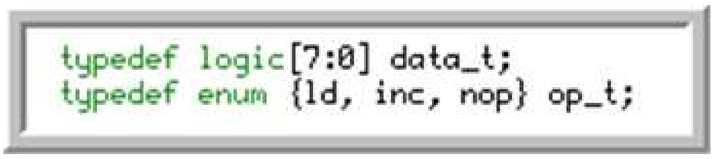
Lab 2: Creating Classes and Objects

In this lab we will create objects that we'll use in our counter's test bench. These objects will make it easier to randomize our stimulus.

The entire lab should be done in the `classes.sv` file. This will allow the `run.do` file to run the simulation.

Data Types

Our goal is to create a set library of three classes that we will use to generate stimulus to the counter. The classes contain two data members. One of each of the following types:



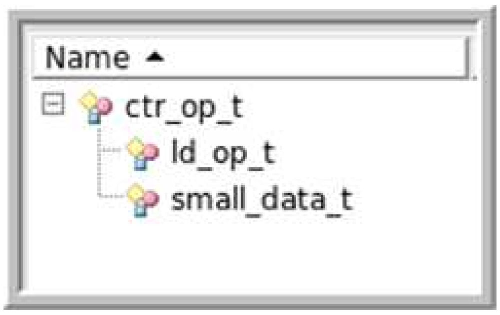
```
typedef logic[7:0] data_t;
typedef enum {ld, inc, nop} op_t;
```

- `data_t` is an eight bit register.
- `op_t` is an enumerated type with the three possible counter operations:
    - `ld` – load the register
    - `inc` – increment the register
    - `nop` – the value in the register remains unchanged.

These types are already defined in the lab file `classes.sv`.

Class Hierarchy

You will use the data types above to create a class library with the following features. The library will implement the following class hierarchy. This chart says that class `ld_op_t` extends class `ctr_op_t`.



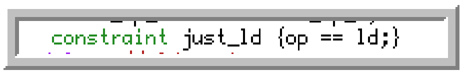
The `ctr_op_t` Class

The ctr_op_t class contains two data members and one method:

- There is one randomizable data member of type `data_t`
- There is one randomizable data member of type `op_t`
- There is one method. A function called `convert2string` which returns a string with the values in the object.

The `ld_opt_t` Class

This class is an extension of the `ctr_op_t` class. It contains a constraint that limits it to producing load operations:



```
  constraint just_ld {op == ld;}
```

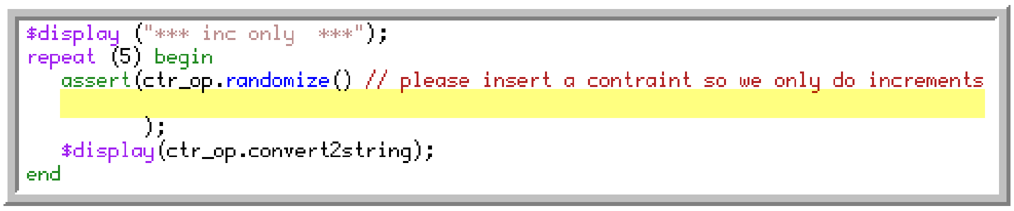
The `small_data_t` Class

This class extends the `ctr_op_t` class. It contains a constraint that limits it to producing data values that are less than one hundred and one.

It is similar to the constraint in `ld_op_t`.

Randomizing using the `with` statement

We want to generate five objects that only produce increment operations. Please modify the `randomize` call using a `with` statement to implement this constraint:

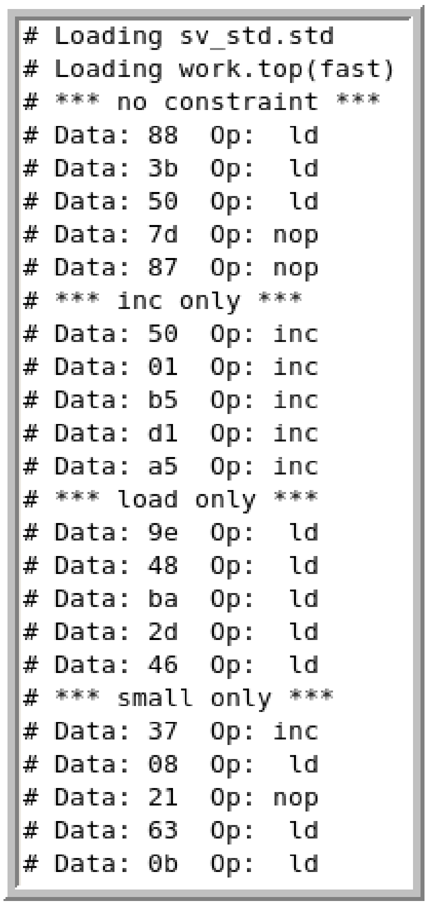


```
$display ("*** inc only  ***");
repeat (5) begin
   assert(ctr_op.randomize() // please insert a contraint so we only do increments

          );
   $display(ctr_op.convert2string);
end
```

Running the Example

If you have written this example correctly and execute the run.do script you will see the following:



```
# Loading sv_std.std
# Loading work.top(fast)
# *** no constraint ***
# Data: 88  Op:  ld
# Data: 3b  Op:  ld
# Data: 50  Op:  ld
# Data: 7d  Op: nop
# Data: 87  Op: nop
# *** inc only ***
# Data: 50  Op: inc
# Data: 01  Op: inc
# Data: b5  Op: inc
# Data: d1  Op: inc
# Data: a5  Op: inc
# *** load only ***
# Data: 9e  Op:  ld
# Data: 48  Op:  ld
# Data: ba  Op:  ld
# Data: 2d  Op:  ld
# Data: 46  Op:  ld
# *** small only ***
# Data: 37  Op: inc
# Data: 08  Op:  ld
# Data: 21  Op: nop
# Data: 63  Op:  ld
# Data: 0b  Op:  ld
```