Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu), Classes: MW 16:40-18:30, URBAN 204, Office Hours: MW 14:00-15:00.

Project 2: IR Code Generation (Complete)

(Due Wednesday 2/1/12)

This project is to complete the IR code generator for the MINI compiler. The new areas are *new object*, *method declaration* and *invocation*, *object field*, *"this"* node, and *variable reference*.

Symbol Table and Typechecker

Both the symbol table and the typechecker are needed in this version of IR code generator. They are passed in as parameters to the `IrgenVisitor` constructor:

```
public IrgenVisitor(Table tab, TypeVisitor tv) { ... }
```

New Object

A `NewObj` node should be translated into a `CALL` to `"malloc"` with a single argument representing the size of the object, followed by a sequence of statements for initializing the object's class variables.

A special note: the initialization expressions saved in the the symbol table entry for this purpose are `AST` expressions, and need to be translated into `IR` expressions. Furthermore, this translation should be conducted in the proper scope environment — the `IrgenVisitor` variable `currClass` needs to point to the object's class.

Method Declaration

There are two issues to resolve. First, we need to construct an unique label for the method. Second, we need to figure out the values of the two middle parameters to a `FUNC` node, `varCnt` and `argCnt`. `varCnt` represents the total number of local variables, and `argCnt` represents the maximum number of arguments of a call in the method's body (excluding calls to system routines).

The solution for unique label is to concatenate the class name and the method name together, with an underscore (`_`) in between. (With one exception: for the method `main`, no change is needed.) To do this, you need to keep track of the class scope information in the IrgenVisitor. (Recall the use of `currClass` in the typechecker project.)

For `varCnt`, you simply count the local variables. In the symbol table class `MethodRec`, there is a method, `int localCnt()`, for this purpose. For `argCnt`, the value has to come from recursive visit to the method's body. Think about declaring a class variable at the top level to keep track of this value, and whenever a `Call` or `CallStmt` node is encountered, check to see if this variable needs to be updated.

Method Invocation

Handling method invocation (i.e. `Call` and `CallStmt` nodes) also involves two issues: constructing an unique label and setting up the access link.

For constructing an unique label, you first need to figure out the class that the method belongs to (which could be a different class than the current class in which the call happens.) The proper steps are:

- Perform typechecking on the `obj` component of the `Call/CallStmt` node, which should return an `ObjType` representing the `obj`'s class.

  An Important Detail: The `TypeVisitor` program has its own copy of `currClass` and `currMethod`. These two variables need to be set to the current environment before typechecking can work. Use `typeChecker.setClass(currClass)` and `typeChecker.setMethod(currMethod)` to do this.

- Look up the `MethodRec` with the class info.

- Finally, concatenate class name and method name together (with an underscore in between).

The access link is a pointer to the object record, which provides access to instance variables from within a method's body. To implement this link, you need to add the object record (which is the the IR translation of the `obj` component of the `Call/CallStmt` node) as the zeroth element of the argument list of the `Call/CallStmt` node. For instance, an AST `Call` node with two arguments will be translated into an IR `CALL` node with three arguments, in which the very first argument is the IR representation of the `obj` component.

Object Field

Object field in the form of a `Field` node is translated into a `FIELD` node

```
(Field obj var)  =>  (FIELD <obj's record> <var's idx>)
```

Note that to compute the variable's index, you need to know which class the variable belongs to. Do similar things as in `Call/CallStmt` nodes, i.e. performing typechecking on the `obj` component, and get the class info.

"This" Node

"This" represents a reference to the current class object. We assume that it is only used within a method's scope. Since a pointer to the current class's object record is passed in to the method as the zeroth parameter, the correct translation for `This` node should be `PARAM(0)`.

Variable Reference

In the IR tree representation, variables (i.e. `Id` nodes) are no longer referenced through their names. The correct forms for them dependent on their categories:

- *Class variables* — They are stored in an object record (which is allocated when a new object is created), and are accessed through offsets from the record's main pointer (i.e. starting address). A class variable's offset value is defined as the variable's index minus one. The variable's index can be obtained from the symbol table.

  When a standing-alone `Id` node is identified as a class variable, it should be translated into:

  ```
  (FIELD PARAM(0) <var's idx>-1)
  ```

  In this case, it is equivalent to having a `"This ."` prefix in front of the `Id`.

  Note that when an `Id` node appears inside a `Field` node, it is taken care of by the visit routine of the `Field` node. In that case, there is no need to recursively visit the `Id` node.

- *Local variables* — A method's local variables are stored in an abstract array `VAR`. Their indices correspond to their positions in the method declaration: the first variable is `(VAR 1)`, the second is `(VAR 2)`, and so on.

- *Method's parameters* — They are stored in an abstract array `PARAM`, and indexed by their positions in the method declaration as well: `(PARAM 1)`, `(PARAM 2)`, and so on.

Code Organization

Copy and decompress the file `322p2.tar`. Code organization is similar to the last project, with the exception that two additional modules, `symbol` and `typechk`, are included. Your program should be called `IrgenVisitor.java`, and should be placed in the directory `irgen`. The run script is now called `runi`.

What and How to Turn in Your Program

Submit your program `IrgenVisitor.java` through the "Dropbox" on the D2L class webpage.