Programming Assignment V

15-李晓畅-凌之寒

1. 代码生成实现

1.1 代码生成简述

在 PA5 handout 中给出了实现静态语义检查的大概流程：

> There are many possible ways to write the code generator. One reasonable strategy is to perform code generation in two passes. The first pass decides the object layout for each class, particularly the offset at which each attribute is stored in an object. Using this information, the second pass recursively walks each feature and generates stack machine code for each expression.

类似于在 PA4 中的实现，还是先进行对类和对象的布局，决定各个属性的位置，这还是相当于在类节点上进行代码生成；再进行递归调用，从每一个属性向下调用，递归生成代码。

某种意义上说，PA5 中我们的工作似乎 PA4 中的工作是很类似的。同样是先构造继承树，基于继承树信息决定类的情况，再对下一层的属性、表达式节点通过统一的接口进行调用，不同的节点基于自己的特殊情况进行不同处理，递归向下的进行代码生成。

这种类似似乎并不是巧合。概念上说，语义分析要做的是解读，检查其中矛盾的地方或不符合习惯的地方，其基础是理解代码，或者转化成的 AST 到底想要说什么；而代码生成同样是在理解的基础上进行映射，在保持意思不变的基础上将一种表达形式转化为另一种表达形式。

具体任务将包括下面几个方面：

> 1. Determine and emit code for global constants, such as prototype objects.
> 2. Determine and emit code for global tables, such as the **class_nameTab**, the **class_objTab**, and the dispatch tables.
> 3. Determine and emit code for the initialization method of each class.
> 4. Determine and emit code for each method definition.

PA4 的设计对本次任务的实现很有参考价值。任务实际上是相似的，框架都是进行遍历，只是目的不同，那么在不同节点上进行的操作也会不同。语义分析实现对代码内容的检查所以遍历到一个节点我们要检查记录，而这里的代码生成侧重于生成代码，所以我们要释放代码，并留下记录，供其他部分使用。

所以我们设计了和 PA4 中类似的结构：

```
// similar to our class table in PA4
// difference is we need a tag to identify, a offset to locate; no check but need to gen code
// but we always use name inside
// by the way, SymbolTable we use in PA4 is the base class here qwq
class CgenClassTable : public SymbolTable<Symbol,CgenNode> {…

// CgenNode <--> Class
class CgenNode : public class__class { …

// similar to our env_t in PA4
// but stronger, better packed
// it is necessary for a complex job
class EnvTable {…

class BoolConst …
```

$CgenClassTable$，PA4 $ClassTable$的升级版。类似的内核，不同的接口和功能。

$CgenNode$，一个信息更丰富的类节点，代码生成要实际带入很多东西，所以需要更多信息，所以要丰富节点的表达能力。

$EnvTable$，它有着更好封装，更强功能。

$BoolConst$，它大概是一个例子。

下面介绍也将围绕$CgenClassTable$和$EnvTable$如何实现、如何参与代码生成展开。

1.2 类节点和类表

1.2.1 类节点数据结构简述

官方实现为我们提供了一个新的类节点`class CgenNode : public class__class {`。

```
// CgenNode <--> Class
class CgenNode : public class__class {

private:
   int chain_depth_;
   int class_tag_;
   int descendants_cnt_;                    // number of child(s)
   CgenNodeP parentnd;                      // Parent of class
   List<CgenNode> *children;                // Children of class
   Basicness basic_status;                  // `Basic' if class is basic
                                            // `NotBasic' otherwise

public:
   CgenNode(Class_ c,
            Basicness bstatus,
            CgenClassTableP class_table);

   void add_child(CgenNodeP child);

   // dull----------------------------------------
```

这也带来了一些好处。它记录了自己的所有孩子和双亲，这样就为我们的$CgenClassTable$节约了保存图的结构。而且找孩子变得很方便了。

而且可以利用很多已经给出的代码。

1.2.2 $CgenClassTable$实现概述

这是本次任务的核心。

```
// similar to our class table in PA4
// difference is we need a tag to identify, a offset to locate; no check but need to gen code
// but we always use name inside
// by the way, SymbolTable we use in PA4 is the base class here qwq
class CgenClassTable : public SymbolTable<Symbol,CgenNode> {…
```

$CgenClassTable$继承于$SymbolTable<Symbol,CgenNode>$,而后者我们在 PA4 中就将其应用于保存名字和类、作用域的关系了。这里官方给出的应用也是类似的，它将被用于保存类名和节点之间的关系。COOL 似乎不支持内嵌对象，所以对类而言没有作用域的说法，所以这里并不设计域的界定。

首先可以考察属性的定义。主要的结构包括下面这些：

```
// classes
List<CgenNode> *nds;
// tag -> name
std::map<int, Symbol> class_tag_map_;
// similar to our class_map in PA4
// name -> node
std::map<Symbol, CgenNodeP> name_to_cgen_map_;
// class -> attr(list) defined
// attributs are always private
std::map<Symbol, std::vector<attr_class*>>   class_attr_map_;
// class -> method(list) defined
std::map<Symbol, std::vector<method_class*>> class_method_map_;

// class, attr -> offset
std::map<Symbol, std::map<Symbol, int>>      attr_offset_map_;
// class, method -> offset
std::map<Symbol, std::map<Symbol, int>>      meth_offset_map_;

// class(self) -> class(defined met), method
std::map<Symbol, std::vector<std::pair<Symbol, Symbol>>> dispatch_tab_map_;
// class -> parents
std::map<Symbol, std::vector<CgenNodeP>> parent_chain_map_;
```

虽然看起来繁杂，但实际上表达是意思同 PA4$ClassTable$是类似地。

和外界交互时我们需要将 Tag 映射为名字。

访问具体信息时我们要将名字映射为节点。

PA4 里每次查节点找信息还是很麻烦的，这里直接将它分为属性和方法拿出来作为映射，之后也会发现因为 COOL 独特的要求，也就是属性私有，方法公有的条件下使用这样的结构还是很方便的。

记录名字的偏移。毫无疑问，这是为代码生成做准备的。

其他也可以单独爬树得到结果，这里就写一个图把他固定下来也是可行的选择。

归根结底，我们的类表还是两种结构，一种是内部数据间的结构，因为我们总是使用名字作为内部类的标识，所以我们需要将其映射到偏移或 tag；另一种是同外部交互的结构，将名字映射到节点，这和我们在 PA4 中做的是一样的。

其他还有一些变量，包括输出流、基类的 TAG、已经标注的标签序号和当前节点（也就是类），此处不再赘述。

然后我们可以考察主要方法。

下列是我们在$CgenClassTable$中实现的几个核心方法，这同我们在 1.1 代码生成简述中描述的任务是完全对应的。

```
// The following methods emit code for
// constants and global declarations.
    // task 1
   void code_global_data();
   void code_global_text();
   void code_bools(int);
   void code_select_gc();
   void code_constants();

    // task 2
   void code_class_nametabs();      // emit class_nameTab
   void code_class_objtabs();       // emit class_objtab
   void code_object_disptabs();     // emit dispatch_tab

    // task 3
   void code_protobjs();
   void code_object_inits();

    // task 4
   void code_methods();
```

就像他们的名字表述的那样，他们会对应生成代码。之后我们会具体阐述详细情况。

就像我们在 PA4 中面临的那样，构建继承图仍然是必须实现的任务，对应包含下面的方法：

```
// The following creates an inheritance graph from
// a list of classes.  The graph is implemented as
// a tree of `CgenNode', and class names are placed
// in the base class symbol table.

   void install_basic_classes();
   void install_class(CgenNodeP nd);
   void install_classes(Classes cs);
   // assign tag and number of childs to class
   void install_classtags();

   void install_attrs_and_methods();
   void install_name_to_cgen();

   void build_inheritance_tree();
   void set_relations(CgenNodeP nd);
```

类似 PA4 中的初始化方法，区别在于我们将功能分解了。各个部分的初始化我们将它分为不同的函数实现。毕竟我们新增了很多结构，分别实现、分别测试是有意义。实现方法类似 PA4 中的初始化方法，比如：

```
void CgenClassTable::install_name_to_cgen() {
    CgenNodeP curr_cgen;
    for (List<CgenNode> *l = nds; l; l = l->tl()) {
        curr_cgen = l->hd();
        name_to_cgen_map_[curr_cgen->get_name()] = curr_cgen;
    }
}
```

简单为我们需要用到的 map 进行一个初始化就可以了。

Install_attrs_and_methods()就很复杂了：

```
void CgenClassTable::install_attrs_and_methods() {
    CgenNodeP curr_cgennode;
    Features curr_features;
    // install features defined by the class
    // method_map and attr_map only have features defined
    for (List<CgenNode> *l = nds; l; l = l->tl()) { ···
    // install offset
    for (List<CgenNode> *l = nds; l; l = l->tl()) {···
}
```

加载方法和属性做一遍遍历节点，加载偏移再遍历一次。

比如在加载属性时：

```
for (List<CgenNode> *l = nds; l; l = l->tl()) {
    curr_cgennode = l->hd();
    curr_features = curr_cgennode->get_features();
    Feature curr_feature;
    Symbol curr_cgen_name = curr_cgennode->get_name();
    for (int i = curr_features->first(); curr_features->more(i); i = curr_features->next(i)) {
        curr_feature = curr_features->nth(i);
        // for method
        if (curr_feature->is_method()) {···
        // for attr
        else {···
    }
}
```

其实就遍历，然后分别加入即可。加载偏移也是类似的。

详细代码可参考附录。

1.2.3 代码生成简述

下面我们会简述几类核心代码生成功能的实现方法。Task1 已由框架给出。

考察*code_class_nametabs*()的实现：

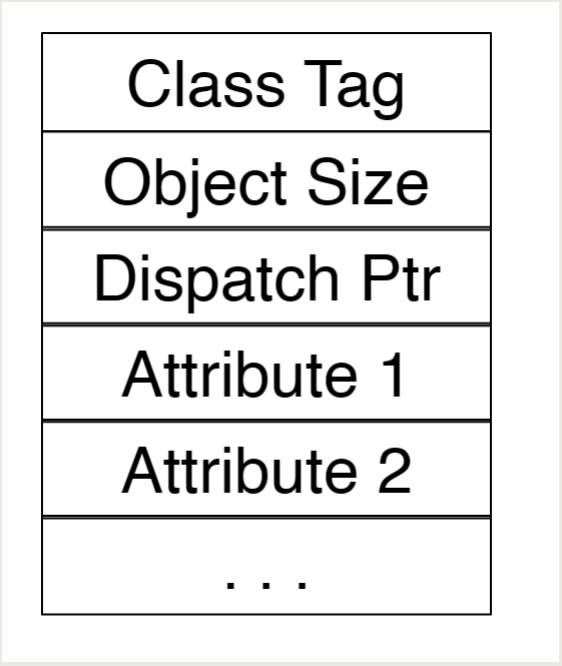
```
void CgenClassTable::code_class_nametabs() {
    str << CLASSNAMETAB << LABEL;
    int len = class_tag_map_.size();

    StringEntry* str_entry;
    for (int i = 0; i < len; i++) {
        Symbol name = class_tag_map_[i];
        str_entry = stringtable.lookup_string(name->get_string());
        str << WORD;
        str_entry->code_ref(str);
        str << endl;
    }
}
```

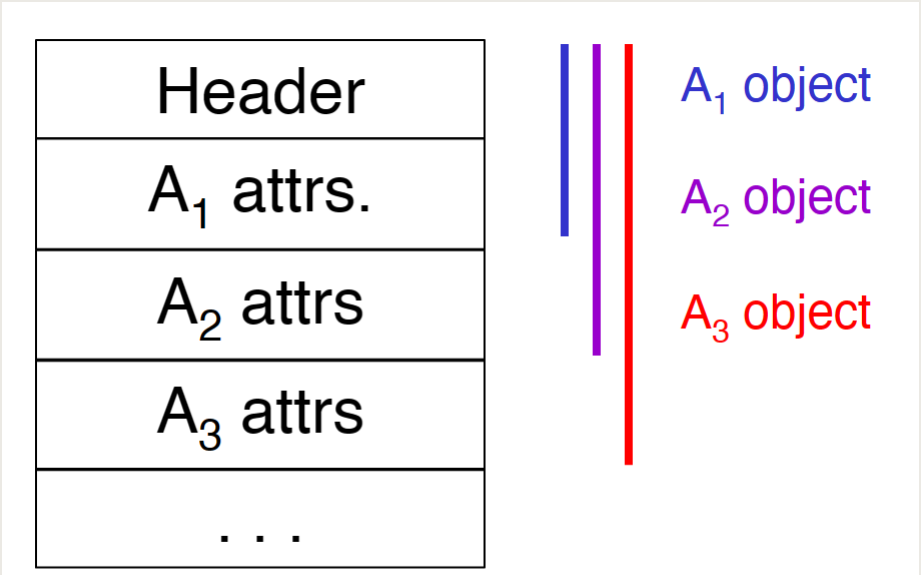
按照对应格式，输出每个类的名字就可以了。

Task2 其他方法就是类似的了。Task3 就比较复杂了。

下面简单以*void CgenClassTable::code_object_inits*()为例介绍：

首先我们需要知道一个类是如何布局的。可以参考下面的两张图：





所以我们首先要分配下$Class\ Tag$，$Object\ Size$和$Dispatch\ Ptr$。

这些部分每个类初始化都需要实现，所以我们就为它单独设计了一种 emit 方法。`emit_start_frame(str);`

然后，我们要加载类的父亲们的属性和自己的属性。自己的简单一个 move 就可以实现我们的需求。父亲们的也有对应方法。

然后对应输出就可以了。合起来大概像下面展示的：

```
void CgenClassTable::code_object_inits() {
    CgenNodeP curr_cgen;
    for (List<CgenNode> *l = nds; l; l = l->tl()) {
        curr_cgen = l->hd();
        curr_cgenclass_ = curr_cgen;
        emit_init_ref(curr_cgen->get_name(), str);
        str << LABEL;
        emit_start_frame(str);
        envTable->enterframe();

        // install parent's
        CgenNodeP parent = curr_cgen->get_parentnd();
        if (parent && parent->get_name() != No_class) {…
        // install self's
        // so you see the benefits of saperating attr and meth
        const attrList& curr_attrs = class_attr_map_[curr_cgen->get_name()];
        for (auto attr : curr_attrs) {…
        emit_move(ACC, SELF, str);
        emit_end_frame(str);
        envTable->exitframe();
        emit_return(str);
    }
}
```

当然，这里把属性单独打一个表的好处就体现出来了 qwq。

其他 Task3 实现与其类似。

Task4 的$void\ code_methods()$可以被认为是开启下一阶段代码生成的接口，在其中他会通过调用表达式的代码生成方法：`method->expr->code(str);`。因为设计到环境表，所以这会在下一部分阐述。

1.2.4 其他公共方法

同样的，构建好的$CgenClassTable$不仅要为自己服务，还要为其他阶段服务。所以我们需要实现一些支持方法。

```
// convenient for labeling
int get_labelid_and_add() { return labelid_++; }

CgenNodeP get_cgennode(Symbol name) { return name_to_cgen_map_[name]; }
CgenNodeP get_curr_class() const { return curr_cgenclass_; }

 // return T/F for find or not, real return on offset
bool get_attr_offset(Symbol cls, Symbol attr, int *offset);
bool get_meth_offset(Symbol cls, Symbol meth, int *offset);

ostream& codege_str() {
    return str;
}

void code();
CgenNodeP root();
```

打标签时可以用第一个方法。

就像他们的名字阐述的那样，他们大概就是访问私有变量然后对应返回即可。

上面分开给属性和方法保存映射还是有意义的，这里的方法中可以看到这一点。

Code（）是代码生成的接口，通过调用 code 可以实现代码生成。它将调用上面提到的五个生成方法实现代码生成。它会在初始化类表时，完成对前面定义的结构体后被调用以完成代码生成。

就像上面阐述的那样，因为节点自带了很多信息，所以我们就不需要维护继承图了，我们只需要访问根就可以爬到任何希望的地方。

1.3 环境表和代码生成简述

1.3.1 环境表数据结构简述

PA4 的环境表在代码生成上有些不够。最主要体现在我们没有很好包含对偏移量的支持。所以我们重新组织了这一类，实现一个符号表，符号表内的每一个表项将把符号映射成 tag。大概像下面这样。

```
// similar to our env_t in PA4
// but stronger, better packed
// it is necessary for a complex job
class EnvTable {…
```

虽然我们也很想直接用框架给出的符号表，但有一点问题在于框架给出的 value 是一个指针。

```
// class EnvTable : public SymbolTable<Symbol, int> {…
```

所以我们添加了一些属性：

```
int formal_fp_offset_ = DEFAULT_OBJFIELDS;      // forlam parm offset
int local_fp_offset_ = -1;                      // local var offset
int last_local_fp_offset_ = -1;                 // prv fp
```

然后照着框架的符号表画了一个：

```
EnvTable() = default;

void enterframe(){…
void exitframe(){…
void enterscope(){…
void exitscope(){ …

void add_formal_id(Symbol name){…
void add_local_id(Symbol name){…
// name -> offset
bool lookup(Symbol name, int *offset){…
```

功能完全一样，这里就不详细阐述了。

1.3.2 从类到表达式

下面我们将阐述$void\ CgenClassTable::code_methods()$的实现。

COOL 不支持单独定义函数，类似于 Java，方法都在类里。所以我们遍历类就可以进而获取所有方法了：

```
void CgenClassTable::code_methods() {
    CgenNodeP curr_cgennode;
    for (List<CgenNode> *l = nds; l; l = l->tl()) {…
}
```

下面将对具体方法实施处理：

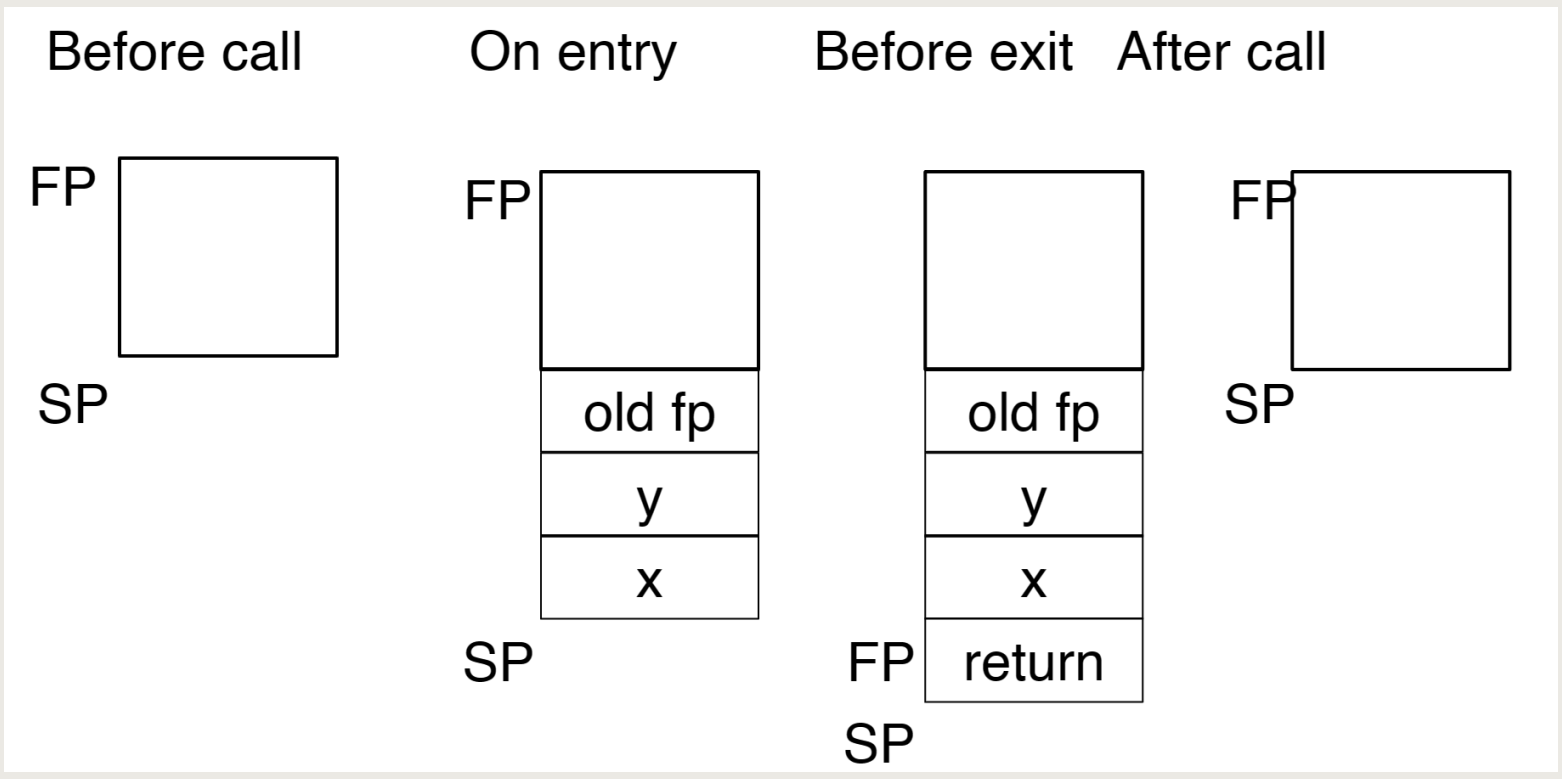
```
for (List<CgenNode> *l = nds; l; l = l->tl()) {
    curr_cgennode = l->hd();
    curr_cgenclass_ = curr_cgennode;
    Symbol cgenname = curr_cgenclass_->get_name();
    if (is_basic_class(cgenname)) { // basic method need to do nothing…
    // all methods public means class only need to emit their own methods
    // dispatch will tell which method is actually called
    const methodList& methods = class_method_map_[cgenname];
    Formals curr_formals;
    std::list<Formal> formal_list;
    for (auto method : methods) {…
}
```

基本类并不需要我们实现方法。要么没有，要么已经由框架实现了。

对于一个类就遍历它所定义的方法就可以了。因为方法是公有的，所以只写自己定义的代码是正确的。这也是为什么前面的方法表我们并不登记继承的方法，而偏移表却需要登记继承方法。

对方法的代码生成除了涉及表达式的生成，还涉及几个因素：进入函数前后都需要进行维护。

考察调用示例：



我们不难发现我们需要保存 fp、返回地址和形参列表。

```
for (auto method : methods) {
    formal_list.clear();
    envTable->enterframe();
    curr_formals = method->formals;
    for (int i = curr_formals->first(); curr_formals->more(i); i = curr_formals->next(i)) {…
    for (auto formal : formal_list) {…
    emit_method_ref(curr_cgenclass_->get_name(), method->get_name(), str);
    str << LABEL;
    emit_start_frame(str);
    method->expr->code(str);
    emit_end_frame(str);
    emit_addiu(SP, SP, formal_list.size() * WORD_SIZE, str);
    emit_return(str);
    envTable->exitframe();
}
```

我们在这里写一部分，另一部分就交给 dispatch 了。

对应进行处理之后就可以进入表达式生成。

完成后需要恢复到调用前状态。

下面任务就是在特定的表达式节点上进行的了。

1.3.3 表达式生成简述

我们同样为表达式节点定义统一的接口：

```
class Expression_class : public tree_node {
public:
   tree_node *copy()     { return copy_Expression(); }
   virtual Expression copy_Expression() = 0;
   virtual void code(ostream &s) = 0;

#ifdef Expression_EXTRAS
   Expression_EXTRAS
#endif
};
```

具体的表达式需要实现$code()$方法，以实现对代码的生成。

下面以$void\ cond_class::code(ostream\ \&s)$为例说明：

```
void cond_class::code(ostream &s) {
    pred->code(s);
    // load two params for compared
    emit_load(T1, ATTR_BASE_OFFSET, ACC, s);
    emit_move(T2, ZERO, s);

    int out_lebal = codegen_classtable->get_labelid_and_add();
    int false_lebal = codegen_classtable->get_labelid_and_add();

    emit_beq(T1, T2, false_lebal, s);
    // true cond
    then_exp->code(s);
    emit_branch(out_lebal, s);
    // false cond
    emit_label_def(false_lebal, s);
    else_exp->code(s);
    emit_label_def(out_lebal, s);
}
```

加载两个比较变量，生成真假两种条件下的 label，生成分支和对应代码即可。

其他表达式的代码生成与之类似。

1.3.4 总结：代码是怎么生成的

大概是这样的流程：

```
void program_class::cgen(ostream &os)
{
  // spim wants comments to start with '#'
  os << "# start of generated code\n";

  initialize_constants();
  envTable = new EnvTable();
  codegen_classtable = new CgenClassTable(classes,os);

  os << "\n# end of generated code\n";
}
```

代码生成将从程序节点开始，初始化$CgenClassTable$的过程将开启各部分的生成。

```
CgenClassTable::CgenClassTable(Classes classes, ostream& s) : nds(NULL) , str(s),⋯
```

在初始化中首先将加载信息到对应地结构中：

```
install_basic_classes();
install_classes(classes);
install_name_to_cgen();
build_inheritance_tree();
install_classtags();
install_attrs_and_methods();

code();
exitscope();
```

对$code()$的调用将正式开启代码生成。

在其中我们将调用之前实现的几种生成方法，分步骤实现代码生成。

```
void CgenClassTable::code(){
  code_global_data();
  code_select_gc();
  code_constants();

//                Add your code to emit
//                  - prototype objects
//                  - class_nameTab
//                  - dispatch tables
//
  code_global_text();
//                Add your code to emit
//                  - object initializer
//                  - the class methods
//                  - etc...
  code_class_nametabs();
  code_class_objtabs();
  code_object_disptabs();
  code_protobjs();
  code_object_inits();
  code_methods();

}
```

最后的$code_method()$将把代码生成推进到第二阶段，进而开启对具体表达式的处理。比如上面提到的$cond$节点。

在这一切结束后就实现了代码生成。

2. 测试构造和测试结果

类似前一次实验中我们做的，我们还是照例构造一些测试样例。当然，在代码生成阶段没有错误，所以就不用构造错误样例了。下面展示的是正确的测试样例。主要还是 cool 文档中一些比较复杂的 example。

另外，我们也为一些特殊情况构造了样例，比如：

```
--let let let LET

class LLLLLLLET {
  a : Int <- let x : Int, value : Int <- x + 1 in let value : Int <- 1 in x + value;
  getA() : Int {a};
};

Class Main {
  main() : Int {
    (new LLLLLLLET).getA()
  };
};
```

为 let 专门构造的样例。

和一些其他情况，包括可以的继承和方法。

```
--Class inherit and method

class A {
  compute(a:Int, b:Int) : Int {a+b};
  foo() : A {new A};
};

class A1 inherits A {
  i : Int <- 1;
  getI() : Int {i};
  foo() : A {new A1};
};

class A2 inherits A {
};

class A11 inherits A1 {
  compute(x:Int, y:Int) : Int {x-y+i};
};

class A12 inherits A1 {
  b : Bool;
  compute(b:Int, i:Int) : Int {b*i};
};

class A111 inherits A11 {
  j : Int <- 9;
  getJ() : Int {j};
};

class A121 inherits A12 {
  foo() : A {new A122};
};

class A122 inherits A12 {
  compute2(a:Bool) : Bool {a = b};
};

Class Main {
  a : A111;
  main() : Int{
    (new A121).foo().compute(a.getJ(), (new A121).getI())
  };
};
```

均可通过。

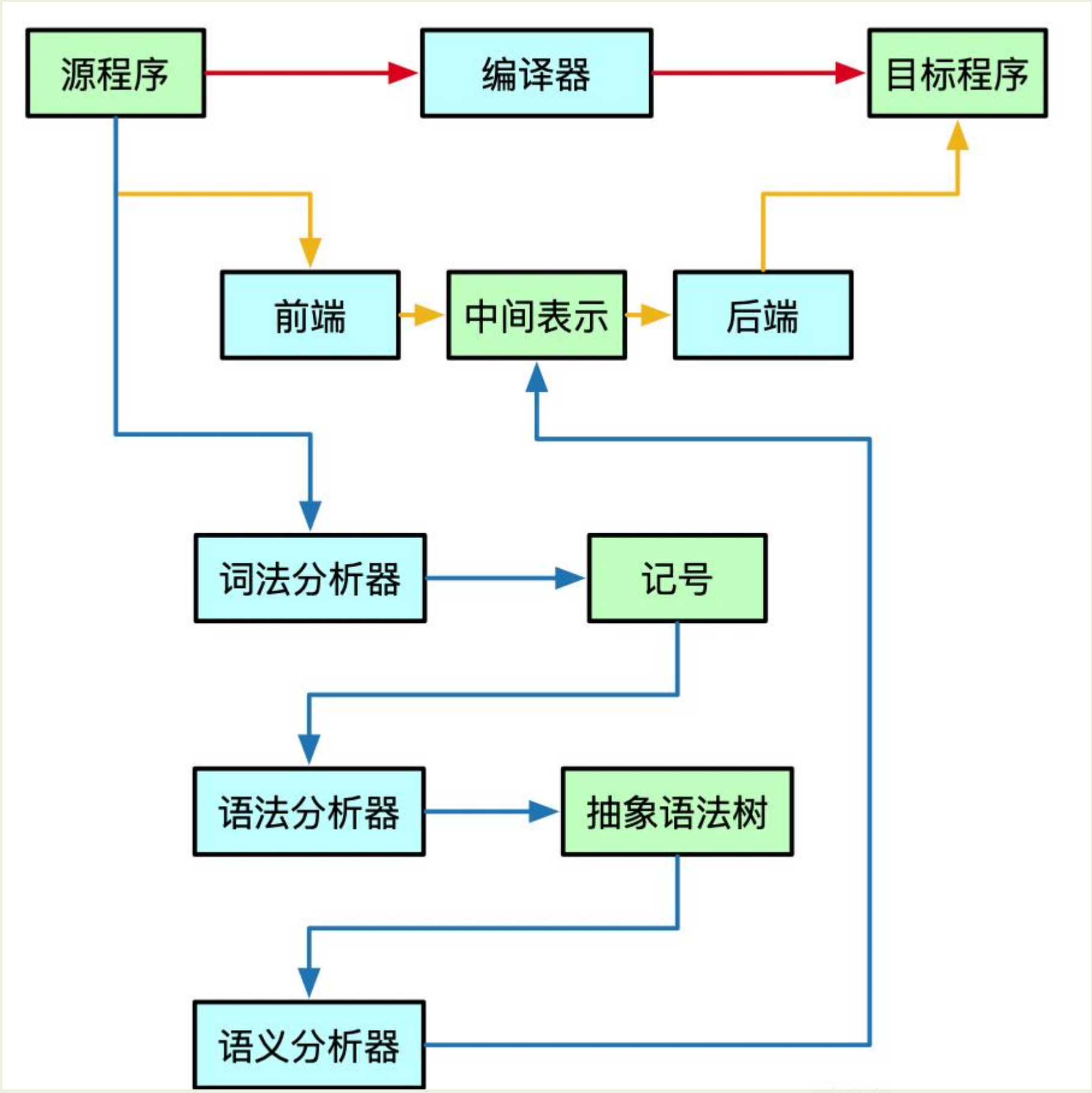
```
root@computer:/home/Compile-Lab/assignments/PA5r# gmake dotest

Running code generator on example.cl

./mycoolc example.cl
```

3. 实验总结

> 在经历过漫长的各种准备工作后，我们终于到了最终的代码生成阶段 QwQ。
>
> 在代码生成阶段的设计和实现中，我们惊奇地发现代码生成和语义分析有着类似的实际内核。由于不论是代码生成还是语义分析，实现这一阶段的前提条件都是对相应代码段的理解，对上下文信息的再组织和利用。我们需要真正理解代码段希望表达的意思，而非类似于词法分析或者语法分析那种主要只是匹配其形式，或者某种文法。具体到实现代码上，我们为了获取全文信息，我们需要遍历整个语法树，并将信息组织起来为后续使用提供方便，而不是类似于前两个阶段的简单匹配、识别一种模式。
>
> 代码生成和语义分析还是存在一些差异。归结而言语义分析是理解代码，查找其是否有同上下文矛盾的地方；而代码生成已经假定代码正确性，只需要进行语言转化，类似于翻译的工作即可。
>
> 在具体实现中，我们基于任务划分阶段，基于数据结构组织方法，最终实现了复杂的工程。
>
> 在具体任务的不同阶段，我们采取了不同的策略。类似于语义分析，我们还是应该构建一颗方便使用的继承树；有了继承树之后，我们应该规范各种类型节点代码生成的统一规范，再利用具体信息为对应语句生成代码即可。
>
> 面向对象虽然很方便，但是实现起来会为我们一些麻烦。

本学期编译的项目看起来就告一段落了。我们通过几个任务实现了编译器的不同阶段，体会到了他们的差异；在总结和分析中我们又不难发现他们类似的内核和本质。我们遇到了很多问题，但克服问题的过程又给我们了很多启发。总之，我们从本次各阶段的任务都获得了很大收获。

最后还是一张编译过程示意图作为结束。



附录

1. cgen.h 代码

```
#include <assert.h>
#include <stdio.h>
#include <vector>
#include <map>
#include <list>
#include "emit.h"
#include "cool-tree.h"
#include "symtab.h"

enum Basicness     {Basic, NotBasic};
#define TRUE 1
```

```
#define FALSE 0
#define ATTR_BASE_OFFSET 3
#define DISPATCH_OFFSET 2

class CgenClassTable;
typedef CgenClassTable *CgenClassTableP;
typedef std::vector<attr_class*> attrList;
typedef std::vector<method_class*> methodList;

class CgenNode;
typedef CgenNode *CgenNodeP;

// similar to our class table in PA4
// difference is we need a tag to identify, a offset to locate; no check but need to gen code
// but we always use name inside
// by the way, SymbolTable we use in PA4 is the base class here qwq
class CgenClassTable : public SymbolTable<Symbol,CgenNode> {
private:
    // classes
    List<CgenNode> *nds;
    // tag -> name
    std::map<int, Symbol> class_tag_map_;
    // similar to our class_map in PA4
    // name -> node
    std::map<Symbol, CgenNodeP> name_to_cgen_map_;
    // class -> attr(list) defined
    // attributs are always private
    std::map<Symbol, std::vector<attr_class*>>   class_attr_map_;
    // class -> method(list) defined
    std::map<Symbol, std::vector<method_class*>> class_method_map_;

    // class, attr -> offset
    std::map<Symbol, std::map<Symbol, int>>      attr_offset_map_;
    // class, method -> offset
    std::map<Symbol, std::map<Symbol, int>>      meth_offset_map_;

    // class(self) -> class(defined met), method
    std::map<Symbol, std::vector<std::pair<Symbol, Symbol>>> dispatch_tab_map_;
    // class -> parents
    std::map<Symbol, std::vector<CgenNodeP>> parent_chain_map_;

    ostream& str;
    int stringclasstag;
    int intclasstag;
    int boolclasstag;
    int labelid_;
    // point to class handling now
    CgenNodeP curr_cgenclass_;


// The following methods emit code for
// constants and global declarations.
    // task 1
   void code_global_data();
   void code_global_text();
   void code_bools(int);
   void code_select_gc();
   void code_constants();

    // task 2
   void code_class_nametabs();      // emit class_nameTab
   void code_class_objtabs();       // emit class_objtab
   void code_object_disptabs();     // emit dispatch_tab

    // task 3
   void code_protobjs();
   void code_object_inits();

    // task 4
```

```
81.    void code_methods();
82.
83. // The following creates an inheritance graph from
84. // a list of classes.  The graph is implemented as
85. // a tree of `CgenNode', and class names are placed
86. // in the base class symbol table.
87.
88.    void install_basic_classes();
89.    void install_class(CgenNodeP nd);
90.    void install_classes(Classes cs);
91.    // assign tag and number of childs to class
92.    void install_classtags();
93.
94.    void install_attrs_and_methods();
95.    void install_name_to_cgen();
96.
97.    void build_inheritance_tree();
98.    void set_relations(CgenNodeP nd);
99. public:
100.    CgenClassTable(Classes, ostream& str);
101.    // convenient for labeling
102.    int get_labelid_and_add() { return labelid_++; }
103.
104.    CgenNodeP get_cgennode(Symbol name) { return name_to_cgen_map_[name]; }
105.    CgenNodeP get_curr_class() const { return curr_cgenclass_; }
106.
107.     // return T/F for find or not, real return on offset
108.    bool get_attr_offset(Symbol cls, Symbol attr, int *offset);
109.    bool get_meth_offset(Symbol cls, Symbol meth, int *offset);
110.
111.    ostream& codege_str() {
112.        return str;
113.    }
114.
115.    void code();
116.    CgenNodeP root();
117. };
118.
119. // CgenNode <--> Class
120. class CgenNode : public class__class {
121.
122. private:
123.    int chain_depth_;
124.    int class_tag_;
125.    int descendants_cnt_;                    // number of child(s)
126.    CgenNodeP parentnd;                      // Parent of class
127.    List<CgenNode> *children;                // Children of class
128.    Basicness basic_status;                  // `Basic' if class is basic
129.                                             // `NotBasic' otherwise
130.
131. public:
132.    CgenNode(Class_ c,
133.             Basicness bstatus,
134.             CgenClassTableP class_table);
135.
136.    void add_child(CgenNodeP child);
137.
138.    // dull------------------------------------------
139.    List<CgenNode> *get_children() { return children; }
140.    void set_parentnd(CgenNodeP p);
141.    void set_classtag(int tag) { class_tag_ = tag; }
142.    void set_chain_depth(int depth) { chain_depth_ = depth; }
143.    void set_descendants_cnt(int descnt) { descendants_cnt_ = descnt; }
144.    int  get_classtag() const { return class_tag_; }
145.    int  get_chain_depth() const { return chain_depth_; }
146.    int  get_descendants_cnt() const { return  descendants_cnt_; }
147.    CgenNodeP get_parentnd() { return parentnd; }
148.    std::vector<CgenNodeP> get_parents_list();
149.    int basic() { return (basic_status == Basic); }
```

```
};

// similar to our env_t in PA4
// but stronger, better packed
// it is necessary for a complex job
class EnvTable {
public:
    typedef std::list<std::pair<Symbol, int>> symbol2offsetList;
private:
    std::list<std::list<std::pair<Symbol, int>>> envlist_;
    // exit scop <- [ [(symble1, tag1), (symble2,tag2), ...], [...], [...]] -> enter scop
    //               |<-               one scop        ->|
    int formal_fp_offset_ = DEFAULT_OBJFIELDS;      // forlam parm offset
    int local_fp_offset_ = -1;                     // local var offset
    int last_local_fp_offset_ = -1;                // prv fp

public:
    EnvTable() = default;

    void enterframe(){
        formal_fp_offset_ = DEFAULT_OBJFIELDS;      // ref 'run time'
        local_fp_offset_ = -1;
        last_local_fp_offset_ = -1;
        enterscope();
    }
    void exitframe(){
        exitscope();
        formal_fp_offset_ = DEFAULT_OBJFIELDS;
        local_fp_offset_ = -1;
        last_local_fp_offset_ = -1;
    }
    void enterscope(){
        last_local_fp_offset_ = local_fp_offset_;
        envlist_.push_back({});
    }
    void exitscope(){
        envlist_.pop_back();
        local_fp_offset_ = last_local_fp_offset_;
    }

    void add_formal_id(Symbol name){
        //std::cout << "#push name " << name << " and " << offset << endl;
        envlist_.back().push_back({name, formal_fp_offset_++});
    }
    void add_local_id(Symbol name){
        envlist_.back().push_back({name, local_fp_offset_--});
    }
    // name -> offset
    bool lookup(Symbol name, int *offset){
        //std::cout <<"# find name " << name << " " << envlist_.size() << endl;
        for (auto rit = envlist_.rbegin(); rit != envlist_.rend(); ++rit) {
            const symbol2offsetList &sym2off_list = *rit;
            for (auto rlit = sym2off_list.rbegin(); rlit != sym2off_list.rend(); ++rlit) {
                if (rlit->first == name) {
                    *offset = rlit->second;
                    return true;
                }
            }
        }
        return false;
    }
};

// class EnvTable : public SymbolTable<Symbol, int> {
//     private:
//     int formal_fp_offset_ = DEFAULT_OBJFIELDS;      // forlam parm offset
//     int local_fp_offset_ = -1;                     // local var offset
//     int last_local_fp_offset_ = -1;
//     public:
```

```
//     /*
//     SP0 ->  |   FP0     |
//             |   SELF    |
//             |   formal  |
//     FP1 ->  |   RA      |
//     SP1 ->  |           |
//     */
//     // prepare to call
//     // a f call is sure to enter a new scop
//     void enterframe(){
//         formal_fp_offset_ = DEFAULT_OBJFIELDS;      // ref 'run time'
//         local_fp_offset_ = -1;
//         last_local_fp_offset_ = -1;
//         enterscope();
//     }
//     // call done
//     void exitframe(){
//         exitscope();
//         formal_fp_offset_ = DEFAULT_OBJFIELDS;
//         local_fp_offset_ = -1;
//         last_local_fp_offset_ = -1;
//     }
//     void enterscope(){
//         last_local_fp_offset_ = local_fp_offset_;
//         SymbolTable<Symbol, int> :: enterscope();
//     }
//     void exitscope(){
//         SymbolTable<Symbol, int> :: exitscope();
//         local_fp_offset_ = last_local_fp_offset_;
//     }
//     void add_formal_id(Symbol name){
//         //std::cout << "#push name " << name << " and " << offset << endl;
//         addid(name, (int*)(long)(formal_fp_offset_++));
//     }
//     void add_local_id(Symbol name){
//         // envlist_.back().push_back({name, local_fp_offset_--});
//         addid(name, (int*)(long)(formal_fp_offset_--));
//     }
//     bool lookup(Symbol name){
//     }
// };

class BoolConst
{
 private:
  int val;
 public:
  BoolConst(int);
  void code_def(ostream&, int boolclasstag);
  void code_ref(ostream&) const;
};

```

2. cgen.cc 代码

```
//**************************************************************
//
// Code generator SKELETON
//
// Read the comments carefully. Make sure to
//    initialize the base class tags in
//       `CgenClassTable::CgenClassTable'
//
//    Add the label for the dispatch tables to
//       `IntEntry::code_def'
//       `StringEntry::code_def'
//       `BoolConst::code_def'
```

```
13. //
14. //    Add code to emit everyting else that is needed
15. //       in `CgenClassTable::code'
16. //
17. //
18. // The files as provided will produce code to begin the code
19. // segments, declare globals, and emit constants.  You must
20. // fill in the rest.
21. //
22. //*****************************************************************
23. #include <algorithm>
24. #include <functional>
25.
26. #include "cgen.h"
27. #include "cgen_gc.h"
28.
29. extern void emit_string_constant(ostream& str, char *s);
30. extern int cgen_debug;
31.
32. #define DEBUG_FLAG "debug_flag\n"
33.
34. //
35. // Three symbols from the semantic analyzer (semant.cc) are used.
36. // If e : No_type, then no code is generated for e.
37. // Special code is generated for new SELF_TYPE.
38. // The name "self" also generates code different from other references.
39. //
40. ////////////////////////////////////////////////////////////////////////
41. //
42. // Symbols
43. //
44. // For convenience, a large number of symbols are predefined here.
45. // These symbols include the primitive type and method names, as well
46. // as fixed names used by the runtime system.
47. //
48. ////////////////////////////////////////////////////////////////////////
49. Symbol
50.        arg,
51.        arg2,
52.        Bool,
53.        concat,
54.        cool_abort,
55.        copy,
56.        Int,
57.        in_int,
58.        in_string,
59.        IO,
60.        length,
61.        Main,
62.        main_meth,
63.        No_class,
64.        No_type,
65.        Object,
66.        out_int,
67.        out_string,
68.        prim_slot,
69.        self,
70.        SELF_TYPE,
71.        Str,
72.        str_field,
73.        substr,
74.        type_name,
75.        val;
76. //
77. // Initializing the predefined symbols.
78. //
79. static void initialize_constants(void)
80. {
81.   arg         = idtable.add_string("arg");
```

```
  arg2        = idtable.add_string("arg2");
  Bool        = idtable.add_string("Bool");
  concat      = idtable.add_string("concat");
  cool_abort  = idtable.add_string("abort");
  copy        = idtable.add_string("copy");
  Int         = idtable.add_string("Int");
  in_int      = idtable.add_string("in_int");
  in_string   = idtable.add_string("in_string");
  IO          = idtable.add_string("IO");
  length      = idtable.add_string("length");
  Main        = idtable.add_string("Main");
  main_meth   = idtable.add_string("main");
//   _no_class is a symbol that can't be the name of any
//   user-defined class.
  No_class    = idtable.add_string("_no_class");
  No_type     = idtable.add_string("_no_type");
  Object      = idtable.add_string("Object");
  out_int     = idtable.add_string("out_int");
  out_string  = idtable.add_string("out_string");
  prim_slot   = idtable.add_string("_prim_slot");
  self        = idtable.add_string("self");
  SELF_TYPE   = idtable.add_string("SELF_TYPE");
  Str         = idtable.add_string("String");
  str_field   = idtable.add_string("_str_field");
  substr      = idtable.add_string("substr");
  type_name   = idtable.add_string("type_name");
  val         = idtable.add_string("_val");
}

static char *gc_init_names[] =
  { "_NoGC_Init", "_GenGC_Init", "_ScnGC_Init" };
static char *gc_collect_names[] =
  { "_NoGC_Collect", "_GenGC_Collect", "_ScnGC_Collect" };
static CgenClassTable *codegen_classtable = nullptr;

static bool is_basic_class(Symbol name) {
    return name == Object || name == Str || name == Bool || name == Int || name == IO;
}

static EnvTable* envTable = nullptr;
//  BoolConst is a class that implements code generation for operations
//  on the two booleans, which are given global names here.
BoolConst falsebool(FALSE);
BoolConst truebool(TRUE);

//*********************************************************
//
// Define method for code generation
//
// This is the method called by the compiler driver
// `cgtest.cc'. cgen takes an `ostream' to which the assembly will be
// emmitted, and it passes this and the class list of the
// code generator tree to the constructor for `CgenClassTable'.
// That constructor performs all of the work of the code
// generator.
//
//*********************************************************

void program_class::cgen(ostream &os)
{
  // spim wants comments to start with '#'
  os << "# start of generated code\n";

  initialize_constants();
  envTable = new EnvTable();
  codegen_classtable = new CgenClassTable(classes,os);

  os << "\n# end of generated code\n";
}
```

```
151.
152. ///////////////////////////////////////////////////////////////////////////
153. //
154. //  emit_* procedures
155. //
156. //  emit_X  writes code for operation "X" to the output stream.
157. //  There is an emit_X for each opcode X, as well as emit_ functions
158. //  for generating names according to the naming conventions (see emit.h)
159. //  and calls to support functions defined in the trap handler.
160. //
161. //  Register names and addresses are passed as strings.  See `emit.h'
162. //  for symbolic names you can use to refer to the strings.
163. //
164. ///////////////////////////////////////////////////////////////////////////
165.
166. static void emit_load(char *dest_reg, int offset, char *source_reg, ostream& s)
167. {
168.   s << LW << dest_reg << " " << offset * WORD_SIZE << "(" << source_reg << ")"
169.     << endl;
170. }
171.
172. static void emit_store(char *source_reg, int offset, char *dest_reg, ostream& s)
173. {
174.   s << SW << source_reg << " " << offset * WORD_SIZE << "(" << dest_reg << ")"
175.       << endl;
176. }
177.
178. static void emit_load_imm(char *dest_reg, int val, ostream& s)
179. { s << LI << dest_reg << " " << val << endl; }
180.
181. static void emit_load_address(char *dest_reg, char *address, ostream& s)
182. { s << LA << dest_reg << " " << address << endl; }
183.
184. static void emit_partial_load_address(char *dest_reg, ostream& s)
185. { s << LA << dest_reg << " "; }
186.
187. static void emit_load_bool(char *dest, const BoolConst& b, ostream& s)
188. {
189.   emit_partial_load_address(dest,s);
190.   b.code_ref(s);
191.   s << endl;
192. }
193.
194. static void emit_load_string(char *dest, StringEntry *str, ostream& s)
195. {
196.   emit_partial_load_address(dest,s);
197.   str->code_ref(s);
198.   s << endl;
199. }
200.
201. static void emit_load_int(char *dest, IntEntry *i, ostream& s)
202. {
203.   emit_partial_load_address(dest,s);
204.   i->code_ref(s);
205.   s << endl;
206. }
207.
208. static void emit_move(char *dest_reg, char *source_reg, ostream& s)
209. { s << MOVE << dest_reg << " " << source_reg << endl; }
210.
211. static void emit_neg(char *dest, char *src1, ostream& s)
212. { s << NEG << dest << " " << src1 << endl; }
213.
214. static void emit_add(char *dest, char *src1, char *src2, ostream& s)
215. { s << ADD << dest << " " << src1 << " " << src2 << endl; }
216.
217. static void emit_addu(char *dest, char *src1, char *src2, ostream& s)
218. { s << ADDU << dest << " " << src1 << " " << src2 << endl; }
219.
```

```
220. static void emit_addiu(char *dest, char *src1, int imm, ostream& s)
221. { s << ADDIU << dest << " " << src1 << " " << imm << endl; }
222.
223. static void emit_div(char *dest, char *src1, char *src2, ostream& s)
224. { s << DIV << dest << " " << src1 << " " << src2 << endl; }
225.
226. static void emit_mul(char *dest, char *src1, char *src2, ostream& s)
227. { s << MUL << dest << " " << src1 << " " << src2 << endl; }
228.
229. static void emit_sub(char *dest, char *src1, char *src2, ostream& s)
230. { s << SUB << dest << " " << src1 << " " << src2 << endl; }
231.
232. static void emit_sll(char *dest, char *src1, int num, ostream& s)
233. { s << SLL << dest << " " << src1 << " " << num << endl; }
234.
235. static void emit_jalr(char *dest, ostream& s)
236. { s << JALR << "\t" << dest << endl; }
237.
238. static void emit_jal(char *address,ostream &s)
239. { s << JAL << address << endl; }
240.
241. static void emit_return(ostream& s)
242. { s << RET << endl; }
243.
244. static void emit_gc_assign(ostream& s)
245. { s << JAL << "_GenGC_Assign" << endl; }
246.
247. static void emit_disptable_ref(Symbol sym, ostream& s)
248. {  s << sym << DISPTAB_SUFFIX; }
249.
250. static void emit_init_ref(Symbol sym, ostream& s)
251. { s << sym << CLASSINIT_SUFFIX; }
252.
253. static void emit_label_ref(int l, ostream &s)
254. { s << "label" << l; }
255.
256. static void emit_protobj_ref(Symbol sym, ostream& s)
257. { s << sym << PROTOBJ_SUFFIX; }
258.
259. static void emit_method_ref(Symbol classname, Symbol methodname, ostream& s)
260. { s << classname << METHOD_SEP << methodname; }
261.
262. static void emit_label_def(int l, ostream &s)
263. {
264.   emit_label_ref(l,s);
265.   s << ":" << endl;
266. }
267.
268. static void emit_beqz(char *source, int label, ostream &s)
269. {
270.   s << BEQZ << source << " ";
271.   emit_label_ref(label,s);
272.   s << endl;
273. }
274.
275. static void emit_beq(char *src1, char *src2, int label, ostream &s)
276. {
277.   s << BEQ << src1 << " " << src2 << " ";
278.   emit_label_ref(label,s);
279.   s << endl;
280. }
281.
282. static void emit_bne(char *src1, char *src2, int label, ostream &s)
283. {
284.   s << BNE << src1 << " " << src2 << " ";
285.   emit_label_ref(label,s);
286.   s << endl;
287. }
288.
```

```
289. static void emit_bleq(char *src1, char *src2, int label, ostream &s)
290. {
291.   s << BLEQ << src1 << " " << src2 << " ";
292.   emit_label_ref(label,s);
293.   s << endl;
294. }
295.
296. static void emit_blt(char *src1, char *src2, int label, ostream &s)
297. {
298.   s << BLT << src1 << " " << src2 << " ";
299.   emit_label_ref(label,s);
300.   s << endl;
301. }
302.
303. static void emit_blti(char *src1, int imm, int label, ostream &s)
304. {
305.   s << BLT << src1 << " " << imm << " ";
306.   emit_label_ref(label,s);
307.   s << endl;
308. }
309.
310. static void emit_bgti(char *src1, int imm, int label, ostream &s)
311. {
312.   s << BGT << src1 << " " << imm << " ";
313.   emit_label_ref(label,s);
314.   s << endl;
315. }
316.
317. static void emit_branch(int l, ostream& s)
318. {
319.   s << BRANCH;
320.   emit_label_ref(l,s);
321.   s << endl;
322. }
323.
324. //
325. // Push a register on the stack. The stack grows towards smaller addresses.
326. //
327. static void emit_push(char *reg, ostream& str)
328. {
329.   emit_store(reg,0,SP,str);
330.   emit_addiu(SP,SP,-4,str);
331. }
332.
333. //
334. // Fetch the integer value in an Int object.
335. // Emits code to fetch the integer value of the Integer object pointed
336. // to by register source into the register dest
337. //
338. static void emit_fetch_int(char *dest, char *source, ostream& s)
339. { emit_load(dest, DEFAULT_OBJFIELDS, source, s); }
340.
341. //
342. // Emits code to store the integer value contained in register source
343. // into the Integer object pointed to by dest.
344. //
345. static void emit_store_int(char *source, char *dest, ostream& s)
346. { emit_store(source, DEFAULT_OBJFIELDS, dest, s); }
347.
348. static void emit_test_collector(ostream &s)
349. {
350.   emit_push(ACC, s);
351.   emit_move(ACC, SP, s); // stack end
352.   emit_move(A1, ZERO, s); // allocate nothing
353.   s << JAL << gc_collect_names[cgen_Memmgr] << endl;
354.   emit_addiu(SP,SP,4,s);
355.   emit_load(ACC,0,SP,s);
356. }
357.
```

```
358. static void emit_gc_check(char *source, ostream &s)
359. {
360.   if (source != (char*)A1) emit_move(A1, source, s);
361.   s << JAL << "_gc_check" << endl;
362. }
363.
364. static void emit_start_frame(ostream &s) {
365.     emit_addiu(SP, SP, -12, s);
366.     emit_store(FP, 3, SP, s);
367.     emit_store(SELF, 2, SP, s);
368.     emit_store(RA, 1, SP, s);
369.     emit_addiu(FP, SP, 4, s);
370.     emit_move(SELF, ACC, s);
371. }
372.
373. static void emit_end_frame(ostream &s) {
374.     emit_load(FP, 3, SP, s);
375.     emit_load(SELF, 2, SP, s);
376.     emit_load(RA, 1, SP, s);
377.     emit_addiu(SP, SP, 12, s);
378. }
379.
380. static void emit_abort(int lebal, int lineno, ostream &s) {
381.     emit_bne(ACC, ZERO, lebal, s);
382.     s << LA << ACC << " str_const0" << endl;
383.     emit_load_imm(T1, lineno, s);
384.     emit_jal("_dispatch_abort", s);
385. }
386.
387. static void emit_gc_update(char *src, int offset, ostream &s) {
388.     if (cgen_Memmgr == GC_GENGC) {
389.         emit_addiu(A1, src, WORD_SIZE * offset, s);
390.         emit_gc_assign(s);
391.    }
392. }
393.
394. static void emit_load_t1_t2(ostream &s, Expression e1, Expression e2) {
395.     // get two expression's return val to t1, t2
396.     e1->code(s);
397.     emit_push(ACC, s); // 将 e1 产生的地址压栈
398.     e2->code(s);
399.
400.     emit_load(T1, 1, SP, s);
401.     emit_move(T2, ACC, s);
402.     emit_addiu(SP, SP, 4, s);
403. }
404.
405. static void emit_breakpoint_debug(ostream &s) {
406.     StringEntry* strentry = stringtable.lookup_string(DEBUG_FLAG);
407.     emit_push(ACC, s);
408.     emit_load_string(ACC, strentry, s);
409.     emit_push(ACC, s);
410.     emit_jal("IO.out_string", s);
411.     emit_addiu(SP, SP, 4, s);   // no need to recover ~~
412.     emit_load(ACC, 1, SP, s);   // recover ACC
413.     emit_addiu(SP, SP, 4, s);
414. }
415.
416. //////////////////////////////////////////////////////////////////////////
417. //
418. // coding strings, ints, and booleans
419. //
420. // Cool has three kinds of constants: strings, ints, and booleans.
421. // This section defines code generation for each type.
422. //
423. // All string constants are listed in the global "stringtable" and have
424. // type StringEntry.  StringEntry methods are defined both for String
425. // constant definitions and references.
426. //
```

```
427. // All integer constants are listed in the global "inttable" and have
428. // type IntEntry.  IntEntry methods are defined for Int
429. // constant definitions and references.
430. //
431. // Since there are only two Bool values, there is no need for a table.
432. // The two booleans are represented by instances of the class BoolConst,
433. // which defines the definition and reference methods for Bools.
434. //
435. ///////////////////////////////////////////////////////////////////////////////
436.
437. //
438. // Strings
439. //
440. void StringEntry::code_ref(ostream& s)
441. {
442.   s << STRCONST_PREFIX << index;
443. }
444.
445. //
446. // Emit code for a constant String.
447. // You should fill in the code naming the dispatch table.
448. //
449.
450. void StringEntry::code_def(ostream& s, int stringclasstag)
451. {
452.   IntEntryP lensym = inttable.add_int(len);
453.
454.   // Add -1 eye catcher
455.   s << WORD << "-1" << endl;
456.
457.   code_ref(s);  s  << LABEL                                             // label
458.       << WORD << stringclasstag << endl                                 // tag
459.       << WORD << (DEFAULT_OBJFIELDS + STRING_SLOTS + (len+4)/4) << endl // size
460.       << WORD;
461.     emit_disptable_ref(Str, s);
462.
463.  /***** Add dispatch information for class String ******/
464.       s << endl;                                              // dispatch table
465.       s << WORD;  lensym->code_ref(s);  s << endl;            // string length
466.   emit_string_constant(s,str);                                // ascii string
467.   s << ALIGN;                                                 // align to word
468. }
469.
470. //
471. // StrTable::code_string
472. // Generate a string object definition for every string constant in the
473. // stringtable.
474. //
475. void StrTable::code_string_table(ostream& s, int stringclasstag)
476. {
477.   for (List<StringEntry> *l = tbl; l; l = l->tl())
478.     l->hd()->code_def(s,stringclasstag);
479. }
480.
481. //
482. // Ints
483. //
484. void IntEntry::code_ref(ostream &s)
485. {
486.   s << INTCONST_PREFIX << index;
487. }
488.
489. //
490. // Emit code for a constant Integer.
491. // You should fill in the code naming the dispatch table.
492. //
493.
494. void IntEntry::code_def(ostream &s, int intclasstag)
495. {
```

```
  // Add -1 eye catcher
  s << WORD << "-1" << endl;
  code_ref(s);  s << LABEL                                // label
      << WORD << intclasstag << endl                      // class tag
      << WORD << (DEFAULT_OBJFIELDS + INT_SLOTS) << endl  // object size
      << WORD;
    emit_disptable_ref(Int, s);

 /***** Add dispatch information for class Int ******/

      s << endl;                                          // dispatch table
      s << WORD << str << endl;                           // integer value
}

//
// IntTable::code_string_table
// Generate an Int object definition for every Int constant in the
// inttable.
//
void IntTable::code_string_table(ostream &s, int intclasstag)
{
  for (List<IntEntry> *l = tbl; l; l = l->tl())
      l->hd()->code_def(s,intclasstag);
}

//
// Bools
//
BoolConst::BoolConst(int i) : val(i) { assert(i == 0 || i == 1); }

void BoolConst::code_ref(ostream& s) const
{
  s << BOOLCONST_PREFIX << val;
}

//
// Emit code for a constant Bool.
// You should fill in the code naming the dispatch table.
//

void BoolConst::code_def(ostream& s, int boolclasstag)
{
  // Add -1 eye catcher
  s << WORD << "-1" << endl;

  code_ref(s);  s << LABEL                                // label
      << WORD << boolclasstag << endl                     // class tag
      << WORD << (DEFAULT_OBJFIELDS + BOOL_SLOTS) << endl   // object size
      << WORD;
    emit_disptable_ref(Bool, s);

 /***** Add dispatch information for class Bool ******/

      s << endl;                                          // dispatch table
      s << WORD << val << endl;                           // value (0 or 1)
}

//////////////////////////////////////////////////////////////////////////////
//
//  CgenClassTable methods
//
//////////////////////////////////////////////////////////////////////////////

//***************************************************
//
//  Emit code to start the .data segment and to
//  declare the global names.
//
//***************************************************
```

```
565.
566. void CgenClassTable::code_global_data()
567. {
568.   Symbol main    = idtable.lookup_string(MAINNAME);
569.   Symbol string  = idtable.lookup_string(STRINGNAME);
570.   Symbol integer = idtable.lookup_string(INTNAME);
571.   Symbol boolc   = idtable.lookup_string(BOOLNAME);
572.
573.   str << "\t.data\n" << ALIGN;
574.   //
575.   // The following global names must be defined first.
576.   //
577.   str << GLOBAL << CLASSNAMETAB << endl;
578.   str << GLOBAL; emit_protobj_ref(main,str);    str << endl;
579.   str << GLOBAL; emit_protobj_ref(integer,str); str << endl;
580.   str << GLOBAL; emit_protobj_ref(string,str);  str << endl;
581.   str << GLOBAL; falsebool.code_ref(str);  str << endl;
582.   str << GLOBAL; truebool.code_ref(str);   str << endl;
583.   str << GLOBAL << INTTAG << endl;
584.   str << GLOBAL << BOOLTAG << endl;
585.   str << GLOBAL << STRINGTAG << endl;
586.
587.   //
588.   // We also need to know the tag of the Int, String, and Bool classes
589.   // during code generation.
590.   //
591.   str << INTTAG << LABEL
592.       << WORD << intclasstag << endl;
593.   str << BOOLTAG << LABEL
594.       << WORD << boolclasstag << endl;
595.   str << STRINGTAG << LABEL
596.       << WORD << stringclasstag << endl;
597. }
598.
599. //***************************************************
600. //
601. //  Emit code to start the .text segment and to
602. //  declare the global names.
603. //
604. //***************************************************
605.
606. void CgenClassTable::code_global_text()
607. {
608.   str << GLOBAL << HEAP_START << endl
609.       << HEAP_START << LABEL
610.       << WORD << 0 << endl
611.       << "\t.text" << endl
612.       << GLOBAL;
613.   emit_init_ref(idtable.add_string("Main"), str);
614.   str << endl << GLOBAL;
615.   emit_init_ref(idtable.add_string("Int"),str);
616.   str << endl << GLOBAL;
617.   emit_init_ref(idtable.add_string("String"),str);
618.   str << endl << GLOBAL;
619.   emit_init_ref(idtable.add_string("Bool"),str);
620.   str << endl << GLOBAL;
621.   emit_method_ref(idtable.add_string("Main"), idtable.add_string("main"), str);
622.   str << endl;
623. }
624.
625. void CgenClassTable::code_bools(int boolclasstag)
626. {
627.   falsebool.code_def(str,boolclasstag);
628.   truebool.code_def(str,boolclasstag);
629. }
630.
631. void CgenClassTable::code_select_gc()
632. {
633.   //
```

```
  // Generate GC choice constants (pointers to GC functions)
  //
  str << GLOBAL << "_MemMgr_INITIALIZER" << endl;
  str << "_MemMgr_INITIALIZER:" << endl;
  str << WORD << gc_init_names[cgen_Memmgr] << endl;
  str << GLOBAL << "_MemMgr_COLLECTOR" << endl;
  str << "_MemMgr_COLLECTOR:" << endl;
  str << WORD << gc_collect_names[cgen_Memmgr] << endl;
  str << GLOBAL << "_MemMgr_TEST" << endl;
  str << "_MemMgr_TEST:" << endl;
  str << WORD << (cgen_Memmgr_Test == GC_TEST) << endl;
}

//*****************************************************
//
// Emit code to reserve space for and initialize all of
// the constants.  Class names should have been added to
// the string table (in the supplied code, is is done
// during the construction of the inheritance graph), and
// code for emitting string constants as a side effect adds
// the string's length to the integer table.  The constants
// are emmitted by running through the stringtable and inttable
// and producing code for each entry.
//
//*****************************************************

void CgenClassTable::code_constants()
{
  //
  // Add constants that are required by the code generator.
  //
  stringtable.add_string("");
  stringtable.add_string(DEBUG_FLAG); // for test
  inttable.add_string("0");
  stringtable.code_string_table(str, stringclasstag);
  inttable.code_string_table(str, intclasstag);
  code_bools(boolclasstag);
}

CgenClassTable::CgenClassTable(Classes classes, ostream& s) : nds(NULL) , str(s),
labelid_(0),
curr_cgenclass_(nullptr)
{
   stringclasstag = 0 /* Change to your String class tag here */;
   intclasstag =    0 /* Change to your Int class tag here */;
   boolclasstag =   0 /* Change to your Bool class tag here */;

   codegen_classtable = this;
   enterscope();
   if (cgen_debug) cout << "Building CgenClassTable" << endl;
   install_basic_classes();
   install_classes(classes);
   install_name_to_cgen();
   build_inheritance_tree();
   install_classtags();
   install_attrs_and_methods();

   code();
   exitscope();
}

void CgenClassTable::install_basic_classes()
{

// The tree package uses these globals to annotate the classes built below.
  //curr_lineno  = 0;
  Symbol filename = stringtable.add_string("<basic class>");

//
```

```
// A few special class names are installed in the lookup table but not
// the class list.  Thus, these classes exist, but are not part of the
// inheritance hierarchy.
// No_class serves as the parent of Object and the other special classes.
// SELF_TYPE is the self class; it cannot be redefined or inherited.
// prim_slot is a class known to the code generator.
//
  addid(No_class,
    new CgenNode(class_(No_class,No_class,nil_Features(),filename),
                Basic,this));
  addid(SELF_TYPE,
    new CgenNode(class_(SELF_TYPE,No_class,nil_Features(),filename),
                Basic,this));
  addid(prim_slot,
    new CgenNode(class_(prim_slot,No_class,nil_Features(),filename),
                Basic,this));

//
// The Object class has no parent class. Its methods are
//        cool_abort() : Object    aborts the program
//        type_name() : Str        returns a string representation of class name
//        copy() : SELF_TYPE       returns a copy of the object
//
// There is no need for method bodies in the basic classes---these
// are already built in to the runtime system.
//
  install_class(
   new CgenNode(
    class_(Object,
       No_class,
       append_Features(
           append_Features(
           single_Features(method(cool_abort, nil_Formals(), Object, no_expr())),
           single_Features(method(type_name, nil_Formals(), Str, no_expr()))),
           single_Features(method(copy, nil_Formals(), SELF_TYPE, no_expr()))),
       filename),
    Basic,this));

//
// The IO class inherits from Object. Its methods are
//        out_string(Str) : SELF_TYPE         writes a string to the output
//        out_int(Int) : SELF_TYPE              "    an int    "  "     "
//        in_string() : Str                   reads a string from the input
//        in_int() : Int                        "   an int     "  "     "
//
   install_class(
    new CgenNode(
     class_(IO,
           Object,
           append_Features(
           append_Features(
           append_Features(
           single_Features(method(out_string, single_Formals(formal(arg, Str)),
                      SELF_TYPE, no_expr())),
           single_Features(method(out_int, single_Formals(formal(arg, Int)),
                      SELF_TYPE, no_expr()))),
           single_Features(method(in_string, nil_Formals(), Str, no_expr()))),
           single_Features(method(in_int, nil_Formals(), Int, no_expr()))),
       filename),
    Basic,this));

//
// The Int class has no methods and only a single attribute, the
// "val" for the integer.
//
   install_class(
    new CgenNode(
     class_(Int,
        Object,
```

```
            single_Features(attr(val, prim_slot, no_expr())),
        filename),
     Basic,this));

//
// Bool also has only the "val" slot.
//
    install_class(
     new CgenNode(
      class_(Bool, Object, single_Features(attr(val, prim_slot, no_expr())),filename),
      Basic,this));

//
// The class Str has a number of slots and operations:
//       val                                  ???
//       str_field                            the string itself
//       length() : Int                       length of the string
//       concat(arg: Str) : Str               string concatenation
//       substr(arg: Int, arg2: Int): Str     substring
//
   install_class(
    new CgenNode(
      class_(Str,
         Object,
             append_Features(
             append_Features(
             append_Features(
             append_Features(
             single_Features(attr(val, Int, no_expr())),
            single_Features(attr(str_field, prim_slot, no_expr())),
            single_Features(method(length, nil_Formals(), Int, no_expr())),
            single_Features(method(concat,
                   single_Formals(formal(arg, Str)),
                   Str,
                   no_expr())),
        single_Features(method(substr,
                   append_Formals(single_Formals(formal(arg, Int)),
                          single_Formals(formal(arg2, Int))),
                   Str,
                   no_expr())),
         filename),
        Basic,this));

}

bool CgenClassTable::get_attr_offset(Symbol cls, Symbol attr, int *offset) {
    if (attr_offset_map_.find(cls) == attr_offset_map_.end()) {
        return false;
    }
    if (attr_offset_map_[cls].find(attr) == attr_offset_map_[cls].end()) {
        return false;
    }
    *offset = attr_offset_map_[cls][attr];
    return true;
}
// CgenClassTable::install_class
// CgenClassTable::install_classes
//
// install_classes enters a list of classes in the symbol table.
//
void CgenClassTable::install_class(CgenNodeP nd)
{
  Symbol name = nd->get_name();

  if (probe(name))
    {
      return;
    }

```

```
841.   // The class name is legal, so add it to the list of classes
842.   // and the symbol table.
843.   nds = new List<CgenNode>(nd,nds);
844.   addid(name,nd);
845. }
846.
847. void CgenClassTable::install_classes(Classes cs)
848. {
849.   for(int i = cs->first(); cs->more(i); i = cs->next(i))
850.     install_class(new CgenNode(cs->nth(i),NotBasic,this));
851. }
852.
853. void CgenClassTable::install_classtags() {
854.     int curr_tag = 0;
855.     CgenNodeP curr_cgennode = name_to_cgen_map_[Object];
856.
857.     std::function<void(CgenNodeP)> dfs_set_tags = [&](CgenNodeP curr_cgen) {
858.         curr_cgen->set_classtag(curr_tag);
859.         class_tag_map_[curr_tag] = curr_cgen->get_name();
860.         // str << "# set the class " << curr_cgen->get_name() << " the tag is " << curr_tag << endl;
861.         Symbol curr_name = curr_cgen->get_name();
862.         if (curr_name == Str) {
863.             stringclasstag = curr_tag;
864.         } else if (curr_name == Bool) {
865.             boolclasstag = curr_tag;
866.         } else if (curr_name == Int) {
867.             intclasstag = curr_tag;
868.         }
869.         curr_tag++;
870.         for (List<CgenNode> *l = curr_cgen->get_children(); l; l = l->tl()) {
871.             dfs_set_tags(l->hd());
872.         }
873.     };
874.     std::function<int(CgenNodeP)> set_des_cnt = [&](CgenNodeP curr_cgen) {
875.         int des_cnt = 1;
876.         for (List<CgenNode> *l = curr_cgen->get_children(); l; l = l->tl()) {
877.             des_cnt += set_des_cnt(l->hd());
878.         }
879.         curr_cgen->set_descendants_cnt(des_cnt - 1);
880.         return des_cnt;
881.     };
882.     dfs_set_tags(curr_cgennode);
883.     set_des_cnt(curr_cgennode);
884. }
885.
886. void CgenClassTable::install_attrs_and_methods() {
887.     CgenNodeP curr_cgennode;
888.     Features curr_features;
889.     // install features defined by the class
890.     // method_map and attr_map only have features defined
891.     for (List<CgenNode> *l = nds; l; l = l->tl()) {
892.         curr_cgennode = l->hd();
893.         curr_features = curr_cgennode->get_features();
894.         Feature curr_feature;
895.         Symbol curr_cgen_name = curr_cgennode->get_name();
896.         for (int i = curr_features->first(); curr_features->more(i); i = curr_features->next(i)) {
897.             curr_feature = curr_features->nth(i);
898.             // for method
899.             if (curr_feature->is_method()) {
900.                 class_method_map_[curr_cgen_name].push_back(static_cast<method_class*>(curr_feature));
901.             }
902.             // for attr
903.             else {
904.                 class_attr_map_[curr_cgennode->get_name()].push_back(static_cast<attr_class*>(curr_feature));
905.             }
906.         }
907.     }
908.     // install offset
909.     for (List<CgenNode> *l = nds; l; l = l->tl()) {
```

```
910.         curr_cgennode = l->hd();
911.         Symbol curr_name = curr_cgennode->get_name();
912.         // acquire all parents
913.         auto chain = curr_cgennode->get_parents_list();
914.         for (auto parent : chain) {
915.             // child will inherit parent's methods
916.             auto methods = class_method_map_[parent->get_name()];
917.             for (auto method : methods) {
918.                 Symbol meth_name = method->get_name();
919.                 // undefined before
920.                 if (meth_offset_map_[curr_name].find(meth_name) == meth_offset_map_[curr_name].end()) {
921.                     dispatch_tab_map_[curr_name].push_back({parent->get_name(), meth_name});
922.                     // defined it in the tail
923.                     meth_offset_map_[curr_name][meth_name] = dispatch_tab_map_[curr_name].size() - 1;
924.                 }
925.                 else {
926.                     int meth_offset = meth_offset_map_[curr_name][meth_name];
927.                     dispatch_tab_map_[curr_name][meth_offset] = {parent->get_name(), meth_name};
928.                 }
929.             }
930.         }
931.         curr_cgennode->set_chain_depth(chain.size());
932.         parent_chain_map_[curr_name] = std::move(chain);
933.     }
934. }
935.
936. void CgenClassTable::install_name_to_cgen() {
937.     CgenNodeP curr_cgen;
938.     for (List<CgenNode> *l = nds; l; l = l->tl()) {
939.         curr_cgen = l->hd();
940.         name_to_cgen_map_[curr_cgen->get_name()] = curr_cgen;
941.     }
942. }
943. //
944. // CgenClassTable::build_inheritance_tree
945. //
946. void CgenClassTable::build_inheritance_tree()
947. {
948.   for(List<CgenNode> *l = nds; l; l = l->tl())
949.       set_relations(l->hd());
950. }
951.
952. //
953. // CgenClassTable::set_relations
954. //
955. // Takes a CgenNode and locates its, and its parent's, inheritance nodes
956. // via the class table.  Parent and child pointers are added as appropriate.
957. //
958. void CgenClassTable::set_relations(CgenNodeP nd)
959. {
960.   CgenNode *parent_node = probe(nd->get_parent());
961.   nd->set_parentnd(parent_node);
962.   parent_node->add_child(nd);
963. }
964.
965. void CgenNode::add_child(CgenNodeP n)
966. {
967.   children = new List<CgenNode>(n,children);
968. }
969.
970. void CgenNode::set_parentnd(CgenNodeP p)
971. {
972.   assert(parentnd == NULL);
973.   assert(p != NULL);
974.   parentnd = p;
975. }
976.
977. std::vector<CgenNodeP> CgenNode::get_parents_list() {
978.     CgenNodeP pa = this;
```

```
    std::vector<CgenNodeP> parents;
    while (true) {
        parents.push_back(pa);
        if (pa->get_name() == Object) {
            break;
        }
        pa = pa->get_parentnd();
    }
    std::reverse(parents.begin(), parents.end());
    return parents;
}

void CgenClassTable::code_class_nametabs() {
    str << CLASSNAMETAB << LABEL;
    int len = class_tag_map_.size();

    StringEntry* str_entry;
    for (int i = 0; i < len; i++) {
        Symbol name = class_tag_map_[i];
        str_entry = stringtable.lookup_string(name->get_string());
        str << WORD;
        str_entry->code_ref(str);
        str << endl;
    }
}

void CgenClassTable::code_class_objtabs() {

    str << CLASSOBJTAB << LABEL;
    int len = class_tag_map_.size();

    StringEntry* str_entry;
    for (int i = 0; i < len; i++) {
        Symbol name = class_tag_map_[i];
        str << WORD;
        emit_protobj_ref(name, str);
        str << endl;

        str << WORD;
        emit_init_ref(name, str);
        str << endl;
    }
}

bool CgenClassTable::get_meth_offset(Symbol cls, Symbol meth, int *offset) {
    if (meth_offset_map_.find(cls) == meth_offset_map_.end()) {
        return false;
    }
    auto find_meth = meth_offset_map_[cls].find(meth);
    if (find_meth != meth_offset_map_[cls].end()) {
        // std::cout << "# find the meth is from " << find_meth->second << endl;
        *offset = find_meth->second;
        return true;
    }
    return false;
}

void CgenClassTable::code_object_disptabs() {
    List<CgenNode> *nd_list = nds;
    CgenNode *head;

    for (auto &[class_name, distab] : dispatch_tab_map_) {
        emit_disptable_ref(class_name, str);
        str << LABEL;
        for (auto &[meth_class, meth_name] : distab) {
            str << WORD;
            emit_method_ref(meth_class, meth_name, str);
            str << endl;
        }
```

```
1048.     }
1049.
1050. }
1051.
1052. void CgenClassTable::code_protobjs() {
1053.     CgenNodeP curr_cgennode, curr_parent;
1054.     for (List<CgenNode> *l = nds; l; l = l->tl()) {
1055.         curr_cgennode = l->hd();
1056.         Symbol curr_name = curr_cgennode->get_name();
1057.         str << WORD << -1 << endl;
1058.         emit_protobj_ref(curr_name, str);
1059.         str << LABEL;
1060.         str << WORD << curr_cgennode->get_classtag() << endl;
1061.         int attr_cnt = class_attr_map_[curr_name].size();
1062.         auto parents = curr_cgennode->get_parents_list();
1063.         for (auto parent : parents) {
1064.             Symbol parent_name = parent->get_name();
1065.             if (parent_name == curr_cgennode->get_name()) {
1066.                 continue;
1067.             }
1068.             attr_cnt += class_attr_map_[parent_name].size();
1069.         }
1070.         str << WORD << attr_cnt + ATTR_BASE_OFFSET << endl;
1071.         str << WORD;
1072.         emit_disptable_ref(curr_name, str);
1073.         str << endl;
1074.         int offset = ATTR_BASE_OFFSET; //
1075.         for (auto parent : parents) {
1076.             Symbol parent_name = parent->get_name();
1077.             const attrList& curr_attr_list = class_attr_map_[parent_name];
1078.             for (auto attr : curr_attr_list) {
1079.                 // 先打印出来 name+attrname 试试看
1080.                 // str << parent_name << " " << attr->get_name() << endl;
1081.                 Symbol attr_type = attr->get_type();
1082.                 str << WORD;
1083.                 attr_offset_map_[curr_name][attr->get_name()] = offset++;
1084.                 if (attr_type == Str) {   // 这个地方的处理尚待修改
1085.                     StringEntry *strentry = stringtable.lookup_string("");
1086.                     strentry->code_ref(str);
1087.                 } else if (attr_type == Bool) {
1088.                     falsebool.code_ref(str);
1089.                 } else if (attr_type == Int) {
1090.                     IntEntry *intentry = inttable.lookup_string("0");
1091.                     intentry->code_ref(str);
1092.                 } else {
1093.                     str << 0;
1094.                 }
1095.                 // str << " " << attr_offset_map_[curr_name][attr] << " ";
1096.                 str << endl;
1097.             }
1098.         }
1099.     }
1100. }
1101.
1102. void CgenClassTable::code_object_inits() {
1103.     CgenNodeP curr_cgen;
1104.     for (List<CgenNode> *l = nds; l; l = l->tl()) {
1105.         curr_cgen = l->hd();
1106.         curr_cgenclass_ = curr_cgen;
1107.         emit_init_ref(curr_cgen->get_name(), str);
1108.         str << LABEL;
1109.         emit_start_frame(str);
1110.         envTable->enterframe();
1111.
1112.         // install parent's
1113.         CgenNodeP parent = curr_cgen->get_parentnd();
1114.         if (parent && parent->get_name() != No_class) {
1115.             str << JAL;
1116.             emit_init_ref(parent->get_name(), str);
```

```
            str << endl;
        }
        // install self's
        // so you see the benefits of saperating attr and meth
        const attrList& curr_attrs = class_attr_map_[curr_cgen->get_name()];
        for (auto attr : curr_attrs) {
            Expression init_expr = attr->get_init();
            Symbol attr_type = attr->get_type();
            if (!init_expr->is_empty()) {
                init_expr->code(str);
                int attr_off = attr_offset_map_[curr_cgen->get_name()][attr->get_name()];
                emit_store(ACC, attr_off, SELF, str);
                emit_gc_update(SELF, attr_off, str);
            }
        }
        emit_move(ACC, SELF, str);
        emit_end_frame(str);
        envTable->exitframe();
        emit_return(str);
    }
}

void CgenClassTable::code_methods() {
    CgenNodeP curr_cgennode;
    for (List<CgenNode> *l = nds; l; l = l->tl()) {
        curr_cgennode = l->hd();
        curr_cgenclass_ = curr_cgennode;
        Symbol cgenname = curr_cgenclass_->get_name();
        if (is_basic_class(cgenname)) { // basic method need to do nothing
            continue;
        }
        // all methods public means class only need to emit their own methods
        // dispatch will tell which method is actually called
        const methodList& methods = class_method_map_[cgenname];
        Formals curr_formals;
        std::list<Formal> formal_list;
        for (auto method : methods) {
            formal_list.clear();
            envTable->enterframe();
            curr_formals = method->formals;
            for (int i = curr_formals->first(); curr_formals->more(i); i = curr_formals->next(i)) {
                formal_list.push_front(curr_formals->nth(i));
            }
            for (auto formal : formal_list) {
                envTable->add_formal_id(formal->get_name());
            }
            emit_method_ref(curr_cgenclass_->get_name(), method->get_name(), str);
            str << LABEL;
            emit_start_frame(str);
            method->expr->code(str);
            emit_end_frame(str);
            emit_addiu(SP, SP, formal_list.size() * WORD_SIZE, str);
            emit_return(str);
            envTable->exitframe();
        }
    }
}

void CgenClassTable::code(){
  code_global_data();
  code_select_gc();
  code_constants();

//                Add your code to emit
//                  - prototype objects
//                  - class_nameTab
//                  - dispatch tables
//
  code_global_text();
```

```
1186. //                Add your code to emit
1187. //                  - object initializer
1188. //                  - the class methods
1189. //                  - etc...
1190.   code_class_nametabs();
1191.   code_class_objtabs();
1192.   code_object_disptabs();
1193.   code_protobjs();
1194.   code_object_inits();
1195.   code_methods();
1196.
1197. }
1198.
1199. CgenNodeP CgenClassTable::root()
1200. {
1201.    return probe(Object);
1202. }
1203.
1204. ///////////////////////////////////////////////////////////////////////
1205. //
1206. // CgenNode methods
1207. //
1208. ///////////////////////////////////////////////////////////////////////
1209.
1210. CgenNode::CgenNode(Class_ nd, Basicness bstatus, CgenClassTableP ct) :
1211.    class__class((const class__class &) *nd),
1212.    chain_depth_(0),
1213.    class_tag_(0),
1214.    parentnd(NULL),
1215.    children(NULL),
1216.    basic_status(bstatus)
1217. {
1218.    stringtable.add_string(name->get_string());          // Add class name to string table
1219. }
1220.
1221. //******************************************************************
1222. //
1223. //   Fill in the following methods to produce code for the
1224. //   appropriate expression.  You may add or remove parameters
1225. //   as you wish, but if you do, remember to change the parameters
1226. //   of the declarations in `cool-tree.h'  Sample code for
1227. //   constant integers, strings, and booleans are provided.
1228. //
1229. //******************************************************************
1230.
1231. void assign_class::code(ostream &s) {
1232.     expr->code(s);
1233.     CgenNodeP curr_cgen = codegen_classtable->get_curr_class();
1234.     int offset;
1235.     if (envTable->lookup(name, &offset)) {
1236.         emit_store(ACC, offset, FP, s);
1237.         emit_gc_update(FP, offset, s);
1238.         return;
1239.     }
1240.     if (codegen_classtable->get_attr_offset(curr_cgen->get_name(), name, &offset)) { // 属于 attr 类型的
1241.         emit_store(ACC, offset, SELF, s);
1242.         emit_gc_update(SELF, offset, s);
1243.     }
1244. }
1245.
1246. void static_dispatch_class::code(ostream &s) {
1247.     Expression curr_expr;
1248.     // save parameters
1249.     for (int i = actual->first(); actual->more(i); i = actual->next(i)) {
1250.         curr_expr = actual->nth(i);
1251.         curr_expr->code(s);
1252.         emit_push(ACC, s);
1253.     }
1254.     expr->code(s);
```

```
1255.
1256.     int lebalid = codegen_classtable->get_labelid_and_add();
1257.
1258.     emit_abort(lebalid, get_line_number(), s);
1259.
1260.     emit_label_def(lebalid, s);
1261.     // emit_load(T1, DISPTABLE_OFFSET, ACC, s);
1262.     std::string suffix = DISPTAB_SUFFIX;
1263.     std::string distab_addr = type_name->get_string() + suffix;
1264.     emit_load_address(T1, const_cast<char*>(distab_addr.c_str()), s);
1265.     int offset;
1266.     codegen_classtable->get_meth_offset(type_name, name, &offset);
1267.
1268.     emit_load(T1, offset, T1, s);
1269.     emit_jalr(T1, s);
1270. }
1271.
1272. void dispatch_class::code(ostream &s) {
1273.     Expression curr_expr;
1274.     for (int i = actual->first(); actual->more(i); i = actual->next(i)) {
1275.         curr_expr = actual->nth(i);
1276.         curr_expr->code(s);
1277.         emit_push(ACC, s);
1278.     }
1279.     expr->code(s);
1280.     int lebalid = codegen_classtable->get_labelid_and_add();
1281.     emit_abort(lebalid, get_line_number(), s);
1282.     emit_label_def(lebalid, s);
1283.     emit_load(T1, DISPTABLE_OFFSET, ACC, s);
1284.     int offset;
1285.     Symbol expr_type = expr->get_type();
1286.     if (expr_type == SELF_TYPE) {
1287.         expr_type = codegen_classtable->get_curr_class()->get_name();
1288.     }
1289.     codegen_classtable->get_meth_offset(expr_type, name, &offset);
1290.     emit_load(T1, offset, T1, s);
1291.
1292.     emit_jalr(T1, s);
1293. }
1294.
1295. void cond_class::code(ostream &s) {
1296.     pred->code(s);
1297.     // load two params for compare
1298.     emit_load(T1, ATTR_BASE_OFFSET, ACC, s);
1299.     emit_move(T2, ZERO, s);
1300.
1301.     int out_lebal = codegen_classtable->get_labelid_and_add();
1302.     int false_lebal = codegen_classtable->get_labelid_and_add();
1303.
1304.     emit_beq(T1, T2, false_lebal, s);
1305.     // true cond
1306.     then_exp->code(s);
1307.     emit_branch(out_lebal, s);
1308.     // false cond
1309.     emit_label_def(false_lebal, s);
1310.     else_exp->code(s);
1311.     emit_label_def(out_lebal, s);
1312. }
1313.
1314. void loop_class::code(ostream &s) {
1315.     int start_lebal = codegen_classtable->get_labelid_and_add();
1316.     int end_lebal = codegen_classtable->get_labelid_and_add();
1317.
1318.     emit_label_def(start_lebal, s);
1319.     pred->code(s);
1320.     emit_load(T1, ATTR_BASE_OFFSET, ACC, s);
1321.     emit_beq(T1, ZERO, end_lebal, s);
1322.
1323.     body->code(s);
```

```
1324.     emit_branch(start_lebal, s);
1325.     emit_label_def(end_lebal, s);
1326. 
1327.     emit_move(ACC, ZERO, s);
1328. }
1329. 
1330. void typcase_class::code(ostream &s) {
1331.     expr->code(s);
1332. 
1333.     int no_void_lebal = codegen_classtable->get_labelid_and_add();
1334.     int out_lebal = codegen_classtable->get_labelid_and_add();
1335.     int notmatch_lebal = codegen_classtable->get_labelid_and_add();
1336.     emit_bne(ACC, ZERO, no_void_lebal, s);
1337. 
1338.     s << LA << ACC << " str_const0" << endl;
1339.     emit_load_imm(T1, get_line_number(), s);
1340.     emit_jal("_case_abort2", s);
1341. 
1342.     emit_label_def(no_void_lebal, s);
1343.     emit_load(T1, TAG_OFFSET, ACC, s);
1344. 
1345.     std::vector<Case> sorted_cases;
1346.     sorted_cases.reserve(cases->len());
1347. 
1348.     Case curr_case;
1349.     for (int i = cases->first(); cases->more(i); i = cases->next(i)) {
1350.         curr_case = cases->nth(i);
1351.         sorted_cases.push_back(curr_case);
1352.     }
1353.     std::function<bool(Case, Case)> sort_comp = [&](Case a, Case b)-> bool {
1354.         return codegen_classtable->get_cgennode(a->get_type_decl())->get_chain_depth() >
1355.             codegen_classtable->get_cgennode(b->get_type_decl())->get_chain_depth();
1356.     };
1357.     std::sort(sorted_cases.begin(), sorted_cases.end(), sort_comp);
1358. 
1359.     for (auto case_class : sorted_cases) {
1360.         // emit_branch(out_lebal, s);
1361.         Symbol cgen_type = case_class->get_type_decl();
1362.         CgenNodeP cgen = codegen_classtable->get_cgennode(cgen_type);
1363.         int next_case_lebal = codegen_classtable->get_labelid_and_add();
1364.         int start_tag = cgen->get_classtag();
1365.         int end_tag = start_tag + cgen->get_descendants_cnt();
1366.         emit_blti(T1, start_tag, next_case_lebal, s);
1367.         emit_bgti(T1, end_tag, next_case_lebal, s);
1368. 
1369.         emit_push(ACC, s);
1370.         envTable->enterscope();
1371.         envTable->add_local_id(case_class->get_name());
1372. 
1373.         case_class->get_expr()->code(s);
1374. 
1375.         envTable->exitscope();
1376.         emit_addiu(SP, SP, 4, s);
1377. 
1378.         emit_branch(out_lebal, s);
1379.         emit_label_def(next_case_lebal, s);
1380. 
1381.     }
1382. 
1383.     emit_label_def(notmatch_lebal, s);
1384.     emit_jal("_case_abort", s);
1385.     emit_label_def(out_lebal, s);
1386. }
1387. 
1388. void block_class::code(ostream &s) {
1389.     Expressions exprs = body;
1390.     Expression expr;
1391.     for (int i = exprs->first(); exprs->more(i); i = exprs->next(i)) {
1392.         expr = exprs->nth(i);
```

```
        expr->code(s);
    }
}

void let_class::code(ostream &s) {
    init->code(s);
    if (init->is_empty()) {
        if (type_decl == Int) {
            emit_load_int(ACC, inttable.lookup_string("0"), s);
        } else if (type_decl == Str) {
            emit_load_string(ACC, stringtable.lookup_string(""), s);
        } else if (type_decl == Bool) {
            emit_load_bool(ACC, falsebool, s);
        }
    }

    emit_push(ACC, s);
    envTable->enterscope();
    envTable->add_local_id(identifier);

    body->code(s);

    envTable->exitscope();
    emit_addiu(SP, SP, 4, s);
}

void plus_class::code(ostream &s) {
    // s << "# coding plus class\n";
    e1->code(s);
    emit_push(ACC, s);
    e2->code(s);
    emit_jal("Object.copy", s);

    emit_load(T1, 1, SP, s);
    emit_load(T2, ATTR_BASE_OFFSET, T1, s);
    emit_load(T3, ATTR_BASE_OFFSET, ACC, s);
    emit_addiu(SP, SP, 4, s);
    emit_add(T3, T2, T3, s);
    emit_store(T3, ATTR_BASE_OFFSET, ACC, s);
}

void sub_class::code(ostream &s) {
    e1->code(s);
    emit_push(ACC, s);
    e2->code(s);
    emit_jal("Object.copy", s);

    emit_load(T1, 1, SP, s);
    emit_load(T2, ATTR_BASE_OFFSET, T1, s);
    emit_load(T3, ATTR_BASE_OFFSET, ACC, s);
    emit_addiu(SP, SP, 4, s);

    emit_sub(T3, T2, T3, s);
    emit_store(T3, ATTR_BASE_OFFSET, ACC, s);
}

void mul_class::code(ostream &s) {
    e1->code(s);
    emit_push(ACC, s);
    e2->code(s);
    emit_jal("Object.copy", s);

    emit_load(T1, 1, SP, s);
    emit_load(T2, ATTR_BASE_OFFSET, T1, s);
    emit_load(T3, ATTR_BASE_OFFSET, ACC, s);
    emit_addiu(SP, SP, 4, s);

    emit_mul(T3, T2, T3, s);
    emit_store(T3, ATTR_BASE_OFFSET, ACC, s);
```

```
}

void divide_class::code(ostream &s) {
    e1->code(s);
    emit_push(ACC, s);
    e2->code(s);
    emit_jal("Object.copy", s);

    emit_load(T1, 1, SP, s);
    emit_load(T2, ATTR_BASE_OFFSET, T1, s);
    emit_load(T3, ATTR_BASE_OFFSET, ACC, s);
    emit_addiu(SP, SP, 4, s);

    emit_div(T3, T2, T3, s);
    emit_store(T3, ATTR_BASE_OFFSET, ACC, s);
}

void neg_class::code(ostream &s) {
    e1->code(s);
    emit_jal("Object.copy", s);
    emit_load(T1, ATTR_BASE_OFFSET, ACC, s);
    emit_neg(T1, T1, s);

    emit_store(T1, ATTR_BASE_OFFSET, ACC, s);
}

void lt_class::code(ostream &s) {
    emit_load_t1_t2(s, e1, e2);
    // get two val to t1, t2
    emit_load(T1, ATTR_BASE_OFFSET, T1, s);
    emit_load(T2, ATTR_BASE_OFFSET, T2, s);
    // create label
    int out_lebal = codegen_classtable->get_labelid_and_add();

    emit_load_bool(ACC, truebool, s);
    emit_blt(T1, T2, out_lebal, s);
    emit_load_bool(ACC, falsebool, s);
    emit_label_def(out_lebal, s);
}

// alike
void eq_class::code(ostream &s) {
    emit_load_t1_t2(s, e1, e2);
    Symbol e1type = e1->get_type();
    Symbol e2type = e2->get_type();
    if ((e1type == Int || e1type == Bool || e1type == Str)
        && (e2type == Int || e2type == Bool || e2type == Str)) {
        emit_load_bool(ACC, truebool, s);
        emit_load_bool(A1, falsebool, s);
        emit_jal("equality_test", s);
        return;
    }
    int lebalid = codegen_classtable->get_labelid_and_add();

    emit_load_bool(ACC, truebool, s);
    emit_beq(T1, T2, lebalid, s);
    emit_load_bool(ACC, falsebool, s);
    emit_label_def(lebalid, s);
}

void leq_class::code(ostream &s) {
    emit_load_t1_t2(s, e1, e2);
    emit_load(T1, ATTR_BASE_OFFSET, T1, s);
    emit_load(T2, ATTR_BASE_OFFSET, T2, s);

    int lebalid = codegen_classtable->get_labelid_and_add();

    emit_load_bool(ACC, truebool, s);
    emit_bleq(T1, T2, lebalid, s);
```

```
1531.     emit_load_bool(ACC, falsebool, s);
1532.     emit_label_def(lebalid, s);
1533. }
1534.
1535. void comp_class::code(ostream &s) {
1536.     e1->code(s);
1537.     emit_load(T1, ATTR_BASE_OFFSET, ACC, s);
1538.
1539.     int lebalid = codegen_classtable->get_labelid_and_add();
1540.
1541.     emit_load_bool(ACC, truebool, s);
1542.     emit_beq(T1, ZERO, lebalid, s);
1543.     emit_load_bool(ACC, falsebool, s);
1544.     emit_label_def(lebalid, s);
1545. }
1546.
1547. void int_const_class::code(ostream& s)
1548. {
1549.   //
1550.   // Need to be sure we have an IntEntry *, not an arbitrary Symbol
1551.   //
1552.   emit_load_int(ACC,inttable.lookup_string(token->get_string()),s);
1553. }
1554.
1555. void string_const_class::code(ostream& s)
1556. {
1557.   emit_load_string(ACC,stringtable.lookup_string(token->get_string()),s);
1558. }
1559.
1560. void bool_const_class::code(ostream& s)
1561. {
1562.   emit_load_bool(ACC, BoolConst(val), s);
1563. }
1564.
1565. void new__class::code(ostream &s) {
1566.     std::string object_name = type_name->get_string();
1567.     // new a SELF
1568.     if (type_name == SELF_TYPE) {
1569.         emit_load(T1, 0, ACC, s);   // load tag
1570.         emit_load_address(T2, CLASSOBJTAB, s); // load obj
1571.
1572.         emit_sll(T1, T1, 3, s);     // t1 * 8
1573.
1574.         emit_add(T2, T1, T2, s);
1575.         emit_load(ACC, 0, T2, s);
1576.         emit_jal("Object.copy", s);
1577.
1578.         emit_load(T1, 0, ACC, s);
1579.         emit_load_address(T2, CLASSOBJTAB, s);
1580.
1581.         emit_sll(T1, T1, 3, s);
1582.
1583.         emit_add(T2, T1, T2, s);
1584.         emit_load(A1, 1, T2, s);
1585.         emit_jalr(A1, s);
1586.
1587.     } else {
1588.         std::string protobj_object = object_name + PROTOBJ_SUFFIX;
1589.         emit_load_address(ACC, const_cast<char *>(protobj_object.c_str()), s);
1590.         emit_jal("Object.copy", s);
1591.         std::string init_object = object_name + CLASSINIT_SUFFIX;
1592.         emit_jal(const_cast<char *>(init_object.c_str()), s);
1593.     }
1594. }
1595.
1596. void isvoid_class::code(ostream &s) {
1597.     e1->code(s);
1598.     emit_move(T1, ACC, s);
1599.
```

```
1600.     int lebal_id = codegen_classtable->get_labelid_and_add();
1601.     emit_load_bool(ACC, truebool, s);
1602.     emit_beq(T1, ZERO, lebal_id, s);
1603.     emit_load_bool(ACC, falsebool, s);
1604.     emit_label_def(lebal_id, s);
1605. }
1606.
1607. // return 0 for no exp
1608. void no_expr_class::code(ostream &s) {
1609.     emit_move(ACC, ZERO, s);
1610. }
1611.
1612. void object_class::code(ostream &s) {
1613.     if (name == self) {
1614.         emit_move(ACC, SELF, s);
1615.         return;
1616.     }
1617.     CgenNodeP curr_cgen = codegen_classtable->get_curr_class();
1618.     int offset;
1619.     if (envTable->lookup(name, &offset)) {
1620.         emit_load(ACC, offset, FP, s);
1621.         emit_gc_update(FP, offset, s);
1622.         return;
1623.     }
1624.     if (codegen_classtable->get_attr_offset(curr_cgen->get_name(), name, &offset)) { // 属于 attr 类型的
1625.         emit_load(ACC, offset, SELF, s);
1626.         emit_gc_update(SELF, offset, s);
1627.     }
1628. }
1629.
```

3. cool-tree.h 代码

```
 1. #ifndef COOL_TREE_H
 2. #define COOL_TREE_H
 3. //////////////////////////////////////////////////////////
 4. //
 5. // file: cool-tree.h
 6. //
 7. // This file defines classes for each phylum and constructor
 8. //
 9. //////////////////////////////////////////////////////////
10.
11. #include "tree.h"
12. #include "cool-tree.handcode.h"
13.
14. // define the class for phylum
15. // define simple phylum - Program
16. typedef class Program_class *Program;
17.
18. class Program_class : public tree_node {
19. public:
20.    tree_node *copy()     { return copy_Program(); }
21.    virtual Program copy_Program() = 0;
22.
23. #ifdef Program_EXTRAS
24.    Program_EXTRAS
25. #endif
26. };
27.
28. // define simple phylum - Class_
29. typedef class Class__class *Class_;
30.
31. class Class__class : public tree_node {
32. public:
33.    tree_node *copy()     { return copy_Class_(); }
34.    virtual Class_ copy_Class_() = 0;
```

```
35.
36. #ifdef Class__EXTRAS
37.    Class__EXTRAS
38. #endif
39. };
40.
41. // define simple phylum - Feature
42. typedef class Feature_class *Feature;
43.
44. class Feature_class : public tree_node {
45. public:
46.    tree_node *copy()     { return copy_Feature(); }
47.    virtual Feature copy_Feature() = 0;
48.
49. #ifdef Feature_EXTRAS
50.    Feature_EXTRAS
51. #endif
52. };
53.
54. // define simple phylum - Formal
55. typedef class Formal_class *Formal;
56.
57. class Formal_class : public tree_node {
58. public:
59.    tree_node *copy()     { return copy_Formal(); }
60.    virtual Formal copy_Formal() = 0;
61.
62. #ifdef Formal_EXTRAS
63.    Formal_EXTRAS
64. #endif
65. };
66.
67. // define simple phylum - Expression
68. typedef class Expression_class *Expression;
69.
70. class Expression_class : public tree_node {
71. public:
72.    tree_node *copy()     { return copy_Expression(); }
73.    virtual Expression copy_Expression() = 0;
74.    virtual void code(ostream &s) = 0;
75.
76. #ifdef Expression_EXTRAS
77.    Expression_EXTRAS
78. #endif
79. };
80.
81. // define simple phylum - Case
82. typedef class Case_class *Case;
83.
84. class Case_class : public tree_node {
85. public:
86.    tree_node *copy()     { return copy_Case(); }
87.    virtual Case copy_Case() = 0;
88.
89. #ifdef Case_EXTRAS
90.    Case_EXTRAS
91. #endif
92. };
93.
94. // define the class for phylum - LIST
95. // define list phlyum - Classes
96. typedef list_node<Class_> Classes_class;
97. typedef Classes_class *Classes;
98.
99. // define list phlyum - Features
100. typedef list_node<Feature> Features_class;
101. typedef Features_class *Features;
102.
103. // define list phlyum - Formals
```

```
104. typedef list_node<Formal> Formals_class;
105. typedef Formals_class *Formals;
106. 
107. // define list phlyum - Expressions
108. typedef list_node<Expression> Expressions_class;
109. typedef Expressions_class *Expressions;
110. 
111. // define list phlyum - Cases
112. typedef list_node<Case> Cases_class;
113. typedef Cases_class *Cases;
114. 
115. // define the class for constructors
116. // define constructor - program
117. class program_class : public Program_class {
118. public:
119.    Classes classes;
120. public:
121.    program_class(Classes a1) {
122.       classes = a1;
123.    }
124.    Program copy_Program();
125.    void dump(ostream& stream, int n);
126. 
127. #ifdef Program_SHARED_EXTRAS
128.    Program_SHARED_EXTRAS
129. #endif
130. #ifdef program_EXTRAS
131.    program_EXTRAS
132. #endif
133. };
134. 
135. // define constructor - class_
136. class class__class : public Class__class {
137. public:
138.    Symbol name;
139.    Symbol parent;
140.    Features features;
141.    Symbol filename;
142. public:
143.    class__class(Symbol a1, Symbol a2, Features a3, Symbol a4) {
144.       name = a1;
145.       parent = a2;
146.       features = a3;
147.       filename = a4;
148.    }
149.    Class_ copy_Class_();
150.    void dump(ostream& stream, int n);
151. 
152. #ifdef Class__SHARED_EXTRAS
153.    Class__SHARED_EXTRAS
154. #endif
155. #ifdef class__EXTRAS
156.    class__EXTRAS
157. #endif
158. };
159. 
160. // define constructor - method
161. class method_class : public Feature_class {
162. public:
163.    Symbol name;
164.    Formals formals;
165.    Symbol return_type;
166.    Expression expr;
167. public:
168.    method_class(Symbol a1, Formals a2, Symbol a3, Expression a4) {
169.       name = a1;
170.       formals = a2;
171.       return_type = a3;
172.       expr = a4;
```

```
173.    }
174.    Feature copy_Feature();
175.    void dump(ostream& stream, int n);
176.
177. #ifdef Feature_SHARED_EXTRAS
178.    Feature_SHARED_EXTRAS
179. #endif
180. #ifdef method_EXTRAS
181.    method_EXTRAS
182. #endif
183. };
184.
185. // define constructor - attr
186. class attr_class : public Feature_class {
187. public:
188.    Symbol name;
189.    Symbol type_decl;
190.    Expression init;
191. public:
192.    attr_class(Symbol a1, Symbol a2, Expression a3) {
193.       name = a1;
194.       type_decl = a2;
195.       init = a3;
196.    }
197.    Feature copy_Feature();
198.    void dump(ostream& stream, int n);
199.
200. #ifdef Feature_SHARED_EXTRAS
201.    Feature_SHARED_EXTRAS
202. #endif
203. #ifdef attr_EXTRAS
204.    attr_EXTRAS
205. #endif
206. };
207.
208. // define constructor - formal
209. class formal_class : public Formal_class {
210. public:
211.    Symbol name;
212.    Symbol type_decl;
213. public:
214.    formal_class(Symbol a1, Symbol a2) {
215.       name = a1;
216.       type_decl = a2;
217.    }
218.    Formal copy_Formal();
219.    void dump(ostream& stream, int n);
220.
221. #ifdef Formal_SHARED_EXTRAS
222.    Formal_SHARED_EXTRAS
223. #endif
224. #ifdef formal_EXTRAS
225.    formal_EXTRAS
226. #endif
227. };
228.
229. // define constructor - branch
230. class branch_class : public Case_class {
231. public:
232.    Symbol name;
233.    Symbol type_decl;
234.    Expression expr;
235. public:
236.    branch_class(Symbol a1, Symbol a2, Expression a3) {
237.       name = a1;
238.       type_decl = a2;
239.       expr = a3;
240.    }
241.    Case copy_Case();
```

```
242.    void dump(ostream& stream, int n);
243.
244. #ifdef Case_SHARED_EXTRAS
245.    Case_SHARED_EXTRAS
246. #endif
247. #ifdef branch_EXTRAS
248.    branch_EXTRAS
249. #endif
250. };
251.
252. // define constructor - assign
253. class assign_class : public Expression_class {
254. public:
255.    Symbol name;
256.    Expression expr;
257. public:
258.    assign_class(Symbol a1, Expression a2) {
259.       name = a1;
260.       expr = a2;
261.    }
262.    Expression copy_Expression();
263.    void dump(ostream& stream, int n);
264.
265. #ifdef Expression_SHARED_EXTRAS
266.    Expression_SHARED_EXTRAS
267. #endif
268. #ifdef assign_EXTRAS
269.    assign_EXTRAS
270. #endif
271. };
272.
273. // define constructor - static_dispatch
274. class static_dispatch_class : public Expression_class {
275. public:
276.    Expression expr;
277.    Symbol type_name;
278.    Symbol name;
279.    Expressions actual;
280. public:
281.    static_dispatch_class(Expression a1, Symbol a2, Symbol a3, Expressions a4) {
282.       expr = a1;
283.       type_name = a2;
284.       name = a3;
285.       actual = a4;
286.    }
287.    Expression copy_Expression();
288.    void dump(ostream& stream, int n);
289.
290. #ifdef Expression_SHARED_EXTRAS
291.    Expression_SHARED_EXTRAS
292. #endif
293. #ifdef static_dispatch_EXTRAS
294.    static_dispatch_EXTRAS
295. #endif
296. };
297.
298. // define constructor - dispatch
299. class dispatch_class : public Expression_class {
300. public:
301.    Expression expr;
302.    Symbol name;
303.    Expressions actual;
304. public:
305.    dispatch_class(Expression a1, Symbol a2, Expressions a3) {
306.       expr = a1;
307.       name = a2;
308.       actual = a3;
309.    }
310.    Expression copy_Expression();
```

```
311.    void dump(ostream& stream, int n);
312.
313. #ifdef Expression_SHARED_EXTRAS
314.    Expression_SHARED_EXTRAS
315. #endif
316. #ifdef dispatch_EXTRAS
317.    dispatch_EXTRAS
318. #endif
319. };
320.
321. // define constructor - cond
322. class cond_class : public Expression_class {
323. public:
324.    Expression pred;
325.    Expression then_exp;
326.    Expression else_exp;
327. public:
328.    cond_class(Expression a1, Expression a2, Expression a3) {
329.       pred = a1;
330.       then_exp = a2;
331.       else_exp = a3;
332.    }
333.    Expression copy_Expression();
334.    void dump(ostream& stream, int n);
335.
336. #ifdef Expression_SHARED_EXTRAS
337.    Expression_SHARED_EXTRAS
338. #endif
339. #ifdef cond_EXTRAS
340.    cond_EXTRAS
341. #endif
342. };
343.
344. // define constructor - loop
345. class loop_class : public Expression_class {
346. public:
347.    Expression pred;
348.    Expression body;
349. public:
350.    loop_class(Expression a1, Expression a2) {
351.       pred = a1;
352.       body = a2;
353.    }
354.    Expression copy_Expression();
355.    void dump(ostream& stream, int n);
356.
357. #ifdef Expression_SHARED_EXTRAS
358.    Expression_SHARED_EXTRAS
359. #endif
360. #ifdef loop_EXTRAS
361.    loop_EXTRAS
362. #endif
363. };
364.
365. // define constructor - typcase
366. class typcase_class : public Expression_class {
367. public:
368.    Expression expr;
369.    Cases cases;
370. public:
371.    typcase_class(Expression a1, Cases a2) {
372.       expr = a1;
373.       cases = a2;
374.    }
375.    Expression copy_Expression();
376.    void dump(ostream& stream, int n);
377.
378. #ifdef Expression_SHARED_EXTRAS
379.    Expression_SHARED_EXTRAS
```

```
#endif
#ifdef typcase_EXTRAS
   typcase_EXTRAS
#endif
};

// define constructor - block
class block_class : public Expression_class {
public:
   Expressions body;
public:
   block_class(Expressions a1) {
      body = a1;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef block_EXTRAS
   block_EXTRAS
#endif
};

// define constructor - let
class let_class : public Expression_class {
public:
   Symbol identifier;
   Symbol type_decl;
   Expression init;
   Expression body;
public:
   let_class(Symbol a1, Symbol a2, Expression a3, Expression a4) {
      identifier = a1;
      type_decl = a2;
      init = a3;
      body = a4;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef let_EXTRAS
   let_EXTRAS
#endif
};

// define constructor - plus
class plus_class : public Expression_class {
public:
   Expression e1;
   Expression e2;
public:
   plus_class(Expression a1, Expression a2) {
      e1 = a1;
      e2 = a2;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef plus_EXTRAS
   plus_EXTRAS
#endif
```

```
449. };
450.
451. // define constructor - sub
452. class sub_class : public Expression_class {
453. public:
454.    Expression e1;
455.    Expression e2;
456. public:
457.    sub_class(Expression a1, Expression a2) {
458.       e1 = a1;
459.       e2 = a2;
460.    }
461.    Expression copy_Expression();
462.    void dump(ostream& stream, int n);
463.
464. #ifdef Expression_SHARED_EXTRAS
465.    Expression_SHARED_EXTRAS
466. #endif
467. #ifdef sub_EXTRAS
468.    sub_EXTRAS
469. #endif
470. };
471.
472. // define constructor - mul
473. class mul_class : public Expression_class {
474. public:
475.    Expression e1;
476.    Expression e2;
477. public:
478.    mul_class(Expression a1, Expression a2) {
479.       e1 = a1;
480.       e2 = a2;
481.    }
482.    Expression copy_Expression();
483.    void dump(ostream& stream, int n);
484.
485. #ifdef Expression_SHARED_EXTRAS
486.    Expression_SHARED_EXTRAS
487. #endif
488. #ifdef mul_EXTRAS
489.    mul_EXTRAS
490. #endif
491. };
492.
493. // define constructor - divide
494. class divide_class : public Expression_class {
495. public:
496.    Expression e1;
497.    Expression e2;
498. public:
499.    divide_class(Expression a1, Expression a2) {
500.       e1 = a1;
501.       e2 = a2;
502.    }
503.    Expression copy_Expression();
504.    void dump(ostream& stream, int n);
505.
506. #ifdef Expression_SHARED_EXTRAS
507.    Expression_SHARED_EXTRAS
508. #endif
509. #ifdef divide_EXTRAS
510.    divide_EXTRAS
511. #endif
512. };
513.
514. // define constructor - neg
515. class neg_class : public Expression_class {
516. public:
517.    Expression e1;
```

```
public:
   neg_class(Expression a1) {
      e1 = a1;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef neg_EXTRAS
   neg_EXTRAS
#endif
};

// define constructor - lt
class lt_class : public Expression_class {
public:
   Expression e1;
   Expression e2;
public:
   lt_class(Expression a1, Expression a2) {
      e1 = a1;
      e2 = a2;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef lt_EXTRAS
   lt_EXTRAS
#endif
};

// define constructor - eq
class eq_class : public Expression_class {
public:
   Expression e1;
   Expression e2;
public:
   eq_class(Expression a1, Expression a2) {
      e1 = a1;
      e2 = a2;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef eq_EXTRAS
   eq_EXTRAS
#endif
};

// define constructor - leq
class leq_class : public Expression_class {
public:
   Expression e1;
   Expression e2;
public:
   leq_class(Expression a1, Expression a2) {
      e1 = a1;
      e2 = a2;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);
```

```
587.
588. #ifdef Expression_SHARED_EXTRAS
589.    Expression_SHARED_EXTRAS
590. #endif
591. #ifdef leq_EXTRAS
592.    leq_EXTRAS
593. #endif
594. };
595.
596. // define constructor - comp
597. class comp_class : public Expression_class {
598. public:
599.    Expression e1;
600. public:
601.    comp_class(Expression a1) {
602.       e1 = a1;
603.    }
604.    Expression copy_Expression();
605.    void dump(ostream& stream, int n);
606.
607. #ifdef Expression_SHARED_EXTRAS
608.    Expression_SHARED_EXTRAS
609. #endif
610. #ifdef comp_EXTRAS
611.    comp_EXTRAS
612. #endif
613. };
614.
615. // define constructor - int_const
616. class int_const_class : public Expression_class {
617. public:
618.    Symbol token;
619. public:
620.    int_const_class(Symbol a1) {
621.       token = a1;
622.    }
623.    Expression copy_Expression();
624.    void dump(ostream& stream, int n);
625.
626. #ifdef Expression_SHARED_EXTRAS
627.    Expression_SHARED_EXTRAS
628. #endif
629. #ifdef int_const_EXTRAS
630.    int_const_EXTRAS
631. #endif
632. };
633.
634. // define constructor - bool_const
635. class bool_const_class : public Expression_class {
636. public:
637.    Boolean val;
638. public:
639.    bool_const_class(Boolean a1) {
640.       val = a1;
641.    }
642.    Expression copy_Expression();
643.    void dump(ostream& stream, int n);
644.
645. #ifdef Expression_SHARED_EXTRAS
646.    Expression_SHARED_EXTRAS
647. #endif
648. #ifdef bool_const_EXTRAS
649.    bool_const_EXTRAS
650. #endif
651. };
652.
653. // define constructor - string_const
654. class string_const_class : public Expression_class {
655. public:
```

```
656.    Symbol token;
657. public:
658.    string_const_class(Symbol a1) {
659.       token = a1;
660.    }
661.    Expression copy_Expression();
662.    void dump(ostream& stream, int n);
663.
664. #ifdef Expression_SHARED_EXTRAS
665.    Expression_SHARED_EXTRAS
666. #endif
667. #ifdef string_const_EXTRAS
668.    string_const_EXTRAS
669. #endif
670. };
671.
672. // define constructor - new_
673. class new__class : public Expression_class {
674. public:
675.    Symbol type_name;
676. public:
677.    new__class(Symbol a1) {
678.       type_name = a1;
679.    }
680.    Expression copy_Expression();
681.    void dump(ostream& stream, int n);
682.
683. #ifdef Expression_SHARED_EXTRAS
684.    Expression_SHARED_EXTRAS
685. #endif
686. #ifdef new__EXTRAS
687.    new__EXTRAS
688. #endif
689. };
690.
691. // define constructor - isvoid
692. class isvoid_class : public Expression_class {
693. public:
694.    Expression e1;
695. public:
696.    isvoid_class(Expression a1) {
697.       e1 = a1;
698.    }
699.    Expression copy_Expression();
700.    void dump(ostream& stream, int n);
701.
702. #ifdef Expression_SHARED_EXTRAS
703.    Expression_SHARED_EXTRAS
704. #endif
705. #ifdef isvoid_EXTRAS
706.    isvoid_EXTRAS
707. #endif
708. };
709.
710. // define constructor - no_expr
711. class no_expr_class : public Expression_class {
712. public:
713. public:
714.    no_expr_class() {
715.    }
716.    Expression copy_Expression();
717.    void dump(ostream& stream, int n);
718.
719. #ifdef Expression_SHARED_EXTRAS
720.    Expression_SHARED_EXTRAS
721. #endif
722. #ifdef no_expr_EXTRAS
723.    no_expr_EXTRAS
724. #endif
```

```
};

// define constructor - object
class object_class : public Expression_class {
public:
   Symbol name;
public:
   object_class(Symbol a1) {
      name = a1;
   }
   Expression copy_Expression();
   void dump(ostream& stream, int n);

#ifdef Expression_SHARED_EXTRAS
   Expression_SHARED_EXTRAS
#endif
#ifdef object_EXTRAS
   object_EXTRAS
#endif
};

// define the prototypes of the interface
Classes nil_Classes();
Classes single_Classes(Class_);
Classes append_Classes(Classes, Classes);
Features nil_Features();
Features single_Features(Feature);
Features append_Features(Features, Features);
Formals nil_Formals();
Formals single_Formals(Formal);
Formals append_Formals(Formals, Formals);
Expressions nil_Expressions();
Expressions single_Expressions(Expression);
Expressions append_Expressions(Expressions, Expressions);
Cases nil_Cases();
Cases single_Cases(Case);
Cases append_Cases(Cases, Cases);
Program program(Classes);
Class_ class_(Symbol, Symbol, Features, Symbol);
Feature method(Symbol, Formals, Symbol, Expression);
Feature attr(Symbol, Symbol, Expression);
Formal formal(Symbol, Symbol);
Case branch(Symbol, Symbol, Expression);
Expression assign(Symbol, Expression);
Expression static_dispatch(Expression, Symbol, Symbol, Expressions);
Expression dispatch(Expression, Symbol, Expressions);
Expression cond(Expression, Expression, Expression);
Expression loop(Expression, Expression);
Expression typcase(Expression, Cases);
Expression block(Expressions);
Expression let(Symbol, Symbol, Expression, Expression);
Expression plus(Expression, Expression);
Expression sub(Expression, Expression);
Expression mul(Expression, Expression);
Expression divide(Expression, Expression);
Expression neg(Expression);
Expression lt(Expression, Expression);
Expression eq(Expression, Expression);
Expression leq(Expression, Expression);
Expression comp(Expression);
Expression int_const(Symbol);
Expression bool_const(Boolean);
Expression string_const(Symbol);
Expression new_(Symbol);
Expression isvoid(Expression);
Expression no_expr();
Expression object(Symbol);
#endif
```