Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu), Classes: MW 4:40-6:30pm, DLC304, Office Hours: MW 2-3pm.

Project 4: Symbol Tables
(Due Wednesday 11/16/11)

This project is to implement a visitor program for the MINI AST nodes to collect symbol information from the input program and store them in a symbol table.

MINI Compiler's Symbol Table

As discussed in class, MINI compiler's symbol table has a multi-level structure. At the top level, a single `Table` provides the main interface to the compiler. Under this level, there are separate `ClassRec`s for storing information about classes, `MethodRec`s for storing information about methods, and `VarRec`s for storing information about variables:

```
public class Table {
  private Hashtable<String,ClassRec> classes;
}

public class ClassRec {
  private Id id;
  private ClassRec parent;
  private Vector<VarRec> class_vars;
  private Hashtable<String,MethodRec> methods;
}

public class MethodRec {
  private Id id;
  private Type rtype;
  private Vector<VarRec> params;
  private Vector<VarRec> locals;
}

public class VarRec {
  public static final int CLASS=0, LOCAL=1, PARAM=2;
  private Id id;
  private Type type;
  private int kind;
  private int idx;
  private Exp init;
}
```

This set of symbol table definition files are provided to you.

The SymbolVisitor Program

Your task is to write a visitor program, `SymbolVisitor.java`, to collect symbol information from a given MINI program, and store the information in corresponding components of the symbol table.

`SymbolVisitor.java` takes the form of a Java visitor program, and implements the (provided) `TypeVI` interface defined over the MINI AST nodes. Since symbol information of a program exist only in declarations, the visitor only needs to visit the declaration sections of the program. Specifically, the visitor program should perform the following operations:

- Create an empty top-level table at the start of the visit.
- Upon visiting a class declaration, create a new `ClassRec` object, and add it to the top-level table.

- Upon visiting a method declaration, create a new `MethodRec` object, and add it to the `methods` table in the corresponding `ClassRec` object.
- Upon visiting a variable declaration in a class, create a new `VarRec` object, and add it to the `class_vars` list in the corresponding `ClassRec` object.
- Upon visiting a parameter or a variable declaration in a method, create a new `VarRec` object, and add it to the `params` list or the `locals` list in the corresponding `MethodRec` object.

Additional Requirements

Error Handling The provided symbol table definition code already includes error-handling code. A `SymbolException` will be thrown when a violation is caught. Your program can simply rely on the provided code to catch many errors. However, the following cases are beyond the scope of the provided code's low-level error detection, and need to be handled directly by your visitor program:

- *"Method* `main` *is missing"* — Every program must have one (and only one) `main` method.
- *"The main class contains variable declarations"* — The static class containing the `main` method is called the main class, and it is not allowed to have variable declarations.
- *"A name in var-decl is already defined as a param in the same method scope"* — Params and local vars of the same method are in the same scope, hence their names have to be distinct.

In each of these cases, your program should throw a `SymbolException`.

Setting-up Class Hierarchy The MINI language allows subclass declarations. The class inheritance information in a program need to be captured and recorded in the symbol table:

- In `ClassRec`, there is a `parent` pointer. For a subclass, it needs to point to the parent class's `ClassRec`.
- In `VarRec`, there is an `idx` field, which records the position of the variable in its scope. For instance, the first local variable and the first parameter in a method both have an index of 1. However, the first class variable in a subclass may have an index other than 1 due to inheritance. Ancestor's variables are "copied" into subclass's scope; new subclass variables are "appended" after them. The index computation for subclass variables need to take those ancestor's variables into consideration.

For both issues, a simple solution is to process parent's `ClassRec` before its children's. However, since a subclass declaration may appear before its parent's declaration in a MINI program, we need to perform a topological sort on class decls based on their inheritance relationship. The class notes *MINI Symbol Table* contains more info about how to implement the method `setupClassHierarchy(ClassDeclList cl)` for this purpose.

Project Organization

Copy and decompress the file `proj4-code.tar/proj4-code.zip`. You'll see four subdirectories:

- `ast` — containing the AST nodes and the visitor interface definitions
- `astpsr` — containing a parser for reading AST programs
- `symbol` — containing the symbol table definitions and a driver program `TestSymbol.java`
- `tst` — containing tests and their expected output.

There is also a `Makefile` and a `runs` script. Your program `SymbolVisitor.java` should be placed in the `symbol` subdirectory. To compile your program, do: `make symbol`; to run with a test, say `tst/test01.ast`, you do: `./runs tst/test01.ast`.

Submission

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