The Iris Final Report

Valerie Zhang (valerie.zhang@tufts.edu)
Josh Kim (joshua.kim634361@tufts.edu)
Ayda Aricanli (ayda.aricanli@tufts.edu)
Timothy Valk (timothy.valk@tufts.edu)
Trevor Sullivan (trevor.sullivan@tufts.edu)

December 2023

Contents

1 Introduction

Iris is a general-purpose, object-oriented programming language that combines features from Java, C++, and Smalltalk. An Iris program is a series of classes – nearly everything is a class! As such, Iris supports inheritance and encapsulation.

Iris puts a spin on normal encapsulation conventions by introducing `permit` class members. These members are accessible by a class's sub-classes as well as the classes whose names are in its private collection `permitted`. However, unlike friend classes in C++, a class's permitted classes may only access the `permit` methods of the class, rather than all private methods.

2 Language Tutorial

2.1 Notation

Code is distinguished from prose by using this `font`. Grammar rules are distinguished using this *font*.

2.2 Welcome to Iris

Hey there! Welcome to Iris. Let's learn how to write, compile, and run our very first Iris program. Note that Iris programs are written in `.iris` files. Let's dive in.

2.3 Hello World!

As introduced above, an Iris program consists of a series of class definitions. The starting point for program execution, familiarly, is still a "main" function, but this "main" function must be defined in a definition of a "Main" class. As such, to write a simple "Hello World" Iris program, let's start with a "Main" class definition:

```
class Main () {
   ..  class definition in here
}
```

Within the "Main" class, a "main" function must be defined under the "public" label, with the "univ" keyword and a return type of int (more on both "public" and "univ" later):

```
class Main () {
   public:
   int univ main() {
      ..  function definition in here
   }
}
```

The Olympus class is a built-in class that contains standard input and standard output functions. Thus, "Hello world!" can printed by calling the `println` function of the Olympus class:

```
class Main () {
   public:
   int univ main() {
      Olympus.println("Hello world!");
   }
}
```

2.4 Compilation and Execution

You have now written a compilable "Hello world!" Iris program! To compile the program – let's call it "hello-world.iris" – simply type: `./irisc hello-world.iris`

To run the compiled program, simply type: `./hello-world.exe`

2.5 Expanding Our Program – Instantiation

With "Hello world!" under our belt, we can continue journeying through Iris with instantiation. Consider the following code:

```
class Goddess () {
   public:
      string name;
      void univ smite() {
         Olympus.print("Bam");
      }
   private:
      int followers;
}

class Main () {
   public:
      int univ main() {
         Olympus.print("Hello world!");
      }
}
```

We now have two classes – our "Main" definition from before, and "Goddess" which has "name" as a public member, "followers" as a private member, and "smite" as a public method. To call "smite", we must instantiate the Goddess class in the "main" function:

```
class Goddess () {
   public:
      string name;
      void univ smite() {
         Olympus.print("Bam!");
      }
   private:
      int followers;
}

class Main () {
   public:
      int univ main() {
         Olympus.print("Hello world!");
         Goddess my_goddess = new Goddess();
         my_animal.noise();
      }
}
```

We should now print "Hello World!" followed by "Bam!"

2.6 Expanding Our Program – Inheritance

A key feature of Iris is the ability for classes to inherit from other classes. A child class inherits all the members of its parent class. Let's define a Dog class and call "noise" from it in "main" in this manner:

```
class Goddess () {
   public:
      string name;
      void univ smite() {
         Olympus.print("Bam!");
      }
   private:
      int followers;
}

class Artemis of Goddess () {
   public:
      void univ motto() {
         Olympus.print("I am the goddess of wisdom.");
      }
   private:
      int arrows;
}

class Main () {
   public:
      int univ main() {
         Olympus.print("Hello world!");
         Artemis my_artemis = new Artemis();
         my_artemis.smite();
      }
}
```

Because Artemis inherits "smite" from Goddess, this should print "Hello World!" followed by "Bam!" once again!

2.7 Expanding Our Program – Permit

At this point, we've built a program that exploits the features of classes. Let's add an Iris-specific feature to top it off: `permit`. A special form of encapsulation, members under the `permit` label are only accessible to another class if they're in the permitted list of that class. Consider a new method, under `permit`, in the Goddess class:

```
class Goddess () {
   public:
      string name;
      void univ smite() {
         Olympus.print("Bam!");
      }
   permit:
      void univ cast_spell() {
         Olympus.print("A spell has been cast on the kingdom!");
      }
```

```
   private:
      int followers;
}
```

A class's permitted list is defined inside the parentheses at the top of the class. Currently, Goddess has no class names in its permitted list, so no class other than Goddess can access the "make_spell" function. For instance, the following code will result in a runtime error:

```
class Goddess () {
   public:
      string name;
      void univ smite() {
         Olympus.print("Bam!");
      }
   permit:
      void univ make_spell() {
         Olympus.print("A spell has been cast on the kingdom!");
      }
   private:
      int followers;
}

class Main () {
   public:
      int univ main() {
         Goddess my_goddess = new Goddess();
         my_goddess.make_spell();
      }
}
```

Main is not in the permitted list of Goddess, and thus, the above code fails. Let's introduce a "Kingdom" class and include it in Goddess's permitted list:

```
class Goddess (Kingdom) {
   public:
      string name;
      void univ smite() {
         Olympus.print("Bam!");
      }
   permit:
      void univ make_spell() {
         Olympus.print("A spell has been cast on the kingdom!");
      }
   private:
      int followers;
}

class Kingdom () {
   public:
      void univ cast_spell() {
         Goddess a_goddess = new Goddess();
         a_goddess.make_spell();
      }
}
```

```
class Main () {
   public:
      int univ main() {
         Kingdom my_kingdom = new Kingdom();
         my_kingdom.cast_spell();
      }
}
```

And voila! A spell has been cast on the kingdom! We are able to call "make_spell" in Kingdom because Kingdom is in the permitted list of Goddess. You now have the building blocks needed to write, compile, and execute interesting and complex Iris programs. Reference the following Language Manual for complete descriptions of syntax and semantics, and more code examples. Go forth!

3 Language Manual: Lexical Conventions

3.1 Comments

The characters `..` introduce a single-line comment.
The characters `.~*` introduce a multi-line comment, which terminates with `*~.`

```
..This is a single line comment
.~* This is a multi-
line comment *~.
```

3.2 Identifier Conventions

Any valid identifier in Iris is an uppercase or lowercase alphabetic character followed by any number of uppercase or lowercase alphabetic characters and/or digits. Uppercase and lowercase letters are distinct. The underscore `_` is also an alphabetic character, and an identifier starting with an underscore is valid.

3.3 Keywords

`if`	`of`	`bool`
`else`	`public`	`string`
`while`	`permit`	`float`
`return`	`private`	`true`
`class`	`void`	`false`
`self`	`univ`	`Olympus`
`new`	`int`	`Object`

3.4 Literals

Iris allows for literals for each primitive data type:

3.4.1 Integer literals

Integer → `0` | `-?[1 - 9]{1}[0 - 9]*`

An integer literal is a sequence of one or more digits. Integer literals can begin with exactly one "`-`", and if so, are assumed to be negative. The only integer literal that can start with 0 is 0. All integer literals are assumed to be in base 10.

3.4.2 Float literals

Float → `[` *Integer* `][.]{1}[0 - 9]+`

A floating literal consists of 3 parts: a sequence of one or more digits, the decimal point, and another sequence of one or more digits. All 3 of these parts MUST be present for a float literal. Floating literals can optionally begin with exactly one "`-`", and if so, are assumed to be negative. All float literals are assumed to be in base 10.

3.4.3 Boolean literals

Boolean → `true` | `false`

3.4.4 String literals

string → `["][` *ascii-literal* `]*["]`

A string literal consists of a series of character literals surrounded by double quotes. Within the string literal, a `"` character must be preceded by a single \.

4 Language Manual: Types

typ → *int* | *bool* | *float* | *string* | *void* | `Object of` *string*

Functions or methods (used interchangeably) can be declared to return and/or take in any of the above types as parameters. Variables can also be set as any of the types. When declaring a variable of some *Class* type (represented by `Object of` *string*) the name of the class encoded in the *string* is used rather than the word `Object`.

An `int` is represented by a 32-bit OCaml integer, where the leading 31 bits encode the integer itself, and the LSB is an integer indicator (if it's on, then the data represents an integer). A *float* is an OCaml float, and a *string* is represented as an OCaml string.

4.1 Primitive Types

Primitive types refer to every type except the `Object of` *string* type. Primitive types are distinguished by the fact that they are not classes. As such, there is no way of directly calling methods on these types with the `.` (dot) operator. Furthermore, creating new instances of these types is simpler than creating new instances of Object types.

```
..  The following line instantiates an integer
int number;
```

4.2 Object

The `Object of` *string* type refers to the pre-defined classes Object and Olympus, as well as any other user-defined classes. This type allows users to create and interact with instances of their own

user-defined classes and write methods returning user-defined classes. Unlike the primitive types, instances of Object variables must be initialized by using the class' corresponding constructor using the `new` keyword. See the Class Instantiation section for more information on `new`.

5 Language Manual: Expressions

expr	→	*literal*	
	|	*var*	5.1
	|	*uop expr*	5.2
	|	*expr postfix*	5.3
	|	*expr binop expr*	5.4
	|	*var assign expr*	5.5
	|	*typ id assign expr*	5.5
	|	*var opassign expr*	5.5
	|	*id.id(expr list)*	5.6
	|	`new` *id* `()`	5.7

5.1 Variable Access

var → *id* | *id.id*

There are 3 types of variables in Iris: formals (function parameters), locals, and class variables. Formals and locals are always accessed using their `id`. Class variables, on the other hand, are accessed differently depending on scope.

A class variable can be referred to with just its `id` if it is within the class it is defined in. Outside of that class, it is called with `id1.id2`, where `id1` is the identifier for an object instance of its class, and `id2` is the identifier for the class variable. If a child class accesses a variable it is overriding from its parent, it will access the most local definition of the variable defined at `id`.

5.2 Unary Operators

uop → `-` | `!`

5.2.1 `-`*expr*

In addition to being the binary operator for subtraction, `-` encodes the negation operator, a right-associative unary operator that negates its operand. It can only be used on an *expr* that evaluates to an `int` or a `float`.

5.2.2 `!`*expr*

The logical-not operator is a right-associative unary operator that is used to compare Booleans. Returns the Boolean literal `true` if its operand is `false`, and `false` if its operand is `true`. It can only be used on an *expr* that evaluates to type `bool`, or `int`. For `int`s, if the value evaluates to `0`, then it returns the Boolean literal `true`. Otherwise, it returns `false`.

5.3 Postfix Unary Operators

postfix → `++` | `--`

5.3.1 *id*`++`

The plus-plus operator is a left-associative unary operator and is used to increment integer variables by 1. Returns its operand, which must be a variable or class variable, plus one.

5.3.2 *id--*

The minus-minus operator is a left-associative unary operator and is used to decrement integer variables by 1. Returns its operand, which must be a variable or class variable, minus one.

5.4 Binary Operators

binop → `!=`
| `==`
| `+`
| `+=`
| `-`
| `-=`
| `*`
| `*=`
| `/`
| `/=`
| `>`
| `>=`
| `<`
| `<=`
| `||`
| `&&`

5.4.1 *expr* `!=` *expr*

The not-equals operator is a left-associative binary operator and is used to compare the equality of expressions that evaluate to the same primitive types. Returns Boolean `true` if its two operands are different. Does not work on strings.

5.4.2 *expr* `==` *expr*

The equal-equals operator is a left-associative binary operator and is used to compare the equality of expressions that evaluate to the same primitive types. Returns Boolean `true` if its two operands are the same. When used on string literals, this operator is syntactic sugar for a call to `Olympus.streq(...)`, which checks the structural equality of strings.

5.4.3 *expr* `+` *expr*

The plus operator is a left-associative binary operator and is used to add expressions that evaluate to a primitive type (`int` or `float`). Returns the sum of its two operands if they are integers or floats, and will concatenate its two operands if they are strings. If a float and an integer are added, the result will be a float.

5.4.4 *expr* `-` *expr*

The minus operator is a left-associative binary operator and is used to subtract expressions that evaluate to `float`s or `int`s. Returns the difference between its first operand and second operand. If one expression is a `float` and the other an `int`, then it will evaluate to a `float`.

5.4.5 *expr* `*` *expr*

The times operator is a left-associative binary operator and is used to multiply expressions that evaluate to `float`s or `int`s. Returns the product of its two operands. If one expression is a `float` and the other an `int`, then it will evaluate to a `float`.

5.4.6 *expr* / *expr*

The division operator is a left-associative binary operator and is used to divide expressions that evaluate to `float`s or `int`s. Returns the result of dividing its first operand by its second. If one expression is a `float` and the other an `int`, then it will evaluate to a `float`.

5.4.7 *expr* > *expr*

The greater-than operator is a left-associative binary operator that is used to compare expressions that evaluate to `int`s or `float`s. Returns Boolean `true` if its first operand is greater than its second operand and `false` otherwise. Each operand must evaluate to the same type.

5.4.8 *expr* >= *expr*

The greater-than-equals operator is a left-associative binary operator that is used to compare expressions that evaluate to `int`s or `float`s. Returns Boolean `true` if its first operand is greater than or equal to its second operand and `false` otherwise. Each operand must evaluate to the same type.

5.4.9 *expr* < *expr*

The less-than operator is a left-associative binary operator that is used to compare expressions that evaluate to `int`s or `float`s. Returns Boolean `true` if its first operand is less than its second operand and `false` otherwise. Each operand must evaluate to the same type.

5.4.10 *expr* <= *expr*

The less-than-equals operator is a left-associative binary operator that is used to expressions that evaluate to `int`s or `float`s. Returns Boolean `true` if its first operand is less than or equal to its second operand and `false` otherwise. Each operand must evaluate to the same type.

5.4.11 *expr* || *expr*

The or operator is a left-associative binary operator that is used on expressions that evaluate to `bool`s. Returns Boolean literal `true` if either of its operands evaluate to `true` and `false` otherwise.

5.4.12 *expr* && *expr*

The and operator is a left-associative binary operator that is used on expressions that evaluate to `bool`s. Returns Boolean literal `true` if both of its operands evaluate to `true` and `false` otherwise.

5.5 Assignment

These are the *binop* operations that also perform assignment. For all of the asignment operators, the right-hand side of the operator must evaluate to the same type as the left-hand side. The *opassign* assignment type can only be used on previously declared variables. All of these expressions return the expression on the right-hand side.

5.5.1 *var* = *expr*

The assignment operator is a left-associative binary operator and is used to assign variable values. A valid assignment expression consists of a right-hand side expression that evalutes to the same type as the variable.

5.5.2 *typ id = expr*

The assignment operator can also be used in conjuction with local variable declarations. A valid expression of this form consists of a right-hand side expression that evalutes to type typ.

opassign → `+=`
| `-=`
| `*=`
| `/=`

5.5.3 *var += expr*

The plus-equals operator is a left-associative binary operator and is used to add and assign expressions that evaluate to primitive types to variables. Returns the sum of its two operands if they are numeric and assigns the value to its left operand, which must be a previously declared variable. If its operands are of string type, performs concatenation similar to `+` and assigns.

5.5.4 *var -= expr*

The minus-equals operator is a left-associative binary operator and is used to subtract and assign expressions that evaluate to primitive types to variables. Returns the difference between its two operands and assigns the result to its left operand, which must be a previously declared variable.

5.5.5 *var *= expr*

The times-equals operator is a left-associative binary operator and is used to multiply and assign expressions that evaluate to primitive types to variables. Returns the product of its two operands and assigns the result to its left operand, which must be a previously declared variable.

5.5.6 *var /= expr*

The divide-equals operator is a left-associative binary operator and is used to divide and assign expressions that evaluate to primitive types to variables. Returns the quotient of its two operands same as divide, and assigns the value to its left operand, which must be a previously declared variable.

5.6 Function Calls

Function calls are made using the `id.id(expr list)` rule, where `expr list` is a series of comma-delimited expressions. There are two types of functions or methods that exist in Iris: class methods and instance methods.

For class methods, the first `id` would be the class name and the second *id* would be the class method name. This is followed by an `expr list` which may be empty or nonempty.

The same goes for instance methods where the first `id` would be the instance name and the second `id` would be the instance method name. This is followed by an `expr list` which may be empty or nonempty.

5.7 Object Instantiation

Object instantiation is done through class constructors. A class constructor is a method defined by the class name followed by parentheses `()`. For example, the `Dog` constructor would be `Dog()`.

To instantiate an object a class constructor much follow the `new` keyword: `new Dog();`. This will allocate a new instance of the particular class on the heap and return the address to it. This address

can be assigned to a declared object instance as seen below:

```
Dog d = new Dog();

Dog d2;
d2 = new Dog();
```

See section 8, Classes, (specifically 8.4).

6 Language Manual: Statements

stmt → {*stmt list*}
| *expr*;
| `return` *expr*;
| `if (` *expr* `) {` *stmt* `}`
| `if (` *expr* `) {` *stmt* `}` *else* `{` *stmt* `}`
| `while (` *expr* `) {` *stmt* `}`
| *typ id*;

6.1 Block

The block statement is a list of statements.

6.2 *expr*

The *expr* statement is one of the expressions defined in 4.

6.3 `return`

The *return* statement contains the keyword `return` followed by an expression. It is used to pass a value out of a function.

6.4 `if`

The `if` statement contains the keyword `if` followed by an expression evaluating to a boolean, a statement representing the true clause, and a statement representing the false clause.

6.5 `else`

The `else` statement can only be used after an `if`; if the boolean condition of the preceding `if` evaluates to `false`, then the statement following the `else` is evaluated.

6.6 `while`

The while statement contains the keyword `while` followed by an expression and a statement. It is used to execute the statement a certain number of times by iterating according to the expression.

6.7 Local Declarations

The local declaration statement is a *typ* followed by an *id*. It is used to create a local variable by binding a type to a name.

7 Language Manual: Declaration

declaration → *variable decl*
| *function decl*
| *class decl*

Declarations introduce entities into the program that include an identifier which can be used in the program to refer to it. There are three types of declarations: variable declarations, function declarations, and class declarations. Variables and functions can only be declared inside a class declaration. Each declaration is packaged with specifiers defined below.

7.1 Identifiers (Names)

The rules for identifiers are as follows:

identifier → `[A-Z | a-z]+[A-Z | a-z | 0-9 | _ ]*`

Each declaration type must have an identifier associated with it.

7.2 Specifiers

There are three types of specifiers in Iris and are defined below:

specifiers → *typ*
| *class-specifiers*
| *univ-specifier-opt*

7.2.1 Type specifiers

Type specifiers are required for variable/instance declarations, and function declarations. These specifiers are any type defined in Iris: a primitive data type or a defined-class in Iris which is referred to using an identifier.

7.2.2 Class specifiers

There are two class specifiers:

class-specifiers → `class`
| *of-identifier-opt*

of-identifier-opt → *nothing*
| `of` *identifier*

The keyword `class` is used to declare a class and is followed by an identifier.

The keyword `of` is used for inheritance to specify the parent class of a new class. This specifier is optional. If not included, the newly defined class will automatically inherit from the Object class. Below are examples of class declarations:

```
class Dog of Animal () { ... }   ..the class Dog inherits from class Animal
class Dog () { ... }             ..the class Dog inherits from class Object
```

7.2.3 The `univ` specifier

univ-specifier-opt → *nothing*
| `univ`

The keyword `univ` is an optional specifier used to declare a method as a class method, and is used within class declarations when declaring a classes' methods. Methods without the univ keyword are instance methods used on instances of a class. The univ keyword cannot be used on a classes member variables. Examples of this are below.

```
..this is a declaration with univ
int univ sum(int a, int b) {
   return a + b;
}

..this is a declaration without univ
int mult(int a, int b) {
   return a * b;
}
```

7.3 Variable Declarations

A variable declaration is defined as follows:

variable-decl → *typ identifier*
| *typ identifier* `=` *expr*

The first rule for variable declarations for class members which is similar to local variable declaration. See section 6, Statements, (specifically 6.8) for more information on local variable declarations.

The second rule allows for simultaneous variable declaration and assignment. See 4 for more on *expr*.

7.4 Method Declarations

A function or method declaration is defined as follows:

function decl → *typ univ-specifier-opt identifier* `(` *variable-decl-list* `){` *stmt* `}`

Like a variable declaration, a function declaration must start with a type specifier followed by an identifier. A function declaration also includes an optional list of arguments which is defined below:

variable-decl-opt → *nothing*
| *variable-decl-list*

A *variable-decl-list* is a list of variable declarations separated by a comma '`,`'. A function declaration will also contain a body of statements which is described in the following section. Below are examples of function declarations:

```
..this is a univ function that takes in 2 int arguments
int univ sum(int a, int b) {
     return a + b
}
```

```
..this is a function that returns the string "a"
string a() {
      return "a";
}
```

7.5 Class Declarations

A class declaration is defined as follows:

class-decl → `class` *identifier of-opt identifier* `(` *permit-classes-opt* `)` `{` *encap-opt-list* `}`

The *permit-classes-opt* is a list of classes that have access to a class' `permit` members and methods, which can be empty or nonempty. See 7.2.4 for more on `permit`. Members can be included in the `permitted` category even when the *permit-classes-opt* is empty. The parentheses `()` must be included in the declaration, even if there are no permitted classes. The *encap-opt-list* is a list of encapsulation states a class may contain. See 7.2 for more on encapsulation. The states may be listed in any order, and may or may not be included. As such, an empty class may be declared with no encapsulation states included. However, member variable and method declarations must be defined under an encapsulation state, else they are assumed to be `public`. The curly braces {} must be included in the declaration. Below are examples of class declarations:

```
..this is a declaration of a class Dog
class Dog of Animal () {
   public:
      int name;
      int age;
      string univ noise() {
         return "bark";
      }
}

..this is a declaration of a class Goddess
class Goddess () {
   public:
      void univ smite {
         ..  print using built-in Olympus class, see "Built-in Classes" for more
         Olympus.print("Bam");
      }
      string univ toString() {
         return "I am a goddess.";
      }
   permit:
      void grantWish(string wish) { }
   private:
      int followers;
}
```

8 Language Manual: Classes

Classes are declared using the `class` keyword followed by the name of the class. If the class inherits from a class other than `Object`, the name is followed by the keyword `of`, followed by another class name – the parent. The body of a class, containing member variables and functions, is surrounded by { and }. The order in which classes are defined matters in two cases:

1. Inheritance: Parent classes must be defined above child classes.
2. Member variables: A class can only have object member variables of classes that have been defined before it.

Outside of these cases, all objects may be instantiated freely within functions, regardless of order.

8.1 The Object Class

The Object Class is the root of all classes in that it is a superclass of every existing and user-defined class. If a class does not inherit from an already existing class, it inherently will inherit from Object.

8.2 Encapsulation

8.2.1 Overview

In Iris, classes have three encapsulation states represented by the keywords `public`, `permit`, and `private`. At least one of these keywords must be provided if a class has a nonempty body. If the class is empty, keywords can be omitted without compiler errors.

Within the body of a class, `public`, `permit`, and `private` are followed by a `:` character. The members and/or methods that follow the keyword – until either another keyword occurs or the end of the class is reached – have its encapsulation level.

8.2.2 `public`

Member variables and functions under the `public` keyword are accessible by any class.

8.2.3 `private`

Member variables and functions under the `private` keyword are accessible internally within that class. They cannot be called by a client of that class.

8.2.4 `permit`

Member variables and functions under the `permit` keyword are conditionally accessible by other classes based on the private collection of permitted class names, surrounded by parentheses, that follow the class name. If a class is included in the private collection, it gains access to the `permit` category as if it were public. Otherwise, `permit` members remain private.

The privately declared collection of zero or more class names is included after the class name declaration using the following syntax: `(ClassA, ClassB, ...)`. The set of parentheses can be empty but cannot be omitted. Two examples of this are:

```
 .~* Simple example given some class Dog *~.
class Dog () {
   ..  class definition in here
}

 .~* Cat inherits from Animal, and Dog has permission to Cat's permit members *~.
class Cat of Animal (Dog) {
   ..  class definition in here
}
```

8.3 Inheritance

Similarly to Java, child classes inherit member variables and functions from a parent class. Child classes can also override their parent's member variables and functions. A child class automatically gets access to the `permit` member variables of its parent, but parents do not automatically get access to that of their children (they must be explicitly added to the child's private collection of permitted classes for this). Children do not inherit their ancestors' permitted list, but they do have access to their ancestors' private and permitted members. See 8.2.4 for an example of a class declaration involving inheritance.

8.4 Instantiation

Classes are passed around as pointers under the hood in Iris, and must be instantiated with the `new` keyword as in the following example.

```
..this will allocate space on the heap for c
ClassName c = new ClassName();
```

Classes not instantiated with the `new` keyword are uninitialized and will result in undefined behavior if an access is attempted.

9 Language Manual: Built-In Classes

9.1 The Main Class

The `Main` class contains exactly one class method: `univ main()`, which runs any Iris program. A user must define this class for any driver program and include the code to run within it.

```
 .~* Example of Main given some defined class Dog *~.
class Main () {
   int univ main() {
      Dog newDog = new Dog();
      string out = newDog.toString();
      Olympus.print(out);
   }
}
```

9.2 The Olympus Class

The `Olympus` class is similar to the `System` class in Java. It contains helpful functions and fields for reading in standard input and printing to standard output or error. It cannot be instantiated again by a user, but user-defined classes may inherit from `Olympus` if they wish.

The Olympus class includes the following methods:

Return	Function	Description
`void`	`univ print(string out);`	Given a *string*, prints it to standard output.
`void`	`univ println(string out);`	Given a *string*, prints it to standard output with a newline terminating the string.
`void`	`univ printerr(string out);`	Given a *string*, prints it to standard error.
`string`	`univ readaline();`	Reads a single line from standard input and returns it as a `string`. A line is defined as a sequence of characters, terminated by a newline or EOF.
`bool`	`univ streq(string s1, string s2);`	Takes in two strings and checks their structural equality. The "==", when used on strings, is syntactic sugar for a call to this function.

Standard input, standard output, and standard error are accessed only through these functions. There is no notion of a "stream" in Iris that the user can declare and pass to other functions.

10 Language Manual: Scope

Similar to C, functions and variables are in scope after their declaration until the end of the block of code they appear in (blocks are denoted with { and }). The rest of this section describes the expected behavior for the scope of variables and functions in a variety of cases.

Formals and local variables declared in functions go out of scope when a function returns. Declaring two variables or two functions of the same name within the same scope is not allowed (e.g. two instances of the Dog class named `d` in `main()`, two member variables in the same class named `age`, or two functions declared in the `Dog` class as `void bark()`). Declaring two classes of the same name is also not allowed.

Declaring a local variable of the same name as a member variable is allowed, but the local variable overshadows the member variable for the scope of that function. A child member variable declaration overshadows a parent member variable of the same name.

Declaring a member function that is already included in a parent class definition is also allowed, and the child definition will override the parent definition.

A member variable and member function of the same name in the same scope is allowed (so `int age;` and `int age()` can both be members of a class).

11 Project Plan

11.1 Process

Our group decided early on that we wanted to pursue an object-oriented language. We were inspired by the approach of Smalltalk – where everything's an object – and combined it with the syntax and conventions of Java and C++. We added our own touch with the `permit` form of encapsulation and thus, Iris – who delivers messages in Greek mythology – was born.

Despite being a team of five, our team found it easy to coordinate meeting times throughout the semester. In general, all work was done during these meetings, with the occasional bug resolved or feature added individually, on one's own time. Since submitting "Hello World!", we had a roadmap of language features we wished to implement and systematically built up the language in this order.

11.2 Project Timeline

20 September, 2023: Project Proposal
16 October, 2023: Scanner and Parser
18 October, 2023: Language Reference Manual: First Draft
8 November, 2023: Hello World!
29 November, 2023: Extended Testsuite
15 December, 2023: Final Compiler

11.3 Roles and Responsibilities

As stated above, our group implemented the overwhelming majority of this project together. We took our individual roles with a grain of salt – while Valerie often took lead on testing, and Trevor often took the lead on system architecture, every member of our group had significant contributions to all parts of the project.

11.4 Development Environment

The Iris compiler is written in OCaml, a functional programming language. The final output of the compiler, a result of codegen, are LLVM instructions, generated by the OCaml LLVM API. The editor used was Visual Studio Code, and the LiveShare extension was often used for simultaneous collaboration. GitHub was used for version control.

11.5 Intermediary Versions

Version Overview

Our commit log in our repository was incorrect for some reason (totaled to about 20 commits even though we had upwards of 100). Don't trust the image; we worked pretty steadily through November.

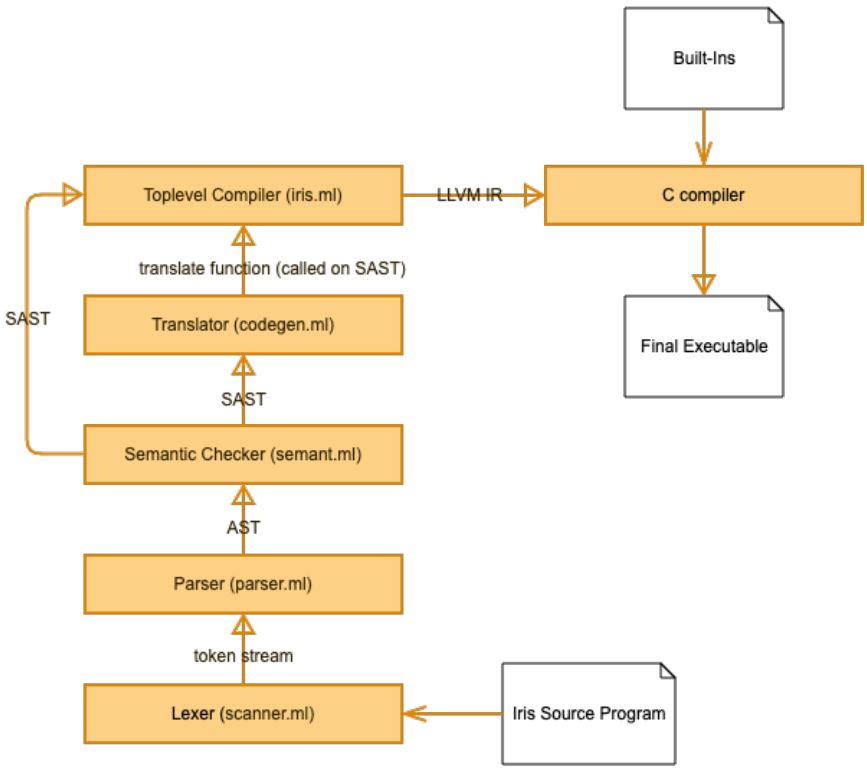
Here is the high-level prose overview of our versions.

1. Scanner/Parser with pretty-printed AST
2. Semant with pretty-printed SAST
3. Semant and Codegen: "Hello world!" and back-end setup for inheritance
4. Semant and Codegen: Incorporating much of Microc's lower-level code into class/program architecture (`exprs`, `stmts`, variables, etc.), testing via the `Main` class
5. Semant and Codegen: Implemented simple encapsulation with `public` and `private`
6. Semant and Codegen: Object instantiation, taking into account inheritance

7. Semant and Codegen: Implemented member functions with inheritance, including overloading and dynamic dispatch
8. Semant and Codegen: Implemented constructors for objects, including the dynamic dispatch case (e.g. `Animal a = new Cat();`)
9. Semant and Codegen: Implemented member variables: overloading, inheritance
10. Semant and Codegen: Implemented `univ` functions
11. Semant and Codegen: Implemented Iris `permit` encapsulation, including the lable and LLVM calls to a linked-in function that checks for permissions

12 Architectural Design

12.1 Architecture Diagram



12.1.1 Lexer (Whole Team)

The lexer, housed in `scanner.ml`, sweeps through an Iris program and compiles it into a series of tokens for use by the parser.

12.1.2 Parser (Whole Team)

The parser, housed in `parser.ml`, builds an abstract syntax tree of an Iris program from a series of tokens provided by the lexer.

12.1.3 Semantic Checker (Whole Team)

Housed in `semant.ml` and `gus.ml`, the semantic checker intakes an abstract syntax tree representing a program and, with the help of a large lookup table, outputs a semantically-checked abstract syntax tree of the program.

12.1.4 Translator (Whole Team)

Housed in `codegen.ml` and `guini.ml`, the translator intakes a semantically-checked abstract syntax tree of a program and emits an LLVM module representing the program. A lot of the work in this component was spearheaded by Trevor and Josh, although we all contributed to the writing, testing, and debugging of the generated LLVM.

12.1.5 Top-level Compiler (Ayda, Starter Code)

The Iris top-level compiler was mostly adapted from the MicroC top-level compiler. It coordinates the other steps of compilation and emits the final LLVM IR. Ayda wrote a lot of the LLVM code generation that linked Iris function calls to their respective C built-ins.

12.1.6 Built-Ins for `stdio` (Valerie, Ayda)

This is the small C file that links standard input and output functions and string conversions for primitive types, for use by our `Olympus` class' functionality. The file also contains a function for string comparison and a helpful function used in codegen for checking permitted classes, written by Trevor, Josh, and Tim. This is not available to user.

12.2 Implementation Summary

12.2.1 Feature: Classes and Inheritance

The whole team worked on implementing classes and inheritance. Since we needed a lookup table encoding all of the classes and the information contained within them for much of the semantic checking, we decided to take a first pass over the program and build a large lookup table that also encodes parent classes. In the semantic checker, we check class types and assignments by traversing through class' parents until we either hit Object, or the class we're looking for, to determine if it is a valid assignment. We also organize each class' functions to prepare to correctly instantiate their function virtual tables in the code generation phase; in order to handle overloaded functions in dynamic dispatch cases, the ordering of the virtual tables must match from parent to child class. Trevor and Valerie took the lead on implementing those features, but we all contributed to testing and debugging. We also implemented inheritance of member variables, which is something Josh took point on.

12.2.2 Feature: Encapsulation

The whole team worked on implementing `public` and `private` encapsulation levels in the semantic and code generation phases. We also all contributed to the semantic checking of permitted members, including the underlying design of the SAST. This included decisions about how to represent member variables and functions that fell under the `permit` label. Trevor, Josh, and Tim took the lead on implementing the code generation for (`permit`, which involved the runtime checking of permitted member variables and classes.

13 Test Plan

13.1 Testing Approach

We tested incrementally by hand as we added features, and often added these by-hand tests to our test suite. We have an automated test script that runs our test suite (the same as the ones submitted for intermediary deliverables) which we've run after every big change to ensure older cases still passed.

13.2 Demo Programs

13.2.1 Animal Guessing Game

Source File

```
.~*
 ~
 ~      AnimalGame.iris
 ~      by Valerie Zhang
 ~
 ~
 ~      This program is an interactive animal guessing game. The program
 ~      selects an animal based on user input and allows for the player
 ~      to make guess ("g"), ask for hints ("h"), or quit ("q"). There
 ~      are a total of 6 hints about different characteristics of the
 ~      animal. Players have up to 3 guesses before the program will reveal
 ~      the answer and quit.
*~.

class Animal (Main) {
    public:

        .. These functions set private member variables
        void setAnimal(string s) {
            animal = s;
        }
        void setType(string s) {
            type = s;
        }
        void setLegs(int i) {
            legs = i;
        }
        void setFur(bool b) {
            fur = b;
        }
        void setFly(bool b) {
            canFly = b;
        }

        .. These functions return the values of private variables
        string getClass() {
            return animal;
        }

        string getType() {
            return type;
        }

        int numLegs() {
```

```
            return legs;
        }

        bool hasFur() {
            return fur;
        }

        bool canFly() {
            return canFly;
        }

        void noise() {
            Olympus.println("Noise");
        }

        void funFact() {
            Olympus.println("There are 5 different animal classes: Mammals, Birds, Insects, Amphibians, and Reptiles!");
        }

    private:
        string animal;
        string type;
        int legs;
        bool fur;
        bool canFly;
}

.~*
 ~  Below are the different animals to guess. They override the noise and
 ~  funFact methods of Animal
*~.

class Cat of Animal () {
    public:
        void noise() {
            Olympus.println("Meow!");
        }

        void funFact() {
            Olympus.println("I can land on all fours!");
        }
}
```

```
class Dog of Animal () {
    public:
        void noise() {
            Olympus.println("Bark!");
        }

        void funFact() {
            Olympus.println("I have a great sense of smell!");
        }
}

class Chicken of Animal () {
    public:
        void noise() {
            Olympus.println("Cluck!");
        }

        void funFact() {
            Olympus.println("I am one of the closest living descendants to dinosaurs!");
        }
}

class Snake of Animal () {
    public:
        void noise() {
            Olympus.println("Hiss!");
        }

        void funFact() {
            Olympus.println("I can be very venomous!");
        }
}

class Frog of Animal () {
    public:
        void noise() {
            Olympus.println("Croak!");
        }

        void funFact() {
            Olympus.println("I have excellent night vision");
        }
}
```

```
class Bee of Animal () {
    public:
        void noise() {
            Olympus.println("Buzz!");
        }

        void funFact() {
            Olympus.println("They make something sweet ");
        }
}

class Richard of Animal () {
    public:
        void noise() {
            Olympus.println("Howdy folks!");
        }

        void funFact() {
            Olympus.println("This is someone you know very well ;)");
        }
}

.~*
 ~ AnimalGame is a class that contains methods to run the game
*~.
class AnimalGame () {
    public:
        .. runs the game including the command loop to make guesses, get hints, and quit
        void run() {
            string input = "";
            int count = 0;

            .. Game start-up text, and getting user input to choose animal
            Olympus.println("Hello and welcome to the animal guessing game!");
            Olympus.println("You will have 3 tries to guess the animal, along with a number of hints.");
            Olympus.println("Please enter a number from 1-6");
            input = Olympus.readaline();

            .. Gets animal to guess
            Animal a = getAnimal(input);

            .. command loop for the game
            while (!Olympus.streq(input, "q") || count != 3) {
                Olympus.print("Please enter g to make a guess, h for a hint, or q to quit: ");
```

```
            input = Olympus.readaline();
            if (Olympus.streq(input, "q")) { .. Player wants to quit
                Olympus.println("Thank you for playing the game. We hope you had fun!!");
                return;
            } else if (Olympus.streq(input, "g")) { .. Player wants to make a guess
                if (!makeGuess(a, count)) { .. Increment if player got guess wrong
                    count++;
                    if (count == 3) {
                        return;
                    }
                }
            } else if (Olympus.streq(input, "h")) { .. Player wants a hint
                giveHint(a);
            }
        }
    }


    .. method to get kind of animal based on user input
    Animal getAnimal(string input) {
        Animal a;
        if (Olympus.streq(input, "1")) {
            Cat c = new Cat();
            c.setAnimal("Mammal");
            c.setType("Cat");
            c.setLegs(4);
            c.setFur(true);
            c.setFly(false);
            a = c;
        } else if (Olympus.streq(input, "2")) {
            Dog d = new Dog();
            d.setAnimal("Mammal");
            d.setType("Dog");
            d.setLegs(4);
            d.setFur(true);
            d.setFly(false);
            a = d;
        } else if (Olympus.streq(input, "3")) {
            Chicken c = new Chicken();
            c.setAnimal("Bird");
            c.setType("Chicken");
            c.setLegs(2);
            c.setFur(false);
```

```
            c.setFly(true);
            a = c;
        } else if (Olympus.streq(input, "4")) {
            Snake s = new Snake();
            s.setAnimal("Reptile");
            s.setType("Snake");
            s.setLegs(0);
            s.setFur(false);
            s.setFly(false);
            a = s;
        } else if (Olympus.streq(input, "5")) {
            Frog f = new Frog();
            f.setAnimal("Amphibian");
            f.setType("Frog");
            f.setLegs(4);
            f.setFur(false);
            f.setFly(false);
            a = f;
        } else if (Olympus.streq(input, "6")) {
            Bee b = new Bee();
            b.setAnimal("Insect");
            b.setType("Bee");
            b.setLegs(6);
            b.setFur(false);
            b.setFly(true);
            a = b;
        } else {
            Richard r = new Richard();
            r.setAnimal("Compiler");
            r.setType("Richard");
            r.setLegs(2);
            r.setFur(false);
            r.setFly(false);
            a = r;
        }
        return a;
    }

    .. method to perform operations to make a guess
    bool makeGuess(Animal a, int count) {

        .. gets user guess
        string input;
        Olympus.print("Please enter your guess: ");
```

```
        input = Olympus.readaline();

        .. checks guess against animal's type
        if (!Olympus.streq(input, a.getType())) {
            Olympus.println("Wrong guess :(");
            count++;
            if (count == 3) {
                Olympus.println("You have used up all your guesses.");
                Olympus.print("The correct animal is: ");
                Olympus.println(a.getType());
                Olympus.println("You'll get 'em next time :)!");
            }
            return false;
        }

        Olympus.println("Ding ding ding! You got it! Thank you for playing the game :D");
        return true;
    }

    .. method to perform operations for giving hints
    void giveHint(Animal a) {

        .. printing for requesting a hint
        Olympus.println("Select one of the options for a hint:");
        Olympus.println("Class");
        Olympus.println("Legs");
        Olympus.println("Fur");
        Olympus.println("Fly");
        Olympus.println("Noise");
        Olympus.println("Fun fact");
        string input = Olympus.readaline();

        .. parse through input
        if (Olympus.streq(input, "Class")) {
            Olympus.print("The animal is part of the class: ");
            Olympus.println(a.getClass());
        } else if (Olympus.streq(input, "Legs")) {
            Olympus.print("The animal has ");
            Olympus.printi(a.numLegs());
            Olympus.println(" legs");
        } else if (Olympus.streq(input, "Fur")) {
            if (a.hasFur()) {
                Olympus.println("The animal does have fur");
```

```
                } else {
                    Olympus.println("The animal does not have fur");
                }
            } else if (Olympus.streq(input, "Fly")) {
                if (a.canFly()) {
                    Olympus.println("The animal can fly");
                } else {
                    Olympus.println("The animal cannot fly");
                }
            } else if (Olympus.streq(input, "Noise")) {
                a.noise();
            } else if (Olympus.streq(input, "Fun fact")) {
                a.funFact();
            } else {
                Olympus.println("We don't have that type of hint :(");
            }
        }

}

.. The main class
class Main () {
    public:
        int univ main() {
            .. Main.run();

            AnimalGame ag = new AnimalGame();
            ag.run();
            return 0;
        }
}
```

LLVM File

Note: lines that are tabbed 4 spaces have been done so for photo formatting. They originally were all on one line

```
; ModuleID = 'Iris'
source_filename = "Iris"

%Object_vtable = type {}
%Animal_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Animal**)*, void (%Animal**)* }
%Animal = type { %Animal_vtable*, i32, [2 x i8*], i8*, i8*, i32, i1, i1 }
%Cat_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Cat**)*, void (%Cat**)* }
%Cat = type { %Cat_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%Dog_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Dog**)*, void (%Dog**)* }
%Dog = type { %Dog_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%Chicken_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Chicken**)*, void (%Chicken**)* }
%Chicken = type { %Chicken_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%Snake_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Snake**)*, void (%Snake**)* }
%Snake = type { %Snake_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%Frog_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Frog**)*, void (%Frog**)* }
%Frog = type { %Frog_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%Bee_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Bee**)*, void (%Bee**)* }
%Bee = type { %Bee_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%Richard_vtable = type { void (%Animal**, i8*)*, void (%Animal**, i8*)*,
    void (%Animal**, i32)*, void (%Animal**, i1)*, void (%Animal**, i1)*,
    i8* (%Animal**)*, i8* (%Animal**)*, i32 (%Animal**)*, i1 (%Animal**)*,
    i1 (%Animal**)*, void (%Richard**)*, void (%Richard**)* }
%Richard = type { %Richard_vtable*, i32, [1 x i8*], i8*, i8*, i32, i1, i1 }
%AnimalGame_vtable = type { void (%AnimalGame**)*, %Animal** (%AnimalGame**, i8*)*,
    i1 (%AnimalGame**, %Animal**, i32)*, void (%AnimalGame**, %Animal**)* }
%AnimalGame = type { %AnimalGame_vtable*, i32, [1 x i8*] }
%Main_vtable = type { i32 ()* }

```

```
@Object_vtable_data = global %Object_vtable zeroinitializer
@fmt = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.2 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.3 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.5 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.6 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.7 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.8 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.9 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.10 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.11 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.12 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.13 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.14 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.15 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.16 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.17 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.18 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.19 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.20 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.21 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.22 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.23 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.24 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.25 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.26 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.27 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.28 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.29 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@fmt.30 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.31 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.32 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str = private unnamed_addr constant [6 x i8] c"Noise\00", align 1
@fmt.33 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.34 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.35 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.36 = private unnamed_addr constant [89 x i8]
    c"There are 5 different animal classes: Mammals, Birds, Insects,
    Amphibians, and Reptiles!\00", align 1
@Animal_vtable_data = global %Animal_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Animal**)* @Animal_noise,
    void (%Animal**)* @Animal_funFact }
@fmt.37 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
```

```
@fmt.38 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.39 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.40 = private unnamed_addr constant [6 x i8] c"Meow!\00", align 1
@fmt.41 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.42 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.43 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.44 = private unnamed_addr constant [25 x i8]
    c"I can land on all fours!\00", align 1
@Cat_vtable_data = global %Cat_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Cat**)* @Cat_noise,
    void (%Cat**)* @Cat_funFact }
@fmt.45 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.46 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.47 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.48 = private unnamed_addr constant [6 x i8] c"Bark!\00", align 1
@fmt.49 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.50 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.51 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.52 = private unnamed_addr constant [31 x i8]
    c"I have a great sense of smell!\00", align 1
@Dog_vtable_data = global %Dog_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Dog**)* @Dog_noise,
    void (%Dog**)* @Dog_funFact }
@fmt.53 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.54 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.55 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.56 = private unnamed_addr constant [7 x i8] c"Cluck!\00", align 1
@fmt.57 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.58 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.59 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.60 = private unnamed_addr constant [57 x i8]
    c"I am one of the closest living descendants to dinosaurs!\00", align 1
@Chicken_vtable_data = global %Chicken_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Chicken**)* @Chicken_noise,
    void (%Chicken**)* @Chicken_funFact }
@fmt.61 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.62 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
```

```
@fmt.63 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.64 = private unnamed_addr constant [6 x i8] c"Hiss!\00", align 1
@fmt.65 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.66 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.67 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.68 = private unnamed_addr constant [24 x i8]
    c"I can be very venomous!\00", align 1
@Snake_vtable_data = global %Snake_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Snake**)* @Snake_noise,
    void (%Snake**)* @Snake_funFact }
@fmt.69 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.70 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.71 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.72 = private unnamed_addr constant [7 x i8] c"Croak!\00", align 1
@fmt.73 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.74 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.75 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.76 = private unnamed_addr constant [30 x i8]
    c"I have excellent night vision\00", align 1
@Frog_vtable_data = global %Frog_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Frog**)* @Frog_noise,
    void (%Frog**)* @Frog_funFact }
@fmt.77 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.78 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.79 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.80 = private unnamed_addr constant [6 x i8] c"Buzz!\00", align 1
@fmt.81 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.82 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.83 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.84 = private unnamed_addr constant [27 x i8]
    c"They make something sweet \00", align 1
@Bee_vtable_data = global %Bee_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Bee**)* @Bee_noise,
    void (%Bee**)* @Bee_funFact }
@fmt.85 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.86 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.87 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
```

```
@global_str.88 = private unnamed_addr constant [13 x i8] c"Howdy folks!\00", align 1
@fmt.89 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.90 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.91 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.92 = private unnamed_addr constant [38 x i8]
    c"This is someone you know very well ;)\00", align 1
@Richard_vtable_data = global %Richard_vtable { void (%Animal**, i8*)* @Animal_setAnimal,
    void (%Animal**, i8*)* @Animal_setType, void (%Animal**, i32)* @Animal_setLegs,
    void (%Animal**, i1)* @Animal_setFur, void (%Animal**, i1)* @Animal_setFly,
    i8* (%Animal**)* @Animal_getClass, i8* (%Animal**)* @Animal_getType,
    i32 (%Animal**)* @Animal_numLegs, i1 (%Animal**)* @Animal_hasFur,
    i1 (%Animal**)* @Animal_canFly, void (%Richard**)* @Richard_noise,
    void (%Richard**)* @Richard_funFact }
@fmt.93 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.94 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.95 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.96 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@global_str.97 = private unnamed_addr constant [47 x i8]
    c"Hello and welcome to the animal guessing game!\00", align 1
@global_str.98 = private unnamed_addr constant [73 x i8]
    c"You will have 3 tries to guess the animal, along with a number of hints.\00", align 1
@global_str.99 = private unnamed_addr constant [31 x i8]
    c"Please enter a number from 1-6\00", align 1
@global_str.100 = private unnamed_addr constant [61 x i8]
    c"Please enter g to make a guess, h for a hint, or q to quit: \00", align 1
@global_str.101 = private unnamed_addr constant [2 x i8] c"q\00", align 1
@global_str.102 = private unnamed_addr constant [54 x i8]
    c"Thank you for playing the game. We hope you had fun!!\00", align 1
@global_str.103 = private unnamed_addr constant [2 x i8] c"g\00", align 1
@global_str.104 = private unnamed_addr constant [2 x i8] c"h\00", align 1
@global_str.105 = private unnamed_addr constant [2 x i8] c"q\00", align 1
@fmt.106 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.107 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.108 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.109 = private unnamed_addr constant [2 x i8] c"1\00", align 1
@temp_str = private unnamed_addr constant [4 x i8] c"Cat\00", align 1
@global_str.110 = private unnamed_addr constant [7 x i8] c"Mammal\00", align 1
@global_str.111 = private unnamed_addr constant [4 x i8] c"Cat\00", align 1
@global_str.112 = private unnamed_addr constant [2 x i8] c"2\00", align 1
@temp_str.113 = private unnamed_addr constant [4 x i8] c"Dog\00", align 1
@global_str.114 = private unnamed_addr constant [7 x i8] c"Mammal\00", align 1
@global_str.115 = private unnamed_addr constant [4 x i8] c"Dog\00", align 1
@global_str.116 = private unnamed_addr constant [2 x i8] c"3\00", align 1
@temp_str.117 = private unnamed_addr constant [8 x i8] c"Chicken\00", align 1
@global_str.118 = private unnamed_addr constant [5 x i8] c"Bird\00", align 1
@global_str.119 = private unnamed_addr constant [8 x i8] c"Chicken\00", align 1
@global_str.120 = private unnamed_addr constant [2 x i8] c"4\00", align 1
@temp_str.121 = private unnamed_addr constant [6 x i8] c"Snake\00", align 1
@global_str.122 = private unnamed_addr constant [8 x i8] c"Reptile\00", align 1
```

```
@global_str.123 = private unnamed_addr constant [6 x i8] c"Snake\00", align 1
@global_str.124 = private unnamed_addr constant [2 x i8] c"5\00", align 1
@temp_str.125 = private unnamed_addr constant [5 x i8] c"Frog\00", align 1
@global_str.126 = private unnamed_addr constant [10 x i8] c"Amphibian\00", align 1
@global_str.127 = private unnamed_addr constant [5 x i8] c"Frog\00", align 1
@global_str.128 = private unnamed_addr constant [2 x i8] c"6\00", align 1
@temp_str.129 = private unnamed_addr constant [4 x i8] c"Bee\00", align 1
@global_str.130 = private unnamed_addr constant [7 x i8] c"Insect\00", align 1
@global_str.131 = private unnamed_addr constant [4 x i8] c"Bee\00", align 1
@temp_str.132 = private unnamed_addr constant [8 x i8] c"Richard\00", align 1
@global_str.133 = private unnamed_addr constant [9 x i8] c"Compiler\00", align 1
@global_str.134 = private unnamed_addr constant [8 x i8] c"Richard\00", align 1
@fmt.135 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.136 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.137 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.138 = private unnamed_addr constant [26 x i8]
    c"Please enter your guess: \00", align 1
@global_str.139 = private unnamed_addr constant [15 x i8]
    c"Wrong guess :(\00", align 1
@global_str.140 = private unnamed_addr constant [35 x i8]
    c"You have used up all your guesses.\00", align 1
@global_str.141 = private unnamed_addr constant [24 x i8]
    c"The correct animal is: \00", align 1
@global_str.142 = private unnamed_addr constant [29 x i8]
    c"You'll get 'em next time :)!\00", align 1
@global_str.143 = private unnamed_addr constant [62 x i8]
    c"Ding ding ding! You got it! Thank you for playing the game :D\00", align 1
@fmt.144 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.145 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.146 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.147 = private unnamed_addr constant [38 x i8]
    c"Select one of the options for a hint:\00", align 1
@global_str.148 = private unnamed_addr constant [6 x i8] c"Class\00", align 1
@global_str.149 = private unnamed_addr constant [5 x i8] c"Legs\00", align 1
@global_str.150 = private unnamed_addr constant [4 x i8] c"Fur\00", align 1
@global_str.151 = private unnamed_addr constant [4 x i8] c"Fly\00", align 1
@global_str.152 = private unnamed_addr constant [6 x i8] c"Noise\00", align 1
@global_str.153 = private unnamed_addr constant [9 x i8] c"Fun fact\00", align 1
@global_str.154 = private unnamed_addr constant [6 x i8] c"Class\00", align 1
@global_str.155 = private unnamed_addr constant [34 x i8]
    c"The animal is part of the class: \00", align 1
@global_str.156 = private unnamed_addr constant [5 x i8] c"Legs\00", align 1
@global_str.157 = private unnamed_addr constant [16 x i8] c"The animal has \00", align 1
@global_str.158 = private unnamed_addr constant [6 x i8] c" legs\00", align 1
@global_str.159 = private unnamed_addr constant [4 x i8] c"Fur\00", align 1
@global_str.160 = private unnamed_addr constant [25 x i8]
    c"The animal does have fur\00", align 1
@global_str.161 = private unnamed_addr constant [29 x i8]
    c"The animal does not have fur\00", align 1
```

```
@global_str.162 = private unnamed_addr constant [4 x i8] c"Fly\00", align 1
@global_str.163 = private unnamed_addr constant [19 x i8] c"The animal can fly\00", align 1
@global_str.164 = private unnamed_addr constant [22 x i8] c"The animal cannot fly\00", align 1
@global_str.165 = private unnamed_addr constant [6 x i8] c"Noise\00", align 1
@global_str.166 = private unnamed_addr constant [9 x i8] c"Fun fact\00", align 1
@global_str.167 = private unnamed_addr constant [35 x i8]
    c"We don't have that type of hint :(\00", align 1
@AnimalGame_vtable_data = global %AnimalGame_vtable {
        void (%AnimalGame**)* @AnimalGame_run, %Animal** (%AnimalGame**, i8*)* @AnimalGame_getAnimal,
        i1 (%AnimalGame**, %Animal**, i32)* @AnimalGame_makeGuess,
        void (%AnimalGame**, %Animal**)* @AnimalGame_giveHint }
@fmt.168 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.169 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.170 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@temp_str.171 = private unnamed_addr constant [11 x i8] c"AnimalGame\00", align 1
@Main_vtable_data = global %Main_vtable { i32 ()* @main }

declare i32 @printf(i8*, ...)

declare i32 @printerr(i8*, ...)

declare i8** @readaline(...)

declare i1 @streq(i8*, i8*, ...)

declare void @class_permitted(i8*, i8**, i32, ...)

define void @Animal_setAnimal(%Animal** %0, i8* %1) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %s = alloca i8*, align 8
  store i8* %1, i8** %s, align 8
  %s2 = load i8*, i8** %s, align 8
  %temp3 = load %Animal*, %Animal** %temp, align 8
  %selfanimal = getelementptr inbounds %Animal, %Animal* %temp3, i32 0, i32 3
  store i8* %s2, i8** %selfanimal, align 8
  ret void
}

define void @Animal_setType(%Animal** %0, i8* %1) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %s = alloca i8*, align 8
  store i8* %1, i8** %s, align 8
  %s2 = load i8*, i8** %s, align 8
```

```
344     %temp3 = load %Animal*, %Animal** %temp, align 8
345     %selftype = getelementptr inbounds %Animal, %Animal* %temp3, i32 0, i32 4
346     store i8* %s2, i8** %selftype, align 8
347     ret void
348   }
349
350   define void @Animal_setLegs(%Animal** %0, i32 %1) {
351   entry:
352     %temp = alloca %Animal*, align 8
353     %temp1 = load %Animal*, %Animal** %0, align 8
354     store %Animal* %temp1, %Animal** %temp, align 8
355     %i = alloca i32, align 4
356     store i32 %1, i32* %i, align 4
357     %i2 = load i32, i32* %i, align 4
358     %temp3 = load %Animal*, %Animal** %temp, align 8
359     %selflegs = getelementptr inbounds %Animal, %Animal* %temp3, i32 0, i32 5
360     store i32 %i2, i32* %selflegs, align 4
361     ret void
362   }
363
364   define void @Animal_setFur(%Animal** %0, i1 %1) {
365   entry:
366     %temp = alloca %Animal*, align 8
367     %temp1 = load %Animal*, %Animal** %0, align 8
368     store %Animal* %temp1, %Animal** %temp, align 8
369     %b = alloca i1, align 1
370     store i1 %1, i1* %b, align 1
371     %b2 = load i1, i1* %b, align 1
372     %temp3 = load %Animal*, %Animal** %temp, align 8
373     %selffur = getelementptr inbounds %Animal, %Animal* %temp3, i32 0, i32 6
374     store i1 %b2, i1* %selffur, align 1
375     ret void
376   }
377
378   define void @Animal_setFly(%Animal** %0, i1 %1) {
379   entry:
380     %temp = alloca %Animal*, align 8
381     %temp1 = load %Animal*, %Animal** %0, align 8
382     store %Animal* %temp1, %Animal** %temp, align 8
383     %b = alloca i1, align 1
384     store i1 %1, i1* %b, align 1
385     %b2 = load i1, i1* %b, align 1
386     %temp3 = load %Animal*, %Animal** %temp, align 8
387     %selfcanFly = getelementptr inbounds %Animal, %Animal* %temp3, i32 0, i32 7
388     store i1 %b2, i1* %selfcanFly, align 1
389     ret void
390   }
391
392   define i8* @Animal_getClass(%Animal** %0) {
```

```
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %temp2 = load %Animal*, %Animal** %temp, align 8
  %selfanimal = getelementptr inbounds %Animal, %Animal* %temp2, i32 0, i32 3
  %selfanimal3 = load i8*, i8** %selfanimal, align 8
  ret i8* %selfanimal3
}

define i8* @Animal_getType(%Animal** %0) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %temp2 = load %Animal*, %Animal** %temp, align 8
  %selftype = getelementptr inbounds %Animal, %Animal* %temp2, i32 0, i32 4
  %selftype3 = load i8*, i8** %selftype, align 8
  ret i8* %selftype3
}

define i32 @Animal_numLegs(%Animal** %0) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %temp2 = load %Animal*, %Animal** %temp, align 8
  %selflegs = getelementptr inbounds %Animal, %Animal* %temp2, i32 0, i32 5
  %selflegs3 = load i32, i32* %selflegs, align 4
  ret i32 %selflegs3
}

define i1 @Animal_hasFur(%Animal** %0) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %temp2 = load %Animal*, %Animal** %temp, align 8
  %selffur = getelementptr inbounds %Animal, %Animal* %temp2, i32 0, i32 6
  %selffur3 = load i1, i1* %selffur, align 1
  ret i1 %selffur3
}

define i1 @Animal_canFly(%Animal** %0) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
```

```
441     %temp2 = load %Animal*, %Animal** %temp, align 8
442     %selfcanFly = getelementptr inbounds %Animal, %Animal* %temp2, i32 0, i32 7
443     %selfcanFly3 = load i1, i1* %selfcanFly, align 1
444     ret i1 %selfcanFly3
445   }
446
447   define void @Animal_noise(%Animal** %0) {
448   entry:
449     %temp = alloca %Animal*, align 8
450     %temp1 = load %Animal*, %Animal** %0, align 8
451     store %Animal* %temp1, %Animal** %temp, align 8
452     %printf = call i32 (i8*, ...) @printf(
453         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.31, i32 0, i32 0),
454         i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str, i32 0, i32 0))
455     ret void
456   }
457
458   define void @Animal_funFact(%Animal** %0) {
459   entry:
460     %temp = alloca %Animal*, align 8
461     %temp1 = load %Animal*, %Animal** %0, align 8
462     store %Animal* %temp1, %Animal** %temp, align 8
463     %printf = call i32 (i8*, ...) @printf(
464       i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.34, i32 0, i32 0),
465       i8* getelementptr inbounds ([89 x i8], [89 x i8]* @global_str.36, i32 0, i32 0))
466     ret void
467   }
468
469   define void @Cat_noise(%Cat** %0) {
470   entry:
471     %temp = alloca %Cat*, align 8
472     %temp1 = load %Cat*, %Cat** %0, align 8
473     store %Cat* %temp1, %Cat** %temp, align 8
474     %printf = call i32 (i8*, ...) @printf(
475       i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.38, i32 0, i32 0),
476       i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.40, i32 0, i32 0))
477     ret void
478   }
479
480   define void @Cat_funFact(%Cat** %0) {
481   entry:
482     %temp = alloca %Cat*, align 8
483     %temp1 = load %Cat*, %Cat** %0, align 8
484     store %Cat* %temp1, %Cat** %temp, align 8
485     %printf = call i32 (i8*, ...) @printf(
486       i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.42, i32 0, i32 0),
487       i8* getelementptr inbounds ([25 x i8], [25 x i8]* @global_str.44, i32 0, i32 0))
488     ret void
489   }
```

```
490
491   define void @Dog_noise(%Dog** %0) {
492   entry:
493     %temp = alloca %Dog*, align 8
494     %temp1 = load %Dog*, %Dog** %0, align 8
495     store %Dog* %temp1, %Dog** %temp, align 8
496     %printf = call i32 (i8*, ...) @printf(
497         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.46, i32 0, i32 0),
498         i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.48, i32 0, i32 0))
499     ret void
500   }
501
502   define void @Dog_funFact(%Dog** %0) {
503   entry:
504     %temp = alloca %Dog*, align 8
505     %temp1 = load %Dog*, %Dog** %0, align 8
506     store %Dog* %temp1, %Dog** %temp, align 8
507     %printf = call i32 (i8*, ...) @printf(
508         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.50, i32 0, i32 0),
509         i8* getelementptr inbounds ([31 x i8], [31 x i8]* @global_str.52, i32 0, i32 0))
510     ret void
511   }
512
513   define void @Chicken_noise(%Chicken** %0) {
514   entry:
515     %temp = alloca %Chicken*, align 8
516     %temp1 = load %Chicken*, %Chicken** %0, align 8
517     store %Chicken* %temp1, %Chicken** %temp, align 8
518     %printf = call i32 (i8*, ...) @printf(
519         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.54, i32 0, i32 0),
520         i8* getelementptr inbounds ([7 x i8], [7 x i8]* @global_str.56, i32 0, i32 0))
521     ret void
522   }
523
524   define void @Chicken_funFact(%Chicken** %0) {
525   entry:
526     %temp = alloca %Chicken*, align 8
527     %temp1 = load %Chicken*, %Chicken** %0, align 8
528     store %Chicken* %temp1, %Chicken** %temp, align 8
529     %printf = call i32 (i8*, ...) @printf(
530         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.58, i32 0, i32 0),
531         i8* getelementptr inbounds ([57 x i8], [57 x i8]* @global_str.60, i32 0, i32 0))
532     ret void
533   }
534
535   define void @Snake_noise(%Snake** %0) {
536   entry:
537     %temp = alloca %Snake*, align 8
```

```
538     %temp1 = load %Snake*, %Snake** %0, align 8
539     store %Snake* %temp1, %Snake** %temp, align 8
540     %printf = call i32 (i8*, ...) @printf(
541         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.62, i32 0, i32 0),
542         i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.64, i32 0, i32 0))
543     ret void
544   }
545
546   define void @Snake_funFact(%Snake** %0) {
547   entry:
548     %temp = alloca %Snake*, align 8
549     %temp1 = load %Snake*, %Snake** %0, align 8
550     store %Snake* %temp1, %Snake** %temp, align 8
551     %printf = call i32 (i8*, ...) @printf(
552         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.66, i32 0, i32 0),
553         i8* getelementptr inbounds ([24 x i8], [24 x i8]* @global_str.68, i32 0, i32 0))
554     ret void
555   }
556
557   define void @Frog_noise(%Frog** %0) {
558   entry:
559     %temp = alloca %Frog*, align 8
560     %temp1 = load %Frog*, %Frog** %0, align 8
561     store %Frog* %temp1, %Frog** %temp, align 8
562     %printf = call i32 (i8*, ...) @printf(
563         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.70, i32 0, i32 0),
564         i8* getelementptr inbounds ([7 x i8], [7 x i8]* @global_str.72, i32 0, i32 0))
565     ret void
566   }
567
568   define void @Frog_funFact(%Frog** %0) {
569   entry:
570     %temp = alloca %Frog*, align 8
571     %temp1 = load %Frog*, %Frog** %0, align 8
572     store %Frog* %temp1, %Frog** %temp, align 8
573     %printf = call i32 (i8*, ...) @printf(
574         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.74, i32 0, i32 0),
575         i8* getelementptr inbounds ([30 x i8], [30 x i8]* @global_str.76, i32 0, i32 0))
576     ret void
577   }
578
579   define void @Bee_noise(%Bee** %0) {
580   entry:
581     %temp = alloca %Bee*, align 8
582     %temp1 = load %Bee*, %Bee** %0, align 8
583     store %Bee* %temp1, %Bee** %temp, align 8
584     %printf = call i32 (i8*, ...) @printf(
585         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.78, i32 0, i32 0),
586         i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.80, i32 0, i32 0))
```

```
587     ret void
588   }
589
590   define void @Bee_funFact(%Bee** %0) {
591   entry:
592     %temp = alloca %Bee*, align 8
593     %temp1 = load %Bee*, %Bee** %0, align 8
594     store %Bee* %temp1, %Bee** %temp, align 8
595     %printf = call i32 (i8*, ...) @printf(
596         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.82, i32 0, i32 0),
597         i8* getelementptr inbounds ([27 x i8], [27 x i8]* @global_str.84, i32 0, i32 0))
598     ret void
599   }
600
601   define void @Richard_noise(%Richard** %0) {
602   entry:
603     %temp = alloca %Richard*, align 8
604     %temp1 = load %Richard*, %Richard** %0, align 8
605     store %Richard* %temp1, %Richard** %temp, align 8
606     %printf = call i32 (i8*, ...) @printf(
607         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.86, i32 0, i32 0),
608         i8* getelementptr inbounds ([13 x i8], [13 x i8]* @global_str.88, i32 0, i32 0))
609     ret void
610   }
611
612   define void @Richard_funFact(%Richard** %0) {
613   entry:
614     %temp = alloca %Richard*, align 8
615     %temp1 = load %Richard*, %Richard** %0, align 8
616     store %Richard* %temp1, %Richard** %temp, align 8
617     %printf = call i32 (i8*, ...) @printf(
618         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.90, i32 0, i32 0),
619         i8* getelementptr inbounds ([38 x i8], [38 x i8]* @global_str.92, i32 0, i32 0))
620     ret void
621   }
622
623   define void @AnimalGame_run(%AnimalGame** %0) {
624   entry:
625     %temp = alloca %AnimalGame*, align 8
626     %temp1 = load %AnimalGame*, %AnimalGame** %0, align 8
627     store %AnimalGame* %temp1, %AnimalGame** %temp, align 8
628     %input = alloca i8*, align 8
629     store i8* getelementptr inbounds ([1 x i8],
630         [1 x i8]* @global_str.96, i32 0, i32 0), i8** %input, align 8
631     %count = alloca i32, align 4
632     store i32 0, i32* %count, align 4
633     %printf = call i32 (i8*, ...) @printf(
634         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.94, i32 0, i32 0),
635         i8* getelementptr inbounds ([47 x i8], [47 x i8]* @global_str.97, i32 0, i32 0))
```

```
636     %printf2 = call i32 (i8*, ...) @printf(
637         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.94, i32 0, i32 0),
638         i8* getelementptr inbounds ([73 x i8], [73 x i8]* @global_str.98, i32 0, i32 0))
639     %printf3 = call i32 (i8*, ...) @printf(
640         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.94, i32 0, i32 0),
641         i8* getelementptr inbounds ([31 x i8], [31 x i8]* @global_str.99, i32 0, i32 0))
642     %readaline = call i8** (...) @readaline()
643     %get_temp = load i8*, i8** %readaline, align 8
644     store i8* %get_temp, i8** %input, align 8
645     %temp4 = load %AnimalGame*, %AnimalGame** %temp, align 8
646     %vtable = getelementptr inbounds %AnimalGame, %AnimalGame* %temp4, i32 0, i32 0
647     %input5 = load i8*, i8** %input, align 8
648     %vtable6 = load %AnimalGame_vtable*, %AnimalGame_vtable** %vtable, align 8
649     %fun_to_call = getelementptr inbounds %AnimalGame_vtable,
650         %AnimalGame_vtable* %vtable6, i32 0, i32 1
651     %function = load %Animal** (%AnimalGame**, i8*)*,
652         %Animal** (%AnimalGame**, i8*)** %fun_to_call, align 8
653     %getAnimal_result = call %Animal** %function(%AnimalGame** %temp, i8* %input5)
654     %a = alloca %Animal*, align 8
655     %temp7 = load %Animal*, %Animal** %getAnimal_result, align 8
656     store %Animal* %temp7, %Animal** %a, align 8
657     br label %while
658
659 while:                                            ; preds = %merge, %entry
660     %input44 = load i8*, i8** %input, align 8
661     %streq45 = call i1 (i8*, i8*, ...) @streq(i8* %input44,
662         i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.105, i32 0, i32 0))
663     %tmp46 = xor i1 %streq45, true
664     %count47 = load i32, i32* %count, align 4
665     %tmp48 = icmp ne i32 %count47, 3
666     %tmp49 = or i1 %tmp46, %tmp48
667     br i1 %tmp49, label %while_body, label %merge50
668
669 while_body:                                       ; preds = %while
670     %printf8 = call i32 (i8*, ...) @printf(
671         i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.93, i32 0, i32 0),
672         i8* getelementptr inbounds ([61 x i8], [61 x i8]* @global_str.100, i32 0, i32 0))
673     %readaline9 = call i8** (...) @readaline()
674     %get_temp10 = load i8*, i8** %readaline9, align 8
675     store i8* %get_temp10, i8** %input, align 8
676     %input11 = load i8*, i8** %input, align 8
677     %streq = call i1 (i8*, i8*, ...) @streq(i8* %input11,
678         i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.101, i32 0, i32 0))
679     br i1 %streq, label %then, label %else
680
681 merge:                                            ; preds = %merge15
682     br label %while
683
684 then:                                             ; preds = %while_body
```

```
685     %printf12 = call i32 (i8*, ...) @printf(
686         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.94, i32 0, i32 0),
687         i8* getelementptr inbounds ([54 x i8], [54 x i8]* @global_str.102, i32 0, i32 0))
688     ret void
689
690   else:                                             ; preds = %while_body
691     %input13 = load i8*, i8** %input, align 8
692     %streq14 = call i1 (i8*, i8*, ...) @streq(i8* %input13,
693         i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.103, i32 0, i32 0))
694     br i1 %streq14, label %then16, label %else33
695
696   merge15:                                          ; preds = %merge36, %merge23
697     br label %merge
698
699   then16:                                           ; preds = %else
700     %temp17 = load %AnimalGame*, %AnimalGame** %temp, align 8
701     %vtable18 = getelementptr inbounds %AnimalGame, %AnimalGame* %temp17, i32 0, i32 0
702     %count19 = load i32, i32* %count, align 4
703     %vtable20 = load %AnimalGame_vtable*, %AnimalGame_vtable** %vtable18, align 8
704     %fun_to_call21 = getelementptr inbounds %AnimalGame_vtable,
705         %AnimalGame_vtable* %vtable20, i32 0, i32 2
706     %function22 = load i1 (%AnimalGame**, %Animal**, i32)*,
707         i1 (%AnimalGame**, %Animal**, i32)** %fun_to_call21, align 8
708     %makeGuess_result = call i1 %function22(%AnimalGame** %temp,
709         %Animal** %a, i32 %count19)
710     %tmp = xor i1 %makeGuess_result, true
711     br i1 %tmp, label %then24, label %else32
712
713   merge23:                                          ; preds = %else32, %merge29
714     br label %merge15
715
716   then24:                                           ; preds = %then16
717     %count25 = load i32, i32* %count, align 4
718     %tmp26 = add i32 %count25, 1
719     store i32 %tmp26, i32* %count, align 4
720     %count27 = load i32, i32* %count, align 4
721     %tmp28 = icmp eq i32 %count27, 3
722     br i1 %tmp28, label %then30, label %else31
723
724   merge29:                                          ; preds = %else31
725     br label %merge23
726
727   then30:                                           ; preds = %then24
728     ret void
729
730   else31:                                           ; preds = %then24
731     br label %merge29
732
733   else32:                                           ; preds = %then16
```

```
734     br label %merge23
735
736   else33:                                           ; preds = %else
737     %input34 = load i8*, i8** %input, align 8
738     %streq35 = call i1 (i8*, i8*, ...) @streq(i8* %input34,
739         i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.104, i32 0, i32 0))
740     br i1 %streq35, label %then37, label %else43
741
742   merge36:                                          ; preds = %else43, %then37
743     br label %merge15
744
745   then37:                                           ; preds = %else33
746     %temp38 = load %AnimalGame*, %AnimalGame** %temp, align 8
747     %vtable39 = getelementptr inbounds %AnimalGame, %AnimalGame* %temp38, i32 0, i32 0
748     %vtable40 = load %AnimalGame_vtable*, %AnimalGame_vtable** %vtable39, align 8
749     %fun_to_call41 = getelementptr inbounds %AnimalGame_vtable,
750         %AnimalGame_vtable* %vtable40, i32 0, i32 3
751     %function42 = load void (%AnimalGame**, %Animal**)*,
752         void (%AnimalGame**, %Animal**)** %fun_to_call41, align 8
753     call void %function42(%AnimalGame** %temp, %Animal** %a)
754     br label %merge36
755
756   else43:                                           ; preds = %else33
757     br label %merge36
758
759   merge50:                                          ; preds = %while
760     ret void
761   }
762
763   define %Animal** @AnimalGame_getAnimal(%AnimalGame** %0, i8* %1) {
764   entry:
765     %temp = alloca %AnimalGame*, align 8
766     %temp1 = load %AnimalGame*, %AnimalGame** %0, align 8
767     store %AnimalGame* %temp1, %AnimalGame** %temp, align 8
768     %input = alloca i8*, align 8
769     store i8* %1, i8** %input, align 8
770     %a = alloca %Animal*, align 8
771     %input2 = load i8*, i8** %input, align 8
772     %streq = call i1 (i8*, i8*, ...) @streq(i8* %input2,
773         i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.109, i32 0, i32 0))
774     br i1 %streq, label %then, label %else
775
776   merge:                                            ; preds = %merge35, %then
777     ret %Animal** %a
778
779   then:                                             ; preds = %entry
780     %malloccall = tail call i8* @malloc(i32 ptrtoint
781         (%Cat* getelementptr (%Cat, %Cat* null, i32 1) to i32))
782     %Cat = bitcast i8* %malloccall to %Cat*
```

```
%vtable = getelementptr inbounds %Cat, %Cat* %Cat, i32 0, i32 0
store %Cat_vtable* @Cat_vtable_data, %Cat_vtable** %vtable, align 8
%temp3 = alloca %Cat*, align 8
%num_permitted = getelementptr inbounds %Cat, %Cat* %Cat, i32 0, i32 1
store i32 1, i32* %num_permitted, align 4
%permit_list = getelementptr inbounds %Cat, %Cat* %Cat, i32 0, i32 2
store [1 x i8*] [i8* getelementptr inbounds
    ([4 x i8], [4 x i8]* @temp_str, i32 0, i32 0)], [1 x i8*]* %permit_list, align 8
store %Cat* %Cat, %Cat** %temp3, align 8
%c = alloca %Cat*, align 8
%temp4 = load %Cat*, %Cat** %temp3, align 8
store %Cat* %temp4, %Cat** %c, align 8
%temp5 = load %Cat*, %Cat** %c, align 8
%vtable6 = getelementptr inbounds %Cat, %Cat* %temp5, i32 0, i32 0
%vtable7 = load %Cat_vtable*, %Cat_vtable** %vtable6, align 8
%fun_to_call = getelementptr inbounds %Cat_vtable,
    %Cat_vtable* %vtable7, i32 0, i32 0
%function = load void (%Animal**, i8*)*,
    void (%Animal**, i8*)** %fun_to_call, align 8
%arg_cast = bitcast %Cat** %c to %Animal**
call void %function(%Animal** %arg_cast,
    i8* getelementptr inbounds ([7 x i8], [7 x i8]* @global_str.110, i32 0, i32 0))
%temp8 = load %Cat*, %Cat** %c, align 8
%vtable9 = getelementptr inbounds %Cat, %Cat* %temp8, i32 0, i32 0
%vtable10 = load %Cat_vtable*, %Cat_vtable** %vtable9, align 8
%fun_to_call11 = getelementptr inbounds %Cat_vtable,
    %Cat_vtable* %vtable10, i32 0, i32 1
%function12 = load void (%Animal**, i8*)*,
    void (%Animal**, i8*)** %fun_to_call11, align 8
%arg_cast13 = bitcast %Cat** %c to %Animal**
call void %function12(%Animal** %arg_cast13,
    i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.111, i32 0, i32 0))
%temp14 = load %Cat*, %Cat** %c, align 8
%vtable15 = getelementptr inbounds %Cat, %Cat* %temp14, i32 0, i32 0
%vtable16 = load %Cat_vtable*, %Cat_vtable** %vtable15, align 8
%fun_to_call17 = getelementptr inbounds %Cat_vtable,
    %Cat_vtable* %vtable16, i32 0, i32 2
%function18 = load void (%Animal**, i32)*,
    void (%Animal**, i32)** %fun_to_call17, align 8
%arg_cast19 = bitcast %Cat** %c to %Animal**
call void %function18(%Animal** %arg_cast19, i32 4)
%temp20 = load %Cat*, %Cat** %c, align 8
%vtable21 = getelementptr inbounds %Cat, %Cat* %temp20, i32 0, i32 0
%vtable22 = load %Cat_vtable*, %Cat_vtable** %vtable21, align 8
%fun_to_call23 = getelementptr inbounds %Cat_vtable,
    %Cat_vtable* %vtable22, i32 0, i32 3
%function24 = load void (%Animal**, i1)*,
    void (%Animal**, i1)** %fun_to_call23, align 8
```

```
  %arg_cast25 = bitcast %Cat** %c to %Animal**
  call void %function24(%Animal** %arg_cast25, i1 true)
  %temp26 = load %Cat*, %Cat** %c, align 8
  %vtable27 = getelementptr inbounds %Cat, %Cat* %temp26, i32 0, i32 0
  %vtable28 = load %Cat_vtable*, %Cat_vtable** %vtable27, align 8
  %fun_to_call29 = getelementptr inbounds %Cat_vtable,
      %Cat_vtable* %vtable28, i32 0, i32 4
  %function30 = load void (%Animal**, i1)*,
      void (%Animal**, i1)** %fun_to_call29, align 8
  %arg_cast31 = bitcast %Cat** %c to %Animal**
  call void %function30(%Animal** %arg_cast31, i1 false)
  %cast_assign = bitcast %Cat** %c to %Animal**
  %temp32 = load %Animal*, %Animal** %cast_assign, align 8
  store %Animal* %temp32, %Animal** %a, align 8
  br label %merge

else:                                             ; preds = %entry
  %input33 = load i8*, i8** %input, align 8
  %streq34 = call i1 (i8*, i8*, ...) @streq(i8* %input33,
      i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.112, i32 0, i32 0))
  br i1 %streq34, label %then36, label %else75

merge35:                                          ; preds = %merge78, %then36
  br label %merge

then36:                                           ; preds = %else
  %malloccall37 = tail call i8* @malloc(i32 ptrtoint
      (%Dog* getelementptr (%Dog, %Dog* null, i32 1) to i32))
  %Dog = bitcast i8* %malloccall37 to %Dog*
  %vtable38 = getelementptr inbounds %Dog, %Dog* %Dog, i32 0, i32 0
  store %Dog_vtable* @Dog_vtable_data, %Dog_vtable** %vtable38, align 8
  %temp39 = alloca %Dog*, align 8
  %num_permitted40 = getelementptr inbounds %Dog, %Dog* %Dog, i32 0, i32 1
  store i32 1, i32* %num_permitted40, align 4
  %permit_list41 = getelementptr inbounds %Dog, %Dog* %Dog, i32 0, i32 2
  store [1 x i8*] [i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @temp_str.113, i32 0, i32 0)], [1 x i8*]* %permit_list41, align 8
  store %Dog* %Dog, %Dog** %temp39, align 8
  %d = alloca %Dog*, align 8
  %temp42 = load %Dog*, %Dog** %temp39, align 8
  store %Dog* %temp42, %Dog** %d, align 8
  %temp43 = load %Dog*, %Dog** %d, align 8
  %vtable44 = getelementptr inbounds %Dog, %Dog* %temp43, i32 0, i32 0
  %vtable45 = load %Dog_vtable*, %Dog_vtable** %vtable44, align 8
  %fun_to_call46 = getelementptr inbounds %Dog_vtable,
      %Dog_vtable* %vtable45, i32 0, i32 0
  %function47 = load void (%Animal**, i8*)*,
      void (%Animal**, i8*)** %fun_to_call46, align 8
  %arg_cast48 = bitcast %Dog** %d to %Animal**
```

```
880     call void %function47(%Animal** %arg_cast48,
881         i8* getelementptr inbounds ([7 x i8], [7 x i8]* @global_str.114, i32 0, i32 0))
882     %temp49 = load %Dog*, %Dog** %d, align 8
883     %vtable50 = getelementptr inbounds %Dog, %Dog* %temp49, i32 0, i32 0
884     %vtable51 = load %Dog_vtable*, %Dog_vtable** %vtable50, align 8
885     %fun_to_call52 = getelementptr inbounds %Dog_vtable,
886         %Dog_vtable* %vtable51, i32 0, i32 1
887     %function53 = load void (%Animal**, i8*)*,
888         void (%Animal**, i8*)** %fun_to_call52, align 8
889     %arg_cast54 = bitcast %Dog** %d to %Animal**
890     call void %function53(%Animal** %arg_cast54,
891         i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.115, i32 0, i32 0))
892     %temp55 = load %Dog*, %Dog** %d, align 8
893     %vtable56 = getelementptr inbounds %Dog, %Dog* %temp55, i32 0, i32 0
894     %vtable57 = load %Dog_vtable*, %Dog_vtable** %vtable56, align 8
895     %fun_to_call58 = getelementptr inbounds %Dog_vtable,
896         %Dog_vtable* %vtable57, i32 0, i32 2
897     %function59 = load void (%Animal**, i32)*,
898         void (%Animal**, i32)** %fun_to_call58, align 8
899     %arg_cast60 = bitcast %Dog** %d to %Animal**
900     call void %function59(%Animal** %arg_cast60, i32 4)
901     %temp61 = load %Dog*, %Dog** %d, align 8
902     %vtable62 = getelementptr inbounds %Dog,
903         %Dog* %temp61, i32 0, i32 0
904     %vtable63 = load %Dog_vtable*, %Dog_vtable** %vtable62, align 8
905     %fun_to_call64 = getelementptr inbounds %Dog_vtable,
906         %Dog_vtable* %vtable63, i32 0, i32 3
907     %function65 = load void (%Animal**, i1)*,
908         void (%Animal**, i1)** %fun_to_call64, align 8
909     %arg_cast66 = bitcast %Dog** %d to %Animal**
910     call void %function65(%Animal** %arg_cast66, i1 true)
911     %temp67 = load %Dog*, %Dog** %d, align 8
912     %vtable68 = getelementptr inbounds %Dog, %Dog* %temp67, i32 0, i32 0
913     %vtable69 = load %Dog_vtable*, %Dog_vtable** %vtable68, align 8
914     %fun_to_call70 = getelementptr inbounds %Dog_vtable,
915         %Dog_vtable* %vtable69, i32 0, i32 4
916     %function71 = load void (%Animal**, i1)*,
917         void (%Animal**, i1)** %fun_to_call70, align 8
918     %arg_cast72 = bitcast %Dog** %d to %Animal**
919     call void %function71(%Animal** %arg_cast72, i1 false)
920     %cast_assign73 = bitcast %Dog** %d to %Animal**
921     %temp74 = load %Animal*, %Animal** %cast_assign73, align 8
922     store %Animal* %temp74, %Animal** %a, align 8
923     br label %merge35
924
925   else75:                                           ; preds = %else
926     %input76 = load i8*, i8** %input, align 8
927     %streq77 = call i1 (i8*, i8*, ...) @streq(i8* %input76,
928         i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.116, i32 0, i32 0))
```

```
  br i1 %streq77, label %then79, label %else119

merge78:                                          ; preds = %merge122, %then79
  br label %merge35

then79:                                           ; preds = %else75
  %malloccall80 = tail call i8* @malloc(i32 ptrtoint
      (%Chicken* getelementptr (%Chicken, %Chicken* null, i32 1) to i32))
  %Chicken = bitcast i8* %malloccall80 to %Chicken*
  %vtable81 = getelementptr inbounds %Chicken, %Chicken* %Chicken, i32 0, i32 0
  store %Chicken_vtable* @Chicken_vtable_data, %Chicken_vtable** %vtable81, align 8
  %temp82 = alloca %Chicken*, align 8
  %num_permitted83 = getelementptr inbounds %Chicken,
      %Chicken* %Chicken, i32 0, i32 1
  store i32 1, i32* %num_permitted83, align 4
  %permit_list84 = getelementptr inbounds %Chicken,
      %Chicken* %Chicken, i32 0, i32 2
  store [1 x i8*] [i8* getelementptr inbounds
      ([8 x i8], [8 x i8]* @temp_str.117, i32 0, i32 0)], [1 x i8*]* %permit_list84, align 8
  store %Chicken* %Chicken, %Chicken** %temp82, align 8
  %c85 = alloca %Chicken*, align 8
  %temp86 = load %Chicken*, %Chicken** %temp82, align 8
  store %Chicken* %temp86, %Chicken** %c85, align 8
  %temp87 = load %Chicken*, %Chicken** %c85, align 8
  %vtable88 = getelementptr inbounds %Chicken, %Chicken* %temp87, i32 0, i32 0
  %vtable89 = load %Chicken_vtable*, %Chicken_vtable** %vtable88, align 8
  %fun_to_call90 = getelementptr inbounds %Chicken_vtable,
      %Chicken_vtable* %vtable89, i32 0, i32 0
  %function91 = load void (%Animal**, i8*)*, void (%Animal**, i8*)** %fun_to_call90, align 8
  %arg_cast92 = bitcast %Chicken** %c85 to %Animal**
  call void %function91(%Animal** %arg_cast92,
      i8* getelementptr inbounds ([5 x i8], [5 x i8]* @global_str.118, i32 0, i32 0))
  %temp93 = load %Chicken*, %Chicken** %c85, align 8
  %vtable94 = getelementptr inbounds %Chicken, %Chicken* %temp93, i32 0, i32 0
  %vtable95 = load %Chicken_vtable*, %Chicken_vtable** %vtable94, align 8
  %fun_to_call96 = getelementptr inbounds %Chicken_vtable,
      %Chicken_vtable* %vtable95, i32 0, i32 1
  %function97 = load void (%Animal**, i8*)*,
      void (%Animal**, i8*)** %fun_to_call96, align 8
  %arg_cast98 = bitcast %Chicken** %c85 to %Animal**
  call void %function97(%Animal** %arg_cast98,
      i8* getelementptr inbounds ([8 x i8], [8 x i8]* @global_str.119, i32 0, i32 0))
  %temp99 = load %Chicken*, %Chicken** %c85, align 8
  %vtable100 = getelementptr inbounds %Chicken, %Chicken* %temp99, i32 0, i32 0
  %vtable101 = load %Chicken_vtable*, %Chicken_vtable** %vtable100, align 8
  %fun_to_call102 = getelementptr inbounds %Chicken_vtable,
      %Chicken_vtable* %vtable101, i32 0, i32 2
  %function103 = load void (%Animal**, i32)*,
      void (%Animal**, i32)** %fun_to_call102, align 8
```

```
978       %arg_cast104 = bitcast %Chicken** %c85 to %Animal**
979       call void %function103(%Animal** %arg_cast104, i32 2)
980       %temp105 = load %Chicken*, %Chicken** %c85, align 8
981       %vtable106 = getelementptr inbounds %Chicken, %Chicken* %temp105, i32 0, i32 0
982       %vtable107 = load %Chicken_vtable*, %Chicken_vtable** %vtable106, align 8
983       %fun_to_call108 = getelementptr inbounds %Chicken_vtable,
984           %Chicken_vtable* %vtable107, i32 0, i32 3
985       %function109 = load void (%Animal**, i1)*,
986           void (%Animal**, i1)** %fun_to_call108, align 8
987       %arg_cast110 = bitcast %Chicken** %c85 to %Animal**
988       call void %function109(%Animal** %arg_cast110, i1 false)
989       %temp111 = load %Chicken*, %Chicken** %c85, align 8
990       %vtable112 = getelementptr inbounds %Chicken, %Chicken* %temp111, i32 0, i32 0
991       %vtable113 = load %Chicken_vtable*, %Chicken_vtable** %vtable112, align 8
992       %fun_to_call114 = getelementptr inbounds %Chicken_vtable,
993           %Chicken_vtable* %vtable113, i32 0, i32 4
994       %function115 = load void (%Animal**, i1)*,
995           void (%Animal**, i1)** %fun_to_call114, align 8
996       %arg_cast116 = bitcast %Chicken** %c85 to %Animal**
997       call void %function115(%Animal** %arg_cast116, i1 true)
998       %cast_assign117 = bitcast %Chicken** %c85 to %Animal**
999       %temp118 = load %Animal*, %Animal** %cast_assign117, align 8
1000      store %Animal* %temp118, %Animal** %a, align 8
1001      br label %merge78
1002
1003    else119:                                          ; preds = %else75
1004      %input120 = load i8*, i8** %input, align 8
1005      %streq121 = call i1 (i8*, i8*, ...) @streq(i8* %input120,
1006          i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.120, i32 0, i32 0))
1007      br i1 %streq121, label %then123, label %else162
1008
1009    merge122:                                         ; preds = %merge165, %then123
1010      br label %merge78
1011
1012    then123:                                          ; preds = %else119
1013      %malloccall124 = tail call i8* @malloc(i32 ptrtoint
1014          (%Snake* getelementptr (%Snake, %Snake* null, i32 1) to i32))
1015      %Snake = bitcast i8* %malloccall124 to %Snake*
1016      %vtable125 = getelementptr inbounds %Snake, %Snake* %Snake, i32 0, i32 0
1017      store %Snake_vtable* @Snake_vtable_data, %Snake_vtable** %vtable125, align 8
1018      %temp126 = alloca %Snake*, align 8
1019      %num_permitted127 = getelementptr inbounds %Snake, %Snake* %Snake, i32 0, i32 1
1020      store i32 1, i32* %num_permitted127, align 4
1021      %permit_list128 = getelementptr inbounds %Snake, %Snake* %Snake, i32 0, i32 2
1022      store [1 x i8*] [i8* getelementptr inbounds ([6 x i8],
1023          [6 x i8]* @temp_str.121, i32 0, i32 0)], [1 x i8*]* %permit_list128, align 8
1024      store %Snake* %Snake, %Snake** %temp126, align 8
1025      %s = alloca %Snake*, align 8
```

```
1026    %temp129 = load %Snake*, %Snake** %temp126, align 8
1027    store %Snake* %temp129, %Snake** %s, align 8
1028    %temp130 = load %Snake*, %Snake** %s, align 8
1029    %vtable131 = getelementptr inbounds %Snake, %Snake* %temp130, i32 0, i32 0
1030    %vtable132 = load %Snake_vtable*, %Snake_vtable** %vtable131, align 8
1031    %fun_to_call133 = getelementptr inbounds %Snake_vtable,
1032        %Snake_vtable* %vtable132, i32 0, i32 0
1033    %function134 = load void (%Animal**, i8*)*,
1034        void (%Animal**, i8*)** %fun_to_call133, align 8
1035    %arg_cast135 = bitcast %Snake** %s to %Animal**
1036    call void %function134(%Animal** %arg_cast135,
1037        i8* getelementptr inbounds ([8 x i8], [8 x i8]* @global_str.122, i32 0, i32 0))
1038    %temp136 = load %Snake*, %Snake** %s, align 8
1039    %vtable137 = getelementptr inbounds %Snake, %Snake* %temp136, i32 0, i32 0
1040    %vtable138 = load %Snake_vtable*, %Snake_vtable** %vtable137, align 8
1041    %fun_to_call139 = getelementptr inbounds %Snake_vtable,
1042        %Snake_vtable* %vtable138, i32 0, i32 1
1043    %function140 = load void (%Animal**, i8*)*,
1044        void (%Animal**, i8*)** %fun_to_call139, align 8
1045    %arg_cast141 = bitcast %Snake** %s to %Animal**
1046    call void %function140(%Animal** %arg_cast141,
1047        i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.123, i32 0, i32 0))
1048    %temp142 = load %Snake*, %Snake** %s, align 8
1049    %vtable143 = getelementptr inbounds %Snake, %Snake* %temp142, i32 0, i32 0
1050    %vtable144 = load %Snake_vtable*, %Snake_vtable** %vtable143, align 8
1051    %fun_to_call145 = getelementptr inbounds %Snake_vtable,
1052        %Snake_vtable* %vtable144, i32 0, i32 2
1053    %function146 = load void (%Animal**, i32)*,
1054        void (%Animal**, i32)** %fun_to_call145, align 8
1055    %arg_cast147 = bitcast %Snake** %s to %Animal**
1056    call void %function146(%Animal** %arg_cast147, i32 0)
1057    %temp148 = load %Snake*, %Snake** %s, align 8
1058    %vtable149 = getelementptr inbounds %Snake, %Snake* %temp148, i32 0, i32 0
1059    %vtable150 = load %Snake_vtable*, %Snake_vtable** %vtable149, align 8
1060    %fun_to_call151 = getelementptr inbounds %Snake_vtable,
1061        %Snake_vtable* %vtable150, i32 0, i32 3
1062    %function152 = load void (%Animal**, i1)*,
1063        void (%Animal**, i1)** %fun_to_call151, align 8
1064    %arg_cast153 = bitcast %Snake** %s to %Animal**
1065    call void %function152(%Animal** %arg_cast153, i1 false)
1066    %temp154 = load %Snake*, %Snake** %s, align 8
1067    %vtable155 = getelementptr inbounds %Snake, %Snake* %temp154, i32 0, i32 0
1068    %vtable156 = load %Snake_vtable*, %Snake_vtable** %vtable155, align 8
1069    %fun_to_call157 = getelementptr inbounds %Snake_vtable,
1070        %Snake_vtable* %vtable156, i32 0, i32 4
1071    %function158 = load void (%Animal**, i1)*,
1072        void (%Animal**, i1)** %fun_to_call157, align 8
1073    %arg_cast159 = bitcast %Snake** %s to %Animal**
1074    call void %function158(%Animal** %arg_cast159, i1 false)
```

```
1075      %cast_assign160 = bitcast %Snake** %s to %Animal**
1076      %temp161 = load %Animal*, %Animal** %cast_assign160, align 8
1077      store %Animal* %temp161, %Animal** %a, align 8
1078      br label %merge122
1079
1080    else162:                                          ; preds = %else119
1081      %input163 = load i8*, i8** %input, align 8
1082      %streq164 = call i1 (i8*, i8*, ...) @streq(i8* %input163,
1083          i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.124, i32 0, i32 0))
1084      br i1 %streq164, label %then166, label %else205
1085
1086    merge165:                                         ; preds = %merge208, %then166
1087      br label %merge122
1088
1089    then166:                                          ; preds = %else162
1090      %malloccall167 = tail call i8* @malloc(i32 ptrtoint
1091          (%Frog* getelementptr (%Frog, %Frog* null, i32 1) to i32))
1092      %Frog = bitcast i8* %malloccall167 to %Frog*
1093      %vtable168 = getelementptr inbounds %Frog, %Frog* %Frog, i32 0, i32 0
1094      store %Frog_vtable* @Frog_vtable_data, %Frog_vtable** %vtable168, align 8
1095      %temp169 = alloca %Frog*, align 8
1096      %num_permitted170 = getelementptr inbounds %Frog, %Frog* %Frog, i32 0, i32 1
1097      store i32 1, i32* %num_permitted170, align 4
1098      %permit_list171 = getelementptr inbounds %Frog, %Frog* %Frog, i32 0, i32 2
1099      store [1 x i8*] [i8* getelementptr inbounds ([5 x i8],
1100          [5 x i8]* @temp_str.125, i32 0, i32 0)], [1 x i8*]* %permit_list171, align 8
1101      store %Frog* %Frog, %Frog** %temp169, align 8
1102      %f = alloca %Frog*, align 8
1103      %temp172 = load %Frog*, %Frog** %temp169, align 8
1104      store %Frog* %temp172, %Frog** %f, align 8
1105      %temp173 = load %Frog*, %Frog** %f, align 8
1106      %vtable174 = getelementptr inbounds %Frog, %Frog* %temp173, i32 0, i32 0
1107      %vtable175 = load %Frog_vtable*, %Frog_vtable** %vtable174, align 8
1108      %fun_to_call176 = getelementptr inbounds %Frog_vtable,
1109          %Frog_vtable* %vtable175, i32 0, i32 0
1110      %function177 = load void (%Animal**, i8*)*,
1111          void (%Animal**, i8*)** %fun_to_call176, align 8
1112      %arg_cast178 = bitcast %Frog** %f to %Animal**
1113      call void %function177(%Animal** %arg_cast178,
1114          i8* getelementptr inbounds ([10 x i8], [10 x i8]* @global_str.126, i32 0, i32 0))
1115      %temp179 = load %Frog*, %Frog** %f, align 8
1116      %vtable180 = getelementptr inbounds %Frog, %Frog* %temp179, i32 0, i32 0
1117      %vtable181 = load %Frog_vtable*, %Frog_vtable** %vtable180, align 8
1118      %fun_to_call182 = getelementptr inbounds %Frog_vtable,
1119          %Frog_vtable* %vtable181, i32 0, i32 1
1120      %function183 = load void (%Animal**, i8*)*,
1121          void (%Animal**, i8*)** %fun_to_call182, align 8
1122      %arg_cast184 = bitcast %Frog** %f to %Animal**
```

```
1123      call void %function183(%Animal** %arg_cast184,
1124          i8* getelementptr inbounds ([5 x i8], [5 x i8]* @global_str.127, i32 0, i32 0))
1125      %temp185 = load %Frog*, %Frog** %f, align 8
1126      %vtable186 = getelementptr inbounds %Frog, %Frog* %temp185, i32 0, i32 0
1127      %vtable187 = load %Frog_vtable*, %Frog_vtable** %vtable186, align 8
1128      %fun_to_call188 = getelementptr inbounds %Frog_vtable,
1129          %Frog_vtable* %vtable187, i32 0, i32 2
1130      %function189 = load void (%Animal**, i32)*,
1131          void (%Animal**, i32)** %fun_to_call188, align 8
1132      %arg_cast190 = bitcast %Frog** %f to %Animal**
1133      call void %function189(%Animal** %arg_cast190, i32 4)
1134      %temp191 = load %Frog*, %Frog** %f, align 8
1135      %vtable192 = getelementptr inbounds %Frog,
1136          %Frog* %temp191, i32 0, i32 0
1137      %vtable193 = load %Frog_vtable*, %Frog_vtable** %vtable192, align 8
1138      %fun_to_call194 = getelementptr inbounds %Frog_vtable,
1139          %Frog_vtable* %vtable193, i32 0, i32 3
1140      %function195 = load void (%Animal**, i1)*,
1141          void (%Animal**, i1)** %fun_to_call194, align 8
1142      %arg_cast196 = bitcast %Frog** %f to %Animal**
1143      call void %function195(%Animal** %arg_cast196, i1 false)
1144      %temp197 = load %Frog*, %Frog** %f, align 8
1145      %vtable198 = getelementptr inbounds %Frog, %Frog* %temp197, i32 0, i32 0
1146      %vtable199 = load %Frog_vtable*, %Frog_vtable** %vtable198, align 8
1147      %fun_to_call200 = getelementptr inbounds %Frog_vtable,
1148          %Frog_vtable* %vtable199, i32 0, i32 4
1149      %function201 = load void (%Animal**, i1)*,
1150          void (%Animal**, i1)** %fun_to_call200, align 8
1151      %arg_cast202 = bitcast %Frog** %f to %Animal**
1152      call void %function201(%Animal** %arg_cast202, i1 false)
1153      %cast_assign203 = bitcast %Frog** %f to %Animal**
1154      %temp204 = load %Animal*, %Animal** %cast_assign203, align 8
1155      store %Animal* %temp204, %Animal** %a, align 8
1156      br label %merge165
1157
1158    else205:                                          ; preds = %else162
1159      %input206 = load i8*, i8** %input, align 8
1160      %streq207 = call i1 (i8*, i8*, ...) @streq(i8* %input206,
1161          i8* getelementptr inbounds ([2 x i8], [2 x i8]* @global_str.128, i32 0, i32 0))
1162      br i1 %streq207, label %then209, label %else248
1163
1164    merge208:                                         ; preds = %else248, %then209
1165      br label %merge165
1166
1167    then209:                                          ; preds = %else205
1168      %malloccall210 = tail call i8* @malloc(i32 ptrtoint
1169          (%Bee* getelementptr (%Bee, %Bee* null, i32 1) to i32))
1170      %Bee = bitcast i8* %malloccall210 to %Bee*
1171      %vtable211 = getelementptr inbounds %Bee, %Bee* %Bee, i32 0, i32 0
```

```
1172    store %Bee_vtable* @Bee_vtable_data, %Bee_vtable** %vtable211, align 8
1173    %temp212 = alloca %Bee*, align 8
1174    %num_permitted213 = getelementptr inbounds %Bee, %Bee* %Bee, i32 0, i32 1
1175    store i32 1, i32* %num_permitted213, align 4
1176    %permit_list214 = getelementptr inbounds %Bee, %Bee* %Bee, i32 0, i32 2
1177    store [1 x i8*] [i8* getelementptr inbounds ([4 x i8],
1178        [4 x i8]* @temp_str.129, i32 0, i32 0)], [1 x i8*]* %permit_list214, align 8
1179    store %Bee* %Bee, %Bee** %temp212, align 8
1180    %b = alloca %Bee*, align 8
1181    %temp215 = load %Bee*, %Bee** %temp212, align 8
1182    store %Bee* %temp215, %Bee** %b, align 8
1183    %temp216 = load %Bee*, %Bee** %b, align 8
1184    %vtable217 = getelementptr inbounds %Bee, %Bee* %temp216, i32 0, i32 0
1185    %vtable218 = load %Bee_vtable*, %Bee_vtable** %vtable217, align 8
1186    %fun_to_call219 = getelementptr inbounds %Bee_vtable,
1187        %Bee_vtable* %vtable218, i32 0, i32 0
1188    %function220 = load void (%Animal**, i8*)*,
1189        void (%Animal**, i8*)** %fun_to_call219, align 8
1190    %arg_cast221 = bitcast %Bee** %b to %Animal**
1191    call void %function220(%Animal** %arg_cast221,
1192        i8* getelementptr inbounds ([7 x i8], [7 x i8]* @global_str.130, i32 0, i32 0))
1193    %temp222 = load %Bee*, %Bee** %b, align 8
1194    %vtable223 = getelementptr inbounds %Bee, %Bee* %temp222, i32 0, i32 0
1195    %vtable224 = load %Bee_vtable*, %Bee_vtable** %vtable223, align 8
1196    %fun_to_call225 = getelementptr inbounds %Bee_vtable,
1197        %Bee_vtable* %vtable224, i32 0, i32 1
1198    %function226 = load void (%Animal**, i8*)*,
1199        void (%Animal**, i8*)** %fun_to_call225, align 8
1200    %arg_cast227 = bitcast %Bee** %b to %Animal**
1201    call void %function226(%Animal** %arg_cast227,
1202        i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.131, i32 0, i32 0))
1203    %temp228 = load %Bee*, %Bee** %b, align 8
1204    %vtable229 = getelementptr inbounds %Bee, %Bee* %temp228, i32 0, i32 0
1205    %vtable230 = load %Bee_vtable*, %Bee_vtable** %vtable229, align 8
1206    %fun_to_call231 = getelementptr inbounds %Bee_vtable,
1207        %Bee_vtable* %vtable230, i32 0, i32 2
1208    %function232 = load void (%Animal**, i32)*,
1209        void (%Animal**, i32)** %fun_to_call231, align 8
1210    %arg_cast233 = bitcast %Bee** %b to %Animal**
1211    call void %function232(%Animal** %arg_cast233, i32 6)
1212    %temp234 = load %Bee*, %Bee** %b, align 8
1213    %vtable235 = getelementptr inbounds %Bee, %Bee* %temp234, i32 0, i32 0
1214    %vtable236 = load %Bee_vtable*, %Bee_vtable** %vtable235, align 8
1215    %fun_to_call237 = getelementptr inbounds %Bee_vtable,
1216        %Bee_vtable* %vtable236, i32 0, i32 3
1217    %function238 = load void (%Animal**, i1)*,
1218        void (%Animal**, i1)** %fun_to_call237, align 8
1219    %arg_cast239 = bitcast %Bee** %b to %Animal**
1220    call void %function238(%Animal** %arg_cast239, i1 false)
```

```
    %temp240 = load %Bee*, %Bee** %b, align 8
    %vtable241 = getelementptr inbounds %Bee, %Bee* %temp240, i32 0, i32 0
    %vtable242 = load %Bee_vtable*, %Bee_vtable** %vtable241, align 8
    %fun_to_call243 = getelementptr inbounds %Bee_vtable,
        %Bee_vtable* %vtable242, i32 0, i32 4
    %function244 = load void (%Animal**, i1)*,
        void (%Animal**, i1)** %fun_to_call243, align 8
    %arg_cast245 = bitcast %Bee** %b to %Animal**
    call void %function244(%Animal** %arg_cast245, i1 true)
    %cast_assign246 = bitcast %Bee** %b to %Animal**
    %temp247 = load %Animal*, %Animal** %cast_assign246, align 8
    store %Animal* %temp247, %Animal** %a, align 8
    br label %merge208

else248:                                          ; preds = %else205
    %malloccall249 = tail call i8* @malloc(i32 ptrtoint
        (%Richard* getelementptr (%Richard, %Richard* null, i32 1) to i32))
    %Richard = bitcast i8* %malloccall249 to %Richard*
    %vtable250 = getelementptr inbounds %Richard, %Richard* %Richard, i32 0, i32 0
    store %Richard_vtable* @Richard_vtable_data, %Richard_vtable** %vtable250, align 8
    %temp251 = alloca %Richard*, align 8
    %num_permitted252 = getelementptr inbounds %Richard, %Richard* %Richard, i32 0, i32 1
    store i32 1, i32* %num_permitted252, align 4
    %permit_list253 = getelementptr inbounds %Richard, %Richard* %Richard, i32 0, i32 2
    store [1 x i8*] [i8* getelementptr inbounds ([8 x i8],
        [8 x i8]* @temp_str.132, i32 0, i32 0)], [1 x i8*]* %permit_list253, align 8
    store %Richard* %Richard, %Richard** %temp251, align 8
    %r = alloca %Richard*, align 8
    %temp254 = load %Richard*, %Richard** %temp251, align 8
    store %Richard* %temp254, %Richard** %r, align 8
    %temp255 = load %Richard*, %Richard** %r, align 8
    %vtable256 = getelementptr inbounds %Richard, %Richard* %temp255, i32 0, i32 0
    %vtable257 = load %Richard_vtable*, %Richard_vtable** %vtable256, align 8
    %fun_to_call258 = getelementptr inbounds %Richard_vtable,
        %Richard_vtable* %vtable257, i32 0, i32 0
    %function259 = load void (%Animal**, i8*)*,
        void (%Animal**, i8*)** %fun_to_call258, align 8
    %arg_cast260 = bitcast %Richard** %r to %Animal**
    call void %function259(%Animal** %arg_cast260,
        i8* getelementptr inbounds ([9 x i8], [9 x i8]* @global_str.133, i32 0, i32 0))
    %temp261 = load %Richard*, %Richard** %r, align 8
    %vtable262 = getelementptr inbounds %Richard, %Richard* %temp261, i32 0, i32 0
    %vtable263 = load %Richard_vtable*, %Richard_vtable** %vtable262, align 8
    %fun_to_call264 = getelementptr inbounds %Richard_vtable,
        %Richard_vtable* %vtable263, i32 0, i32 1
    %function265 = load void (%Animal**, i8*)*,
        void (%Animal**, i8*)** %fun_to_call264, align 8
    %arg_cast266 = bitcast %Richard** %r to %Animal**
    call void %function265(%Animal** %arg_cast266,
```

```
1270        i8* getelementptr inbounds ([8 x i8], [8 x i8]* @global_str.134, i32 0, i32 0))
1271      %temp267 = load %Richard*, %Richard** %r, align 8
1272      %vtable268 = getelementptr inbounds %Richard, %Richard* %temp267, i32 0, i32 0
1273      %vtable269 = load %Richard_vtable*, %Richard_vtable** %vtable268, align 8
1274      %fun_to_call270 = getelementptr inbounds %Richard_vtable,
1275          %Richard_vtable* %vtable269, i32 0, i32 2
1276      %function271 = load void (%Animal**, i32)*,
1277          void (%Animal**, i32)** %fun_to_call270, align 8
1278      %arg_cast272 = bitcast %Richard** %r to %Animal**
1279      call void %function271(%Animal** %arg_cast272, i32 2)
1280      %temp273 = load %Richard*, %Richard** %r, align 8
1281      %vtable274 = getelementptr inbounds %Richard, %Richard* %temp273, i32 0, i32 0
1282      %vtable275 = load %Richard_vtable*, %Richard_vtable** %vtable274, align 8
1283      %fun_to_call276 = getelementptr inbounds %Richard_vtable,
1284          %Richard_vtable* %vtable275, i32 0, i32 3
1285      %function277 = load void (%Animal**, i1)*,
1286          void (%Animal**, i1)** %fun_to_call276, align 8
1287      %arg_cast278 = bitcast %Richard** %r to %Animal**
1288      call void %function277(%Animal** %arg_cast278, i1 false)
1289      %temp279 = load %Richard*, %Richard** %r, align 8
1290      %vtable280 = getelementptr inbounds %Richard, %Richard* %temp279, i32 0, i32 0
1291      %vtable281 = load %Richard_vtable*, %Richard_vtable** %vtable280, align 8
1292      %fun_to_call282 = getelementptr inbounds %Richard_vtable,
1293          %Richard_vtable* %vtable281, i32 0, i32 4
1294      %function283 = load void (%Animal**, i1)*,
1295          void (%Animal**, i1)** %fun_to_call282, align 8
1296      %arg_cast284 = bitcast %Richard** %r to %Animal**
1297      call void %function283(%Animal** %arg_cast284, i1 false)
1298      %cast_assign285 = bitcast %Richard** %r to %Animal**
1299      %temp286 = load %Animal*, %Animal** %cast_assign285, align 8
1300      store %Animal* %temp286, %Animal** %a, align 8
1301      br label %merge208
1302    }
1303
1304    define i1 @AnimalGame_makeGuess(%AnimalGame** %0, %Animal** %1, i32 %2) {
1305    entry:
1306      %temp = alloca %AnimalGame*, align 8
1307      %temp1 = load %AnimalGame*, %AnimalGame** %0, align 8
1308      store %AnimalGame* %temp1, %AnimalGame** %temp, align 8
1309      %temp2 = alloca %Animal*, align 8
1310      %temp3 = load %Animal*, %Animal** %1, align 8
1311      store %Animal* %temp3, %Animal** %temp2, align 8
1312      %count = alloca i32, align 4
1313      store i32 %2, i32* %count, align 4
1314      %input = alloca i8*, align 8
1315      %printf = call i32 (i8*, ...) @printf(
1316          i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.135, i32 0, i32 0),
1317          i8* getelementptr inbounds ([26 x i8], [26 x i8]* @global_str.138, i32 0, i32 0))
1318      %readaline = call i8** (...) @readaline()
```

```
1319      %get_temp = load i8*, i8** %readaline, align 8
1320      store i8* %get_temp, i8** %input, align 8
1321      %temp4 = load %Animal*, %Animal** %temp2, align 8
1322      %vtable = getelementptr inbounds %Animal, %Animal* %temp4, i32 0, i32 0
1323      %vtable5 = load %Animal_vtable*, %Animal_vtable** %vtable, align 8
1324      %fun_to_call = getelementptr inbounds %Animal_vtable,
1325          %Animal_vtable* %vtable5, i32 0, i32 6
1326      %function = load i8* (%Animal**)*, i8* (%Animal**)** %fun_to_call, align 8
1327      %getType_result = call i8* %function(%Animal** %temp2)
1328      %input6 = load i8*, i8** %input, align 8
1329      %streq = call i1 (i8*, i8*, ...) @streq(i8* %input6, i8* %getType_result)
1330      %tmp = xor i1 %streq, true
1331      br i1 %tmp, label %then, label %else24
1332
1333    merge:                                            ; preds = %else24
1334      %printf25 = call i32 (i8*, ...) @printf(
1335          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.136, i32 0, i32 0),
1336          i8* getelementptr inbounds ([62 x i8], [62 x i8]* @global_str.143, i32 0, i32 0))
1337      ret i1 true
1338
1339    then:                                             ; preds = %entry
1340      %printf7 = call i32 (i8*, ...) @printf(
1341          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.136, i32 0, i32 0),
1342          i8* getelementptr inbounds ([15 x i8], [15 x i8]* @global_str.139, i32 0, i32 0))
1343      %count8 = load i32, i32* %count, align 4
1344      %tmp9 = add i32 %count8, 1
1345      store i32 %tmp9, i32* %count, align 4
1346      %count10 = load i32, i32* %count, align 4
1347      %tmp11 = icmp eq i32 %count10, 3
1348      br i1 %tmp11, label %then13, label %else
1349
1350    merge12:                                          ; preds = %else, %then13
1351      ret i1 false
1352
1353    then13:                                           ; preds = %then
1354      %printf14 = call i32 (i8*, ...) @printf(
1355          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.136, i32 0, i32 0),
1356          i8* getelementptr inbounds ([35 x i8], [35 x i8]* @global_str.140, i32 0, i32 0))
1357      %printf15 = call i32 (i8*, ...) @printf(
1358          i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.135, i32 0, i32 0),
1359          i8* getelementptr inbounds ([24 x i8], [24 x i8]* @global_str.141, i32 0, i32 0))
1360      %temp16 = load %Animal*, %Animal** %temp2, align 8
1361      %vtable17 = getelementptr inbounds %Animal, %Animal* %temp16, i32 0, i32 0
1362      %vtable18 = load %Animal_vtable*, %Animal_vtable** %vtable17, align 8
1363      %fun_to_call19 = getelementptr inbounds %Animal_vtable, %Animal_vtable* %vtable18, i32 0, i32 6
1364      %function20 = load i8* (%Animal**)*, i8* (%Animal**)** %fun_to_call19, align 8
1365      %getType_result21 = call i8* %function20(%Animal** %temp2)
1366      %printf22 = call i32 (i8*, ...) @printf(
1367          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.136, i32 0, i32 0),
```

```
1368            i8* %getType_result21)
1369      %printf23 = call i32 (i8*, ...) @printf(
1370            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.136, i32 0, i32 0),
1371            i8* getelementptr inbounds ([29 x i8], [29 x i8]* @global_str.142, i32 0, i32 0))
1372      br label %merge12
1373
1374    else:                                             ; preds = %then
1375      br label %merge12
1376
1377    else24:                                           ; preds = %entry
1378      br label %merge
1379    }
1380
1381    define void @AnimalGame_giveHint(%AnimalGame** %0, %Animal** %1) {
1382    entry:
1383      %temp = alloca %AnimalGame*, align 8
1384      %temp1 = load %AnimalGame*, %AnimalGame** %0, align 8
1385      store %AnimalGame* %temp1, %AnimalGame** %temp, align 8
1386      %temp2 = alloca %Animal*, align 8
1387      %temp3 = load %Animal*, %Animal** %1, align 8
1388      store %Animal* %temp3, %Animal** %temp2, align 8
1389      %printf = call i32 (i8*, ...) @printf(
1390            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1391            i8* getelementptr inbounds ([38 x i8], [38 x i8]* @global_str.147, i32 0, i32 0))
1392      %printf4 = call i32 (i8*, ...) @printf(
1393            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1394            i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.148, i32 0, i32 0))
1395      %printf5 = call i32 (i8*, ...) @printf(
1396            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1397            i8* getelementptr inbounds ([5 x i8], [5 x i8]* @global_str.149, i32 0, i32 0))
1398      %printf6 = call i32 (i8*, ...) @printf(
1399            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1400            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.150, i32 0, i32 0))
1401      %printf7 = call i32 (i8*, ...) @printf(
1402            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1403            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.151, i32 0, i32 0))
1404      %printf8 = call i32 (i8*, ...) @printf(
1405            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1406            i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.152, i32 0, i32 0))
1407      %printf9 = call i32 (i8*, ...) @printf(
1408            i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1409            i8* getelementptr inbounds ([9 x i8], [9 x i8]* @global_str.153, i32 0, i32 0))
1410      %readaline = call i8** (...) @readaline()
1411      %get_temp = load i8*, i8** %readaline, align 8
1412      %input = alloca i8*, align 8
1413      store i8* %get_temp, i8** %input, align 8
1414      %input10 = load i8*, i8** %input, align 8
1415      %streq = call i1 (i8*, i8*, ...) @streq(i8* %input10,
1416            i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.154, i32 0, i32 0))
```

```
1417      br i1 %streq, label %then, label %else
1418
1419    merge:                                            ; preds = %merge17, %then
1420      ret void
1421
1422    then:                                             ; preds = %entry
1423      %printf11 = call i32 (i8*, ...) @printf(
1424          i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.144, i32 0, i32 0),
1425          i8* getelementptr inbounds ([34 x i8], [34 x i8]* @global_str.155, i32 0, i32 0))
1426      %temp12 = load %Animal*, %Animal** %temp2, align 8
1427      %vtable = getelementptr inbounds %Animal, %Animal* %temp12, i32 0, i32 0
1428      %vtable13 = load %Animal_vtable*, %Animal_vtable** %vtable, align 8
1429      %fun_to_call = getelementptr inbounds %Animal_vtable, %Animal_vtable* %vtable13, i32 0, i32 5
1430      %function = load i8* (%Animal**)*, i8* (%Animal**)** %fun_to_call, align 8
1431      %getClass_result = call i8* %function(%Animal** %temp2)
1432      %printf14 = call i32 (i8*, ...) @printf(
1433          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0), i8* %getClass_result)
1434      br label %merge
1435
1436    else:                                             ; preds = %entry
1437      %input15 = load i8*, i8** %input, align 8
1438      %streq16 = call i1 (i8*, i8*, ...) @streq(i8* %input15,
1439          i8* getelementptr inbounds ([5 x i8], [5 x i8]* @global_str.156, i32 0, i32 0))
1440      br i1 %streq16, label %then18, label %else27
1441
1442    merge17:                                          ; preds = %merge30, %then18
1443      br label %merge
1444
1445    then18:                                           ; preds = %else
1446      %printf19 = call i32 (i8*, ...) @printf(
1447          i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.144, i32 0, i32 0),
1448          i8* getelementptr inbounds ([16 x i8], [16 x i8]* @global_str.157, i32 0, i32 0))
1449      %temp20 = load %Animal*, %Animal** %temp2, align 8
1450      %vtable21 = getelementptr inbounds %Animal, %Animal* %temp20, i32 0, i32 0
1451      %vtable22 = load %Animal_vtable*, %Animal_vtable** %vtable21, align 8
1452      %fun_to_call23 = getelementptr inbounds %Animal_vtable,
1453          %Animal_vtable* %vtable22, i32 0, i32 7
1454      %function24 = load i32 (%Animal**)*, i32 (%Animal**)** %fun_to_call23, align 8
1455      %numLegs_result = call i32 %function24(%Animal** %temp2)
1456      %printf25 = call i32 (i8*, ...) @printf(
1457          i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.146, i32 0, i32 0),
1458          i32 %numLegs_result)
1459      %printf26 = call i32 (i8*, ...) @printf(
1460          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1461          i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.158, i32 0, i32 0))
1462      br label %merge17
1463
1464    else27:                                           ; preds = %else
1465      %input28 = load i8*, i8** %input, align 8
```

```
1466      %streq29 = call i1 (i8*, i8*, ...) @streq(i8* %input28,
1467          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.159, i32 0, i32 0))
1468      br i1 %streq29, label %then31, label %else42
1469
1470    merge30:                                          ; preds = %merge45, %merge37
1471      br label %merge17
1472
1473    then31:                                           ; preds = %else27
1474      %temp32 = load %Animal*, %Animal** %temp2, align 8
1475      %vtable33 = getelementptr inbounds %Animal, %Animal* %temp32, i32 0, i32 0
1476      %vtable34 = load %Animal_vtable*, %Animal_vtable** %vtable33, align 8
1477      %fun_to_call35 = getelementptr inbounds %Animal_vtable,
1478          %Animal_vtable* %vtable34, i32 0, i32 8
1479      %function36 = load i1 (%Animal**)*, i1 (%Animal**)** %fun_to_call35, align 8
1480      %hasFur_result = call i1 %function36(%Animal** %temp2)
1481      br i1 %hasFur_result, label %then38, label %else40
1482
1483    merge37:                                          ; preds = %else40, %then38
1484      br label %merge30
1485
1486    then38:                                           ; preds = %then31
1487      %printf39 = call i32 (i8*, ...) @printf(
1488          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1489          i8* getelementptr inbounds ([25 x i8], [25 x i8]* @global_str.160, i32 0, i32 0))
1490      br label %merge37
1491
1492    else40:                                           ; preds = %then31
1493      %printf41 = call i32 (i8*, ...) @printf(
1494          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
1495          i8* getelementptr inbounds ([29 x i8], [29 x i8]* @global_str.161, i32 0, i32 0))
1496      br label %merge37
1497
1498    else42:                                           ; preds = %else27
1499      %input43 = load i8*, i8** %input, align 8
1500      %streq44 = call i1 (i8*, i8*, ...) @streq(
1501          i8* %input43,
1502          i8* getelementptr inbounds ([4 x i8], [4 x i8]* @global_str.162, i32 0, i32 0))
1503      br i1 %streq44, label %then46, label %else57
1504
1505    merge45:                                          ; preds = %merge60, %merge52
1506      br label %merge30
1507
1508    then46:                                           ; preds = %else42
1509      %temp47 = load %Animal*, %Animal** %temp2, align 8
1510      %vtable48 = getelementptr inbounds %Animal, %Animal* %temp47, i32 0, i32 0
1511      %vtable49 = load %Animal_vtable*, %Animal_vtable** %vtable48, align 8
1512      %fun_to_call50 = getelementptr inbounds %Animal_vtable,
1513          %Animal_vtable* %vtable49, i32 0, i32 9
1514      %function51 = load i1 (%Animal**)*, i1 (%Animal**)** %fun_to_call50, align 8
```

```
  %canFly_result = call i1 %function51(%Animal** %temp2)
  br i1 %canFly_result, label %then53, label %else55

merge52:                                          ; preds = %else55, %then53
  br label %merge45

then53:                                           ; preds = %then46
  %printf54 = call i32 (i8*, ...) @printf(
      i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
      i8* getelementptr inbounds ([19 x i8], [19 x i8]* @global_str.163, i32 0, i32 0))
  br label %merge52

else55:                                           ; preds = %then46
  %printf56 = call i32 (i8*, ...) @printf(
      i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
      i8* getelementptr inbounds ([22 x i8], [22 x i8]* @global_str.164, i32 0, i32 0))
  br label %merge52

else57:                                           ; preds = %else42
  %input58 = load i8*, i8** %input, align 8
  %streq59 = call i1 (i8*, i8*, ...) @streq(i8* %input58,
      i8* getelementptr inbounds ([6 x i8], [6 x i8]* @global_str.165, i32 0, i32 0))
  br i1 %streq59, label %then61, label %else67

merge60:                                          ; preds = %merge70, %then61
  br label %merge45

then61:                                           ; preds = %else57
  %temp62 = load %Animal*, %Animal** %temp2, align 8
  %vtable63 = getelementptr inbounds %Animal, %Animal* %temp62, i32 0, i32 0
  %vtable64 = load %Animal_vtable*, %Animal_vtable** %vtable63, align 8
  %fun_to_call65 = getelementptr inbounds %Animal_vtable,
      %Animal_vtable* %vtable64, i32 0, i32 10
  %function66 = load void (%Animal**)*, void (%Animal**)** %fun_to_call65, align 8
  call void %function66(%Animal** %temp2)
  br label %merge60

else67:                                           ; preds = %else57
  %input68 = load i8*, i8** %input, align 8
  %streq69 = call i1 (i8*, i8*, ...) @streq(i8* %input68,
      i8* getelementptr inbounds ([9 x i8], [9 x i8]* @global_str.166, i32 0, i32 0))
  br i1 %streq69, label %then71, label %else77

merge70:                                          ; preds = %else77, %then71
  br label %merge60

then71:                                           ; preds = %else67
  %temp72 = load %Animal*, %Animal** %temp2, align 8
  %vtable73 = getelementptr inbounds %Animal, %Animal* %temp72, i32 0, i32 0
```

```
  %vtable74 = load %Animal_vtable*, %Animal_vtable** %vtable73, align 8
  %fun_to_call75 = getelementptr inbounds %Animal_vtable,
      %Animal_vtable* %vtable74, i32 0, i32 11
  %function76 = load void (%Animal**)*, void (%Animal**)** %fun_to_call75, align 8
  call void %function76(%Animal** %temp2)
  br label %merge70

else77:                                           ; preds = %else67
  %printf78 = call i32 (i8*, ...) @printf(
      i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt.145, i32 0, i32 0),
      i8* getelementptr inbounds ([35 x i8], [35 x i8]* @global_str.167, i32 0, i32 0))
  br label %merge70
}

declare noalias i8* @malloc(i32)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint
      (%AnimalGame* getelementptr (%AnimalGame, %AnimalGame* null, i32 1) to i32))
  %AnimalGame = bitcast i8* %malloccall to %AnimalGame*
  %vtable = getelementptr inbounds %AnimalGame,
      %AnimalGame* %AnimalGame, i32 0, i32 0
  store %AnimalGame_vtable* @AnimalGame_vtable_data,
      %AnimalGame_vtable** %vtable, align 8
  %temp = alloca %AnimalGame*, align 8
  %num_permitted = getelementptr inbounds %AnimalGame,
      %AnimalGame* %AnimalGame, i32 0, i32 1
  store i32 1, i32* %num_permitted, align 4
  %permit_list = getelementptr inbounds %AnimalGame,
      %AnimalGame* %AnimalGame, i32 0, i32 2
  store [1 x i8*] [i8* getelementptr inbounds ([11 x i8],
      [11 x i8]* @temp_str.171, i32 0, i32 0)], [1 x i8*]* %permit_list, align 8
  store %AnimalGame* %AnimalGame, %AnimalGame** %temp, align 8
  %ag = alloca %AnimalGame*, align 8
  %temp1 = load %AnimalGame*, %AnimalGame** %temp, align 8
  store %AnimalGame* %temp1, %AnimalGame** %ag, align 8
  %temp2 = load %AnimalGame*, %AnimalGame** %ag, align 8
  %vtable3 = getelementptr inbounds %AnimalGame,
      %AnimalGame* %temp2, i32 0, i32 0
  %vtable4 = load %AnimalGame_vtable*, %AnimalGame_vtable** %vtable3, align 8
  %fun_to_call = getelementptr inbounds %AnimalGame_vtable,
      %AnimalGame_vtable* %vtable4, i32 0, i32 0
  %function = load void (%AnimalGame**)*,
      void (%AnimalGame**)** %fun_to_call, align 8
  call void %function(%AnimalGame** %ag)
  ret i32 0
}
```

13.2.2 Farm Tour

Source file

```
.~*
 ~
 ~      FarmTour.iris
 ~      by: Valerie Zhang
 ~
 ~
 ~      This is an interactive program touring a farm. The program allows
 ~      the user to see 4 different farm animals with their farmer guide
 ~      Farmer Bob. The user can choose to try to call over an animal
 ~      or not. If yes, the animal may ignore the user, if not, Farmer
 ~      Bob will call them over.
 ~
*~.

.. Class definitions of all the animals and farmer
class Animal (Main) {
    public:
        void ignore() {
            Olympus.println("...");
        }

    permit:
        void noise() {
            Olympus.println("Noise");
        }
}

class Farmer of Animal (Main) {
    public:
        .. This function allows the farmer to "call" over the animal
        .. by calling each animal's permit method "noise". The Farmer
        .. class is a permitted class for all the different animal classes
        void callAnimal(Animal a) {
            a.noise();
        }
}

class Cow of Animal (Farmer) {
    permit:
        void noise() {
            Olympus.println("Moooooooooo!");
        }
}

class Pig of Animal (Farmer) {
    permit:
        void noise() {
            Olympus.println("Oink oink!");
        }
```

```
}

class Duck of Animal (Farmer, FarmTour) {
    permit:
        void noise() {
            Olympus.println("Quack!");
        }
}

class Sheep of Animal (Farmer) {
    permit:
        void noise() {
            Olympus.println("Baaaaaaah!");
        }
}

.. class that contains methods to run the farm tour
class FarmTour (Main) {
    public:
        .. runs the tour with the command loop
        void run() {
            Farmer bob = new Farmer();
            string input = "";

            .. printing startup
            Olympus.println("Welcome to Farmer Bob's farm! Let's see what animals we can see.");
            Olympus.println("There's Farmer Bob! Farmer Bob asks: 'What animal would you like to see?'");

            while (true) {
                Olympus.print("Please enter p for pig, co for cow, d for duck, s for sheep, or q to leave: ");
                input = Olympus.readaline();

                if (input == "p") {
                    seePig(bob);
                } else if (input == "co") {
                    seeCow(bob);
                } else if (input == "d") {
                    seeDuck(bob);
                } else if (input == "s") {
                    seeSheep(bob);
                } else if (input == "q") {
                    Olympus.println("Farmer Bob says: 'Thanks for coming to visit!'");
                    return;
                } else {
                    Olympus.println("Farmer Bob says: 'We don't have that animal here, maybe next time!'");
                }

                Olympus.println("Farmer Bob asks: 'What animal would you like to see next?'");
            }
```

```
        Olympus.println("Farmer Bob says: 'That's all the animals we have here, thanks for coming to visit!'");
    }

    .. the following functions perform the operations for "seeing" an animal
    void seePig(Farmer f) {
        Pig p = new Pig();

        string input = "";
        Olympus.println("Farmer Bob says: 'Sure thing! Let's go say hi to my friend Wilbur.'");
        Olympus.println("You approach the pig pen and see a large pig sitting in the mud.");

        Olympus.print("Will you try calling Wilbur? y/n: ");
        input = Olympus.readaline();

        if (input == "y") {
            Olympus.println("You try calling Wilbur");
            p.ignore();
            Olympus.println("It seems Wilbur ignored you.");
        } else {
            Olympus.println("Farmer Bob says: 'Let me call Wilbur over: Wilbur!'");
            f.callAnimal(p);
            Olympus.println("Wilbur walks over and you give him a pat on the nose.");
        }
    }

    void seeCow(Farmer f) {
        Cow c = new Cow();

        string input = "";
        Olympus.println("Farmer Bob says: 'Sure thing! Let's go say hi to my friend Bessie.'");
        Olympus.println("You approach the left side of the field and see a large cow grazing.");

        Olympus.print("Will you try calling Bessie? y/n: ");
        input = Olympus.readaline();

        if (input == "y") {
            Olympus.println("You try calling Bessie");
            c.ignore();
            Olympus.println("It seems Bessie ignored you.");
        } else {
            Olympus.println("Farmer Bob says: 'Let me call Bessie over: Bessie!'");
            f.callAnimal(c);
            Olympus.println("Bessie walks over and you give her a pat on the nose.");
        }

    }

    void seeDuck(Farmer f) {
```

```
        void seeDuck(Farmer f) {
            Duck d = new Duck();

            string input = "";
            Olympus.println("Farmer Bob says: 'Sure thing! Let's go say hi to my friend Waddles.'");
            Olympus.println("You approach a pond and see a duck swimming around.");

            Olympus.print("Will you try calling Waddles? y/n: ");
            input = Olympus.readaline();

            if (input == "y") {
                Olympus.println("You try calling Waddles");
            } else {
                Olympus.println("Farmer Bob says: 'Let me call Waddles over: Waddles!'");
            }
            d.noise();
            Olympus.println("Waddles waddles over and you give her a pat");
        }

        void seeSheep(Farmer f) {
            Sheep s = new Sheep();

            string input = "";
            Olympus.println("Farmer Bob says: 'Sure thing! Let's go say hi to my friend Skittles.'");
            Olympus.println("You approach the right side of the field and see a sheep grazing on grass.");

            Olympus.print("Will you try calling Skittles? y/n: ");
            input = Olympus.readaline();

            if (input == "y") {
                Olympus.println("You try calling Skittles");
                s.ignore();
                Olympus.println("It seems Skittles ignored you.");
            } else {
                Olympus.println("Farmer Bob says: 'Let me call Skittles over: Skittles!'");
                f.callAnimal(s);
                Olympus.println("Skittles approaches and give her a pet.");
            }
        }
}

class Main () {
    public:
        int univ main() {
            FarmTour ft = new FarmTour();
            ft.run();
            return 0;
        }
}
```

LLVM file

Note: lines that are tabbed 4 spaces have been done so for photo formatting. They originally were all on one line

```
; ModuleID = 'Iris'
source_filename = "Iris"

%Object_vtable = type {}
%Animal_vtable = type { void (%Animal**)*, void (%Animal**)* }
%Animal = type { %Animal_vtable*, i32, [2 x i8*] }
%Farmer_vtable = type { void (%Animal**)*, void (%Animal**)*,
    void (%Farmer**, %Animal**)* }
%Farmer = type { %Farmer_vtable*, i32, [2 x i8*] }
%Cow_vtable = type { void (%Animal**)*, void (%Cow**)* }
%Cow = type { %Cow_vtable*, i32, [2 x i8*] }
%Pig_vtable = type { void (%Animal**)*, void (%Pig**)* }
%Pig = type { %Pig_vtable*, i32, [2 x i8*] }
%Duck_vtable = type { void (%Animal**)*, void (%Duck**)* }
%Duck = type { %Duck_vtable*, i32, [3 x i8*] }
%Sheep_vtable = type { void (%Animal**)*, void (%Sheep**)* }
%Sheep = type { %Sheep_vtable*, i32, [2 x i8*] }
%FarmTour_vtable = type { void (%FarmTour**)*, void (%FarmTour**, %Farmer**)*,
    void (%FarmTour**, %Farmer**)*, void (%FarmTour**, %Farmer**)*,
    void (%FarmTour**, %Farmer**)* }
%FarmTour = type { %FarmTour_vtable*, i32, [2 x i8*] }
%Main_vtable = type { i32 ()* }

@Object_vtable_data = global %Object_vtable zeroinitializer
@fmt = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.2 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str = private unnamed_addr constant [4 x i8] c"...\00", align 1
@fmt.3 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.5 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.6 = private unnamed_addr constant [6 x i8] c"Noise\00", align 1
@Animal_vtable_data = global %Animal_vtable { void (%Animal**)* @Animal_ignore,
    void (%Animal**)* @Animal_noise }
@fmt.7 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.8 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.9 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@class_location = private unnamed_addr constant [7 x i8] c"Farmer\00", align 1
@Farmer_vtable_data = global %Farmer_vtable { void (%Animal**)* @Animal_ignore,
    void (%Animal**)* @Animal_noise,
    void (%Farmer**, %Animal**)* @Farmer_callAnimal }
@fmt.10 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.11 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.12 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.13 =
    private unnamed_addr constant [12 x i8] c"Mooooooooo!\00", align 1
@Cow_vtable_data = global %Cow_vtable { void (%Animal**)* @Animal_ignore,
    void (%Cow**)* @Cow_noise }
@fmt.14 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
```

```
@fmt.15 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.16 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.17 =
    private unnamed_addr constant [11 x i8] c"Oink oink!\00", align 1
@Pig_vtable_data = global %Pig_vtable { void (%Animal**)* @Animal_ignore,
    void (%Pig**)* @Pig_noise }
@fmt.18 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.19 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.20 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.21 = private unnamed_addr constant [7 x i8] c"Quack!\00", align 1
@Duck_vtable_data = global %Duck_vtable { void (%Animal**)* @Animal_ignore,
    void (%Duck**)* @Duck_noise }
@fmt.22 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.23 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.24 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@global_str.25 =
    private unnamed_addr constant [11 x i8] c"Baaaaaaah!\00", align 1
@Sheep_vtable_data = global %Sheep_vtable { void (%Animal**)* @Animal_ignore,
    void (%Sheep**)* @Sheep_noise }
@fmt.26 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.27 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.28 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@temp_str = private unnamed_addr constant [7 x i8] c"Farmer\00", align 1
@temp_str.29 = private unnamed_addr constant [5 x i8] c"Main\00", align 1
@global_str.30 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@global_str.31 =
    private unnamed_addr constant [65 x i8]
    c"Welcome to Farmer Bob's farm! Let's see what animals we can see.\00",
    align 1
@global_str.32 = private unnamed_addr constant [74 x i8]
    c"There's Farmer Bob! Farmer Bob asks:
    'What animal would you like to see?'\00", align 1
@global_str.33 = private unnamed_addr constant [77 x i8]
    c"Please enter p for pig, co for cow, d for duck,
    s for sheep, or q to leave: \00", align 1
@global_str.34 = private unnamed_addr constant [2 x i8] c"p\00", align 1
@global_str.35 = private unnamed_addr constant [2 x i8] c"p\00", align 1
@global_str.36 = private unnamed_addr constant [3 x i8] c"co\00", align 1
@global_str.37 = private unnamed_addr constant [3 x i8] c"co\00", align 1
@global_str.38 = private unnamed_addr constant [2 x i8] c"d\00", align 1
@global_str.39 = private unnamed_addr constant [2 x i8] c"d\00", align 1
@global_str.40 = private unnamed_addr constant [2 x i8] c"s\00", align 1
@global_str.41 = private unnamed_addr constant [2 x i8] c"s\00", align 1
@global_str.42 = private unnamed_addr constant [2 x i8] c"q\00", align 1
@global_str.43 = private unnamed_addr constant [2 x i8] c"q\00", align 1
@global_str.44 = private unnamed_addr constant [47 x i8] c"Farmer Bob says:
    'Thanks for coming to visit!'\00", align 1
@global_str.45 = private unnamed_addr constant [68 x i8] c"Farmer Bob says:
    'We don't have that animal here, maybe next time!'\00", align 1
```

```
@global_str.46 = private unnamed_addr constant [59 x i8] c"Farmer Bob asks:
    'What animal would you like to see next?'\00", align 1
@global_str.47 = private unnamed_addr constant [84 x i8] c"Farmer Bob says:
    'That's all the animals we have here, thanks for coming to visit!'\00",
    align 1
@fmt.48 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.49 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.50 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@temp_str.51 = private unnamed_addr constant [4 x i8] c"Pig\00", align 1
@temp_str.52 = private unnamed_addr constant [7 x i8] c"Farmer\00", align 1
@global_str.53 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@global_str.54 = private unnamed_addr constant [68 x i8] c"Farmer Bob says:
    'Sure thing! Let's go say hi to my friend Wilbur.'\00", align 1
@global_str.55 = private unnamed_addr constant [65 x i8]
    c"You approach the pig pen and see a large pig sitting in the mud.\00",
    align 1
@global_str.56 = private unnamed_addr constant [35 x i8]
    c"Will you try calling Wilbur? y/n: \00", align 1
@global_str.57 = private unnamed_addr constant [2 x i8] c"y\00", align 1
@global_str.58 = private unnamed_addr constant [2 x i8] c"y\00", align 1
@global_str.59 = private unnamed_addr constant [23 x i8]
    c"You try calling Wilbur\00", align 1
@global_str.60 = private unnamed_addr constant [29 x i8]
    c"It seems Wilbur ignored you.\00", align 1
@global_str.61 = private unnamed_addr constant [52 x i8]
    c"Farmer Bob says: 'Let me call Wilbur over: Wilbur!'\00", align 1
@global_str.62 = private unnamed_addr constant [54 x i8]
    c"Wilbur walks over and you give him a pat on the nose.\00", align 1
@fmt.63 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.64 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.65 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@temp_str.66 = private unnamed_addr constant [4 x i8] c"Cow\00", align 1
@temp_str.67 = private unnamed_addr constant [7 x i8] c"Farmer\00", align 1
@global_str.68 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@global_str.69 = private unnamed_addr constant [68 x i8]
    c"Farmer Bob says: 'Sure thing! Let's go say hi to my friend Bessie.'\00",
    align 1
@global_str.70 = private unnamed_addr constant [69 x i8]
    c"You approach the left side of the field and see a large cow grazing.\00",
    align 1
@global_str.71 = private unnamed_addr constant [35 x i8]
    c"Will you try calling Bessie? y/n: \00", align 1
@global_str.72 = private unnamed_addr constant [2 x i8] c"y\00", align 1
@global_str.73 = private unnamed_addr constant [2 x i8] c"y\00", align 1
@global_str.74 = private unnamed_addr constant [23 x i8]
    c"You try calling Bessie\00", align 1
@global_str.75 = private unnamed_addr constant [29 x i8]
    c"It seems Bessie ignored you.\00", align 1
@global_str.76 = private unnamed_addr constant [52 x i8]
```

```
148         c"Farmer Bob says: 'Let me call Bessie over: Bessie!'\00", align 1
149     @global_str.77 = private unnamed_addr constant [54 x i8]
150         c"Bessie walks over and you give her a pat on the nose.\00", align 1
151     @fmt.78 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
152     @fmt.79 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
153     @fmt.80 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
154     @temp_str.81 = private unnamed_addr constant [5 x i8] c"Duck\00", align 1
155     @temp_str.82 = private unnamed_addr constant [7 x i8] c"Farmer\00", align 1
156     @temp_str.83 = private unnamed_addr constant [9 x i8] c"FarmTour\00", align 1
157     @global_str.84 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
158     @global_str.85 = private unnamed_addr constant [69 x i8] c"Farmer Bob says:
159         'Sure thing! Let's go say hi to my friend Waddles.'\00", align 1
160     @global_str.86 = private unnamed_addr constant [52 x i8]
161         c"You approach a pond and see a duck swimming around.\00", align 1
162     @global_str.87 = private unnamed_addr constant [36 x i8]
163         c"Will you try calling Waddles? y/n: \00", align 1
164     @class_location.88 = private unnamed_addr constant [9 x i8]
165         c"FarmTour\00", align 1
166     @global_str.89 = private unnamed_addr constant [2 x i8] c"y\00", align 1
167     @global_str.90 = private unnamed_addr constant [2 x i8] c"y\00", align 1
168     @global_str.91 = private unnamed_addr constant [24 x i8]
169         c"You try calling Waddles\00", align 1
170     @class_location.92 = private unnamed_addr constant [9 x i8] c"FarmTour\00",
171         align 1
172     @global_str.93 = private unnamed_addr constant [48 x i8]
173         c"Farmer Bob says: 'Look at that, she came over!'\00", align 1
174     @global_str.94 = private unnamed_addr constant [54 x i8]
175         c"Farmer Bob says: 'Let me call Waddles over: Waddles!'\00", align 1
176     @global_str.95 = private unnamed_addr constant [44 x i8]
177         c"Waddles waddles over and you give her a pat\00", align 1
178     @fmt.96 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
179     @fmt.97 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
180     @fmt.98 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
181     @temp_str.99 = private unnamed_addr constant [6 x i8] c"Sheep\00", align 1
182     @temp_str.100 = private unnamed_addr constant [7 x i8] c"Farmer\00", align 1
183     @global_str.101 = private unnamed_addr constant [1 x i8] zeroinitializer,
184         align 1
185     @global_str.102 = private unnamed_addr constant [70 x i8]
186         c"Farmer Bob says: 'Sure thing! Let's go say hi to my friend Skittles.'\00",
187         align 1
188     @global_str.103 = private unnamed_addr constant [75 x i8]
189         c"You approach the right side of the field and see a
190         sheep grazing on grass.\00", align 1
191     @global_str.104 = private unnamed_addr constant [37 x i8]
192         c"Will you try calling Skittles? y/n: \00", align 1
193     @global_str.105 = private unnamed_addr constant [2 x i8] c"y\00", align 1
194     @global_str.106 = private unnamed_addr constant [2 x i8] c"y\00", align 1
195     @global_str.107 = private unnamed_addr constant [25 x i8]
196         c"You try calling Skittles\00", align 1
```

```
@global_str.108 = private unnamed_addr constant [31 x i8]
    c"It seems Skittles ignored you.\00", align 1
@global_str.109 = private unnamed_addr constant [56 x i8]
    c"Farmer Bob says: 'Let me call Skittles over: Skittles!'\00", align 1
@global_str.110 = private unnamed_addr constant [40 x i8]
    c"Skittles approaches and give her a pet.\00", align 1
@FarmTour_vtable_data =
    global %FarmTour_vtable { void (%FarmTour**)* @FarmTour_run,
    void (%FarmTour**, %Farmer**)* @FarmTour_seePig,
    void (%FarmTour**, %Farmer**)* @FarmTour_seeCow,
    void (%FarmTour**, %Farmer**)* @FarmTour_seeDuck,
    void (%FarmTour**, %Farmer**)* @FarmTour_seeSheep }
@fmt.111 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.112 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.113 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@temp_str.114 = private unnamed_addr constant [9 x i8] c"FarmTour\00", align 1
@temp_str.115 = private unnamed_addr constant [5 x i8] c"Main\00", align 1
@Main_vtable_data = global %Main_vtable { i32 ()* @main }

declare i32 @printf(i8*, ...)

declare i32 @printerr(i8*, ...)

declare i8** @readaline(...)

declare i1 @streq(i8*, i8*, ...)

declare void @class_permitted(i8*, i8**, i32, ...)

define void @Animal_ignore(%Animal** %0) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %printf = call i32 (i8*, ...)
      @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.1, i32 0, i32 0),
      i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @global_str, i32 0, i32 0))
  ret void
}

define void @Animal_noise(%Animal** %0) {
entry:
  %temp = alloca %Animal*, align 8
  %temp1 = load %Animal*, %Animal** %0, align 8
  store %Animal* %temp1, %Animal** %temp, align 8
  %printf = call i32 (i8*, ...)
      @printf(i8* getelementptr inbounds
```

```
        ([4 x i8], [4 x i8]* @fmt.4, i32 0, i32 0),
        i8* getelementptr inbounds ([6 x i8],
        [6 x i8]* @global_str.6, i32 0, i32 0))
  ret void
}

define void @Farmer_callAnimal(%Farmer** %0, %Animal** %1) {
entry:
  %temp = alloca %Farmer*, align 8
  %temp1 = load %Farmer*, %Farmer** %0, align 8
  store %Farmer* %temp1, %Farmer** %temp, align 8
  %temp2 = alloca %Animal*, align 8
  %temp3 = load %Animal*, %Animal** %1, align 8
  store %Animal* %temp3, %Animal** %temp2, align 8
  %temp4 = load %Animal*, %Animal** %temp2, align 8
  %permit_length_ptr = getelementptr inbounds %Animal,
    %Animal* %temp4, i32 0, i32 1
  %permit_length = load i32, i32* %permit_length_ptr, align 4
  %permit_list_ptr = getelementptr inbounds %Animal,
    %Animal* %temp4, i32 0, i32 2
  %str_ptr = getelementptr [2 x i8*], [2 x i8*]* %permit_list_ptr, i32 0, i32 0
  call void (i8*, i8**, i32, ...)
      @class_permitted(i8* getelementptr inbounds ([7 x i8],
      [7 x i8]* @class_location, i32 0, i32 0), i8** %str_ptr,
      i32 %permit_length)
  %vtable = getelementptr inbounds %Animal, %Animal* %temp4, i32 0, i32 0
  %vtable5 = load %Animal_vtable*, %Animal_vtable** %vtable, align 8
  %fun_to_call = getelementptr inbounds %Animal_vtable,
      %Animal_vtable* %vtable5, i32 0, i32 1
  %function = load void (%Animal**)*, void (%Animal**)** %fun_to_call, align 8
  call void %function(%Animal** %temp2)
  ret void
}

define void @Cow_noise(%Cow** %0) {
entry:
  %temp = alloca %Cow*, align 8
  %temp1 = load %Cow*, %Cow** %0, align 8
  store %Cow* %temp1, %Cow** %temp, align 8
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.11, i32 0, i32 0),
      i8* getelementptr inbounds ([12 x i8],
      [12 x i8]* @global_str.13, i32 0, i32 0))
  ret void
}

define void @Pig_noise(%Pig** %0) {
entry:
  %temp = alloca %Pig*, align 8
```

```
  %temp1 = load %Pig*, %Pig** %0, align 8
  store %Pig* %temp1, %Pig** %temp, align 8
  %printf = call i32 (i8*, ...)
      @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt.15, i32 0, i32 0),
      i8* getelementptr inbounds ([11 x i8],
      [11 x i8]* @global_str.17, i32 0, i32 0))
  ret void
}

define void @Duck_noise(%Duck** %0) {
entry:
  %temp = alloca %Duck*, align 8
  %temp1 = load %Duck*, %Duck** %0, align 8
  store %Duck* %temp1, %Duck** %temp, align 8
  %printf = call i32 (i8*, ...)
      @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt.19, i32 0, i32 0),
      i8* getelementptr inbounds
      ([7 x i8], [7 x i8]* @global_str.21, i32 0, i32 0))
  ret void
}

define void @Sheep_noise(%Sheep** %0) {
entry:
  %temp = alloca %Sheep*, align 8
  %temp1 = load %Sheep*, %Sheep** %0, align 8
  store %Sheep* %temp1, %Sheep** %temp, align 8
  %printf = call i32 (i8*, ...)
      @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.23, i32 0, i32 0),
      i8* getelementptr inbounds
      ([11 x i8], [11 x i8]* @global_str.25, i32 0, i32 0))
  ret void
}

define void @FarmTour_run(%FarmTour** %0) {
entry:
  %temp = alloca %FarmTour*, align 8
  %temp1 = load %FarmTour*, %FarmTour** %0, align 8
  store %FarmTour* %temp1, %FarmTour** %temp, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint
      (%Farmer* getelementptr (%Farmer, %Farmer* null, i32 1) to i32))
  %Farmer = bitcast i8* %malloccall to %Farmer*
  %vtable = getelementptr inbounds %Farmer, %Farmer* %Farmer, i32 0, i32 0
  store %Farmer_vtable* @Farmer_vtable_data, %Farmer_vtable** %vtable, align 8
  %temp2 = alloca %Farmer*, align 8
  %num_permitted = getelementptr inbounds %Farmer, %Farmer* %Farmer, i32 0, i32 1
  store i32 2, i32* %num_permitted, align 4
```

```
  %permit_list = getelementptr inbounds %Farmer, %Farmer* %Farmer, i32 0, i32 2
  store [2 x i8*] [i8* getelementptr inbounds ([7 x i8],
      [7 x i8]* @temp_str, i32 0, i32 0), i8* getelementptr inbounds
      ([5 x i8], [5 x i8]* @temp_str.29, i32 0, i32 0)],
      [2 x i8*]* %permit_list, align 8
  store %Farmer* %Farmer, %Farmer** %temp2, align 8
  %bob = alloca %Farmer*, align 8
  %temp3 = load %Farmer*, %Farmer** %temp2, align 8
  store %Farmer* %temp3, %Farmer** %bob, align 8
  %input = alloca i8*, align 8
  store i8* getelementptr inbounds ([1 x i8], [
      1 x i8]* @global_str.30, i32 0, i32 0), i8** %input, align 8
  %seen = alloca i32, align 4
  store i32 0, i32* %seen, align 4
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.27, i32 0, i32 0), i8* getelementptr inbounds
      ([65 x i8], [65 x i8]* @global_str.31, i32 0, i32 0))
  %printf4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.27, i32 0, i32 0), i8* getelementptr inbounds
      ([74 x i8], [74 x i8]* @global_str.32, i32 0, i32 0))
  br label %while

while:                                            ; preds = %merge61, %entry
  %seen65 = load i32, i32* %seen, align 4
  %tmp66 = icmp ne i32 %seen65, 4
  br i1 %tmp66, label %while_body, label %merge67

while_body:                                       ; preds = %while
  %printf5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([3 x i8], [3 x i8]* @fmt.26, i32 0, i32 0), i8* getelementptr inbounds
      ([77 x i8], [77 x i8]* @global_str.33, i32 0, i32 0))
  %readaline = call i8** (...) @readaline()
  %get_temp = load i8*, i8** %readaline, align 8
  store i8* %get_temp, i8** %input, align 8
  %input6 = load i8*, i8** %input, align 8
  %input7 = load i8*, i8** %input, align 8
  %streq = call i1 (i8*, i8*, ...) @streq(i8* %input7, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @global_str.35, i32 0, i32 0))
  br i1 %streq, label %then, label %else

merge:                                            ; preds = %merge15, %then
  %seen59 = load i32, i32* %seen, align 4
  %tmp60 = icmp slt i32 %seen59, 4
  br i1 %tmp60, label %then62, label %else64

then:                                             ; preds = %while_body
  %temp8 = load %FarmTour*, %FarmTour** %temp, align 8
  %vtable9 = getelementptr inbounds %FarmTour, %FarmTour* %temp8, i32 0, i32 0
  %vtable10 = load %FarmTour_vtable*, %FarmTour_vtable** %vtable9, align 8
```

```
393       %fun_to_call = getelementptr inbounds %FarmTour_vtable,
394           %FarmTour_vtable* %vtable10, i32 0, i32 1
395       %function = load void (%FarmTour**, %Farmer**)*,
396           void (%FarmTour**, %Farmer**)** %fun_to_call, align 8
397       call void %function(%FarmTour** %temp, %Farmer** %bob)
398       %seen11 = load i32, i32* %seen, align 4
399       %tmp = add i32 %seen11, 1
400       store i32 %tmp, i32* %seen, align 4
401       br label %merge
402
403     else:                                             ; preds = %while_body
404       %input12 = load i8*, i8** %input, align 8
405       %input13 = load i8*, i8** %input, align 8
406       %streq14 = call i1 (i8*, i8*, ...) @streq(i8* %input13, i8* getelementptr
407           inbounds ([3 x i8], [3 x i8]* @global_str.37, i32 0, i32 0))
408       br i1 %streq14, label %then16, label %else24
409
410     merge15:                                          ; preds = %merge28, %then16
411       br label %merge
412
413     then16:                                           ; preds = %else
414       %temp17 = load %FarmTour*, %FarmTour** %temp, align 8
415       %vtable18 = getelementptr inbounds %FarmTour, %FarmTour* %temp17, i32 0, i32 0
416       %vtable19 = load %FarmTour_vtable*, %FarmTour_vtable** %vtable18, align 8
417       %fun_to_call20 = getelementptr inbounds %FarmTour_vtable,
418           %FarmTour_vtable* %vtable19, i32 0, i32 2
419       %function21 = load void (%FarmTour**, %Farmer**)*,
420           void (%FarmTour**, %Farmer**)** %fun_to_call20, align 8
421       call void %function21(%FarmTour** %temp, %Farmer** %bob)
422       %seen22 = load i32, i32* %seen, align 4
423       %tmp23 = add i32 %seen22, 1
424       store i32 %tmp23, i32* %seen, align 4
425       br label %merge15
426
427     else24:                                           ; preds = %else
428       %input25 = load i8*, i8** %input, align 8
429       %input26 = load i8*, i8** %input, align 8
430       %streq27 = call i1 (i8*, i8*, ...) @streq(i8* %input26, i8* getelementptr
431           inbounds ([2 x i8], [2 x i8]* @global_str.39, i32 0, i32 0))
432       br i1 %streq27, label %then29, label %else37
433
434     merge28:                                          ; preds = %merge41, %then29
435       br label %merge15
436
437     then29:                                           ; preds = %else24
438       %temp30 = load %FarmTour*, %FarmTour** %temp, align 8
439       %vtable31 = getelementptr inbounds %FarmTour, %FarmTour* %temp30, i32 0, i32 0
440       %vtable32 = load %FarmTour_vtable*, %FarmTour_vtable** %vtable31, align 8
441       %fun_to_call33 = getelementptr inbounds %FarmTour_vtable,
```

```
        %FarmTour_vtable* %vtable32, i32 0, i32 3
  %function34 = load void (%FarmTour**, %Farmer**)*,
      void (%FarmTour**, %Farmer**)** %fun_to_call33, align 8
  call void %function34(%FarmTour** %temp, %Farmer** %bob)
  %seen35 = load i32, i32* %seen, align 4
  %tmp36 = add i32 %seen35, 1
  store i32 %tmp36, i32* %seen, align 4
  br label %merge28

else37:                                           ; preds = %else24
  %input38 = load i8*, i8** %input, align 8
  %input39 = load i8*, i8** %input, align 8
  %streq40 = call i1 (i8*, i8*, ...) @streq(i8* %input39, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @global_str.41, i32 0, i32 0))
  br i1 %streq40, label %then42, label %else50

merge41:                                          ; preds = %merge54, %then42
  br label %merge28

then42:                                           ; preds = %else37
  %temp43 = load %FarmTour*, %FarmTour** %temp, align 8
  %vtable44 = getelementptr inbounds %FarmTour, %FarmTour* %temp43, i32 0, i32 0
  %vtable45 = load %FarmTour_vtable*, %FarmTour_vtable** %vtable44, align 8
  %fun_to_call46 = getelementptr inbounds %FarmTour_vtable,
      %FarmTour_vtable* %vtable45, i32 0, i32 4
  %function47 = load void (%FarmTour**, %Farmer**)*,
      void (%FarmTour**, %Farmer**)** %fun_to_call46, align 8
  call void %function47(%FarmTour** %temp, %Farmer** %bob)
  %seen48 = load i32, i32* %seen, align 4
  %tmp49 = add i32 %seen48, 1
  store i32 %tmp49, i32* %seen, align 4
  br label %merge41

else50:                                           ; preds = %else37
  %input51 = load i8*, i8** %input, align 8
  %input52 = load i8*, i8** %input, align 8
  %streq53 = call i1 (i8*, i8*, ...) @streq(i8* %input52, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @global_str.43, i32 0, i32 0))
  br i1 %streq53, label %then55, label %else57

merge54:                                          ; preds = %else57
  br label %merge41

then55:                                           ; preds = %else50
  %printf56 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.27, i32 0, i32 0), i8* getelementptr inbounds
      ([47 x i8], [47 x i8]* @global_str.44, i32 0, i32 0))
  ret void

```

```
491     else57:                                           ; preds = %else50
492       %printf58 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
493           ([4 x i8], [4 x i8]* @fmt.27, i32 0, i32 0), i8* getelementptr inbounds
494           ([68 x i8], [68 x i8]* @global_str.45, i32 0, i32 0))
495       br label %merge54
496
497     merge61:                                          ; preds = %else64, %then62
498       br label %while
499
500     then62:                                           ; preds = %merge
501       %printf63 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
502           ([4 x i8], [4 x i8]* @fmt.27, i32 0, i32 0), i8* getelementptr inbounds
503           ([59 x i8], [59 x i8]* @global_str.46, i32 0, i32 0))
504       br label %merge61
505
506     else64:                                           ; preds = %merge
507       br label %merge61
508
509     merge67:                                          ; preds = %while
510       %printf68 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
511           ([4 x i8], [4 x i8]* @fmt.27, i32 0, i32 0), i8* getelementptr inbounds
512           ([84 x i8], [84 x i8]* @global_str.47, i32 0, i32 0))
513       ret void
514     }
515
516     define void @FarmTour_seePig(%FarmTour** %0, %Farmer** %1) {
517     entry:
518       %temp = alloca %FarmTour*, align 8
519       %temp1 = load %FarmTour*, %FarmTour** %0, align 8
520       store %FarmTour* %temp1, %FarmTour** %temp, align 8
521       %temp2 = alloca %Farmer*, align 8
522       %temp3 = load %Farmer*, %Farmer** %1, align 8
523       store %Farmer* %temp3, %Farmer** %temp2, align 8
524       %malloccall = tail call i8* @malloc(i32 ptrtoint (%Pig* getelementptr
525           (%Pig, %Pig* null, i32 1) to i32))
526       %Pig = bitcast i8* %malloccall to %Pig*
527       %vtable = getelementptr inbounds %Pig, %Pig* %Pig, i32 0, i32 0
528       store %Pig_vtable* @Pig_vtable_data, %Pig_vtable** %vtable, align 8
529       %temp4 = alloca %Pig*, align 8
530       %num_permitted = getelementptr inbounds %Pig, %Pig* %Pig, i32 0, i32 1
531       store i32 2, i32* %num_permitted, align 4
532       %permit_list = getelementptr inbounds %Pig, %Pig* %Pig, i32 0, i32 2
533       store [2 x i8*] [i8* getelementptr inbounds ([4 x i8], [4 x i8]*
534           @temp_str.51, i32 0, i32 0), i8* getelementptr inbounds ([7 x i8],
535           [7 x i8]* @temp_str.52, i32 0, i32 0)], [2 x i8*]* %permit_list, align 8
536       store %Pig* %Pig, %Pig** %temp4, align 8
537       %p = alloca %Pig*, align 8
538       %temp5 = load %Pig*, %Pig** %temp4, align 8
539       store %Pig* %temp5, %Pig** %p, align 8
```

```
  %input = alloca i8*, align 8
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @global_str.53, i32 0,
      i32 0), i8** %input, align 8
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.49, i32 0, i32 0), i8* getelementptr inbounds
      ([68 x i8], [68 x i8]* @global_str.54, i32 0, i32 0))
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.49, i32 0, i32 0), i8* getelementptr inbounds
      ([65 x i8], [65 x i8]* @global_str.55, i32 0, i32 0))
  %printf7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
      [3 x i8]* @fmt.48, i32 0, i32 0), i8* getelementptr inbounds ([35 x i8],
      [35 x i8]* @global_str.56, i32 0, i32 0))
  %readaline = call i8** (...) @readaline()
  %get_temp = load i8*, i8** %readaline, align 8
  store i8* %get_temp, i8** %input, align 8
  %input8 = load i8*, i8** %input, align 8
  %input9 = load i8*, i8** %input, align 8
  %streq = call i1 (i8*, i8*, ...) @streq(i8* %input9, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @global_str.58, i32 0, i32 0))
  br i1 %streq, label %then, label %else

merge:                                            ; preds = %else, %then
  ret void

then:                                             ; preds = %entry
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt.49, i32 0, i32 0), i8* getelementptr inbounds
      ([23 x i8], [23 x i8]* @global_str.59, i32 0, i32 0))
  %temp11 = load %Pig*, %Pig** %p, align 8
  %vtable12 = getelementptr inbounds %Pig, %Pig* %temp11, i32 0, i32 0
  %vtable13 = load %Pig_vtable*, %Pig_vtable** %vtable12, align 8
  %fun_to_call = getelementptr inbounds %Pig_vtable, %Pig_vtable* %vtable13,
      i32 0, i32 0
  %function = load void (%Animal**)*, void (%Animal**)** %fun_to_call, align 8
  %arg_cast = bitcast %Pig** %p to %Animal**
  call void %function(%Animal** %arg_cast)
  %printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.49, i32 0, i32 0), i8* getelementptr inbounds
      ([29 x i8], [29 x i8]* @global_str.60, i32 0, i32 0))
  br label %merge

else:                                             ; preds = %entry
  %printf15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.49, i32 0, i32 0), i8* getelementptr inbounds
      ([52 x i8], [52 x i8]* @global_str.61, i32 0, i32 0))
  %temp16 = load %Farmer*, %Farmer** %temp2, align 8
  %vtable17 = getelementptr inbounds %Farmer, %Farmer* %temp16, i32 0, i32 0
  %vtable18 = load %Farmer_vtable*, %Farmer_vtable** %vtable17, align 8
  %fun_to_call19 = getelementptr inbounds %Farmer_vtable,
```

```
589         %Farmer_vtable* %vtable18, i32 0, i32 2
590       %function20 = load void (%Farmer**, %Animal**)*,
591         void (%Farmer**, %Animal**)** %fun_to_call19, align 8
592       %arg_cast21 = bitcast %Pig** %p to %Animal**
593       call void %function20(%Farmer** %temp2, %Animal** %arg_cast21)
594       %printf22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
595         ([4 x i8], [4 x i8]* @fmt.49, i32 0, i32 0), i8* getelementptr inbounds
596         ([54 x i8], [54 x i8]* @global_str.62, i32 0, i32 0))
597       br label %merge
598     }
599
600     define void @FarmTour_seeCow(%FarmTour** %0, %Farmer** %1) {
601     entry:
602       %temp = alloca %FarmTour*, align 8
603       %temp1 = load %FarmTour*, %FarmTour** %0, align 8
604       store %FarmTour* %temp1, %FarmTour** %temp, align 8
605       %temp2 = alloca %Farmer*, align 8
606       %temp3 = load %Farmer*, %Farmer** %1, align 8
607       store %Farmer* %temp3, %Farmer** %temp2, align 8
608       %malloccall = tail call i8* @malloc(i32 ptrtoint (%Cow* getelementptr
609         (%Cow, %Cow* null, i32 1) to i32))
610       %Cow = bitcast i8* %malloccall to %Cow*
611       %vtable = getelementptr inbounds %Cow, %Cow* %Cow, i32 0, i32 0
612       store %Cow_vtable* @Cow_vtable_data, %Cow_vtable** %vtable, align 8
613       %temp4 = alloca %Cow*, align 8
614       %num_permitted = getelementptr inbounds %Cow, %Cow* %Cow, i32 0, i32 1
615       store i32 2, i32* %num_permitted, align 4
616       %permit_list = getelementptr inbounds %Cow, %Cow* %Cow, i32 0, i32 2
617       store [2 x i8*] [i8* getelementptr inbounds ([4 x i8], [4 x i8]*
618         @temp_str.66, i32 0, i32 0), i8* getelementptr inbounds ([7 x i8],
619         [7 x i8]* @temp_str.67, i32 0, i32 0)], [2 x i8*]* %permit_list, align 8
620       store %Cow* %Cow, %Cow** %temp4, align 8
621       %c = alloca %Cow*, align 8
622       %temp5 = load %Cow*, %Cow** %temp4, align 8
623       store %Cow* %temp5, %Cow** %c, align 8
624       %input = alloca i8*, align 8
625       store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @global_str.68,
626         i32 0, i32 0), i8** %input, align 8
627       %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
628         ([4 x i8], [4 x i8]* @fmt.64, i32 0, i32 0), i8* getelementptr inbounds
629         ([68 x i8], [68 x i8]* @global_str.69, i32 0, i32 0))
630       %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
631         ([4 x i8], [4 x i8]* @fmt.64, i32 0, i32 0), i8* getelementptr inbounds
632         ([69 x i8], [69 x i8]* @global_str.70, i32 0, i32 0))
633       %printf7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
634         ([3 x i8], [3 x i8]* @fmt.63, i32 0, i32 0), i8* getelementptr inbounds
635         ([35 x i8], [35 x i8]* @global_str.71, i32 0, i32 0))
636       %readaline = call i8** (...) @readaline()
637       %get_temp = load i8*, i8** %readaline, align 8
```

```
  store i8* %get_temp, i8** %input, align 8
  %input8 = load i8*, i8** %input, align 8
  %input9 = load i8*, i8** %input, align 8
  %streq = call i1 (i8*, i8*, ...) @streq(i8* %input9, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @global_str.73, i32 0, i32 0))
  br i1 %streq, label %then, label %else

merge:                                            ; preds = %else, %then
  ret void

then:                                             ; preds = %entry
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.64, i32 0, i32 0), i8* getelementptr inbounds
      ([23 x i8], [23 x i8]* @global_str.74, i32 0, i32 0))
  %temp11 = load %Cow*, %Cow** %c, align 8
  %vtable12 = getelementptr inbounds %Cow, %Cow* %temp11, i32 0, i32 0
  %vtable13 = load %Cow_vtable*, %Cow_vtable** %vtable12, align 8
  %fun_to_call = getelementptr inbounds %Cow_vtable, %Cow_vtable* %vtable13,
      i32 0, i32 0
  %function = load void (%Animal**)*, void (%Animal**)** %fun_to_call, align 8
  %arg_cast = bitcast %Cow** %c to %Animal**
  call void %function(%Animal** %arg_cast)
  %printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.64, i32 0, i32 0), i8* getelementptr inbounds
      ([29 x i8], [29 x i8]* @global_str.75, i32 0, i32 0))
  br label %merge

else:                                             ; preds = %entry
  %printf15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.64, i32 0, i32 0), i8* getelementptr inbounds
      ([52 x i8], [52 x i8]* @global_str.76, i32 0, i32 0))
  %temp16 = load %Farmer*, %Farmer** %temp2, align 8
  %vtable17 = getelementptr inbounds %Farmer, %Farmer* %temp16, i32 0, i32 0
  %vtable18 = load %Farmer_vtable*, %Farmer_vtable** %vtable17, align 8
  %fun_to_call19 = getelementptr inbounds %Farmer_vtable,
     %Farmer_vtable* %vtable18, i32 0, i32 2
  %function20 = load void (%Farmer**, %Animal**)*,
      void (%Farmer**, %Animal**)** %fun_to_call19, align 8
  %arg_cast21 = bitcast %Cow** %c to %Animal**
  call void %function20(%Farmer** %temp2, %Animal** %arg_cast21)
  %printf22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.64, i32 0, i32 0), i8* getelementptr inbounds
      ([54 x i8], [54 x i8]* @global_str.77, i32 0, i32 0))
  br label %merge
}

define void @FarmTour_seeDuck(%FarmTour** %0, %Farmer** %1) {
entry:
  %temp = alloca %FarmTour*, align 8
```

```
    %temp1 = load %FarmTour*, %FarmTour** %0, align 8
    store %FarmTour* %temp1, %FarmTour** %temp, align 8
    %temp2 = alloca %Farmer*, align 8
    %temp3 = load %Farmer*, %Farmer** %1, align 8
    store %Farmer* %temp3, %Farmer** %temp2, align 8
    %malloccall = tail call i8* @malloc(i32 ptrtoint (%Duck* getelementptr
        (%Duck, %Duck* null, i32 1) to i32))
    %Duck = bitcast i8* %malloccall to %Duck*
    %vtable = getelementptr inbounds %Duck, %Duck* %Duck, i32 0, i32 0
    store %Duck_vtable* @Duck_vtable_data, %Duck_vtable** %vtable, align 8
    %temp4 = alloca %Duck*, align 8
    %num_permitted = getelementptr inbounds %Duck, %Duck* %Duck, i32 0, i32 1
    store i32 3, i32* %num_permitted, align 4
    %permit_list = getelementptr inbounds %Duck, %Duck* %Duck, i32 0, i32 2
    store [3 x i8*] [i8* getelementptr inbounds ([5 x i8],
        [5 x i8]* @temp_str.81, i32 0, i32 0), i8* getelementptr inbounds
        ([7 x i8], [7 x i8]* @temp_str.82, i32 0, i32 0), i8* getelementptr
        inbounds ([9 x i8], [9 x i8]* @temp_str.83, i32 0, i32 0)],
        [3 x i8*]* %permit_list, align 8
    store %Duck* %Duck, %Duck** %temp4, align 8
    %d = alloca %Duck*, align 8
    %temp5 = load %Duck*, %Duck** %temp4, align 8
    store %Duck* %temp5, %Duck** %d, align 8
    %input = alloca i8*, align 8
    store i8* getelementptr inbounds ([1 x i8], [1 x i8]*
        @global_str.84, i32 0, i32 0), i8** %input, align 8
    %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
        ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
        ([69 x i8], [69 x i8]* @global_str.85, i32 0, i32 0))
    %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
        ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
        ([52 x i8], [52 x i8]* @global_str.86, i32 0, i32 0))
    %printf7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
        ([3 x i8], [3 x i8]* @fmt.78, i32 0, i32 0), i8* getelementptr inbounds
        ([36 x i8], [36 x i8]* @global_str.87, i32 0, i32 0))
    %readaline = call i8** (...) @readaline()
    %get_temp = load i8*, i8** %readaline, align 8
    store i8* %get_temp, i8** %input, align 8
    %temp8 = load %Duck*, %Duck** %d, align 8
    %permit_length_ptr = getelementptr inbounds %Duck, %Duck* %temp8, i32 0, i32 1
    %permit_length = load i32, i32* %permit_length_ptr, align 4
    %permit_list_ptr = getelementptr inbounds %Duck, %Duck* %temp8, i32 0, i32 2
    %str_ptr = getelementptr [3 x i8*], [3 x i8*]* %permit_list_ptr, i32 0, i32 0
    call void (i8*, i8**, i32, ...) @class_permitted(i8* getelementptr inbounds
        ([9 x i8], [9 x i8]* @class_location.88, i32 0, i32 0),
        i8** %str_ptr, i32 %permit_length)
    %vtable9 = getelementptr inbounds %Duck, %Duck* %temp8, i32 0, i32 0
    %vtable10 = load %Duck_vtable*, %Duck_vtable** %vtable9, align 8
    %fun_to_call = getelementptr inbounds %Duck_vtable, %Duck_vtable* %vtable10,
```

```
        i32 0, i32 1
    %function = load void (%Duck**)*, void (%Duck**)** %fun_to_call, align 8
    call void %function(%Duck** %d)
    %input11 = load i8*, i8** %input, align 8
    %input12 = load i8*, i8** %input, align 8
    %streq = call i1 (i8*, i8*, ...) @streq(i8* %input12, i8* getelementptr
        inbounds ([2 x i8], [2 x i8]* @global_str.90, i32 0, i32 0))
    br i1 %streq, label %then, label %else

merge:                                            ; preds = %else, %then
    ret void

then:                                             ; preds = %entry
    %printf13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
        ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
        ([24 x i8], [24 x i8]* @global_str.91, i32 0, i32 0))
    %temp14 = load %Duck*, %Duck** %d, align 8
    %permit_length_ptr15 = getelementptr inbounds %Duck, %Duck* %temp14, i32 0,
        i32 1
    %permit_length16 = load i32, i32* %permit_length_ptr15, align 4
    %permit_list_ptr17 = getelementptr inbounds %Duck, %Duck* %temp14, i32 0,
        i32 2
    %str_ptr18 = getelementptr [3 x i8*], [3 x i8*]* %permit_list_ptr17, i32 0,
        i32 0
    call void (i8*, i8**, i32, ...) @class_permitted(i8* getelementptr inbounds
        ([9 x i8], [9 x i8]* @class_location.92, i32 0, i32 0), i8** %str_ptr18,
        i32 %permit_length16)
    %vtable19 = getelementptr inbounds %Duck, %Duck* %temp14, i32 0, i32 0
    %vtable20 = load %Duck_vtable*, %Duck_vtable** %vtable19, align 8
    %fun_to_call21 = getelementptr inbounds %Duck_vtable, %Duck_vtable* %vtable20,
        i32 0, i32 1
    %function22 = load void (%Duck**)*, void (%Duck**)** %fun_to_call21, align 8
    call void %function22(%Duck** %d)
    %printf23 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
        ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
        ([48 x i8], [48 x i8]* @global_str.93, i32 0, i32 0))
    br label %merge

else:                                             ; preds = %entry
    %printf24 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
        ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
        ([54 x i8], [54 x i8]* @global_str.94, i32 0, i32 0))
    %temp25 = load %Farmer*, %Farmer** %temp2, align 8
    %vtable26 = getelementptr inbounds %Farmer, %Farmer* %temp25, i32 0, i32 0
    %vtable27 = load %Farmer_vtable*, %Farmer_vtable** %vtable26, align 8
    %fun_to_call28 = getelementptr inbounds %Farmer_vtable, %Farmer_vtable*
        %vtable27, i32 0, i32 2
    %function29 = load void (%Farmer**, %Animal**)*,
        void (%Farmer**, %Animal**)** %fun_to_call28, align 8
```

```
785       %arg_cast = bitcast %Duck** %d to %Animal**
786       call void %function29(%Farmer** %temp2, %Animal** %arg_cast)
787       %printf30 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
788           ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
789           ([44 x i8], [44 x i8]* @global_str.95, i32 0, i32 0))
790       br label %merge
791     }
792
793     define void @FarmTour_seeSheep(%FarmTour** %0, %Farmer** %1) {
794     entry:
795       %temp = alloca %FarmTour*, align 8
796       %temp1 = load %FarmTour*, %FarmTour** %0, align 8
797       store %FarmTour* %temp1, %FarmTour** %temp, align 8
798       %temp2 = alloca %Farmer*, align 8
799       %temp3 = load %Farmer*, %Farmer** %1, align 8
800       store %Farmer* %temp3, %Farmer** %temp2, align 8
801       %malloccall = tail call i8* @malloc(i32 ptrtoint (%Sheep* getelementptr
802           (%Sheep, %Sheep* null, i32 1) to i32))
803       %Sheep = bitcast i8* %malloccall to %Sheep*
804       %vtable = getelementptr inbounds %Sheep, %Sheep* %Sheep, i32 0, i32 0
805       store %Sheep_vtable* @Sheep_vtable_data, %Sheep_vtable** %vtable, align 8
806       %temp4 = alloca %Sheep*, align 8
807       %num_permitted = getelementptr inbounds %Sheep, %Sheep* %Sheep, i32 0, i32 1
808       store i32 2, i32* %num_permitted, align 4
809       %permit_list = getelementptr inbounds %Sheep, %Sheep* %Sheep, i32 0, i32 2
810       store [2 x i8*] [i8* getelementptr inbounds ([6 x i8], [6 x i8]* @temp_str.99,
811           i32 0, i32 0), i8* getelementptr inbounds ([7 x i8], [7 x i8]*
812           @temp_str.100, i32 0, i32 0)], [2 x i8*]* %permit_list, align 8
813       store %Sheep* %Sheep, %Sheep** %temp4, align 8
814       %s = alloca %Sheep*, align 8
815       %temp5 = load %Sheep*, %Sheep** %temp4, align 8
816       store %Sheep* %temp5, %Sheep** %s, align 8
817       %input = alloca i8*, align 8
818       store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @global_str.101, i32 0,
819           i32 0), i8** %input, align 8
820       %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
821           ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
822           ([70 x i8], [70 x i8]* @global_str.102, i32 0, i32 0))
823       %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
824           ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
825           ([75 x i8], [75 x i8]* @global_str.103, i32 0, i32 0))
826       %printf7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
827           ([3 x i8], [3 x i8]* @fmt.96, i32 0, i32 0), i8* getelementptr inbounds
828           ([37 x i8], [37 x i8]* @global_str.104, i32 0, i32 0))
829       %readaline = call i8** (...) @readaline()
830       %get_temp = load i8*, i8** %readaline, align 8
831       store i8* %get_temp, i8** %input, align 8
832       %input8 = load i8*, i8** %input, align 8
833       %input9 = load i8*, i8** %input, align 8
```

```
        void (%Farmer**, %Animal**)** %fun_to_call28, align 8
  %arg_cast = bitcast %Duck** %d to %Animal**
  call void %function29(%Farmer** %temp2, %Animal** %arg_cast)
  %printf30 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.79, i32 0, i32 0), i8* getelementptr inbounds
      ([44 x i8], [44 x i8]* @global_str.95, i32 0, i32 0))
  br label %merge
}

define void @FarmTour_seeSheep(%FarmTour** %0, %Farmer** %1) {
entry:
  %temp = alloca %FarmTour*, align 8
  %temp1 = load %FarmTour*, %FarmTour** %0, align 8
  store %FarmTour* %temp1, %FarmTour** %temp, align 8
  %temp2 = alloca %Farmer*, align 8
  %temp3 = load %Farmer*, %Farmer** %1, align 8
  store %Farmer* %temp3, %Farmer** %temp2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%Sheep* getelementptr
      (%Sheep, %Sheep* null, i32 1) to i32))
  %Sheep = bitcast i8* %malloccall to %Sheep*
  %vtable = getelementptr inbounds %Sheep, %Sheep* %Sheep, i32 0, i32 0
  store %Sheep_vtable* @Sheep_vtable_data, %Sheep_vtable** %vtable, align 8
  %temp4 = alloca %Sheep*, align 8
  %num_permitted = getelementptr inbounds %Sheep, %Sheep* %Sheep, i32 0, i32 1
  store i32 2, i32* %num_permitted, align 4
  %permit_list = getelementptr inbounds %Sheep, %Sheep* %Sheep, i32 0, i32 2
  store [2 x i8*] [i8* getelementptr inbounds ([6 x i8], [6 x i8]* @temp_str.99,
      i32 0, i32 0), i8* getelementptr inbounds ([7 x i8], [7 x i8]*
      @temp_str.100, i32 0, i32 0)], [2 x i8*]* %permit_list, align 8
  store %Sheep* %Sheep, %Sheep** %temp4, align 8
  %s = alloca %Sheep*, align 8
  %temp5 = load %Sheep*, %Sheep** %temp4, align 8
  store %Sheep* %temp5, %Sheep** %s, align 8
  %input = alloca i8*, align 8
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @global_str.101, i32 0,
      i32 0), i8** %input, align 8
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
      ([70 x i8], [70 x i8]* @global_str.102, i32 0, i32 0))
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
      ([75 x i8], [75 x i8]* @global_str.103, i32 0, i32 0))
  %printf7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([3 x i8], [3 x i8]* @fmt.96, i32 0, i32 0), i8* getelementptr inbounds
      ([37 x i8], [37 x i8]* @global_str.104, i32 0, i32 0))
  %readaline = call i8** (...) @readaline()
  %get_temp = load i8*, i8** %readaline, align 8
  store i8* %get_temp, i8** %input, align 8
  %input8 = load i8*, i8** %input, align 8
```

```
  %input9 = load i8*, i8** %input, align 8
  %streq = call i1 (i8*, i8*, ...) @streq(i8* %input9, i8* getelementptr
      inbounds ([2 x i8], [2 x i8]* @global_str.106, i32 0, i32 0))
  br i1 %streq, label %then, label %else

merge:                                            ; preds = %else, %then
  ret void

then:                                             ; preds = %entry
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
      ([25 x i8], [25 x i8]* @global_str.107, i32 0, i32 0))
  %temp11 = load %Sheep*, %Sheep** %s, align 8
  %vtable12 = getelementptr inbounds %Sheep, %Sheep* %temp11, i32 0, i32 0
  %vtable13 = load %Sheep_vtable*, %Sheep_vtable** %vtable12, align 8
  %fun_to_call = getelementptr inbounds %Sheep_vtable, %Sheep_vtable* %vtable13,
      i32 0, i32 0
  %function = load void (%Animal**)*, void (%Animal**)** %fun_to_call, align 8
  %arg_cast = bitcast %Sheep** %s to %Animal**
  call void %function(%Animal** %arg_cast)
  %printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
      ([31 x i8], [31 x i8]* @global_str.108, i32 0, i32 0))
  br label %merge

else:                                             ; preds = %entry
  %printf15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
      ([56 x i8], [56 x i8]* @global_str.109, i32 0, i32 0))
  %temp16 = load %Farmer*, %Farmer** %temp2, align 8
  %vtable17 = getelementptr inbounds %Farmer, %Farmer* %temp16, i32 0, i32 0
  %vtable18 = load %Farmer_vtable*, %Farmer_vtable** %vtable17, align 8
  %fun_to_call19 = getelementptr inbounds %Farmer_vtable,
      %Farmer_vtable* %vtable18, i32 0, i32 2
  %function20 = load void (%Farmer**, %Animal**)*,
      void (%Farmer**, %Animal**)** %fun_to_call19, align 8
  %arg_cast21 = bitcast %Sheep** %s to %Animal**
  call void %function20(%Farmer** %temp2, %Animal** %arg_cast21)
  %printf22 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
      ([4 x i8], [4 x i8]* @fmt.97, i32 0, i32 0), i8* getelementptr inbounds
      ([40 x i8], [40 x i8]* @global_str.110, i32 0, i32 0))
  br label %merge
}

declare noalias i8* @malloc(i32)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%FarmTour* getelementptr
```

```
882         (%FarmTour, %FarmTour* null, i32 1) to i32))
883       %FarmTour = bitcast i8* %malloccall to %FarmTour*
884       %vtable = getelementptr inbounds %FarmTour, %FarmTour* %FarmTour, i32 0, i32 0
885       store %FarmTour_vtable* @FarmTour_vtable_data, %FarmTour_vtable** %vtable,
886         align 8
887       %temp = alloca %FarmTour*, align 8
888       %num_permitted = getelementptr inbounds %FarmTour, %FarmTour* %FarmTour,
889         i32 0, i32 1
890       store i32 2, i32* %num_permitted, align 4
891       %permit_list = getelementptr inbounds %FarmTour, %FarmTour* %FarmTour,
892         i32 0, i32 2
893       store [2 x i8*] [i8* getelementptr inbounds ([9 x i8], [9 x i8]*
894         @temp_str.114, i32 0, i32 0), i8* getelementptr inbounds
895         ([5 x i8], [5 x i8]* @temp_str.115, i32 0, i32 0)],
896         [2 x i8*]* %permit_list, align 8
897       store %FarmTour* %FarmTour, %FarmTour** %temp, align 8
898       %ft = alloca %FarmTour*, align 8
899       %temp1 = load %FarmTour*, %FarmTour** %temp, align 8
900       store %FarmTour* %temp1, %FarmTour** %ft, align 8
901       %temp2 = load %FarmTour*, %FarmTour** %ft, align 8
902       %vtable3 = getelementptr inbounds %FarmTour, %FarmTour* %temp2, i32 0, i32 0
903       %vtable4 = load %FarmTour_vtable*, %FarmTour_vtable** %vtable3, align 8
904       %fun_to_call = getelementptr inbounds %FarmTour_vtable,
905         %FarmTour_vtable* %vtable4, i32 0, i32 0
906       %function = load void (%FarmTour**)*, void (%FarmTour**)** %fun_to_call,
907         align 8
908       call void %function(%FarmTour** %ft)
909       ret i32 0
910     }
911 +
```

14 Lessons Learned

14.1 Valerie

These past few months have been such a whirlwind of a ride. I've had so much personal growth and growth in the relationship with my team. One lesson I learned is the importance of stepping up even if you don't feel confident in your knowledge or abilities. I had heard horror stories of groups with one slacking team member and I wanted to make sure I wasn't that type of teammate. However, I also had severe doubts in my ability to contribute to the project. That being said, I pushed myself to get into the weeds of codegen and semant while also keeping in mind the deliverables and deadlines we had coming up. Even when you don't think you can contribute to one portion, there is always something else that can be done whether that be going to office hours when we have a bug or working on other parts of a deliverable that are needed. One final thought and important realization that I've come to looking back on this project is that your team can really make or break the project. I was lucky to have found a team of all hardworking individuals that communicated well and I meshed with on a personal and professional level. I'm truly grateful for the opportunity to work with them and build something together.

To future students, do not underestimate codegen and please start early, I truly can't emphasize this enough. Even if you think implementing something will not be challenging, it definitely will be.

14.2 Ayda

Where to start? I feel like my brain has grown several sizes over the course of this semester, and yet I'm still learning something new every time we work on this compiler. I suppose the best thing I've learned is that who you work with on long-term projects really matters. I was really lucky to have a group this wonderful, but I can imagine how frightful it could be to have to work in a group that doesn't mesh well. The project has definitely made me gain a deep appreciation for how difficult building a compiler is. I was quite nervous coming into this class, as I've never been part of a long-term group project like this, and it covered things I've never touched before. I'm realizing how critical it is to have a solid understanding of both semantic theory and low-level implementation details for such a project. That was one of the challenges for me, since I find conceptual work easier and more enjoyable than the nitty-gritty of implementation. However, I feel like I've gained a new appreciation for the importance of how high-level design and low-level implementation work together, as opposed to thinking of each being important in its own vacuum, and learned how to think about both together. I'm really proud of the work we've done together on Iris, and I know I'll remember this course as one of my favorites (even if it thoroughly wiped the floor with me at times).

14.3 Tim

Compilers is unlike any class I've taken at Tufts. A semester-long group project where you're, in essence, thrown to the wolves is daunting, but it's been an incredible experience. I have learned so much about the interworkings of a compiler – how each phase is connected, how assembly gets generated, and the hurdles one has to clear to implement one. Codegen, especially, was a challenging part of the project to me. I never felt completely comfortable with the LLVM API, and that made it challenging to implement the ideas we visualized in C. I thoroughly enjoyed the Scanner / Parser and Semant phases and got the hang of OCaml pretty quickly. But the best part of the project was hands down the team I got to work with and got to know closely over the few months. I'm grateful to have had such hardworking and passionate team members. We all motivate each other to push through times of seg faults and confusion. This class is so hands-off, and although it can be frustrating to not be directly given the keys to success, it makes you learn more by being independent. Looking back, Iris is a piece of work I can feel incredibly proud of and accomplished by.

14.4 Trevor

This semester has been a spectacular, illuminating experience. When we decided to implement an object-oriented language for our compiler, I had no idea what that would look like at a machine level (turns out it's fairly complicated). I've realized that there are so many abstractions in programming (and in life in general) that we take for granted.

I've also learned so much about system and protocol design. Good modularity is key to success, but implementing it begs the question: what information do the other modules need? We changed our S-AST at least 5 times, each time because we realized that codegen needed access to extra information or the information had to be presented differently. We were lucky that the changes weren't too difficult to make; these kinds of major changes are impossible on other systems. Thorough and open-ended protocol design is incredibly important and makes implementing new features much easier (and, in some cases, possible). You can never put too much thought into your system's design.

Above all else, I've discovered that the most important part of any major project is your team. During this project, I've had the pleasure of working with passionate, persevering, and kind people. I was so lucky to have a team that I got along with on a personal and professional level. When I look back on this class in the future, I know that the first thing I'll remember is my team.

14.5 Josh

When I first joined Compilers, I remember being pretty scared because I had no idea what making a compiler would even entail – I can say with certainty that I had no idea about what llvm was at the start of the semester. However, the way the course is structured made it easy to ease into the entire process. After the final weeks of finishing the project, I think that my understanding of how everything works and comes together has greatly improved.

In particular, when I first started working on the project, I was completely confused by codegen. Part of the reason why it was so difficult was because it seemed like such a huge mountain to scale. The most important part of finishing this project was finding out how we could break the project into manageable parts. Having to learn about everything on the spot was definitely scary, but what made it all great was having a team around me that I could rely on. In the last month, we've had many struggles with getting Iris to work. Yet, everyone in the group would always put in the effort to understand and resolve the problem. It always felt like everyone was invested in the project, which made it easier to work together. This class was a great experience, and I'm thankful that I had such a friendly and smart team for the entire semester.

14.6 General Advice

Our greatest advice is not to underestimate the amount of time code generation, and figuring out the LLVM OCaml API, is going to take. That took up the vast majority of the time we spent on this project, and it mainly is implemented in the final third of the course. Plan ahead for that and start as early as possible. Building reference code in LLVM from other languages that implement your special features also helped us a lot.

For more implementation-specific advice, be sure to work in subgroups and delegate tasks to different people, even if you do work mostly in a group setting. Manage your versions by committing often, just in case. It never hurts to be able to roll back to a working version. Another strategy that saved us a lot of time was using distinct modules and helper functions for our lookup tables, since we had to change our lookup tables several times as we discovered new information we needed to implement the code generation phase. By implementing these in somewhat distinct modules and accessing values by helper functions, we saved a ton of time each time we had to make a change in the data structure.

15 Appendix

15.1 iris.ml

```
(* Top-level of the MicroC compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate LLVM IR,
   and dump the module *)
(* Edited by Ayda Aricanli (copied from MicroC)*)
(* FULL COMPILER (COPIED FROM MICROC) *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./iris.native [-a|-s|-l|-c] [file.iris]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
  Llvm_analysis.assert_valid_module m;
  print_string (Llvm.string_of_llmodule m)
```

15.2 codegen.ml

```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

(* Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk *)

(* We'll refer to Llvm and Ast constructs with module names *)
module L = Llvm
module A = Ast
module S = Sast
open Sast
open Guini
(* open Ast *)

module StringMap = Map.Make(String)
module IntMap = Map.Make(Int)

(* accessing variable in a struct type: use build_struct_gep *)

let translate (classes : sclass_decl list) =
  let context  = L.global_context () in
  (* Add types to the context so we can use them in our LLVM code *)
  let i32_t    = L.i32_type    context
  and i8_t     = L.i8_type     context
  and i1_t     = L.i1_type     context
  and float_t  = L.double_type context
  and void_t   = L.void_type   context
  in

  let string_t = L.pointer_type i8_t
  (* Create an LLVM module -- this is a "container" into which we'll
     generate actual code *)
  and the_module = L.create_module context "Iris" in

  (* Use ltype_map (line 106) to use this function without passing in tmap *)
  let ltype_of_typ tmap = function
      A.Int               -> i32_t
```

```
48      | A.Bool              -> i1_t
49      | A.Float             -> float_t
50      | A.Void              -> void_t
51      | A.String            -> string_t
52      | A.Char              -> i8_t
53      | A.Object (ctyp)     ->
54        let type_not_found_err = "Class type " ^ ctyp ^ " is not defined."
55        in
56          (try
57            StringMap.find ctyp tmap
58          with
59            | Not_found -> raise (Failure (type_not_found_err)))
60        in
61    let get_typ_name = function
62        A.Object (ctyp) -> ctyp
63      | _ -> raise (Failure "This is not an object type")
64
65    in let is_object typ =
66      (try let _ = get_typ_name typ
67           in true
68        with
69        | Failure _ -> false)
70
71      in
72      let formal_typ_of_typ_map type_map typ = (match typ with
73              | A.Object (_) -> L.pointer_type (L.pointer_type (ltype_of_typ type_map typ))
74              | _               -> ltype_of_typ type_map typ)
75
76    in
77      let populate_type_map context (counter, (tmap, chunguini)) sc_decl =
78        let c_name   = sc_decl.sclass_name
79        (* build vritual table *)
80
81        and all_vars = sc_decl.svars
82        and perm_len = List.length sc_decl.spermitted
83          in
84            let perm_arr = L.array_type string_t perm_len
85            in let (typs, _) = (List.split all_vars)
86        (* i32_t to hold position in array of vtables *)
87              in let all_typs =
88                i32_t :: (i32_t :: (perm_arr :: (List.map (formal_typ_of_typ_map tmap) typs)))
89                and (new_var_index_map, _) =
90                  List.fold_left (fun (acc, count) (typ, var_name) ->
91                    ((StringMap.add var_name (count, typ) acc), (count + 1)) ) (StringMap.empty, 3) all_vars
92                in let (new_fun_index_map, _) =
93                  List.fold_left
```

```
                    (fun (acc, count) fun_decl ->
                      ((StringMap.add fun_decl.sfname (count, fun_decl) acc), (count + 1)) )
                    (StringMap.empty, 0) sc_decl.smeths

    in
        let arr_vars = Array.of_list all_typs
        in
          let new_struct = (L.named_struct_type context c_name)
          in
            let _ = L.struct_set_body new_struct arr_vars false
    in (counter + 1, (StringMap.add c_name new_struct tmap, StringMap.add c_name ((sc_decl, ((i1_t),
        (L.const_int i1_t 0))), (new_var_index_map, new_fun_index_map)) chunguini))

    in
      let (_, (type_map, chunguini)) =
        List.fold_left (populate_type_map context) (0, (StringMap.empty, StringMap.empty)) classes

        in let ltype_map = ltype_of_typ type_map
  in
  let formal_typ_of_typ typ = (match typ with
  | A.Object (_) -> L.pointer_type (L.pointer_type (ltype_map typ))
  | _            -> ltype_map typ)

in
(* making vtable types *)
let make_vtable_typ context (guini, vtable_list) sc_decl =
  let all_funcs = sc_decl.smeths
  in
    (* list of function return/formal types (styp) *)
    let all_ftypes  = List.map (fun sfunc -> sfunc.styp) all_funcs
    and all_formals = List.map (fun sfunc -> fst (List.split sfunc.sformals)) all_funcs

      (* list of function return/formal lltypes *)
      in let ret_fltyps    = List.map formal_typ_of_typ all_ftypes
      and formal_fltyps = (List.map (fun typs -> Array.of_list (List.map formal_typ_of_typ typs)) all_formals )
      in
        let all_args = (List.combine ret_fltyps formal_fltyps)
        in
          let func_ltypes = List.map (fun (ret, forms) -> (L.pointer_type (L.function_type ret forms))) all_args
          in
            (* vtable lltype *)
            let vtable_struct = L.named_struct_type context (sc_decl.sclass_name ^ "_vtable")
            in let _ = L.struct_set_body vtable_struct (Array.of_list func_ltypes) false
            in let dummy_args = L.const_named_struct vtable_struct [| L.const_int i32_t 0 |]
            in let dummy_vtable_inst =
                L.define_global (sc_decl.sclass_name ^ "_vtable_data") dummy_args the_module
```

```
                in
                  let ((decl, (_, _)), mem_maps) = StringMap.find sc_decl.sclass_name guini
            in ((StringMap.add sc_decl.sclass_name
                ((decl, (vtable_struct, dummy_vtable_inst)),
                mem_maps) guini), vtable_list @ [L.pointer_type vtable_struct])

    in let (chunguini, _) = List.fold_left (make_vtable_typ context) (chunguini, []) classes
    in let update_ltypes (sc_decl : sclass_decl) =
        let curr_lltype = (ltype_map (A.Object sc_decl.sclass_name))
          in
            let struct_arr = (L.struct_element_types curr_lltype)
        in
                let _ = Array.set struct_arr 0 (L.pointer_type (get_vtable_lltype chunguini sc_decl.sclass_name))
                in
                  L.struct_set_body curr_lltype struct_arr false
      in let _ = List.map update_ltypes classes

  (* built-ins *)
    in let print_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |]
    in let print_func : L.llvalue =
      L.declare_function "printf" print_t the_module in

    let printerr_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |]
    in let printerr_func : L.llvalue =
      L.declare_function "printerr" printerr_t the_module in

    let getline_t : L.lltype =
      L.var_arg_function_type (L.pointer_type string_t) [| |]
    in let getline_func : L.llvalue =
      L.declare_function "readaline" getline_t the_module in

    let streq_t : L.lltype =
      L.var_arg_function_type i1_t [| string_t ; string_t|]
    in let streq_func : L.llvalue =
      L.declare_function "streq" streq_t the_module in

    let permi_t : L.lltype =
      L.var_arg_function_type void_t [| string_t ; (L.pointer_type string_t) ; i32_t|]
    in let permi_func : L.llvalue =
      L.declare_function "class_permitted" permi_t the_module in

    let build_class_functions (cls : sclass_decl) =
      let curr_name = cls.sclass_name in

```

```
186     (* Fill in the body of the given function *)
187      let build_function_llvals (acc : (L.llvalue * sfunc_decl) list) (func : sfunc_decl) =
188       let func_name = (if (curr_name = "Main") && (func.sfname = "main")
189         then "main"
190         else func.sorigin ^ "_" ^ func.sfname)
191       in
192         if ((func.sorigin = curr_name)) then
193           let func_type =
194             L.function_type (formal_typ_of_typ func.styp)
195               (Array.of_list (List.map formal_typ_of_typ (fst (List.split func.sformals))))
196           in
197             let func_ll = (L.define_function func_name func_type the_module, func)
198               in func_ll :: acc
199             else (Option.get (L.lookup_function func_name the_module), func) :: acc
200       in
201    let build_function_body (func_ll : (L.llvalue * sfunc_decl)) =
202      let the_function = snd func_ll in
203      let the_function_llval = fst func_ll in
204      let formal_list = (snd func_ll).sformals in
205      (if (the_function.sorigin <> curr_name) then ()
206       else
207      let builder = L.builder_at_end context (L.entry_block (the_function_llval)) in
208      let format_str = L.build_global_stringptr "%s" "fmt" builder
209      and format_str_ln = L.build_global_stringptr "%s\n" "fmt" builder
210      and format_str_int = L.build_global_stringptr "%d" "fmt" builder in
211       let rec expr builder m ((t, e) : sexpr)  =
212       let check_permit class_location mem_lval =
213         let permit_length_ptr = L.build_struct_gep mem_lval 1 "permit_length_ptr" builder in
214         let permit_length = L.build_load permit_length_ptr "permit_length" builder in
215         let permit_list = L.build_struct_gep mem_lval 2 "permit_list_ptr" builder in
216         let str_ptr =
217           L.build_gep permit_list [| L.const_int i32_t 0 ; L.const_int i32_t 0 |] "str_ptr" builder in
218         let class_string = L.build_global_stringptr class_location "class_location" builder in
219         let _ = L.build_call permi_func [| class_string ; str_ptr ; permit_length |] "" builder
220  in () in
221       match e with
222           SLiteral   i -> (L.const_int i32_t i, m)
223         | SBoolLit   b -> (L.const_int i1_t (if b then 1 else 0), m)
224         | SStringLit s -> (L.build_global_stringptr s "global_str" builder, m)
225         | SFliteral  s -> (L.const_float_of_string float_t s, m)
226         | SId n -> let (t, llval) =
227           (try (StringMap.find n m) with Not_found -> raise (Failure ("couldn't find " ^ n))) in
228                      (if (is_object t) then (llval, m) else (L.build_load llval n builder, m))
229         | SUnop (uop, e) ->
230           let (lval, m') = expr builder m e
231           in (match uop with
232               Neg -> (match t with
```

```
                    A.Int -> (L.build_neg lval "tmp" builder, m')
                  | A.Float -> (L.build_fneg lval "tmp" builder, m')
                  | _ -> raise (Failure "negation only for int or float"))
    | Not -> (L.build_not lval "tmp" builder, m'))
| SBinop(e1, bnop, e2) ->
    let (e1', m') = expr builder m e1 in
    let (e2', m'') = expr builder m' e2 in
    let (typ, _) = e1 in
    let llval = if typ = A.Float then (match bnop with
      A.Add     -> L.build_fadd
    | A.Sub     -> L.build_fsub
    | A.Mult    -> L.build_fmul
    | A.Div     -> L.build_fdiv
    | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
    | A.Neq     -> L.build_fcmp L.Fcmp.One
    | A.Less    -> L.build_fcmp L.Fcmp.Olt
    | A.Leq     -> L.build_fcmp L.Fcmp.Ole
    | A.Greater -> L.build_fcmp L.Fcmp.Ogt
    | A.Geq     -> L.build_fcmp L.Fcmp.Oge
    | A.And | A.Or ->
        raise (Failure "internal error: semant should have rejected and/or on float")
    ) e1' e2' "tmp" builder
    else if typ = A.String then (match bnop with
      A.Equal -> fst (expr builder m (A.Bool, SCall("Olympus", "streq", [e1 ; e2])))
      | _ -> raise (Failure "Invalid operation on string")
    )
  else (match bnop with
      A.Add     -> L.build_add
    | A.Sub     -> L.build_sub
    | A.Mult    -> L.build_mul
    | A.Div     -> L.build_sdiv
    | A.And     -> L.build_and
    | A.Or      -> L.build_or
    | A.Equal   -> L.build_icmp L.Icmp.Eq
    | A.Neq     -> L.build_icmp L.Icmp.Ne
    | A.Less    -> L.build_icmp L.Icmp.Slt
    | A.Leq     -> L.build_icmp L.Icmp.Sle
    | A.Greater -> L.build_icmp L.Icmp.Sgt
    | A.Geq     -> L.build_icmp L.Icmp.Sge
    ) e1' e2' "tmp" builder
    in
      (llval, m'')
| SDoubleOp (n, doubleop) -> let load_val = L.build_load (snd (StringMap.find n m)) n builder
   in let llval = (match t with
        A.Float ->
        let (e', _) = expr builder m (Float, SFliteral("1.0"))
        in
```

```
                (match doubleop with
                  A.PPlus  -> L.build_fadd load_val e' "tmp" builder
                | A.MMinus -> L.build_fsub load_val e' "tmp" builder)
            | A.Int ->
              let (e', _) = expr builder m (Int, SLiteral(1))
              in
                (match doubleop with
                  A.PPlus -> L.build_add load_val e' "tmp" builder
                | A.MMinus -> L.build_sub load_val e' "tmp" builder)
            | _     -> raise (Failure "binop not implemented for this type"))
           in let _ = L.build_store llval (snd (StringMap.find n m)) builder
          in (llval, m)
      | SAssign (n, e) -> let (e', m') = (expr builder m e) in
          let (t, llval) = (try (StringMap.find n m)
            with Not_found -> raise (Failure ("couldn't find " ^ n))) in
          (if (is_object t) then
            let lltypel = formal_typ_of_typ t
            and lltyper = L.type_of e'
            in let bitcasted = (if lltyper <> lltypel
              then L.build_bitcast e' lltypel "cast_assign" builder else e')
            in
            let temp = L.build_load bitcasted "temp" builder in
            let _ = L.build_store temp llval builder
            in (llval, m')
          else let _ = (L.build_store e' llval builder) in (llval, m'))
      | SDeclAssign (t, n, e) ->
        let e' = fst (expr builder m e)
        in let lltypel = formal_typ_of_typ t
        in let lltype = ltype_map t
        in let m' = (if (is_object t) then
          let local = L.build_alloca (L.pointer_type lltype) n builder in
          let lltyper = L.type_of e'
          in let bitcasted = (if lltyper <> lltypel
              then L.build_bitcast e' lltypel "cast_assign" builder else e')
          in
          let temp = L.build_load bitcasted "temp" builder in
           let _ = L.build_store temp local builder in
           StringMap.add n (t, local) m
        else
        let local = L.build_alloca lltype n builder
          in let _ = L.build_store e' local builder
          in StringMap.add n (t, local) m)
              in (e', m')
      | SClassVarAssign(name, var, e) ->
          let e' = fst (expr builder m e)
          and (typ, lvalptr) = (try (StringMap.find name m)
            with Not_found -> raise (Failure ("couldn't find " ^ name)))
```

```
      in let lval = L.build_load lvalptr "temp" builder
      in
          let cname = get_typ_name typ
          in let var_typ = get_mem_var chunguini cname var
          in
              let gep = L.build_struct_gep lval (get_var_index chunguini cname var) (name ^ var) builder
              in let obj_class = get_class_decl chunguini cname
              in let is_permit = List.exists (fun (_, var_n) -> var_n = var) obj_class.spermitted_vars
              in let _ = (if (not is_permit) then () else check_permit curr_name lval)
              in (if (is_object var_typ) then
                let lltyper = L.type_of e'
                and lltypel = formal_typ_of_typ var_typ
                in let bitcasted = (if (lltypel <> lltyper)
                  then L.build_bitcast e' lltypel "cast_assign" builder else e')
                in let _ = L.build_store bitcasted gep builder
                in (gep, m)
                else let _ = (L.build_store e' gep builder) in (gep, m))
    | SCall("Olympus", "print", [e]) ->
      (L.build_call print_func [| format_str ; (fst (expr builder m e)) |]
      "printf" builder, m)
    | SCall("Olympus", "println", [e]) ->
      (L.build_call print_func [| format_str_ln ; (fst (expr builder m e)) |]
      "printf" builder, m)
    | SCall("Olympus", "printi", [e]) ->
        (L.build_call print_func [| format_str_int ; (fst (expr builder m e)) |]
        "printf" builder, m)
    | SCall("Olympus", "printerr", [e]) ->
        (L.build_call printerr_func [| (fst (expr builder m e)) |]
        "printf" builder, m)
    | SCall("Olympus", "streq", [e_str1 ; e_str2]) ->
        (L.build_call streq_func [| (fst (expr builder m e_str1)) ; (fst (expr builder m e_str2))|]
        "streq" builder, m)
    | SCall("Olympus", "readaline", []) ->
      let call = L.build_call getline_func [| |] "readaline" builder in
      let bcast = L.build_bitcast call (L.pointer_type string_t) "temp" builder in
      (L.build_load bcast "get_temp" builder, m)
    | SCall(caller, func_name, e_list) ->
      let is_univ =
          (try let _ = (ltype_map (Object(caller))) in true
            with _ -> false)
      in
        let ((class_name, vtable), arg_lls) =
          (if is_univ
            then ((caller, get_vtable_ll chunguini caller), (fst (List.split (List.map (expr builder m) e_list))))
          else
            let (typ, lvalptr) =
              (try StringMap.find caller m
```

```
374                     with Not_found ->
375                       raise (Failure
376                         ("codegen.ml " ^ (string_of_int __LINE__) ^  ": " ^ caller ^ " of " ^ func_name ^ " not found")))
377                   in let lval = L.build_load lvalptr "temp" builder
378                  in let cname = (get_typ_name typ)
379                in let fun_encap = (get_fun_decl chunguini cname func_name).sencap
380                in let _ = (if (fun_encap <> "permit:") then () else check_permit curr_name lval)
381              in let vtable_ptr = L.build_struct_gep lval 0 "vtable" builder
382                in ((cname, L.build_load vtable_ptr "vtable" builder),
383                    lvalptr :: (fst (List.split (List.map (expr builder m) e_list))))
384                      in let function_index = get_fun_index chunguini class_name (func_name)
385                      in let code_fun = L.build_struct_gep vtable function_index "fun_to_call" builder
386                      in let func = L.build_load code_fun "function" builder
387                  in let convert_objs (llval_list : L.llvalue list) =
388                    let convert_arg ((ftyp, _), llvalr) =
389                     (if (is_object ftyp) then
390                     let lltypel = formal_typ_of_typ ftyp
391                     and lltyper = L.type_of llvalr
392                     in (if (lltypel <> lltyper) then L.build_bitcast llvalr lltypel "arg_cast" builder else llvalr)
393                    else llvalr)
394                   in let curr_fun = get_fun_decl chunguini class_name func_name
395                   in let curr_form_list = curr_fun.sformals
396                in List.map convert_arg (List.combine curr_form_list llval_list)
397                  in let converted_args = convert_objs arg_lls
398                    in let arg_arr = Array.of_list converted_args
399                    in let function_typ = get_fun_decl chunguini class_name func_name
400                    in let ret_ty = function_typ.styp
401                    in let result = (match ret_ty with
402                      A.Void -> ""
403                    | _ -> func_name ^ "_result")
404                    in let call = L.build_call func arg_arr result builder
405                  in (call, m)
406            | SClassVar(name, var) ->
407              let (typ, lvalptr) = StringMap.find name m
408            in let lval = L.build_load lvalptr "temp" builder
409              in
410                let cname = get_typ_name typ
411                in
412                    let gep = L.build_struct_gep lval (get_var_index chunguini cname var) (name ^ var) builder
413                   in let obj_class = get_class_decl chunguini cname
414                   in let is_permit = List.exists (fun (_, var_n) -> var_n = var) obj_class.spermitted_vars
415                   in let _ = (if (not is_permit) then () else check_permit curr_name lval) in
416                     (L.build_load gep (name ^ var) builder, m)
417            | SOpAssign (n, op, e) -> let load_val =  L.build_load (snd (StringMap.find n m)) n builder
418                  in
419                    let (e', m') = expr builder m e
420                    in let llval = (match t with
```

```
421                     A.Float -> (match op with
422                       A.Peq -> L.build_fadd
423                     | A.Meq -> L.build_fsub
424                     | A.Teq -> L.build_fmul
425                     | A.Deq -> L.build_fdiv)
426                       load_val e' "tmp" builder
427                   | A.Int -> (match op with
428                       A.Peq -> L.build_add
429                     | A.Meq -> L.build_sub
430                     | A.Teq -> L.build_mul
431                     | A.Deq -> L.build_sdiv)
432                       load_val e' "tmp" builder
433                   | _ -> raise (Failure "Arithmetic Assign not implemented for this type"))
434                  in let _ = L.build_store llval (snd (StringMap.find n m)) builder
435                    in (llval, m')
436           | SNewObject (n) ->
437             let t = A.Object n
438             in let local = L.build_malloc (ltype_map t) n builder in
439             let gep = L.build_struct_gep local 0 "vtable" builder in
440             let vtable_ll = get_vtable_ll chunguini n
441             in let _ = L.build_store vtable_ll gep builder in
442            let alloca = L.build_alloca (L.pointer_type (ltype_map t)) "temp" builder in
443            let perm_list = get_permit_list chunguini n in
444            let num_perms = List.length perm_list in
445
446             let num_perms_llval = L.const_int i32_t num_perms in
447             let gep_num_perms = L.build_struct_gep local 1 "num_permitted" builder in
448             let _ = L.build_store num_perms_llval gep_num_perms builder in
449
450             let perm_llval_arr = Array.of_list
451               (List.map (fun str -> L.build_global_stringptr str "temp_str" builder) perm_list) in
452             let perm_arr_llval = L.const_array string_t perm_llval_arr in
453             let gep_perm = L.build_struct_gep local 2 "permit_list" builder in
454             let _ = L.build_store perm_arr_llval gep_perm builder in
455
456             let _ = L.build_store local alloca builder in
457             (alloca, m) (* stores type of the local var so can be used in expr ^*)
458
459           | SNoexpr -> (L.const_int i32_t 0, m)
460
461       in
462
463       (* Each basic block in a program ends with a "terminator" instruction i.e.
464       one that ends the basic block. By definition, these instructions must
465       indicate which basic block comes next -- they typically yield "void" value
466       and produce control flow, not values *)
467       (* Invoke "instr builder" if the current block doesn't already
```

```
468             have a terminator (e.g., a branch). *)
469         let add_terminal builder instr =
470                                 (* The current block where we're inserting instr *)
471           match L.block_terminator (L.insertion_block builder) with
472             Some _ -> ()
473           | None -> ignore (instr builder) in
474
475         (* Imperative nature of statement processing entails imperative OCaml *)
476         let rec stmt (builder, map) = function
477             SBlock sl -> List.fold_left stmt (builder, map) sl
478             (* Generate code for this expression, return resulting builder *)
479           | SExpr e -> let (_, map') = expr builder map e in (builder, map')
480           | SReturn e -> let _ = match the_function.styp with
481                                   (* Special "return nothing" instr *)
482                                   A.Void -> L.build_ret_void builder
483                                   (* Build return statement *)
484                                 | A.Object(c_name) ->
485                                   let e' = fst (expr builder map e)
486                                   in let lltyper = L.type_of e'
487                                   and lltypel = formal_typ_of_typ (A.Object c_name)
488                                   in let bitcasted =
489                                     (if lltyper <> lltypel then L.build_bitcast e' lltypel "ret_bitcast" builder else e')
490                                   in L.build_ret bitcasted builder
491
492                                 | _ -> L.build_ret (fst (expr builder map e)) builder
493                          in (builder, map)
494           | SIf(pred, then_stmt, else_stmt) ->
495               let (bool_llval, map') = expr builder map pred in
496               (* Add "merge" basic block to our function's list of blocks *)
497               let merge_bb = L.append_block context "merge" the_function_llval in
498               (* Partial function used to generate branch to merge block *)
499               let branch_instr = L.build_br merge_bb in
500
501               let then_bb = L.append_block context "then" the_function_llval in
502               (* Position builder in "then" block and build the statement *)
503               let (then_builder, _) = stmt ((L.builder_at_end context then_bb), map') then_stmt in
504               (* Add a branch to the "then" block (to th merge block)
505               if a terminator doesn't already exist for the "then" block *)
506               let () = add_terminal then_builder branch_instr in
507
508                     (* Identical to stuff we did for "then" *)
509               let else_bb = L.append_block context "else" the_function_llval in
510               let (else_builder, _) = stmt ((L.builder_at_end context else_bb), map') else_stmt in
511               let () = add_terminal else_builder branch_instr in
512
513               (* Generate initial branch instruction perform the selection of "then"
514               or "else". Note we're using the builder we had access to at the start
```

```
515                 of this alternative. *)
516                 let _ = L.build_cond_br bool_llval then_bb else_bb builder in
517                 (* Move to the merge block for further instruction building *)
518
519                 (L.builder_at_end context merge_bb, map')
520
521               | SWhile (predicate, body) ->
522                   (* First create basic block for condition instructions -- this will
523                   serve as destination in the case of a loop *)
524                 let pred_bb = L.append_block context "while" the_function_llval in
525                       (* In current block, branch to predicate to execute the condition *)
526                 let _ = L.build_br pred_bb builder in
527                 (* Create the body's block, generate the code for it, and add a branch
528                 back to the predicate block (we always jump back at the end of a while
529                 loop's body, unless we returned or something) *)
530                 let body_bb = L.append_block context "while_body" the_function_llval in
531                 let (while_builder, _) = stmt ((L.builder_at_end context body_bb), map) body in
532                 let () = add_terminal while_builder (L.build_br pred_bb) in
533
534                       (* Generate the predicate code in the predicate block *)
535                 let pred_builder = L.builder_at_end context pred_bb in
536                 let (bool_val, _) = expr pred_builder map predicate in
537
538                       (* Hook everything up *)
539                 let merge_bb = L.append_block context "merge" the_function_llval in
540                 let _ = L.build_cond_br bool_val body_bb merge_bb pred_builder in
541                   (L.builder_at_end context merge_bb, map)
542
543             | SLocal (t, n) ->
544               let local = (if (is_object t) then
545                 L.build_alloca (L.pointer_type (ltype_map t)) n builder
546                 else
547                 L.build_alloca (ltype_map t) n builder)
548
549               in (builder, StringMap.add n (t, local) map)
550             | _ -> let not_implemented_err = "Codegen: not implemented yet!" in
551                   raise (Failure not_implemented_err)
552
553           (* Build the code for each statement in the function *)
554           in let formals_llvals = Array.to_list (L.params the_function_llval)
555         in let build_alloca_formals m ((t, n), l) =
556
557           let formal =
558             (if (is_object t)
559             then (let alloca = (L.build_alloca (L.pointer_type (ltype_map t)) "temp" builder)
560               in let temp = L.build_load l "temp" builder
561               in let _ = L.build_store temp alloca builder
```

```
562             in alloca)
563               else
564                 let formal = L.build_alloca (ltype_map t) n builder in
565                 let _ = L.build_store l formal builder in formal)
566             in (StringMap.add n (t, formal) m)
567
568               in let formal_map =
569                 List.fold_left build_alloca_formals StringMap.empty (List.combine formal_list formals_llvals)
570               in let (builder, _) = stmt (builder, formal_map) (SBlock the_function.sbody) in
571       (* Add a return if the last block falls off the end *)
572       add_terminal builder (match the_function.styp with
573           A.Void -> L.build_ret_void
574         | A.Float -> L.build_ret (L.const_float float_t 0.0)
575         | t -> L.build_ret (L.const_int (ltype_map t) 0)))
576     in
577     (* add logic that skips over inherited functions *)
578       let llval_list = List.rev (List.fold_left build_function_llvals [] cls.smeths )
579       in let _ = List.iter build_function_body llval_list
580       in let table_lltyp = get_vtable_lltype chunguini curr_name
581       in let table_llval = get_vtable_ll chunguini curr_name
582       in let table = L.const_named_struct table_lltyp (Array.of_list (fst (List.split llval_list)))
583       in let _ = L.delete_global table_llval
584   in
585       L.define_global (curr_name ^ "_vtable_data") table the_module
586
587     in let _ = List.map build_class_functions classes
588
589   in the_module
590
```

15.3 guini.ml

```
(* This module contains helper functions to access fields in chunguini,
used to keep track of information necessary to generate accurate LLVM code *)
(* Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk *)

module StringMap = Map.Make(String)
open Sast

'a StringMap.t -> string -> 'a
let get_class chunguini cname =
  let class_not_defined = "class " ^ cname ^ " not defined"
  in
    (try StringMap.find cname chunguini with
    Not_found -> raise (Failure class_not_defined))

(('a * ('b * 'c)) * 'd) StringMap.t -> string -> 'b
let get_vtable_lltype chunguini cname =
    let ((_, (vtable_lltyp, _)), _) = get_class chunguini cname
    in vtable_lltyp


(('a * ('b * 'c)) * 'd) StringMap.t -> string -> 'a
let get_class_decl chunguini cname =
  let ((decl, (_, _)), _) = get_class chunguini cname
in decl

((sclass_decl * ('a * 'b)) * 'c) StringMap.t -> string -> string list
let get_permit_list chunguini cname =
  let ((decl, (_, _)), _) = get_class chunguini cname
in decl.spermitted

('a * (('b * 'c) StringMap.t * 'd)) StringMap.t -> string -> string -> 'b
let get_var_index chunguini cname vname =
  let (_, (vmap, _)) = get_class chunguini cname
    in fst (try StringMap.find vname vmap with
    Not_found -> raise (Failure ("class " ^ cname ^ " does not contain variable " ^ vname)))

('a * (('b * 'c) StringMap.t * 'd)) StringMap.t -> string -> string -> 'c
let get_mem_var chunguini cname vname =
  let (_, (vmap, _)) = get_class chunguini cname
    in snd (try StringMap.find vname vmap with
    Not_found -> raise (Failure ("class " ^ cname ^ " does not contain variable " ^ vname)))

(('a * ('b * 'c)) * 'd) StringMap.t -> string -> 'c
let get_vtable_ll chunguini cname =
  let ((_, (_, vtable_ll)), _) = get_class chunguini cname
    in vtable_ll
```

```
('a * ('b * ('c * 'd) StringMap.t)) StringMap.t -> string -> string -> 'c
let get_fun_index chunguini cname fname =
  let (_, (_, fmap)) = get_class chunguini cname
  in fst (try StringMap.find fname fmap with
  Not_found -> raise (Failure ("class " ^ cname ^ " does not contain function " ^ fname) ))

('weak18 * ('a * ('b * 'c) StringMap.t)) StringMap.t -> string -> string -> 'c
let get_fun_decl chunguini cname fname =
  let (_, (_, fmap)) = get_class chunguini cname
  in snd (try StringMap.find fname fmap with
  Not_found -> raise (Failure ("class " ^ cname ^ " does not contain function " ^ fname) ))
```

15.4 semant.ml

```
open Ast
open Sast
open Gus

module StringMap = Map.Make(String)

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each defined class *)
  class_decl list -> sclass_decl list
  let check classes =

    (* dummy class definitions for use during semantic checking *)
    let olympus_print = { univ = true;
                          typ = Void;
                          fname = "print";
                          formals = [(String, "out")];
                          body = [Expr(Noexpr)]}
    in let olympus_println = { univ = true;
                               typ = Void;
                               fname = "println";
                               formals = [(String, "out")];
                               body = [Expr(Noexpr)]}
    in let olympus_printerr = { univ = true;
                                typ = Void;
                                fname = "printerr";
                                formals = [(String, "out")];
                                body = [Expr(Noexpr)]}
    in let olympus_printi = { univ = true;
                              typ = Void;
                              fname = "printi";
                              formals = [(Int, "i")];
                              body = [Expr(Noexpr)]}
    in let olympus_readaline = { univ = true;
                                 typ = String;
                                 fname = "readaline";
                                 formals = [];
                                 body = [Expr(Noexpr)]}
    in let olympus_streq = { univ = true;
                             typ = Bool;
                             fname = "streq";
                             formals = [(String, "a") ; (String, "b")];
                             body = [Expr(Noexpr)]}
```

```
  in let olympus_class = { class_name = "Olympus";
                           parent_name = "Object";
                           permitted = [];
                           mems = [("public",   (MemberFun olympus_print)
                                              :: (MemberFun olympus_readaline)
                                              :: (MemberFun olympus_println)
                                              :: (MemberFun olympus_printerr)
                                              :: (MemberFun olympus_printi)
                                              :: (MemberFun olympus_streq)
                                              :: [])] }
  in let object_class =  { class_name = "Object";
                           parent_name = "";
                           permitted = [];
                           mems = [] }
  in let big_chungus = List.fold_left build_chungus
                                      StringMap.empty
                                      (object_class :: (olympus_class :: classes))
    in let classes = object_class :: classes
      in let name_compare c1 c2 = compare c1.class_name c2.class_name in
      let check_dups checked a_class =
        let dup_err = "Compilation error: Duplicate class name " ^ a_class.class_name
          in match checked with
          (* No duplicate bindings *)
          (first_class :: _) when a_class.class_name = first_class.class_name ->
                raise (Failure dup_err)
          | _ -> a_class :: checked
  in let _ = List.fold_left check_dups [] (List.sort name_compare classes)

(* helper functions !!!! WOOOOOO *)

in let build_sast (cdecls : class_decl list) =

  let is_object = function
    | Object (n) -> let _ = find_class big_chungus n in true
    | _ -> false

    in let get_object_name = function
      | Object (n) -> n
      | _ -> "not an object!"

    in let mismatch_types err lhval rhval =
    (if lhval <> rhval then
        (if ((is_object lhval) && (is_object rhval)) then
          let lhcls = get_object_name lhval and rhcls = get_object_name rhval in
```

```
          (if (is_ancestor big_chungus rhcls lhcls) then
              true
          else
              raise (Failure ("class " ^ lhcls ^ " is not an ancestor of " ^ rhcls)))
      else raise (Failure (err)))
  else true)

in let check_class (scls_accum : sclass_decl list) (cls : class_decl) =
  let curr_class = cls.class_name
  and parent_class = cls.parent_name

  (*  *)
in let check_function (func : func_decl) =
  let _ = (match func.typ with
      Object(n) -> let _ = find_class big_chungus n in ()
    | _         -> ())
  in let rec check_expr m (e : expr) =
    match e with
      Literal l -> ((Int, SLiteral l), m)
    | BoolLit b -> ((Bool, SBoolLit b), m)
    | StringLit s -> ((String, SStringLit(s)), m)
    | Fliteral f -> ((Float, SFliteral(f)), m)
    | Id n -> (try let (t, _) = StringMap.find n m
                   in ((t, SId(n)), m)
               with Not_found ->
                 (try let (_, (t, _)) = find_var big_chungus curr_class n
                 in ((t, SClassVar("self", n)), m)
                 with _ -> raise (Failure ( "variable " ^ n ^ " not found"))))
    | Unop (uop, e) -> let ((t, e'), m') = check_expr m e in
      let wrong_type_err = "type: " ^ (string_of_typ t) ^
                           " is invalid for unop" ^ (string_of_uop Not)
                       in
                         let ty = (match uop with
                           Neg when t = Int || t = Float -> t
                         | Not when t = Bool -> Bool
                         | _  -> raise (Failure ( wrong_type_err)))
                         in
                      ((ty, SUnop(uop, (ty, e'))), m')
    | Binop (e1, op, e2) ->
          let ((t1, e1'), m') = check_expr m e1 in
          let ((t2, e2'), m'') = check_expr m' e2 in
          (* All binary operators require operands of the same type *)
          let same = t1 = t2 in
          (* Determine expression type based on operator and operand types *)
          let ty = match op with
            Add | Sub | Mult | Div when same && t1 = Int   -> Int
```

```
      | Add | Sub | Mult | Div when same && t1 = Float -> Float
      | Equal | Neq           when same               -> Bool
      | Less | Leq | Greater | Geq
                when same && (t1 = Int || t1 = Float) -> Bool
      | And | Or when same && t1 = Bool               -> Bool
      | _ -> raise (Failure ( "illegal binary operator " ^
                              string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                              string_of_typ t2 ^ " in " ^ string_of_expr e))
    in ((ty, SBinop((t1, e1'), op, (t2, e2'))), m'')
| DoubleOp (n, doubleop) -> let (typ, _) = (try StringMap.find n m
   with Not_found -> raise (Failure ( "Variable " ^ n ^ " not declared")))
   in (match typ with
                          Int   -> ((Int, SDoubleOp(n, doubleop)), m)
                        | Float -> ((Float, SDoubleOp(n, doubleop)), m)
                        | _     -> raise (Failure ((string_of_typ typ) ^
                                                   " is not a valid type for double op")))
| Call (caller, function_string, (args : expr list)) ->
  let err = "class " ^ caller ^ " not defined" in
  let _ = err in
  let ((_, encap), func_d) =
  (match caller with
  "self" -> find_func big_chungus curr_class function_string
  | _    ->
    let is_univ = (try let _ = find_func big_chungus caller function_string
                       in true
                   with _ -> false) in
    (if is_univ then let (tple, func_e) = find_func big_chungus caller function_string
      in (if (func_e.univ = true)
          then (tple, func_e)
          else raise (Failure (func_e.fname ^ " is not a class method")))
        else
      let (typ, _) = (try StringMap.find caller m
        with
            Not_found -> raise (Failure (caller ^ " is not a local variable")))
      in
        let cname = (match typ with
          | Object (c) -> c
          | _ -> raise (Failure ( (string_of_typ typ) ^ "Not an object")))
        in
          let (tple, func_e) = find_func big_chungus cname function_string
      in (if func_e.univ <> true
        then (tple, func_e)
        else raise (Failure (func_e.fname ^ " is a class method")))))
  in (if ((encap = "private:") && (caller <> "self"))
      then raise (Failure ( "function " ^ function_string ^ " is not accessible"))
```

```
    else
      let sxpr_list = List.map (check_expr m) args
          in let sl  = List.map (fun ((t, sexpr), _) -> (t, sexpr)) sxpr_list
            in let args = (try List.combine sl func_d.formals
              with _ -> raise (Failure ( "Number of arguments doesn't match")))
              in let _ = List.map (fun ((t1, _), (t2, _)) ->
                let  mismatch_err =
                    function_string ^ " argument of type " ^ string_of_typ t1
                    ^ " does not match argument of  " ^ string_of_typ t2
                in mismatch_types mismatch_err t2 t1) args
          in ((func_d.typ, SCall(caller, function_string, sl)), m))
| Assign (n, e) ->
  (try let var = StringMap.find n m
      and (sexpr, m') = check_expr m e in
      let typ1 = fst var and typ2 = fst sexpr in
      let  mismatch_err = "variable " ^ n ^ " has type " ^ string_of_typ (typ1)
      ^ ", but an expression with type " ^ string_of_typ (typ2)
      ^ " was found." in
      let _ = mismatch_types mismatch_err typ1 typ2 in
        ((typ2, SAssign(n, sexpr)), m')
  with Not_found ->
    (try let (_, var) = find_var big_chungus curr_class n
      and (sexpr, m') = check_expr m e in
      let typ1 = fst var and typ2 = fst sexpr in
      let  mismatch_err = "member " ^ n ^ " has type " ^ (string_of_typ (typ1) )
      ^ ", but an expression with type " ^ string_of_typ (typ2)
      ^ " was found." in
      let _ = mismatch_types mismatch_err typ1 typ2 in
        ((typ2, SClassVarAssign("self", n, sexpr)), m')
  with _ -> raise (Failure ( "variable " ^ n ^ " not found"))))
| DeclAssign (t, n, e) ->
  let (sexpr, m') = (check_expr m e) in
  let expr_type = fst sexpr in
  let mismatch_err = "variable " ^ n ^ " has type " ^ string_of_typ (t)
      ^ ", but an expression with type " ^ string_of_typ (fst sexpr)
      ^ " was found." in
  let _ = mismatch_types mismatch_err t expr_type in
  let m'' = StringMap.add n (t, n) m' in
      ((expr_type, SDeclAssign(t, n, sexpr)), m'')
| ClassVar (inst_name, mem) ->
  let (class_typ, _) = StringMap.find inst_name m
  in
    let cname = (match class_typ with
      Object (c) -> c
      | _ -> raise (Failure ( "variable " ^ inst_name ^ " not an object")))
```

```
    in
      let (encap_level, (mem_type, _)) = find_var big_chungus cname mem
      in
        (match encap_level with
          "private:" -> raise (Failure ( "member " ^ mem ^ " is not accessible"))
          | _       ->  ((mem_type, SClassVar(inst_name, mem)), m))

  | ClassVarAssign (inst_name, mem, e) ->
    let (class_typ, _) = StringMap.find inst_name m
    in
      let cname = (match class_typ with
        Object (c) -> c
        | _ -> raise (Failure ( "variable " ^ inst_name ^ " not an object")))
      in
        (* pull out type from chungus variable *)
        let (encap_level, (mem_type, _)) = find_var big_chungus cname mem
      in
        (match encap_level with
          "private:" -> raise (Failure ( "member " ^ mem ^ " is not accessible"))
        | _         -> let (sexpr, m') = (check_expr m e) in
        let expr_type = fst sexpr in
        let  mismatch_err = "class variable " ^ mem ^ " has type " ^ string_of_typ (mem_type)
        ^ ", but an expression with type " ^ string_of_typ (expr_type)
        ^ " was found." in
        let _ = mismatch_types mismatch_err mem_type expr_type in
          ((mem_type, SClassVarAssign(inst_name, mem, sexpr)), m'))
  | OpAssign (var, op, e) ->
        let _ = (try StringMap.find var m
          with Not_found -> raise (Failure ( "variable" ^ var ^ "has not been declared"))) in
        let ((t, e'), m'') = check_expr m e in
        (* Determine expression type based on operator and operand types *)
        let ty = match op with
          Peq | Meq | Teq | Deq when t = Int -> Int
        | Peq | Meq | Teq | Deq when t = Float -> Float
        | _ -> raise (Failure ( string_of_typ t ^ " not valid for op assign"))
      in ((ty, SOpAssign(var, op, (t, e'))), m'')
  | NewObject (n) -> let _ = find_class big_chungus n in
                       ((Object (n), (SNewObject (n))), m)
  | Noexpr -> ((Void, SNoexpr), m)

in let rec check_stmt m (s : stmt) =
    let not_implemented_err = "not implemented stmt: " ^ string_of_stmt s
  in let void_err = "void type cannot be used to declare variable: " ^ string_of_stmt s
  in match s with
    Expr e -> let (checked_expr, new_m) = check_expr m e in
```

```
  (SExpr(checked_expr), new_m)
| Local (t, n) -> (match t with
    Void -> raise (Failure ( void_err))
  | _ ->  (try let _ = StringMap.find n m in
               raise (Failure ( "local variable " ^ n ^ " already exists"))
           with Not_found -> ((SLocal (t, n)), StringMap.add n (t, n) m)))

| If(p, b1, b2) ->
  let (b, m1) = check_expr m p in
    let (typ, _) = b in
      let _ = (match typ with
      Bool -> ()
      | _ -> raise (Failure ( "Predicate in if statement is not a bool"))) in
  let (stmt1, _) = check_stmt m1 b1 in
  let (stmt2, _) = check_stmt m1 b2 in
  (SIf(b, stmt1, stmt2), m)

| While(p, s) ->
  let (b, m1) = check_expr m p in
    let (typ, _) = b in
      let _ = (match typ with
        Bool -> ()
      | _ -> raise (Failure ( "Predicate in while loop is not a bool"))) in
  let (stmt1, _) = check_stmt m1 s in
  (SWhile(b, stmt1), m)

| Return e -> let ((t, e'), m') = check_expr m e in
  (* func is the argument of check_func (to be written) *)
  let  mismatch_err =  "return gives " ^ string_of_typ t ^ " expected " ^
  string_of_typ func.typ ^ " in " ^ string_of_expr e in
  let _ = mismatch_t   (typ * class_name) StringMap.t
  (SReturn (t, e'), m')

| Block sl ->
  let rec check_stmt_list map slist = match slist with
      [Return _ as s]   -> [check_stmt map s]
    | Return _ :: _     -> raise (Failure ("nothing may follow a return"))
    | Block sl :: ss    -> check_stmt_list map (sl @ ss)  (* Flatten blocks *)
    | stmt1 :: ss ->
      let (checked_stmt, map') = (check_stmt map stmt1)
      in
        (checked_stmt, map') :: check_stmt_list map' ss
    | []          -> []
  in let ret_stmts = List.map (fun (s, _) -> s) (check_stmt_list m sl)
  in (SBlock(ret_stmts), m)
```

```
    | _ -> raise (Failure not_implemented_err)
    (* in let member_vars =  *)
    in let locals = List.fold_left
                    (fun acc (typ, name) -> StringMap.add name (typ, name) acc)
                    StringMap.empty func.formals

  in {suniv = func.univ;
      styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      sbody = (let (checked_block, _) = check_stmt locals (Block (func.body))
               in (match checked_block with
               SBlock(checked_stmt_list) -> checked_stmt_list
               | _ -> raise (Failure "whoops!")));
      sorigin = get_func_origin big_chungus curr_class func.fname;
      sencap = get_func_encap big_chungus curr_class func.fname}

(* check_encap, check_class... *)
  in let check_mem enc ((vars, permitted_vars), meths) (mem : member)  =
      (match mem with
       MemberFun(f) ->
         let sfunc = check_function f
         in ((vars, permitted_vars), (sfunc :: meths))
      | MemberVar(v) ->
       let typ = fst v
       in let _ =
       (match typ with
              Object(n) -> let _ = find_class big_chungus n in ()
            | _         -> ())
       in (match enc with
       "permit:" -> ((v :: vars, v :: permitted_vars), meths)
       | _ -> ((v :: vars , permitted_vars), meths)))

  in let sort_fcns (fs : sfunc_decl list) (ps : sfunc_decl list) =
    let replace_local (p : sfunc_decl) =
      if (get_func_origin big_chungus curr_class p.sfname = curr_class)
       then List.find (fun func1 -> func1.sfname = p.sfname) fs else p
    in
    let delete_local flist (f : sfunc_decl) =
      if (is_meth big_chungus parent_class f.sfname) then flist else f::flist
    in
    let ps_updated = List.map replace_local ps
    in
      ps_updated @ (List.fold_left delete_local [] fs)
```

```
      in let check_encap lists ((enc_level, mems) : encap) =
        List.fold_left (check_mem enc_level) lists mems

        in let ((vars, permitted_vars), meths) = List.fold_left check_encap
                                                                ((([], []), [])
                                                                cls.mems
        in
          let curr_meths = (if (curr_class <> "Object") then
          let sp_class = List.find (fun p_class -> p_class.sclass_name = parent_class)
                                   scls_accum
          in
            sort_fcns meths sp_class.smeths
          else meths)
          in let curr_vars = (if (curr_class <> "Object") then
            let sp_class = List.find (fun p_class -> p_class.sclass_name = parent_class)
                                     scls_accum
            in
              sp_class.svars @ (List.rev vars)
            else (List.rev vars))

          in let curr_perm_vars = (if (curr_class <> "Object") then
            let sp_class = List.find (fun p_class -> p_class.sclass_name = parent_class)
                                     scls_accum
            in
              sp_class.spermitted_vars @ (List.rev permitted_vars)
            else (List.rev permitted_vars))
    in
        { sclass_name = cls.class_name;
          sparent_name = cls.parent_name;
          spermitted = cls.permitted;
          svars = curr_vars;
          spermitted_vars = curr_perm_vars;
          smeths = curr_meths
        } :: scls_accum
  in List.fold_left check_class [] cdecls

(* semantically check all, then make sure main exists *)
in let sclasses = build_sast classes

  in let add_self (sclass : sclass_decl) =
    let add_self_to_func (sfunc : sfunc_decl) =
      let new_formals = (if sfunc.suniv
                         then sfunc.sformals
                         else (Object (sfunc.sorigin), "self") :: sfunc.sformals)
      in
```

```
        { suniv = sfunc.suniv;
          styp = sfunc.styp;
          sfname = sfunc.sfname;
          sformals = new_formals;
          sbody = sfunc.sbody;
          sorigin = sfunc.sorigin;
          sencap = sfunc.sencap
        }
    in
      { sclass_name = sclass.sclass_name;
        sparent_name = sclass.sparent_name;
        spermitted = sclass.sclass_name :: sclass.spermitted;
        svars = sclass.svars;
        spermitted_vars = sclass.spermitted_vars;
        smeths = List.map add_self_to_func sclass.smeths
      }

  in let classes_with_self = List.map add_self sclasses
in List.rev classes_with_self
```

15.5 gus.ml

```
(* Module for big_chungus (large lookup/symbol table) *)
(* Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk *)

module StringMap = Map.Make(String)
open Ast

let find_class chungus cname =
  let class_not_defined = "class " ^ cname ^ " not defined" in
  (try StringMap.find cname chungus with
  Not_found -> raise (Failure class_not_defined))

let find_parent_name chungus cname =
  let cdecl = find_class chungus cname
  in fst (fst cdecl)

let find_func chungus cname fname =
  let func_not_found = "function " ^ fname ^ " not defined " ^ "in class " ^ cname in
  let c = find_class chungus cname in
  let funcs = snd (snd c) in
  (try StringMap.find fname funcs with Not_found -> raise (Failure func_not_found))

let get_func_encap chungus cname fname =
  let ((_, encap), _) = find_func chungus cname fname in encap

let is_meth chungus cname fname =
  (try let _ = find_func chungus cname fname
  in true
with _ -> false)

let is_var chungus cname vname =
  (try let (_, (vars, _)) = find_class chungus cname
    in
      let _ = StringMap.find vname vars
      in true
      with _ -> false)

let get_func_origin chungus cname fname =
    let ((origin, _), _) = find_func chungus cname fname in origin


let find_var chungus cname vname =
  let var_not_found = "variable " ^ vname ^ " not defined " ^ "in class " ^ cname  in
  let (_, (vars, _)) = find_class chungus cname in
  (try StringMap.find vname vars with Not_found -> raise (Failure var_not_found))
```

```
let compare_fdecls fdecl1 fdecl2 =
  let pname = fdecl1.fname and cname = fdecl2.fname
  in
    if (fdecl1.univ <> fdecl2.univ)
      then raise (Failure ("Function " ^ cname ^ "must be univ"))

    else if (fdecl1.typ <> fdecl2.typ)
      then raise (Failure ("Function " ^ cname ^ "must return " ^ string_of_typ fdecl1.typ))
    else let all_formals = (try List.combine fdecl1.formals fdecl2.formals
      with _ -> raise (Failure ("Number of arguments in " ^ pname ^ " and " ^ cname ^ " do not match")))
    in List.fold_left (fun _ ((typ1, _), (typ2, _)) ->
      if (typ1 <> typ2) then raise (Failure ("formal types don't match")) else true) true all_formals

let dup_memvar symbols p_name (_, cvar_name) =
  (try let _ = find_var symbols p_name cvar_name
       in raise (Failure ("variable " ^ cvar_name ^ " already exists in " ^ p_name))
    with _ -> ())

  let rec is_ancestor chungus child ancestor =
    if (child <> ancestor) then
      (if (child = "Object") then false
      else
        let parent = find_parent_name chungus child in
        is_ancestor chungus parent ancestor)
    else true

  let build_chungus symbols c_decl =
    let parent_permit = (c_decl.parent_name, c_decl.permitted)
    in let _ = (if (c_decl.class_name <> "Object")
                then fst (fst (find_class symbols c_decl.parent_name)) else "object!")
    in
      let add_encap (var_m, fun_m) e =
        let encap = fst e
        in
          let add_member (var_map, fun_map) m =
            (match m with
              MemberVar (typ, name) ->
                let var_already_defined_err = "variable " ^ name ^ " has multiple definitions"
                in
                  (try let _ = StringMap.find name var_map
                    in raise (Failure (var_already_defined_err))
                  with Not_found -> ((StringMap.add name (encap, (typ, name)) var_map), fun_map))
             | MemberFun f ->
                let func_already_defined_err = "function " ^ f.fname ^ " has multiple definitions"
                in
```

```
                (try let _ = StringMap.find f.fname fun_map
                  in raise (Failure (func_already_defined_err))
                with Not_found ->
                  (var_map, (StringMap.add f.fname ((c_decl.class_name, encap), f) fun_map))))
        in
          List.fold_left add_member (var_m, fun_m) (snd e)
    in
      let (var_map, fun_map) =
        List.fold_left add_encap (StringMap.empty, StringMap.empty) c_decl.mems

      in let parent_class_vars = if (c_decl.class_name = "Object")
                                 then StringMap.empty
                                 else fst (snd (find_class symbols c_decl.parent_name))
        in let curr_var_list = StringMap.bindings var_map
        (* function is wrong *)
        in let add_parent_vars map (vname, (encap, bind)) =
            (try
              let _ = StringMap.find vname map
              in raise (Failure ("variable " ^ vname ^ " already exists in"))

            with _ -> StringMap.add vname (encap, bind) map)
        in let full_vmap = List.fold_left add_parent_vars parent_class_vars curr_var_list

      in let parent_class_funcs = if (c_decl.class_name = "Object")
                                  then StringMap.empty
                                 else snd (snd (find_class symbols c_decl.parent_name))
        in let parent_func_list = StringMap.bindings parent_class_funcs

        in let add_parent_funcs map (fname, ((origin, encap), f_decl)) =
            (try
              let ((_, child_encap), child_func) = StringMap.find fname map
              in let _ = if child_encap <> encap
                then raise (Failure "Encap types do not match")
              else compare_fdecls child_func f_decl
              in map

            with Not_found -> StringMap.add fname ((origin, encap), f_decl) map)

        in let full_fmap = List.fold_left add_parent_funcs fun_map parent_func_list

in  let symbol_value = (parent_permit, (full_vmap, full_fmap))
        in
        StringMap.add c_decl.class_name symbol_value symbols
```

```
let rec is_permitted chungus caller_name target_name func_name =
  let target_class = StringMap.find target_name chungus in
  let (funcs, _) = fst (snd target_class)
  in
    (try let (encap, _) = StringMap.find func_name funcs
    in
      (match encap with
        "private:" -> false
      | _ -> true)
    with Not_found ->
      is_permitted chungus caller_name (find_parent_name chungus target_name) func_name)
```

15.6 sast.ml

```
(* Semantically-checked Abstract Syntax Tree and functions for printing it *)
(* Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk  *)

open Ast

type sexpr = typ * sx
and sx =
    SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
  | SStringLit of string
  (* | SCharLit of string *)
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SDoubleOp of string * doubleop
  | SAssign of string * sexpr
  | SDeclAssign of typ * string * sexpr
  | SClassVarAssign of string * string * sexpr
  | SOpAssign of string * op_assign * sexpr
  | SCall of string * string * sexpr list (* TODO: finish *)
  | SClassVar of string * string
  | SNewObject of string
  | SNoexpr

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt
  | SLocal of bind

type sfunc_decl = {
    suniv     : bool;
    styp      : typ;
    sfname    : string;
    sformals  : bind list;
    sbody     : sstmt list;
    sorigin   : string;
    sencap    : string;
  }

(* a member of a class is either a function or a variable *)
type smember =
    SMemberVar of bind
  | SMemberFun of sfunc_decl
```

```
(* each encapsulation label (public, permit, private) has a list of members *)
type sencap = string * smember list

(*
  class_name: name of the class
  parent_name: name of the parent class (defaults to Object)
  mems: list of encaps
  permitted: names of classes with access to permit members
*)

type sclass_decl = {
  sclass_name : string;
  sparent_name : string;
  spermitted: string list;
  svars: bind list;
  (* spermittedvars: bind list; *)
  smeths: sfunc_decl list;  (* change to a tuple of (origin class, sfun_decl) *)
  (* spermittedmeths: sfunc_decl list; *)
  (* smems: sencap list; *)
}

type sprogram = sclass_decl list

(* Pretty-printing functions *)

sexpr -> class_name
let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLiteral(l) -> string_of_int l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFliteral(l) -> l
  | SStringLit(l) -> l
  (*| SSCharList(l) -> l *)
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SClassVarAssign(_, _, _) -> "unimplemented"
  | SCall(cname, fname, el) ->
      cname ^ "." ^ fname ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SClassVar(_, _) -> "classvar"
  | SNewObject(n) -> "new " ^ n ^ "()"
  | SNoexpr -> ""
  | _ -> "hi")     ^ ")"
```

```
let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
  | SLocal(t, s) -> string_of_typ t ^ s

let string_of_suniv = function
    true  -> "univ "
  | false -> ""

let string_of_sfdecl sfdecl =
  string_of_typ sfdecl.styp ^ " " ^ string_of_univ sfdecl.suniv ^
  sfdecl.sfname ^ "(" ^ String.concat ", " (List.map string_of_formal sfdecl.sformals) ^
  ") {\n" ^
  String.concat "" (List.map string_of_sstmt sfdecl.sbody) ^
  "}"



let rec string_of_svars = function
    []                  -> ""
  | bind :: rest -> "\n" ^ string_of_vdecl bind ^ string_of_svars rest


let rec string_of_sfuncs = function
    []                  -> ""
  | sfdecl :: rest -> "\n" ^ string_of_sfdecl sfdecl ^ string_of_sfuncs rest

let string_of_scdecl cdecl =
  "class " ^ cdecl.sclass_name ^ " of " ^ cdecl.sparent_name ^
      " (" ^ String.concat ", " cdecl.spermitted ^ ")" ^
      " {\n"


let string_of_sprogram classes =
  String.concat "\n" (List.map string_of_scdecl classes)
```

15.7 parser.mly

```
/* Ocamlyacc parser for Iris */
/*  Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk */

%{
open Ast
%}

%token SEMI COLON DOT LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN PEQ MEQ TEQ DEQ SELF
%token NOT EQ NEQ LT LEQ GT GEQ AND OR PPLUS MMINUS
%token RETURN IF ELSE FOR WHILE INT BOOL FLOAT VOID STRING UNIV
%token CLASS PUBLIC PERMIT PRIVATE OF NEW
%token <int> LITERAL
%token <bool> BLIT
%token <string> ID FLIT STRINGLIT
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN PEQ MEQ TEQ DEQ
%right DASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%right NOT

%%

program:
  class_decls_opt EOF { List.rev $1 }

class_decls_opt:
    /* nothing */ { [] }
  | class_decls_opt class_decl { $2 :: $1 }

class_decl:

    CLASS ID of_opt LPAREN class_opt RPAREN LBRACE encap_opt_list RBRACE
    { {
      class_name      = $2;
      parent_name     = $3;
      permitted       = $5;
```

```
      mems            = List.rev $8;
    } }

of_opt:
  /* nothing */ { "Object" }
  | OF ID       { $2 }

class_opt:
    /* nothing */ { [] }
  | class_list    { List.rev $1 }

class_list:
    ID                  { [$1] }
  | class_list COMMA ID { $3 :: $1 }

encap_opt_list:
  /* nothing */ { [] }
  | encap_opt_list encap_opt { (fst $2, List.rev (snd $2)) :: $1 }

encap_opt:
    /* mem_decls_opt                { ("", $1)  } */
  | PUBLIC COLON mem_decls_opt    { ("public:", $3)  }
  | PERMIT COLON mem_decls_opt    { ("permit:", $3)  }
  | PRIVATE COLON mem_decls_opt   { ("private:", $3) }

mem_decls_opt:
    /* nothing */ { [] }
 | mem_decls_opt member { $2 :: $1 }

member:
  | var_decl { MemberVar($1) }
  | fun_decl { MemberFun($1) }

fun_decl:
   typ UNIV ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { {
      univ = true;
      typ = $1;
      fname = $3;
      formals = List.rev $5;
      body = List.rev $8 } }
  | typ ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { {
      univ = false;
      typ = $1;
      fname = $2;
      formals = List.rev $4;
```

```
    body = List.rev $7 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { $1 }

formal_list:
    typ ID                   { [($1,$2)]     }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
    INT    { Int   }
  | BOOL   { Bool  }
  | FLOAT  { Float }
  | VOID   { Void  }
  | STRING { String }
  | ID     { Object($1) }

var_decl:
   typ ID SEMI { ($1, $2) }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI                               { Expr $1               }
  | RETURN expr_opt SEMI                    { Return $2             }
  | LBRACE stmt_list RBRACE                 { Block(List.rev $2)    }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7)        }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
                                            { For($3, $5, $7, $9)   }
  | WHILE LPAREN expr RPAREN stmt           { While($3, $5)         }
  | typ ID SEMI                             { Local($1, $2)         }


expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    LITERAL                         { Literal($1)              }
  | FLIT                            { Fliteral($1)             }
  | BLIT                            { BoolLit($1)              }
  | STRINGLIT                       { StringLit($1)            }
  | ID                              { Id($1)                   }
```

```
    | expr PLUS   expr                    { Binop($1, Add,   $3)       }
    | expr MINUS  expr                    { Binop($1, Sub,   $3)       }
    | expr TIMES  expr                    { Binop($1, Mult,  $3)       }
    | expr DIVIDE expr                    { Binop($1, Div,   $3)       }
    | expr EQ     expr                    { Binop($1, Equal, $3)       }
    | expr NEQ    expr                    { Binop($1, Neq,   $3)       }
    | expr LT     expr                    { Binop($1, Less,  $3)       }
    | expr LEQ    expr                    { Binop($1, Leq,   $3)       }
    | expr GT     expr                    { Binop($1, Greater, $3)     }
    | expr GEQ    expr                    { Binop($1, Geq,   $3)       }
    | expr AND    expr                    { Binop($1, And,   $3)       }
    | expr OR     expr                    { Binop($1, Or,    $3)       }
    | MINUS expr %prec NOT                { Unop(Neg, $2)              }
    | NOT expr                            { Unop(Not, $2)              }
    | ID   PPLUS                          { DoubleOp($1, PPlus)        }
    | ID   MMINUS                         { DoubleOp($1, MMinus)       }
    | ID   PEQ    expr                    { OpAssign($1, Peq, $3)      }
    | ID   MEQ    expr                    { OpAssign($1, Meq, $3)      }
    | ID   TEQ    expr                    { OpAssign($1, Teq, $3)      }
    | ID   DEQ    expr                    { OpAssign($1, Deq, $3)      }
    | ID ASSIGN   expr                    { Assign($1, $3)             }
    | typ ID ASSIGN expr %prec DASSIGN    { DeclAssign($1, $2, $4)     }
    | ID DOT ID ASSIGN expr               { ClassVarAssign($1, $3, $5) }
    | ID DOT ID                           { ClassVar($1, $3)           }
    | ID DOT ID LPAREN args_opt RPAREN    { Call($1, $3, $5)           }
    | SELF DOT ID LPAREN args_opt RPAREN  { Call("self", $3, $5)       }
    | ID LPAREN args_opt RPAREN           { Call("self", $1, $3)       }
    | NEW ID LPAREN RPAREN                { NewObject($2)              }
    | LBRACK args_opt RBRACK              { Call("List", "List", $2)   }
    | LPAREN expr RPAREN                  { $2 }

args_opt:
    /* nothing */ { [] }
    | args_list     { List.rev $1 }

args_list:
    expr                    { [$1] }
    | args_list COMMA expr    { $3 :: $1 }
```

15.8 ast.ml

```
(* Iris Abstract Syntax Tree and functions for printing it *)
(* Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk *)

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
          And | Or

(*   +=   -=   *=   /=   *)
type op_assign = Peq | Meq | Teq | Deq

type uop = Neg | Not

(*   ++   --   *)
type doubleop = PPlus | MMinus

type class_name = string
type instance_name = string

type typ = Int | Bool | Float | Void | Char | String | Object of class_name (* Class name *)

type expr =
    Literal of int
  | Fliteral of string (* change to float *)
  | BoolLit of bool
  | StringLit of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | DoubleOp of string * doubleop
  | Assign of string * expr
  | DeclAssign of typ * string * expr
  | ClassVarAssign of instance_name * string * expr (* string is variable name *)
  | OpAssign of string * op_assign * expr
  | Call of string * string * expr list
  | ClassVar of string * string
  | NewObject of string
  | Noexpr

type bind = typ * string

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Local of bind
```

```
(*
  univ: class method indicator
  typ: return type
  fname: function name
  fomals: args
  body: implementation of function
*)
type func_decl = {
    univ : bool;
    typ : typ;
    fname : string;
    formals : bind list;
    body : stmt list;
  }

(* a member of a class is either a function or a variable *)
type member =
    MemberVar of bind
  | MemberFun of func_decl

(* each encapsulation label (public, permit, private) has a list of members *)
type encap = string * member list

(*
  class_name: name of the class
  parent_name: name of the parent class (defaults to Object)
  mems: list of encaps
  permitted: names of classes with access to permit members
*)
type class_decl = {
  class_name : string;
  parent_name : string;
  permitted: string list;
  mems: encap list;
}

type program = class_decl list


(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
```

```
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_opAssign = function
    Peq -> "+="
  | Meq -> "-="
  | Teq -> "*="
  | Deq -> "/="

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let string_of_doubleop = function
    MMinus -> "--"
  | PPlus -> "++"

let string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | Char -> "char"
  | String -> "string"
  | Object(o) -> o

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StringLit(l) -> l
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | DeclAssign(t, v, e) ->
      string_of_typ t ^ " " ^ v ^ " = "^ string_of_expr e
  | ClassVarAssign(c, v, e) ->
      c ^ "." ^ v ^ " = "^ string_of_expr e
  | OpAssign(s, o, e) -> s ^ " " ^ string_of_opAssign o ^ " " ^ string_of_expr e
  | ClassVar(c, v) -> c ^ "." ^ v
```

```
  | Call(c, f, el) ->
      c ^ "." ^ f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | DoubleOp(_, _) -> "double op"(*string_of_expr v ^ string_of_doubleop o *)
  | NewObject(n) -> "new " ^ n ^ "()"
  | Noexpr -> ""

let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ "; " ^ string_of_expr e2 ^ "; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Local(t, id) -> string_of_typ t ^ " " ^ id ^ ";\n"


let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";"

let string_of_formal (t, id) = string_of_typ t ^ " " ^ id

let string_of_univ = function
    true  -> "univ "
  | false -> ""

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^ string_of_univ fdecl.univ ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map string_of_formal fdecl.formals) ^
  ") {\n" ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}"

let rec string_of_members = function
    []                          -> ""
  | (MemberVar(curr) :: rest) -> string_of_vdecl curr ^ "\n" ^ string_of_members rest
  | (MemberFun(curr) :: rest) -> string_of_fdecl curr ^ "\n" ^ string_of_members rest


let rec string_of_encaps = function
    []                  -> ""
  | (str, mems) :: rest -> str ^ "\n" ^ string_of_members mems ^ string_of_encaps rest
```

15.9 scanner.ml

```
(* Ocamllex scanner for Iris *)
(* Written by Ayda Aricanli, Trevor Sullivan, Valerie Zhang, Josh Kim, Tim Valk *)

{ open Parser }

let digit = ['0' - '9']
let digits = digit+

let single_char = [' ' '!' '#'-'&' '('-'[' ']'-'~']
let double_char = ['\\'']['\\' 't' 'r' 'n' 'b']

let single_apo = '\\' '''
let double_apo = '\\' '"'

let char = single_char | double_char | single_apo | '"'
let string_char = single_char | double_char | double_apo | '''

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf }     (* Whitespace *)
| ".."       { singleComment lexbuf }       (* Comments *)
| ".~*"      { multiComment lexbuf }        (* Comments *)
| '.'        { DOT }
| '('        { LPAREN }
| ')'        { RPAREN }
| '{'        { LBRACE }
| '}'        { RBRACE }
| '['        { LBRACK }
| ']'        { RBRACK }
| ';'        { SEMI }
| ':'        { COLON }
| ','        { COMMA }
| '+'        { PLUS }
| '-'        { MINUS }
| '*'        { TIMES }
| '/'        { DIVIDE }
| "++"       { PPLUS }
| "--"       { MMINUS }
| '='        { ASSIGN }
| "+="       { PEQ }
| "-="       { MEQ }
| "*="       { TEQ }
| "/="       { DEQ }
| "=="       { EQ }
| "!="       { NEQ }
| '<'        { LT }
| "<="       { LEQ }
| ">"        { GT }
| ">="       { GEQ }
| "&&"       { AND }
```

```
  | "||"      { OR }
  | "!"       { NOT }
  | "univ"    { UNIV }
  | "class"   { CLASS }
  | "of"      { OF }
  | "new"     { NEW }
  | "public"  { PUBLIC }
  | "permit"  { PERMIT }
  | "private" { PRIVATE }
  | "if"      { IF }
  | "else"    { ELSE }
  | "for"     { FOR }
  | "while"   { WHILE }
  | "return"  { RETURN }
  | "int"     { INT }
  | "bool"    { BOOL }
  | "float"   { FLOAT }
  | "void"    { VOID }
  | "string"  { STRING }
  | "true"    { BLIT(true)  }
  | "false"   { BLIT(false) }
  | "self"    { SELF }
  | digits as lxm { LITERAL(int_of_string lxm) }
  | digits '.'  digit* as lxm { FLIT(lxm) }
  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
  | eof { EOF }
  | ['\"']string_char*['\"'] as lxm { STRINGLIT(String.sub lxm 1 (String.length lxm - 2)) }
  | _ as character { raise (Failure("illegal character " ^ Char.escaped character)) }

and multiComment = parse
  "*~." { token lexbuf }
| _    { multiComment lexbuf }

and singleComment = parse
  "\n"  { token lexbuf }
| _     { singleComment lexbuf }
```

15.10 Olympus.c

```c
/*
 *      Olympus.c
 *      by: Ayda Aricanli, Valerie Zhang, Trevor Sullivan, Josh Kim, Tim Valk
 *      Built-in class for standard I/O and other useful functions used in
 *      codegen
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

char** readaline() {
    char **s = malloc(8);
    *s = malloc(1000);
    fgets(*s, 1000, stdin);
    int i = 0;
    while ((*s)[i] != '\n') {
        i++;
    }
    (*s)[i] = '\0';
    return s;
}

bool streq(char *str1, char *str2) {
    int result = strcmp(str1, str2);

    if (!result) {
        return true;
    }
    return false;
}

void printerr(char* out) {
    fprintf(stderr, "%s\n", out);
}

// Used to check permitted classes in codegen, not accessaible to the user
void class_permitted(char *str1, char **permits, int list_len) {
    for (int i = 0; i < list_len; i++) {
        if (streq(str1, permits[i])) {
            return;
        }
    }
    printf("RUNTIME ERROR: Class %s does not have access!\n", str1);
    exit(1);
}
```