HC-pitfalls

Reserved Keyword

"next" is a reserved keyword for phasers. The frontend will throw an error whenever a local or a field is name next.

IN clause

- Pitfall: It is forbidden to pass addresses of locals on stack to an async IN clause
- Solutions:
    - Use the OUT clause
    - Pass a pointer to a data-structure allocated on the heap

```
// Invalid example => Non-deterministic runtime error
int a = 0;
int * ptr = &a;

finish async IN(ptr) { *ptr = 10; }
// Here the address of 'a' stored in ptr may now
// points to a deallocated portion of the stack.
```

```
// Legal code
int * ptr = malloc(sizeof(int));

finish async IN(ptr) { *ptr = 10; }
// the address pointed to by ptr is
// allocated in the heap and remains so.
```

- Pitfall: Arrays allocated on stack are currently not supported
- Solutions:
    - Work with arrays allocated on the heap

```
// Invalid example => Non-deterministic runtime error
int a [10];

finish async IN(a) { a[0] = 1; }
// Here the address of 'a' may now points
// to a deallocated portion of the stack.
```

- Pitfall: Only a single IN, OUT or INOUT clause can be provided to an async.
- Solution:
    - IN can take several arguments (similarly to a method call invocation).

```
// Legal code: a single IN clause, that takes a list several arguments
async IN(a,b) OUT(x) { code }

// Invalid example => Compile time error (two IN clauses specified)
async IN(a) IN(b) OUT(x) { code }
```

OUT clause

- Pitfall: OUT variables are guaranteed to have been written back only after the enclosing finish scope is finished

```
int a = 0;
int x = 0; // 1
finish {
  async IN(a) OUT(x) {
    // compute something
    x = someValue; // 2
  }
  // here 'x' has an undetermined value could
  // be either the value x from 1 or x from 2
}
// here 'x' contains the value computed in the async
```