CS 164

Zoom:
https://berkeley.zoom.us/j/2348659471

Commands:
mvn clean package

Mac:
mvn clean package
```
java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy \--pass=.s
--dir src/test/data/pa2/sample/ --test

java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=.s \
--dir src/test/data/pa2/sample/expr_binary --test

java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=.s \
--dir src/test/data/pa2/sample/expr_unary --test
[above commands do not work]
```

Correct (?) command:
java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=.r
src/test/data/pa2/sample/expr_unary.py.ast

java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=.s
src/test/data/pa2/sample/expr_unary.py.ast

java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=.s
src/test/data/pa2/sample/strings.py.ast

```
java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy \--pass=.s
```
src/test/data/pa2/sample/expr_unary.py.ast `--test`

```
For output:
java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=.s
src/test/data/pa2/sample/expr_binary.py.ast
```

Windows:

mvn clean package

```
java -cp "chocopy-ref.jar;target/assignment.jar" chocopy.ChocoPy \--pass=.s
--dir src/test/data/pa2/sample/ --test
```

```
java -cp "chocopy-ref.jar;target/assignment.jar" chocopy.ChocoPy \--pass=.s
--dir src/test/data/pa2/sample/ --test
```

Tests:

Passed:

src/test/data/pa2/sample/bad_assign_expr.py.ast
src/test/data/pa2/sample/bad_concat.py.ast
src/test/data/pa2/sample/bad_duplicate_class.py.ast //
src/test/data/pa2/sample/bad_duplicate_class_member.py.ast //
src/test/data/pa2/sample/bad_duplicate_global_2.py.ast
src/test/data/pa2/sample/bad_expr_binary.py.ast
src/test/data/pa2/sample/bad_expr_unary.py.as
src/test/data/pa2/sample/bad_expr_if.py.ast
src/test/data/pa2/sample/bad_list_assign.py.ast
src/test/data/pa2/sample/bad_list_index.py.ast
src/test/data/pa2/sample/bad_type_id.py.ast
src/test/data/pa2/sample/bad_return_top.py.ast
src/test/data/pa2/sample/bad_var_assign.py.ast
src/test/data/pa2/sample/class_def_init.py.ast //
src/test/data/pa2/sample/expr_binary.py.ast
src/test/data/pa2/sample/expr_concat.py.ast
src/test/data/pa2/sample/expr_id.py.ast
src/test/data/pa2/sample/expr_if.py.ast
src/test/data/pa2/sample/stmt_if.py.ast
src/test/data/pa2/sample/expr_int.py.ast
src/test/data/pa2/sample/expr_lists.py.ast
src/test/data/pa2/sample/expr_list_index.py.ast
src/test/data/pa2/sample/expr_unary.py.ast
src/test/data/pa2/sample/expr_var_assign.py.ast
src/test/data/pa2/sample/stmt_for_lists.py.ast
src/test/data/pa2/sample/stmt_for_strings.py.ast //
src/test/data/pa2/sample/stmt_list_assign.py.ast

src/test/data/pa2/sample/stmt_var_assign.py.ast
src/test/data/pa2/sample/stmt_while.py.ast
src/test/data/pa2/sample/strings.py.ast
src/test/data/pa2/sample/stmt_while.py.ast

Test case out file:
java -cp "chocopy-ref.jar;target/assignment.jar" chocopy.ChocoPy --pass=rs --out src\test\data\pa2\sample\func_def_call_new.ast.typed src\test\data\pa2\sample\func_def_call.py

java -cp "chocopy-ref.jar:target/assignment.jar" chocopy.ChocoPy --pass=rs --out
src/test/data/pa2/sample/bad_duplicate_class_member_new.ast.typed
src/test/data/pa2/sample/bad_duplicate_class_member.py

Working on, can complete next, has some issues/partially implemented/dependency issues
Failed

src/test/data/pa2/sample/ast_coverage.py.ast
src/test/data/pa2/sample/bad_class_attr.py.ast
src/test/data/pa2/sample/bad_class_attr_type.py.ast
src/test/data/pa2/sample/bad_class_init_override.py.ast
src/test/data/pa2/sample/bad_class_init_return.py.ast
src/test/data/pa2/sample/bad_class_member_expr.py.ast
src/test/data/pa2/sample/bad_class_method_invoke.py.ast
src/test/data/pa2/sample/bad_class_method_override.py.ast
src/test/data/pa2/sample/bad_class_method_override_attr.py.ast
src/test/data/pa2/sample/bad_class_super.py.ast
src/test/data/pa2/sample/bad_duplicate_global.py.ast
src/test/data/pa2/sample/bad_func_def_call.py.ast
src/test/data/pa2/sample/bad_func_def_return.py.ast
src/test/data/pa2/sample/bad_local_assign.py.ast
src/test/data/pa2/sample/bad_nonlocal_global.py.ast
src/test/data/pa2/sample/bad_none_assign.py.ast << Have to add class def first
src/test/data/pa2/sample/bad_return_missing.py.ast
src/test/data/pa2/sample/bad_shadow_local.py.ast
src/test/data/pa2/sample/bad_shadow_local_2.py.ast

src/test/data/pa2/sample/bad_strings.py.ast << saw someone suggest using global vars
src/test/data/pa2/sample/bad_type_annotation.py.ast

src/test/data/pa2/sample/class_def_assign.py.ast
src/test/data/pa2/sample/class_def_attr.py.ast
src/test/data/pa2/sample/class_def_methods.py.ast
src/test/data/pa2/sample/decl_global_forward.py.ast
src/test/data/pa2/sample/decl_nonlocal_forward.py.ast
src/test/data/pa2/sample/func_def_call.py.ast
src/test/data/pa2/sample/stmt_for_strings.py.ast << if we fix bad_strings this is fixed/vice versa

src/test/data/pa2/sample/class_def_init.py.ast
???

```
In binary expr changed by deleting int from if statemennt

  if (!node.index.equals(Type.INT_TYPE)){
              err(node, "Index is of non is of non-integer type `%s`",
                    node.index);
            return node.setInferredType(Type.OBJECT_TYPE); // throw an
error actually
        } else if (!list.isListType()) {
           // node.list.setInferredType(Type.ListType);
            err(node, "`%s` is not a list type",
                    list);
            return node.setInferredType(Type.OBJECT_TYPE); // throw an
error actually
        }
```

1. How many passes does your semantic analysis perform over the AST? List the names of these
passes with their class names and briefly explain the purpose of each pass.

It takes 2 passes to go over the ASTs. The first pass is called Declaration Analyzer, it is used to store all the global variables, functions, classes, and methods, and create a hierarchy between these structures. It handles all the declarations outside of stmt nodes. The second pass is called Type Checker. This uses the symbol table to identify the inferred types for expressions and analyze the stmt nodes. (The symbol table was extended with VarTable<T> which also tracks hierarchies for the current table.) Both passes look out for errors over the nodes they examine.

2. What was the hardest component to implement in this assignment? Why was it challenging?

The hardest component to implement in this assignment was the idea of classes, superclasses, subclassess, and methods. Although these structures are separate they heavily rely on each other. For example, superclass methods and members need to be accessible to subclasses. Thus, we need to keep track of those outside of the superclass's block, as well as all of the class relationships. As such it was very difficult to map out the relationships and apply them to the concepts of scope and they had in order to have proper behavior. Testing for correctness took a long time, too.

3. When type checking ill-typed expressions, why is it important to recover by inferring the most specific type? What is the problem if we simply infer the type object for every ill-typed expression? Explain your answer with the help of examples in the student contributed/bad types.py test.

It is important to infer with the most specific type because subclasses will always have more or equal members and functions available to them than the superclasses as they inherit anything the superclass contains. If you default to the least specific type "object" for each ill-typed expression, it may be the case that using the expression (i.e. a member expression's identifier) would require accessing a member or method that wouldn't be available to object. In other words, an otherwise legal expression would be flagged as being legal.

For example, in the case of bad_type_id, the most specific type for x would probably be a float or int. However, if x defaults to being an object, you get an illegal expression in which an object is subtracting 1.