a list of classes. The graph is implemented
114     // as
115     // a tree of `CgenNode', and class names are
116     // placed
117     // in the base class symbol table.
118
119     void install_basic_classes();
120     void install_class(CgenNodeP nd);
121     void install_classes(Classes cs);
122     void build_inheritance_tree();

```

```

117 void set_relations(CgenNodeP nd);
118 public:
119   CgenClassTable(Classes, ostream& str);
120   void code();
121   CgenNodeP root();
122   CgenNode* GetClassNode(Symbol name) { return
       probe(name); }
123   std::vector<CgenNode*> GetClassNodes();
124   std::map<Symbol, int> GetClassTags() {
       return m_class_tags; }
125 };
126
127 class CgenNode : public class__class {
128 private:
129   CgenNodeP parentnd;
130   // Parent of class
131   List<CgenNode> *children;
132   // Children of class
133   Basicness basic_status;
134   // 'Basic' if class is basic
135
136   // 'NotBasic' otherwise
137
138   std::vector<CgenNode*> inheritance;
139   std::vector<attr_class*> m_full_attribs;
140   std::map<Symbol, int> m_attrib_idx_tab;
141
142   std::vector<method_class*> m_full_methods;
143   std::map<Symbol, int> m_dispatch_idx_tab;
144   std::map<Symbol, Symbol> m_dispatch_class_
       tab; // method name -> class name
145
146 public:
147   CgenNode(Class_ c,
148             Basicness bstatus,
149             CgenClassTableP class_table);
150
151   void add_child(CgenNodeP child);
152   List<CgenNode> *get_children() { return
       children; }
153   void set_parentnd(CgenNodeP p);
154   CgenNodeP get_parentnd() { return parentnd;
       }
155   int basic() { return (basic_status == Basic)
       ; }
156   int class_tag;
157
158   std::vector<CgenNode*> GetInheritance();

```

```

157   std::vector<attr_class*> GetFullAttribs();
158   std::map<Symbol, int> GetAttribIdxTab();
159
160   std::vector<method_class*> GetFullMethods();
161   std::map<Symbol, int> GetDispatchIdxTab();
162   std::map<Symbol, Symbol> GetDispatchClassTab
       ();
163
164   std::vector<attr_class*> GetAttribs();
165   std::vector<method_class*> GetMethods();
166
167   void code_protObj(ostream& s);
168   void code_init(ostream& s);
169   void code_methods(ostream& s);
170 };
171
172 class BoolConst
173 {
174 private:
175   int val;
176 public:
177   BoolConst(int);
178   void code_def(ostream&, int boolclasstag);
179   void code_ref(ostream&) const;
180 };

```

Listing 4: cgen.h

A.3 cool-tree.h

```

1
2 #ifndef COOL_TREE_H
3 #define COOL_TREE_H
4
5 #include "tree.h"
6 #include "cool-tree.handcode.h"
7
8 // define simple phylum - Program
9 typedef class Program_class *Program;
10
11 class Program_class : public tree_node {
12 public:
13   tree_node *copy() { return copy_Program
       (); }
14   virtual Program copy_Program() = 0;
15
16 #ifdef Program_EXTRAS
17   Program_EXTRAS
18 #endif

```

```

19 };
20
21
22 // define simple phylum - Class_
23 typedef class Class__class *Class_;
24
25 class Class__class : public tree_node {
26 public:
27     tree_node *copy()      { return copy_Class_()
28         ; }
29     virtual Class_ copy_Class_() = 0;
30
31 #ifdef Class__EXTRAS
32     Class__EXTRAS
33 #endif
34 };
35
36 // define simple phylum - Feature
37 typedef class Feature_class *Feature;
38
39 class Feature_class : public tree_node {
40 public:
41     tree_node *copy()      { return copy_Feature
42         (); }
43     virtual Feature copy_Feature() = 0;
44
45 #ifdef Feature_EXTRAS
46     Feature_EXTRAS
47 #endif
48 };
49
50 // define simple phylum - Formal
51 typedef class Formal_class *Formal;
52
53 class Formal_class : public tree_node {
54 public:
55     tree_node *copy()      { return copy_Formal()
56         ; }
57     virtual Formal copy_Formal() = 0;
58
59 #ifdef Formal_EXTRAS
60     Formal_EXTRAS
61 #endif
62 };
63
64 // define simple phylum - Expression
65 typedef class Expression_class *Expression;

```

```

66
67 class Expression_class : public tree_node {
68 public:
69     tree_node *copy()      { return copy_
70         Expression(); }
71     virtual Expression copy_Expression() = 0;
72
73 #ifdef Expression_EXTRAS
74     Expression_EXTRAS
75 #endif
76 };
77
78 // define simple phylum - Case
79 typedef class Case_class *Case;
80
81 class Case_class : public tree_node {
82 public:
83     tree_node *copy()      { return copy_Case();
84         }
85     virtual Case copy_Case() = 0;
86
87 #ifdef Case_EXTRAS
88     Case_EXTRAS
89 #endif
90 };
91
92 // define list phylum - Classes
93 typedef list_node<Class_> Classes_class;
94 typedef Classes_class *Classes;
95
96
97 // define list phylum - Features
98 typedef list_node<Feature> Features_class;
99 typedef Features_class *Features;
100
101
102 // define list phylum - Formals
103 typedef list_node<Formal> Formals_class;
104 typedef Formals_class *Formals;
105
106
107 // define list phylum - Expressions
108 typedef list_node<Expression> Expressions_class
109 ;
110 typedef Expressions_class *Expressions;
111
112 // define list phylum - Cases

```

```

113 typedef list_node<Case> Cases_class;
114 typedef Cases_class *Cases;
115
116 // define the class for constructors
117 // define constructor - program
118 class program_class : public Program_class {
119 public:
120     Classes classes;
121 public:
122     program_class(Classes a1) {
123         classes = a1;
124     }
125     Program copy_Program();
126     void dump(ostream& stream, int n);
127
128 #ifdef Program_SHARED_EXTRAS
129     Program_SHARED_EXTRAS
130 #endif
131 #ifdef program_EXTRAS
132     program_EXTRAS
133 #endif
134 };
135
136 // define constructor - class_
137 class class__class : public Class__class {
138 public:
139     Symbol name;
140     Symbol parent;
141     Features features;
142     Symbol filename;
143 public:
144     class__class(Symbol a1, Symbol a2, Features
145         a3, Symbol a4) {
146         name = a1;
147         parent = a2;
148         features = a3;
149         filename = a4;
150     }
151     Class_ copy_Class_();
152     void dump(ostream& stream, int n);
153
154 #ifdef Class__SHARED_EXTRAS
155     Class__SHARED_EXTRAS
156 #endif
157 #ifdef class__EXTRAS
158     class__EXTRAS
159 #endif
160 };

```

```

162
163 // define constructor - method
164 class method_class : public Feature_class {
165 public:
166     Symbol name;
167     Formals formals;
168     Symbol return_type;
169     Expression expr;
170 public:
171     method_class(Symbol a1, Formals a2, Symbol
172         a3, Expression a4) {
173         name = a1;
174         formals = a2;
175         return_type = a3;
176         expr = a4;
177     }
178     Feature copy_Feature();
179     void dump(ostream& stream, int n);
180     Symbol GetName() { return name; }
181
182 #ifdef Feature_SHARED_EXTRAS
183     Feature_SHARED_EXTRAS
184 #endif
185 #ifdef method_EXTRAS
186     method_EXTRAS
187 #endif
188 };
189
190 // define constructor - attr
191 class attr_class : public Feature_class {
192 public:
193     Symbol name;
194     Symbol type_decl;
195     Expression init;
196 public:
197     attr_class(Symbol a1, Symbol a2, Expression
198         a3) {
199         name = a1;
200         type_decl = a2;
201         init = a3;
202     }
203     Feature copy_Feature();
204     void dump(ostream& stream, int n);
205     Symbol GetName() { return name; }
206
207 #ifdef Feature_SHARED_EXTRAS
208     Feature_SHARED_EXTRAS
209 #endif

```

```

210 #ifdef attr_EXTRAS
211     attr_EXTRAS
212 #endif
213 };
214
215
216 // define constructor - formal
217 class formal_class : public Formal_class {
218 public:
219     Symbol name;
220     Symbol type_decl;
221 public:
222     formal_class(Symbol a1, Symbol a2) {
223         name = a1;
224         type_decl = a2;
225     }
226     Formal copy_Forma();
227     void dump(ostream& stream, int n);
228     Symbol GetName() { return name; }
229
230 #ifdef Formal_SHARED_EXTRAS
231     Formal_SHARED_EXTRAS
232 #endif
233 #ifdef formal_EXTRAS
234     formal_EXTRAS
235 #endif
236 };
237
238
239 // define constructor - branch
240 class branch_class : public Case_class {
241 public:
242     Symbol name;
243     Symbol type_decl;
244     Expression expr;
245 public:
246     branch_class(Symbol a1, Symbol a2,
247         Expression a3) {
248         name = a1;
249         type_decl = a2;
250         expr = a3;
251     }
252     Case copy_Case();
253     void dump(ostream& stream, int n);
254     Symbol GetName() { return name; }
255     Symbol GetTypeDecl() { return type_decl; }
256     Expression GetExpr() { return expr; }
257
258 #ifdef Case_SHARED_EXTRAS
259     Case_SHARED_EXTRAS

```

```

259 #endif
260 #ifdef branch_EXTRAS
261     branch_EXTRAS
262 #endif
263 };
264
265
266 // define constructor - assign
267 class assign_class : public Expression_class {
268 public:
269     Symbol name;
270     Expression expr;
271 public:
272     assign_class(Symbol a1, Expression a2) {
273         name = a1;
274         expr = a2;
275     }
276     Expression copy_Expression();
277     void dump(ostream& stream, int n);
278
279 #ifdef Expression_SHARED_EXTRAS
280     Expression_SHARED_EXTRAS
281 #endif
282 #ifdef assign_EXTRAS
283     assign_EXTRAS
284 #endif
285 };
286
287
288 // define constructor - static_dispatch
289 class static_dispatch_class : public Expression
290     _class {
291 public:
292     Expression expr;
293     Symbol type_name;
294     Symbol name;
295     Expressions actual;
296 public:
297     static_dispatch_class(Expression a1, Symbol
298         a2, Symbol a3, Expressions a4) {
299         expr = a1;
300         type_name = a2;
301         name = a3;
302         actual = a4;
303     }
304     Expression copy_Expression();
305     void dump(ostream& stream, int n);
306     std::vector<Expression> GetActuals() {
307         std::vector<Expression> ret;
308         for (int i = actual->first(); actual->

```

```

307     more(i); i = actual->next(i))
308     ret.push_back(actual->nth(i));
309     return ret;
310 }
311 #ifdef Expression_SHARED_EXTRAS
312     Expression_SHARED_EXTRAS
313 #endif
314 #ifdef static_dispatch_EXTRAS
315     static_dispatch_EXTRAS
316 #endif
317 };
318
319 // define constructor - dispatch
320 class dispatch_class : public Expression_class
321 {
322 public:
323     Expression expr;
324     Symbol name;
325     Expressions actual;
326 public:
327     dispatch_class(Expression a1, Symbol a2,
328         Expressions a3) {
329         expr = a1;
330         name = a2;
331         actual = a3;
332     }
333     Expression copy_Expression();
334     void dump(ostream& stream, int n);
335     std::vector<Expression> GetActuals() {
336         std::vector<Expression> ret;
337         for (int i = actual->first(); actual->
338             more(i); i = actual->next(i))
339             ret.push_back(actual->nth(i));
340         return ret;
341     }
342 #ifdef Expression_SHARED_EXTRAS
343     Expression_SHARED_EXTRAS
344 #endif
345 #ifdef dispatch_EXTRAS
346     dispatch_EXTRAS
347 #endif
348 };
349
350 // define constructor - cond
351 class cond_class : public Expression_class {
352 public:

```

```

353     Expression pred;
354     Expression then_exp;
355     Expression else_exp;
356 public:
357     cond_class(Expression a1, Expression a2,
358         Expression a3) {
359         pred = a1;
360         then_exp = a2;
361         else_exp = a3;
362     }
363     Expression copy_Expression();
364     void dump(ostream& stream, int n);
365 #ifdef Expression_SHARED_EXTRAS
366     Expression_SHARED_EXTRAS
367 #endif
368 #ifdef cond_EXTRAS
369     cond_EXTRAS
370 #endif
371 };
372
373 // define constructor - loop
374 class loop_class : public Expression_class {
375 public:
376     Expression pred;
377     Expression body;
378 public:
379     loop_class(Expression a1, Expression a2) {
380         pred = a1;
381         body = a2;
382     }
383     Expression copy_Expression();
384     void dump(ostream& stream, int n);
385 #ifdef Expression_SHARED_EXTRAS
386     Expression_SHARED_EXTRAS
387 #endif
388 #ifdef loop_EXTRAS
389     loop_EXTRAS
390 #endif
391 };
392
393 // define constructor - typcase
394 class typcase_class : public Expression_class {
395 public:
396     Expression expr;
397     Cases cases;
398 public:

```



```

402     typcase_class(Expression a1, Cases a2) {
403         expr = a1;
404         cases = a2;
405     }
406     Expression copy_Expression();
407     void dump(ostream& stream, int n);
408     std::vector<branch_class*> GetCases() {
409         std::vector<branch_class*> ret;
410         for (int i = cases->first(); cases->more(
411             i); i = cases->next(i))
412             ret.push_back((branch_class*)cases->
413                 nth(i));
414         return ret;
415     }
416 #ifdef Expression_SHARED_EXTRAS
417     Expression_SHARED_EXTRAS
418 #endif
419 #ifdef typcase_EXTRAS
420     typcase_EXTRAS
421 #endif
422 };
423
424 // define constructor - block
425 class block_class : public Expression_class {
426 public:
427     Expressions body;
428 public:
429     block_class(Expressions a1) {
430         body = a1;
431     }
432     Expression copy_Expression();
433     void dump(ostream& stream, int n);
434
435 #ifdef Expression_SHARED_EXTRAS
436     Expression_SHARED_EXTRAS
437 #endif
438 #ifdef block_EXTRAS
439     block_EXTRAS
440 #endif
441 };
442
443 // define constructor - let
444 class let_class : public Expression_class {
445 public:
446     Symbol identifier;
447     Symbol type_decl;
448     Expression init;
449     Expression body;
450 public:
451     let_class(Symbol a1, Symbol a2, Expression
452         a3, Expression a4) {
453         identifier = a1;
454         type_decl = a2;
455         init = a3;
456         body = a4;
457     }
458     Expression copy_Expression();
459     void dump(ostream& stream, int n);
460
461 #ifdef Expression_SHARED_EXTRAS
462     Expression_SHARED_EXTRAS
463 #endif
464 #ifdef let_EXTRAS
465     let_EXTRAS
466 #endif
467 };
468
469 // define constructor - plus
470 class plus_class : public Expression_class {
471 public:
472     Expression e1;
473     Expression e2;
474 public:
475     plus_class(Expression a1, Expression a2) {
476         e1 = a1;
477         e2 = a2;
478     }
479     Expression copy_Expression();
480     void dump(ostream& stream, int n);
481
482 #ifdef Expression_SHARED_EXTRAS
483     Expression_SHARED_EXTRAS
484 #endif
485 #ifdef plus_EXTRAS
486     plus_EXTRAS
487 #endif
488 };
489
490 // define constructor - sub
491 class sub_class : public Expression_class {
492 public:
493     Expression e1;
494     Expression e2;
495 public:
496     sub_class(Expression a1, Expression a2) {

```

```

499     e1 = a1;
500     e2 = a2;
501 }
502 Expression copy_Expression();
503 void dump(ostream& stream, int n);
504
505 #ifdef Expression_SHARED_EXTRAS
506     Expression_SHARED_EXTRAS
507 #endif
508 #ifdef sub_EXTRAS
509     sub_EXTRAS
510 #endif
511 };
512
513 // define constructor - mul
514 class mul_class : public Expression_class {
515 public:
516     Expression e1;
517     Expression e2;
518 public:
519     mul_class(Expression a1, Expression a2) {
520         e1 = a1;
521         e2 = a2;
522     }
523     Expression copy_Expression();
524     void dump(ostream& stream, int n);
525
526 #ifdef Expression_SHARED_EXTRAS
527     Expression_SHARED_EXTRAS
528 #endif
529 #ifdef mul_EXTRAS
530     mul_EXTRAS
531 #endif
532 };
533
534 // define constructor - divide
535 class divide_class : public Expression_class {
536 public:
537     Expression e1;
538     Expression e2;
539 public:
540     divide_class(Expression a1, Expression a2) {
541         e1 = a1;
542         e2 = a2;
543     }
544     Expression copy_Expression();
545     void dump(ostream& stream, int n);
546

```

```

549 #ifdef Expression_SHARED_EXTRAS
550     Expression_SHARED_EXTRAS
551 #endif
552 #ifdef divide_EXTRAS
553     divide_EXTRAS
554 #endif
555 };
556
557 // define constructor - neg
558 class neg_class : public Expression_class {
559 public:
560     Expression e1;
561 public:
562     neg_class(Expression a1) {
563         e1 = a1;
564     }
565     Expression copy_Expression();
566     void dump(ostream& stream, int n);
567
568 #ifdef Expression_SHARED_EXTRAS
569     Expression_SHARED_EXTRAS
570 #endif
571 #ifdef neg_EXTRAS
572     neg_EXTRAS
573 #endif
574 };
575
576 // define constructor - lt
577 class lt_class : public Expression_class {
578 public:
579     Expression e1;
580     Expression e2;
581 public:
582     lt_class(Expression a1, Expression a2) {
583         e1 = a1;
584         e2 = a2;
585     }
586     Expression copy_Expression();
587     void dump(ostream& stream, int n);
588
589 #ifdef Expression_SHARED_EXTRAS
590     Expression_SHARED_EXTRAS
591 #endif
592 #ifdef lt_EXTRAS
593     lt_EXTRAS
594 #endif
595 };
596

```

```

599
600 // define constructor - eq
601 class eq_class : public Expression_class {
602 public:
603     Expression e1;
604     Expression e2;
605 public:
606     eq_class(Expression a1, Expression a2) {
607         e1 = a1;
608         e2 = a2;
609     }
610     Expression copy_Expression();
611     void dump(ostream& stream, int n);
612
613 #ifdef Expression_SHARED_EXTRAS
614     Expression_SHARED_EXTRAS
615 #endif
616 #ifdef eq_EXTRAS
617     eq_EXTRAS
618 #endif
619 };
620
621
622 // define constructor - leq
623 class leq_class : public Expression_class {
624 public:
625     Expression e1;
626     Expression e2;
627 public:
628     leq_class(Expression a1, Expression a2) {
629         e1 = a1;
630         e2 = a2;
631     }
632     Expression copy_Expression();
633     void dump(ostream& stream, int n);
634
635 #ifdef Expression_SHARED_EXTRAS
636     Expression_SHARED_EXTRAS
637 #endif
638 #ifdef leq_EXTRAS
639     leq_EXTRAS
640 #endif
641 };
642
643
644 // define constructor - comp
645 class comp_class : public Expression_class {
646 public:
647     Expression e1;
648 public:

```

```

649     comp_class(Expression a1) {
650         e1 = a1;
651     }
652     Expression copy_Expression();
653     void dump(ostream& stream, int n);
654
655 #ifdef Expression_SHARED_EXTRAS
656     Expression_SHARED_EXTRAS
657 #endif
658 #ifdef comp_EXTRAS
659     comp_EXTRAS
660 #endif
661 };
662
663
664 // define constructor - int_const
665 class int_const_class : public Expression_class
666 {
667 public:
668     Symbol token;
669 public:
670     int_const_class(Symbol a1) {
671         token = a1;
672     }
673     Expression copy_Expression();
674     void dump(ostream& stream, int n);
675
676 #ifdef Expression_SHARED_EXTRAS
677     Expression_SHARED_EXTRAS
678 #endif
679 #ifdef int_const_EXTRAS
680     int_const_EXTRAS
681 #endif
682 };
683
684 // define constructor - bool_const
685 class bool_const_class : public Expression_
686     class {
687 public:
688     Boolean val;
689 public:
690     bool_const_class(Boolean a1) {
691         val = a1;
692     }
693     Expression copy_Expression();
694     void dump(ostream& stream, int n);
695
696 #ifdef Expression_SHARED_EXTRAS
697     Expression_SHARED_EXTRAS

```

```

697 #endif
698 #ifdef bool_const_EXTRAS
699     bool_const_EXTRAS
700 #endif
701 };
702
703
704 // define constructor - string_const
705 class string_const_class : public Expression_
706     class {
707 public:
708     Symbol token;
709 public:
710     string_const_class(Symbol a1) {
711         token = a1;
712     }
713     Expression copy_Expression();
714     void dump(ostream& stream, int n);
715
716 #ifdef Expression_SHARED_EXTRAS
717     Expression_SHARED_EXTRAS
718 #endif
719 #ifdef string_const_EXTRAS
720     string_const_EXTRAS
721 #endif
722 };
723
724 // define constructor - new_
725 class new__class : public Expression_class {
726 public:
727     Symbol type_name;
728 public:
729     new__class(Symbol a1) {
730         type_name = a1;
731     }
732     Expression copy_Expression();
733     void dump(ostream& stream, int n);
734
735 #ifdef Expression_SHARED_EXTRAS
736     Expression_SHARED_EXTRAS
737 #endif
738 #ifdef new__EXTRAS
739     new__EXTRAS
740 #endif
741 };
742
743
744 // define constructor - isvoid
745 class isvoid_class : public Expression_class {

```

```

746 public:
747     Expression e1;
748 public:
749     isvoid_class(Expression a1) {
750         e1 = a1;
751     }
752     Expression copy_Expression();
753     void dump(ostream& stream, int n);
754
755 #ifdef Expression_SHARED_EXTRAS
756     Expression_SHARED_EXTRAS
757 #endif
758 #ifdef isvoid_EXTRAS
759     isvoid_EXTRAS
760 #endif
761 };
762
763
764 // define constructor - no_expr
765 class no_expr_class : public Expression_class {
766 public:
767 public:
768     no_expr_class() {
769     }
770     Expression copy_Expression();
771     void dump(ostream& stream, int n);
772
773 #ifdef Expression_SHARED_EXTRAS
774     Expression_SHARED_EXTRAS
775 #endif
776 #ifdef no_expr_EXTRAS
777     no_expr_EXTRAS
778 #endif
779 };
780
781
782 // define constructor - object
783 class object_class : public Expression_class {
784 public:
785     Symbol name;
786 public:
787     object_class(Symbol a1) {
788         name = a1;
789     }
790     Expression copy_Expression();
791     void dump(ostream& stream, int n);
792
793 #ifdef Expression_SHARED_EXTRAS
794     Expression_SHARED_EXTRAS
795 #endif

```

```

796 #ifdef object_EXTRAS
797     object_EXTRAS
798 #endif
799 };
800
801
802 // define the prototypes of the interface
803 Classes nil_Classes();
804 Classes single_Classes(Class_);
805 Classes append_Classes(Classes, Classes);
806 Features nil_Features();
807 Features single_Features(Feature);
808 Features append_Features(Features, Features);
809 Formals nil_Formals();
810 Formals single_Formals(Formal);
811 Formals append_Formals(Formals, Formals);
812 Expressions nil_Expressions();
813 Expressions single_Expressions(Expression);
814 Expressions append_Expressions(Expressions,
    Expressions);
815 Cases nil_Cases();
816 Cases single_Cases(Case);
817 Cases append_Cases(Cases, Cases);
818 Program program(Classes);
819 Class_ class_(Symbol, Symbol, Features, Symbol)
    ;
820 Feature method(Symbol, Formals, Symbol,
    Expression);
821 Feature attr(Symbol, Symbol, Expression);
822 Formal formal(Symbol, Symbol);
823 Case branch(Symbol, Symbol, Expression);
824 Expression assign(Symbol, Expression);
825 Expression static_dispatch(Expression, Symbol,
    Symbol, Expressions);
826 Expression dispatch(Expression, Symbol,
    Expressions);
827 Expression cond(Expression, Expression,
    Expression);
828 Expression loop(Expression, Expression);
829 Expression typcase(Expression, Cases);
830 Expression block(Expressions);
831 Expression let(Symbol, Symbol, Expression,
    Expression);
832 Expression plus(Expression, Expression);
833 Expression sub(Expression, Expression);
834 Expression mul(Expression, Expression);
835 Expression divide(Expression, Expression);
836 Expression neg(Expression);
837 Expression lt(Expression, Expression);
838 Expression eq(Expression, Expression);

```

```

839 Expression leq(Expression, Expression);
840 Expression comp(Expression);
841 Expression int_const(Symbol);
842 Expression bool_const(Boolean);
843 Expression string_const(Symbol);
844 Expression new_(Symbol);
845 Expression isvoid(Expression);
846 Expression no_expr();
847 Expression object(Symbol);
848
849 #endif

```

Listing 5: cool-tree.h

A.4 cool-tree.handcode.h

```

1 //
2 // The following include files must come first.
3
4 #ifndef COOL_TREE_HANDCODE_H
5 #define COOL_TREE_HANDCODE_H
6
7 #include <iostream>
8 #include "tree.h"
9 #include "cool.h"
10 #include "stringtab.h"
11 #include <vector> // Fixed: Added vector
12     include
13
14 #define yylineno curr_lineno;
15 extern int yylineno;
16
17 inline Boolean copy_Boolean(Boolean b) {return
    b; }
18
19 inline void assert_Boolean(Boolean) {}
20
21 inline void dump_Boolean(ostream& stream, int
    padding, Boolean b)
22 { stream << pad(padding) << (int) b << "\n";
    }
23
24 void dump_Symbol(ostream& stream, int padding,
    Symbol b);
25
26 void assert_Symbol(Symbol b);
27
28 Symbol copy_Symbol(Symbol b);
29
30
31 class Program_class;
32 typedef Program_class *Program;
33
34 class Class__class;
35 typedef Class__class *Class_;

```

```

29 class Feature_class;
30 typedef Feature_class *Feature;
31 class Formal_class;
32 typedef Formal_class *Formal;
33 class Expression_class;
34 typedef Expression_class *Expression;
35 class Case_class;
36 typedef Case_class *Case;
37
38 typedef list_node<Class> Classes_class;
39 typedef Classes_class *Classes;
40 typedef list_node<Feature> Features_class;
41 typedef Features_class *Features;
42 typedef list_node<Formal> Formals_class;
43 typedef Formals_class *Formals;
44 typedef list_node<Expression> Expressions_class
    ;
45 typedef Expressions_class *Expressions;
46 typedef list_node<Case> Cases_class;
47 typedef Cases_class *Cases;
48
49 // Forward declaration
50 class CgenEnvironment;
51
52 #define Program_EXTRAS
    \
53 virtual void cgen(ostream&) = 0;    \
54 virtual void dump_with_types(ostream&, int) =
    0;
55
56 #define program_EXTRAS
    \
57 void cgen(ostream&);                \
58 void dump_with_types(ostream&, int);
59
60 #define Class__EXTRAS                \
61 virtual Symbol get_name() = 0;        \
62 virtual Symbol get_parent() = 0;      \
63 virtual Symbol get_filename() = 0;    \
64 virtual void dump_with_types(ostream&,int) = 0;
65
66 #define class__EXTRAS
    \
67 Symbol get_name()    { return name; }
    \
68 Symbol get_parent() { return parent; }
    \
69 Symbol get_filename() { return filename; }
    \
70 void dump_with_types(ostream&,int);

```

```

71
72 #define Feature_EXTRAS
    \
73 virtual void dump_with_types(ostream&,int) = 0;
    \
74 virtual bool IsMethod() { return false; } //
    Added
75
76 #define Feature_SHARED_EXTRAS
    \
77 void dump_with_types(ostream&,int);
78
79 #define method_EXTRAS \
80 bool IsMethod() { return true; } // Added
81
82 #define attr_EXTRAS \
83 bool IsMethod() { return false; } // Added
84
85 #define Formal_EXTRAS
    \
86 virtual void dump_with_types(ostream&,int) = 0;
87
88 #define formal_EXTRAS
    \
89 void dump_with_types(ostream&,int);
90
91 #define Case_EXTRAS
    \
92 virtual void dump_with_types(ostream&,int) =
    0;
93
94 #define branch_EXTRAS
    \
95 void dump_with_types(ostream&,int);
96
97 #define Expression_EXTRAS                \
98 Symbol type;                            \
99 Symbol get_type() { return type; }      \
100 Expression set_type(Symbol s) { type = s;
    return this; } \
101 virtual void code(ostream&, CgenEnvironment) =
    0; \
102 virtual void dump_with_types(ostream&,int) = 0;
    \
103 void dump_type(ostream&, int);          \
104 Expression_class() { type = (Symbol) NULL; } \
105 virtual bool IsEmpty() { return false; } //
    Added
106
107 #define Expression_SHARED_EXTRAS        \

```

```
108 void code(ostream&, CgenEnvironment);  
    \  
109 void dump_with_types(ostream&,int);  
110  
111 #define no_expr_EXTRAS \  
112 bool IsEmpty() { return true; } // Added  
113  
114 #endif
```

Listing 6: cool-tree.handcode.h