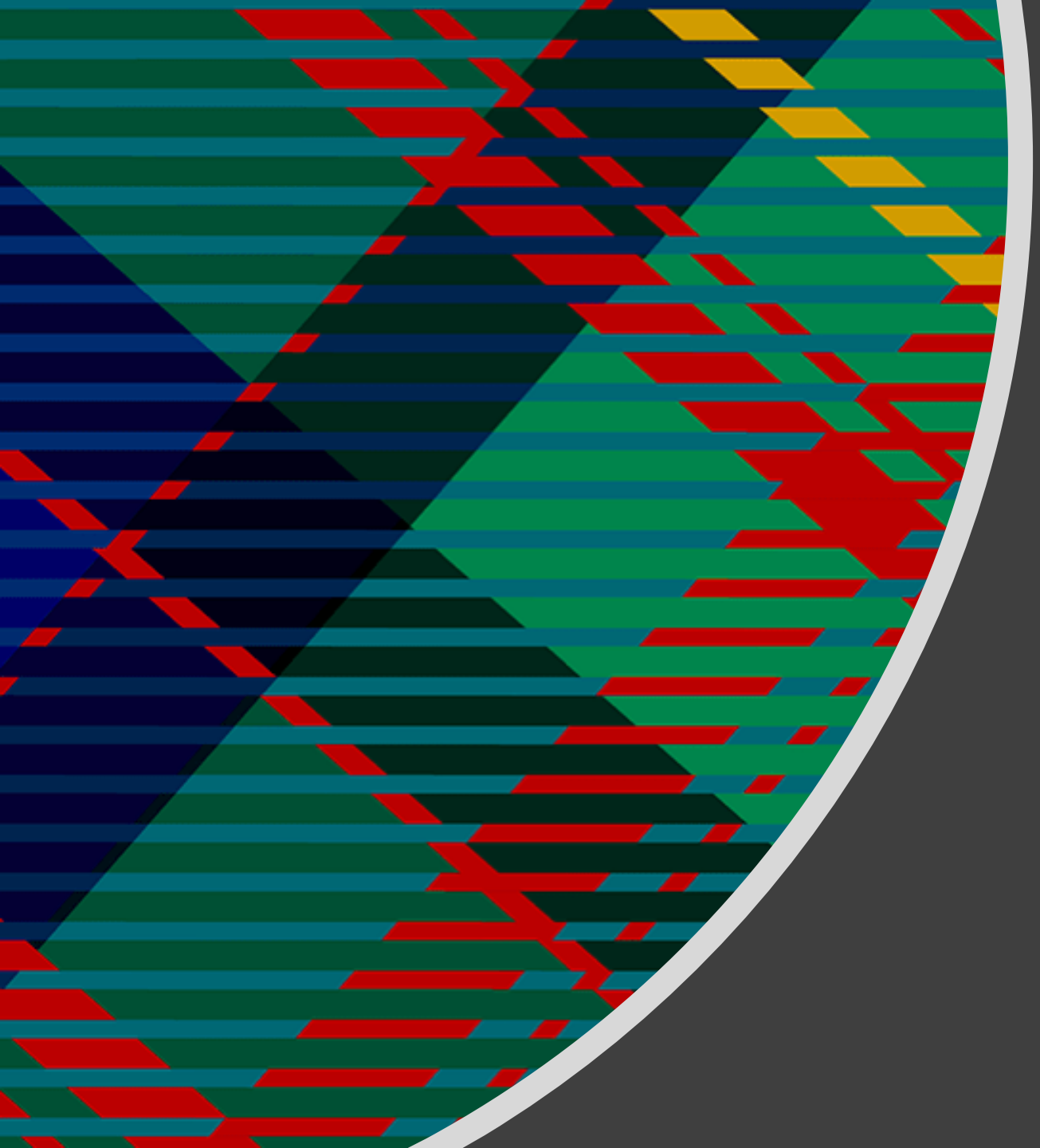

122
Search

Last time

Arrays

- Simple memory model
- Array mechanics
- Aliasing
- Safety

Today

Search

- Linear search
- Safety & correctness
- Contract failure/exploits

Next

Cost

Sorting

Array Search

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23  }
```

What do we return if `x` is not found?

Array Search

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (-1 == \result && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i && i <= n;
23    //@loop_invariant !is_in(x,A,0,i);
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    //@assert !is_in(x,A,0,n);
30    return -1;
31  }
```

Big Bang Theory: Sheldon's neurosis

https://www.youtube.com/watch?v=xVt8_0_2hww

In the news…

Best Undergraduate Computer Engineering Programs (Doctorate)

Computer engineering combines principles of electrical engineering and computer science. These are the top undergraduate schools for computer engineering, where the highest engineering degree offered is a doctorate.

To unlock full rankings, SAT/ACT scores and more, sign up for the U.S. News College Compass!

Carnegie Mellon University

Pittsburgh, PA

#1 in Computer

#6 in Engineering Programs (doctorate) (tie)

Carnegie Mellon University, a private institution in Pittsburgh, is the country's only school founded by industrialist and philanthropist ... more

$55,465 Tuition and Fees | **6,896** Undergraduate Enrollment | **4.4** Reputation Score | SAT, GPA and more

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

Which line might not be safe?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  {
18    for (int i = 0; i < n; i++)
19    {
20      if (A[i] == x) {
21        return i;
22      }
23    }
24    return -1;
25  }
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  {
18    for (int i = 0; i < n; i++)
19    //@loop_invariant 0 <= i;
20    {
21      if (A[i] == x) {
22        return i;
23      }
24    }
24    return -1;
26  }
```

How would we use this code?

Calling search

```
int i = search(12, A, 5);

if (i != -1) {
  // Change 12 to 13
  A[i] = 13;
}
```

Is this code safe?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures \result == -1 ||
18        A[\result] == x;
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    {
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

Calling search

```
int i = search(12, A, 5);

if (i != -1) {
  // Change 12 to 13
  A[i] = 13;
}
```

Is this code safe?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures \result == -1 ||
18        A[\result] == x;
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    {
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

Caller only sees contracts, not implementation

Is this ensures call safe?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures \result == -1 ||
18        (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    {
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

Caller only sees contracts, not implementation

Is this ensures call safe?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures \result == -1 ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    {
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures \result == -1 ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    return -1;
22  }
```

Correctness

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures \result == -1 ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
```

```
bool is_in(int x, int[] A, int lo, int hi)
//@requires 0 <= lo && lo <= hi && hi == \length(A);
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
```

```
bool is_in(int x, int[] A, int lo, int hi)
//@requires 0 <= lo && lo <= hi && hi == \length(A);
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
```

```
bool is_in(int x, int[] A, int lo, int hi)
//@requires 0 <= lo && lo <= hi && hi == \length(A);
{
  if (lo == hi) {
    return false;
  else if (A[lo] == x) {
    return true;
  }
  else {
    return is_in(x, A, lo+1, hi);
  }
}
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    {
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

Is postcondition satisfied?

`0 <= i` by line 22, loop invariant

`i < n` by line 21, loop guard

`A[i] == x` by line 24, if condition

Is search correct: when it returns `i`?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    {
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

Is search correct: when it returns −1?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    //@loop_invariant !is_in(x,A,0,i);
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    return -1;
30  }
```

Is search correct: when it returns –1?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    //@loop_invariant !is_in(x,A,0,i);
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    return -1;
30  }
```

INIT:

`!is_in(x, A, 0, 0)` by definition

PRES:

Assume `!is_in(x, A, 0, i)`

`i' = i+1` by line 21, loop increment

```
!is_in(x, A, 0, i') ==
     !is_in(x, A, 0, i) &&
     A[i] != x
```

Is this a valid loop invariant?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i;
23    //@loop_invariant !is_in(x,A,0,i);
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    return -1;
30  }
```

EXIT:

`!is_in(x, A, 0, i)` by line 23, LI

`i >= n` by negating loop guard

Prove postcondition when returning `-1`?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i && i <=n;
23    //@loop_invariant !is_in(x,A,0,i);
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    return -1;
30  }
```

EXIT:

`!is_in(x, A, 0, i)` by line 23, LI

`i >= n` by negating loop guard

`i <= n` by line 22, LI

Therefore `!is_in(x, A, 0, n)`

Prove postcondition when returning `-1`?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    //@loop_invariant 0 <= i && i <= n;
23    //@loop_invariant !is_in(x,A,0,i);
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    //@assert !is_in(x,A,0,n);
30    return -1;
31  }
```

EXIT:

`!is_in(x, A, 0, i)` by line 23, LI

`i >= n` by negating loop guard

`i <= n` by line 22, LI

Therefore `!is_in(x, A, 0, n)`

Contract Exploitation

Can the caller be sure the function is correct given the contract?

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
30    return -1;
31  }
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    {
23      A[i] = x; // Modify A to contain x
24      return i; // and immediately return i
25      if (A[i] == x) {
26        return i;
27      }
28    }
29    return -1;
30  }
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
22    {
23      A[i] = x+1; // Change each entry something other than x
24      if (A[i] == x) {
25        return i;
26      }
27    }
28    return -1;
29  }
```

```
15  int search(int x, int[] A, int n)
16  //@requires n == \length(A);
17  /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
18      (0 <= \result && \result < n && A[\result] == x);
19    @*/
20  {
21    for (int i = 0; i < n; i++)
24    {
25      if (A[i] == x) {
26        return i;
27      }
28    }
30    return -1;
31  }
```

How fast is this code?