Lab 10: Design Patterns and uC

Learning objectives:

- Practice implementing code using design patterns
- Gain experience reading the uC specification

This labs aims to prepare you for homework 3, which includes the singleton design pattern. It also will help you understand how uC code is structured, which will be useful for projects 4 and 5.

Use the following commands to download and unpack the distribution code:

```
$ wget https://eecs390.github.io/lab/lab10/starter-files.tar.gz
$ tar xzf starter-files.tar.gz
```

1. *Singleton pattern.* Implement an ADT that emulates how a Scheme list works in Python. Recall that a proper list in Scheme is comprised of a sequence of pairs terminated by the empty list. To represent this, implement the `Pair` and `Nil` classes, filling in the methods specified in `scheme_list.py`.

   Modify the `Nil` class to be a singleton, so that there is only one empty list object, matching the behavior of Scheme `eq?` on empty lists.

2. *Visitor pattern.* Complete the implementation of the `GraphGen` class in `gen_call_graph.py`, which uses the visitor pattern to generate a call graph from an abstract syntax tree. A call graph is a graphical representation of the calls that are made between functions. Use the `__current_func` attribute to keep track of the current function definition, adding call targets to the corresponding entry in the `__call_map` attribute.

   You may find the documentation for ast.NodeVisitor helpful.

   When you are done, you can generate the call graph for the solution `list_parser.py` from a previous lab as follows:

   ```
   $ python3 gen_call_graph.py <path to list_parser.py>
   $ dot -Tpdf list_parser.dot -o list_parser.pdf
   ```

3. *uC errors.* Consider the following code samples from the uC language in Project 4. Consult the uC spec and the Project 4 spec to determine the appropriate errors that should be output by your semantic analyzer (if any) as well as which phase the errors are caught in.

uC Code	Error(s) Reported	Phase

`int foo(int a) {`
`  boolean a = false;`
`  return;`
`}`		

`void main(string[] args) {`
`  int b = 0;`
`  int c = b = 7;`
`  5 = 7;`
`}`		

`void int_to_long() {}`		

. . . continued on next page

uC Code	Error(s) Reported	Phase

`struct foo {`
`  boolean b;`
`};`
 	
`void main(string[] args) {`
`  new foo(null);`
`  new foo(true, false);`
`  new foo { 3 };`
`}`		

`void main(string[] args) {`
`  foo f = new foo(args[0]);`
`  bar(f);`
`}`
 	
`void bar(foo f) {`
`  println(f.s);`
`}`
 	
`struct foo {`
`  string s;`
`};`		