Program Verification

- Binary Search
  - The Problem
  - Code Derivation
  - Verification
- Principles

Binary Search

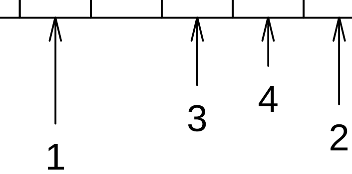
The Problem.

Input: An integer $n \geq 0$ and a sorted array
$x[0] \leq x[1] \leq x[2] \leq ... \leq x[n-1]$.

Output: The integer p tells t's location in
$x[0..n-1]$. If $p = -1$ then t is not in $x[0..n-1]$;
otherwise $0 \leq p < n$ and $t = x[p]$.

The Algorithm. Keep track of a range known to contain t. The range is initially empty, and is shrunk by comparing the middle element to t. This example searches for 50 in $x[0..15]$.

26	26	31	31	32	38	38	41	43	46	50	53	58	59	79	97



Difficulty. The first binary search was published in 1946; when was the first *correct* search published?

Derivation, Step 1

```
initialize range to 0..n-1
loop
    { invariant: mustbe(range) }
    if range is empty,
        break and report that t
        is not in the array
    compute m, the middle of the range
    use m as a probe to shrink the range
        if t is found during
        the shrinking process,
        break and report its position
```

Derivation, Step 2

Represent the range *l..u* by integers *l* and *u*.

```
l = 0; u = n-1
loop
    { invariant: mustbe(l, u) }
    if l > u
        p = -1; break
    m = (l + u) / 2
    use m as a probe to shrink l..u
        if t is found during
        the shrinking process,
        break and note its position
```

Binary Search Code

```
l = 0; u = n-1
loop
    { mustbe(l, u) }
    if l > u
        p = -1; break
    m = (l + u) / 2
    case
        x[m] <  t:  l = m+1
        x[m] == t:  p = m; break
        x[m] >  t:  u = m-1
```

Verifying the Code

```
{ mustbe(0, n-1) }
l = 0; u = n-1
{ mustbe(l, u) }
loop
   { mustbe(l, u) }
   if l > u
   { l > u && mustbe(l, u) }
      { t is not in the array }
      p = -1; break
   { mustbe(l, u) && l <= u }
   m = (l + u) / 2
   { mustbe(l, u) && l <= m <= u }
   case
      x[m] <  t:
         { mustbe(l, u) && cantbe(0, m) }
         { mustbe(m+1, u) }
         l = m+1
         { mustbe(l, u) }
      x[m] == t:
         { x[m] == t }
         p = m; break
      x[m] >  t:
         { mustbe(l, u) && cantbe(m, n) }
         { mustbe(l, m-1) }
         u = m-1
         { mustbe(l, u) }
   { mustbe(l, u) }
```

Binary Search in C

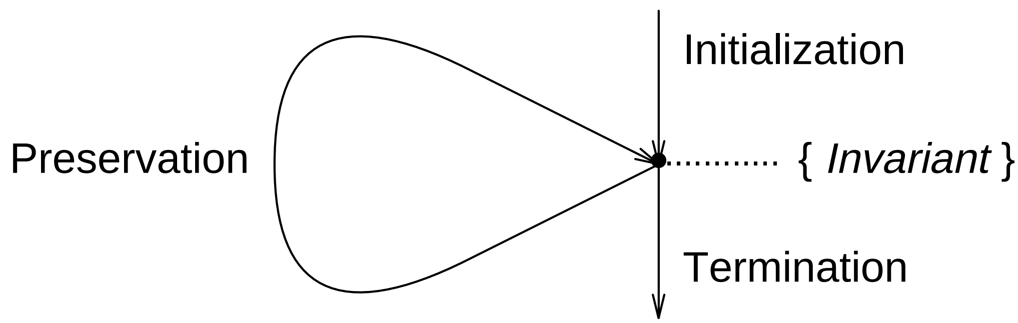
```
int binarysearch(DataType t)
    /* return (any) position
        if t in sorted x[0..n-1] or
        -1 if t is not present */
{   int l, u, m;
    l = 0;
    u = n-1;
    while (l <= u) {
        m = (l + u) / 2;
        if (x[m] < t)
            l = m+1;
        else if (x[m] == t)
            return m;
        else /* x[m] > t */
            u = m-1;
    }
    return -1;
}
```

Principles

Tools of Verification

Assertions

Control Structures: sequential, selection, iteration



Functions

Roles of Verification

Writing subtle code

Walk-throughs, testing, debugging

General: A language for talking about code