Lecture 20 — Loop Invariants

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Loop Invariant

“An invariant of a loop is a property that holds before (and after) each repetition. It is a logical assertion, sometimes programmed as an assertion. Knowing its invariant(s) is essential for understanding the effect of a loop.”

— Wikipedia

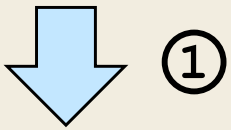
Example Loop

```
function count(int n) -> (int r):
    int i = 0
    //
    while i < n:
        i = i + 1
    //
    return i
```

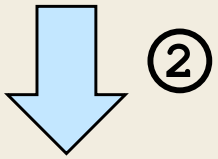
- What is a **loop invariant** here?

Three Rules of Loop Invariants

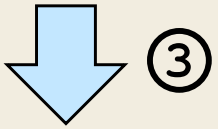
```
function f() requires R ensures E:
```




```
while C where I:
    //
```




```
//
```




```
return ...
```

- **Rule (1):** *Loop invariants must hold on entry*
- **Rule (2):** *Loop invariants must be restored*
- **Rule (3):** *Loop invariants hold afterwards*

Formal Notation

- To **aid explanation** of loop invariant rules, we introduce some notation:

$$\{P\}\ S\ \{Q\}$$

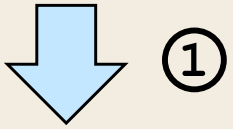
If: `P` is information known **before** statement `S`

Then: `Q` is information known **after** statement `S`

- **Example:** $\{x \geq 0\}\ x = x + 1\ \{x > 0\}$

Loop Invariants must hold on Entry (Rule 1)

```
function f() requires R ensures E:
```

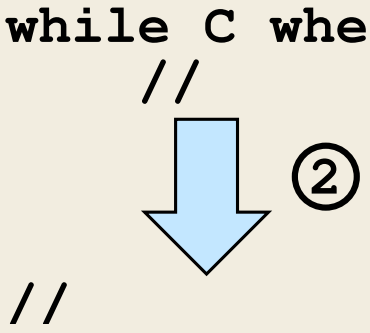



```
while C where I:
    //
```

- **Informally:** Information known at start of loop **must imply** loop invariant
- **Formally:** If $\{R\}\ S\ \{P\}$ then $P \implies I$ must hold

  (where *S* represents all statements **before loop**)

Loop Invariants must be Restored (Rule 2)



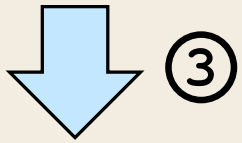
- **Informally:** Assuming only loop invariant and condition hold at start of loop, information known at end **must imply** invariant

- **Formally:** If $\boxed{\{I \wedge C\}\ S\ \{P\}}$ then $\boxed{P \implies I}$ must hold

  (where S represents **loop body**)

Loop Invariants hold afterwards (Rule 3)

```
//
```




```
return ...
```

- **Informally:** Can assume only *loop invariant* and *negated condition* hold after loop
- **Formally:** $\boxed{\{I \wedge \neg C\}\ S\ \{E\}}$ must hold

  (where S represents all statements **after loop**; and E is **postcondition**)

Example 1: Vector Sum

```
function sum(int[] v1, int[] v2) -> (int[] v3)
// Input vectors must have same size
requires |v1| == |v2|
// Result has same size as input
ensures |v1| == |v3|
// Each element of result is sum of corresponding elements in inputs
ensures all { i in 0..|v1| | v3[i] == v1[i] + v2[i] }:
    //
    int i = 0
    int[] ov1 = v1
    //
    while i < |v1|:
        v1[i] = v1[i] + v2[i]
        i = i + 1
    //
    return v1
```

- What **loop invariant** do we need here?

Ghost Variables

A **ghost variable** *is any variable introduced specifically to aid verification in some way, but is unnecessary to execute the program.*

- In vector sum example, variable `ov1` is a **ghost variable** ...

... because can **implement solution** without it ...

... but cannot **verify solution** without it!

Loop Invariant Variables

```
function sum(int[] items) -> (int r)
// No negative elements in items array
requires all { i in 0..|items| | items[i] >= 0 }
// Result cannot be negative
ensures r >= 0:
    //
    int i = 0
    int r = 0
    //
    while i < |items| where i >= 0 && r >= 0:
        r = r + items[i]
        i = i + 1
    //
    return r
```

- Rules imply we need `all {k in 0..items| | items[k] >= 0}|`
- However, special case for variables **not modified** in loop ...

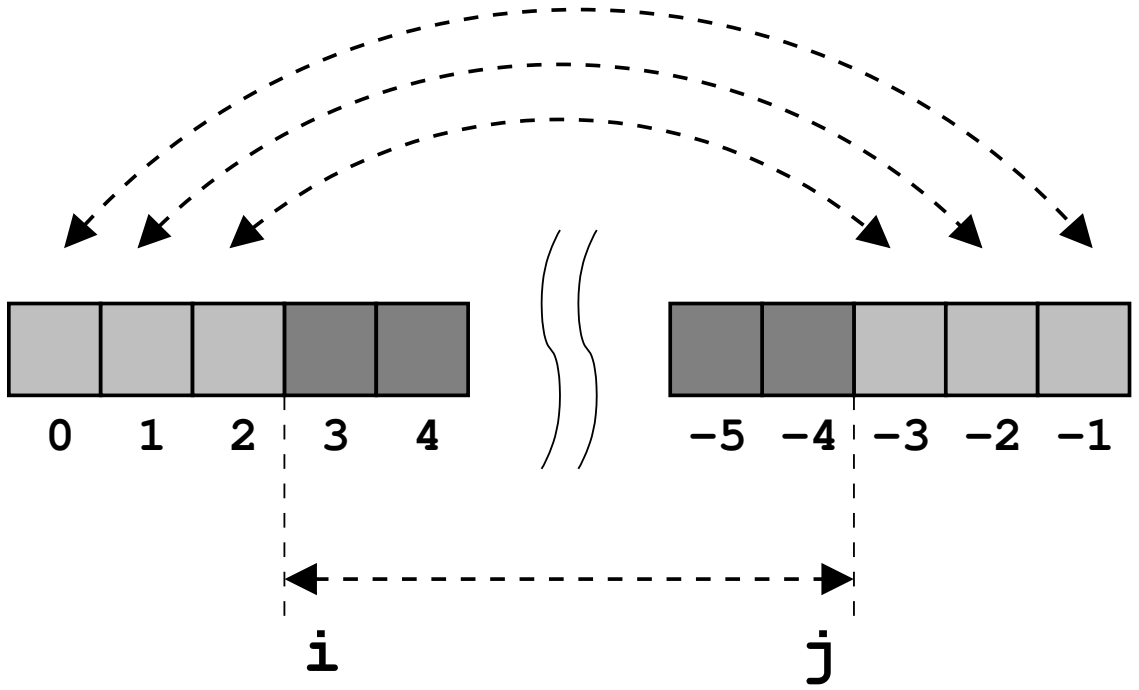
  ... all info about them from before loop **automatically retained**

Example 2: Reversing an Array (Implementation)

```
// In–place reverse of items in an array
function reverse(int[] xs) -> (int[] ys)
// Returned array is same size as input
ensures ...
// All items in return array in reversed order
ensures ...:
    //
    int i = 0
    int j = |xs| - 1
    //
    while i < j:
        //
        int tmp = xs[i]
        xs[i] = xs[j]
        xs[j] = tmp
        j = j - 1
        i = i + 1
    //
    return xs
```

- What **loop invariant** do we need here?

Example 2: Reversing an Array (Diagram)



- Region between `i` and `j` remaining to be reversed