Information-Flow Control, Exercises

EXERCISE 1:

Information-flow Security (i.e. *noninterference*)

- Examples
  - Is this program information-flow secure (i.e. *noninterfering*)?

```
h:=l+4;
l:=l-5;
```

  - Is this program information-flow secure (i.e. *noninterfering*)?

```
while ( (h + l) < 4 ) {
 l:=l+1
}
```

  - Is this program information-flow secure (i.e. *noninterfering*)?

```
l:=h;
l:=0;
```

EXERCISE 2:

Information-flow Control, check (type checking)

Note: the point is not to construct typing derivations, but rather, to test understanding of what the type system checks (e.g. what part of 'check' causes rejection?).

- This program is rejected by a type system for IFC. Why?

```
l:=false;
if h then {
 l:=true;
} else {
 skip;
}
out(l)
```

- This program is rejected by a type system for IFC. Why?

```
l := 0;
while h > 0 {
 l := l + 1;
 h := h - 1;
}
```

- This program is rejected by a type system for IFC. Why?

```
l := 1;
if h mod 2 == 1 then {
  h := 1;
  l := l + 1;
} else {
  h := 0;
  l := l + 1;
}
```

- This program is rejected by a type system for IFC. Why?

```
l:=h;
l:=0
```

- Consider program `if x>0 then y:=z+1 else x:=w; w:=z`. With Γ(w)=H, give the least restrictive labels for Γ(x), Γ(y), Γ(z), such that the program type-checks with initial context L.

EXERCISE 3

Information-flow Control, monitor (run-time reference monitor)

- Consider program `l:=0; if h>0 then {l:=1;} l:=2`. With Γ(`h`)=H and Γ(`l`)=L, explain step by step what the monitored program does when `h` is initially set to `0` and `1` respectively.

EXERCISE 4

Information-flow Challenge

- First four challenges
- Here

EXERCISE 5

Set up Paragon, and get the Paragon code in the assignment to compile and run.