# Gillian-WISL tutorial

## Setup instructions

VSCode should suggest an extension to install: install it. If it doesn't the extension is the "gillian debugger" extension on the marketplace. The configuartion in this repo should be enough for it to work out of the box on the lab machines. If it doesn't, please tell us :)

## General instructions:

You are provided with 2 files: SLL.wisl and DLL.wisl which contain lemmas and functions. The goal is to use wisl and its debugger to insert the right annotations for every function to be succesfully verified. Note that this tutorial uses the "manual" mode of Gillian, that is made for you to learn. In practice, Gillian can infer a lot of these annotations on its own. For every while loop, the invariant is already provided.

Please tell us when you encounter an error and you do not know what to do with it, or if the debugger crashes. You're the first students to do this during a lab. We had fun organising this lab, and hope you'll have fun too!

## Logical commands

- `unfold pred_name(param1, ... paramN)`
- `fold pred_name(param1, ... paramN)`
- `apply lemma_name(param1, ... paramN)`
- `if (bool_exp) { .. } else { .. }`
- `assert {bind: #x, #y, #z} P(#x, #y, #z)`
- invariants: they are written for you.

Some proofs will require that you bind variables using the `assert` statement. For example, let's imagine that you need to apply a lemma, and one of the parameters is the value contained in a cell, at the address contained by variable `t`. I.e, your state contains `t -> ?`, and you want to `apply some_lemma(t, ?)`. However, you do not have any program or logical variable available that contains the right value to use as second parameter.

One solution would be to use a program variable `v := [t]; [[ apply some_lemma(t, v) ]]`. However, that is not what we are trying to achieve, since the proof has now required to modify the program itself!

That's when `assert {bind: ..}` comes in:

```
[[ assert {bind: #v} (t -> #v) ]];
[[ apply some_lemma(t, #v) ]]
```

## Pitfalls

### Syntax

The syntax of WISL is a bit tricky:

- Parenthesize everything, operator precedence is off
- There is a semi-colon BETWEEN each command inside a block, but not at the end
- Logical commands are surrounded by `[[ .. ]]`.

### Automatic unfolding in preconditions

Even in manual mode, Gillian will automatically unfold any predicate in the precondition of a function if it is not recursive. In particular, the `dlist` predicate gets unfolded into its `dlseg` definition automatically.