# CSE 331
# Software Design & Implementation

Autumn 2025

Section 8 – Arrays

# Administrivia

- HW8 released tonight - **Due @ 6pm Friday**
- There WILL be section next Tuesday before the break!
- Quiz on Monday Dec 1st is moved to **Wednesday Dec 3rd**

# Array Notation: Indexing

- Arrays are treated like Lists *mathematically*
  - They are an alternative way to represent Lists in *code*

- To get the jth element, use at(j, L), abbreviated "L[j]"

  at : (List, $\mathbb{N}$) $\rightarrow$ $\mathbb{Z}$

  at(nil , n) := undefined

  at(x :: L , 0) := x

  at(x :: L , n+1) := at(L, n)

  - Must ensure that 0 <= j < n

# More Array Notation

- **Prefix:** All elements from the start to j:
  - A[...j]
- **Suffix:** All elements starting at j to the end:
  - A[j…]
- **Sublist:** All elements in the array from i to j:
  - A[i...j]

**Other useful facts:**

- A[len(A)...] = nil
- A[...-1] = nil
- A[...j-1] ⧺ A[j] = A[...j]
- A[...j-1] ⧺ A[j…] = A
- A[...len(A)-1] = A

# For Any Facts

- Necessary facts about arbitrary parts of an array

- Ex: To show an array is sorted in asc formally:
  - $A[j] < A[j+1]$ *for any* $0 \leq j \leq len(A) - 2$

```
// @requires A[i] < A[i+1] for any 0 <= i < len(A)-1
// @returns false if A[i] /= y for any 0 <= i < len(A)
//          true  otherwise
public boolean bsearch(int[] A, int y) { … }
```

# Array Mutation

- Array mutation can change "for any" facts!

- Ex:

{{ A[j] < A[j+1] for any $0 \le j \le 9$ }}

```
A[0] = 100;
```

{{ (A[j] < A[j+1] for any $\mathbf{1} \le j \le 9$) and A[0] = 100 }}

- Old facts about A[0] could be invalidated!
  - Need to update the range of "for any" facts

# Mutating Arrays (add/remove)

- Can add to the end of an array

{{ A }}
```
A.push(100);
```
{{ A_0 and A = A_0 # [100] }}

{{ A # [100] }}
```
A.push(100);
```
{{ A }}

- Can remove from the end of an array

{{ A }}
```
A.remove(A.size()-1);
```
{{ A_0 and
A = A_0[..len(A_0) – 2] }}

{{ A[..len(A) - 2] }}
```
A.remove(A.size()-1);
```
{{ A }}

# Task 1 – Better Get Proving

The function replace : $(\mathsf{List}, y, z) \to \mathsf{List}$ is defined by

$$\begin{aligned} \mathsf{replace}(\mathsf{nil}, y, z) &:= \mathsf{nil} \\ \mathsf{replace}(x :: L, y, z) &:= z :: \mathsf{replace}(L, y, z) \quad \text{if } x = y \\ \mathsf{replace}(x :: L, y, z) &:= x :: \mathsf{replace}(L, y, z) \quad \text{if } x \neq y \end{aligned}$$

In this problem, we will prove that replace works the same way at the end of the list that it does at the front of the list, i.e.:

$$\begin{aligned} \mathsf{replace}(L \mathbin{+\!\!+} [x], y, z) &= \mathsf{replace}(L, y, z) \mathbin{+\!\!+} [z] \quad \text{if } x = y \\ \mathsf{replace}(L \mathbin{+\!\!+} [x], y, z) &= \mathsf{replace}(L, y, z) \mathbin{+\!\!+} [x] \quad \text{if } x \neq y \end{aligned}$$

**a)** Explain, in your own words, why the following statement, if proven, would imply the one above:

$$\mathsf{replace}(L \mathbin{+\!\!+} [x], y, z) = \mathsf{replace}(L, y, z) \mathbin{+\!\!+} \mathsf{replace}([x], y, z)$$

If $x = y$, then $\mathsf{replace}([x], y, z) = [z]$ by the definition of replace, and if $x \neq y$, then $\mathsf{replace}([x], y, z) = [x]$ by the definition of replace.

# Task 1 – Better Get Proving

claim from a: $\text{replace}(L + [x], y, z) = \text{replace}(L, y, z) + \text{replace}([x], y, z)$

**b)** Explain, in your own words, why the following claim, if proven, would imply the one from part (a):

$$\text{replace}(L + R, y, z) = \text{replace}(L, y, z) + \text{replace}(R, y, z)$$

Setting $R = [x]$ gives the claim from part (a).

# Task 1 – Better Get Proving

**c)** Prove the claim from part (b) by induction <u>on $L$</u>.

$$\text{replace}(L \mathbin{+\!\!+} R, y, z) = \text{replace}(L, y, z) \mathbin{+\!\!+} \text{replace}(R, y, z)$$

# Task 1 – Better Get Proving

Define $P(L)$ to be the claim $\text{replace}(L + R, y, z) = \text{replace}(L, y, z) + \text{replace}(R, y, z)$. We will prove that this holds for all values of $L$ by induction.

**Base Case.** We can see that $P(\text{nil})$ holds as follows:

$$\begin{aligned}\text{replace}(\text{nil} + R, y, z) &= \text{replace}(R, y, z) \\ &= \text{nil} + \text{replace}(R, y, z) \\ &= \text{replace}(\text{nil}, y, z) + \text{replace}(R, y, z) \quad \text{def of replace}\end{aligned}$$

**Inductive Hypothesis.** Suppose that $P(L)$ holds for some arbitrary $L$.

# Task 1 – Better Get Proving

**Inductive Step.** We must show that $P(w :: L)$ holds for any $w$ so we will continue by cases:

Suppose that $w = y$. Then, we can see that

$$\begin{aligned}
&\text{replace}((w :: L) + R, y, z) \\
&= z :: \text{replace}(L + R, y, z) && \text{def of replace (since } w = y) \\
&= z :: (\text{replace}(L, y, z) + \text{replace}(R, y, z)) && \text{Ind. Hyp.} \\
&= (z :: \text{replace}(L, y, z)) + \text{replace}(R, y, z) \\
&= \text{replace}(w :: L, y, z) + \text{replace}(R, y, z) && \text{def of replace (since } w = y)
\end{aligned}$$

# Task 1 – Better Get Proving

**Inductive Step.** We must show that $P(w :: L)$ holds for any $w$ so we will continue by cases:

Now, suppose that $w \neq y$. Then, we can see that

$$
\begin{aligned}
&\text{replace}((w :: L) +\!\!+ R, y, z) \\
&= w :: \text{replace}(L +\!\!+ R, y, z) && \text{def of replace (since } w \neq y) \\
&= w :: (\text{replace}(L, y, z) +\!\!+ \text{replace}(R, y, z)) && \text{Ind. Hyp.} \\
&= (w :: \text{replace}(L, y, z)) +\!\!+ \text{replace}(R, y, z) \\
&= \text{replace}(w :: L, y, z) +\!\!+ \text{replace}(R, y, z) && \text{def of replace (since } w \neq y)
\end{aligned}
$$

Since these two cases are exhaustive, we have proven $P(w :: L)$ holds in general.

**Conclusion.** $P(L)$ holds for all $L$ by induction.

# Task 2 – Jumping Through Loops

```
/**
 * Writes over each copy of y in A with the value z.
 * @param A .. y .. z ..
 * @modifies A
 * @effects A = replace(A_0, y, z)
 */
public void replace(int[] A, int y, int z) { .. }
```

**a)**

```
int i = __________

// Inv: A[.. i] = A_0[.. i] and A[i+1 ..] = replace(A_0[i+1 ..], y, z)
while (__________) {



}
```

# Task 2 – Jumping Through Loops

```
/**
 * Writes over each copy of y in A with the value z.
 * @param A .. y .. z ..
 * @modifies A
 * @effects A = replace(A_0, y, z)
 */
public void replace(int[] A, int y, int z) { .. }
```

```
int i = A.length - 1;

// Inv: A[.. i] = A_0[.. i] and A[i+1 ..] = replace(A_0[i+1 ..], y, z)
while (i >= 0) {
     if (A[i] == y)
          A[i] = z;
     i = i - 1;
}
```

# Task 2 – Jumping Through Loops

```
/**
 * Writes over each copy of y in A with the value z.
 * @param A .. y .. z ..
 * @modifies A
 * @effects A = replace(A_0, y, z)
 */
public void replace(int[] A, int y, int z) { .. }
```

**b)**

```
int i = __________

// Inv: A[.. i-1] = replace(A_0[.. i-1], y, z) and A[i ..] = A_0[i ..]
while (__________) {



}
```

# Task 2 – Jumping Through Loops

```
/**
 * Writes over each copy of y in A with the value z.
 * @param A .. y .. z ..
 * @modifies A
 * @effects A = replace(A_0, y, z)
 */
public void replace(int[] A, int y, int z) { .. }
```

```
int i = 0;

// Inv: A[.. i-1] = replace(A_0[.. i-1], y, z) and A[i ..] = A_0[i ..]
while (i < A.length) {
        if (A[i] == y)
            A[i] = z;
        i = i + 1;
  }
```

# Task 3 – Rally the Loops

Recall the function remove : $(\text{List}, y) \to \text{List}$ defined as follows:

$$\begin{aligned}
\text{remove}(\text{nil}, y) &:= \text{nil} \\
\text{remove}(x :: L, y) &:= \text{remove}(L, y) && \text{if } x = y \\
\text{remove}(x :: L, y) &:= x :: \text{remove}(L, y) && \text{if } x \neq y
\end{aligned}$$

It is possible to prove, as the same manner as we did in Task 1, that the following holds:

$$\begin{aligned}
\text{remove}(L +\!\!+ [x], y) &= \text{remove}(L, y) && \text{if } x = y \\
\text{remove}(L +\!\!+ [x], y) &= \text{remove}(L, y) +\!\!+ [x] && \text{if } x \neq y
\end{aligned}$$

You can use these facts below without proof. Refer to them as "Lemma 2".

In this problem, we will check the correctness of the following code that implements remove. Specifically, it writes $\text{remove}(A, y)$ into some prefix of the array, $A[..j-1]$, and returns $j$.

# Task 3 – Rally the Loops

```
int i = 0;
int j = 0;
```

$\{\{\ P_1:$ ______________________________ $\}\}$

$\{\{\ \text{Inv}:\ A[..j-1] = \text{remove}(A[..i-1], y) \text{ and } A[i..] = A_0[i..]\ \}\}$

```
while (i != A.length) {
  if (A[i] == y) {
```

$\{\{\ P_2:$ ______________________________ $\}\}$

$\{\{\ Q_2:$ ______________________________ $\}\}$

```
  } else {
```

$\{\{\ P_3:$ ______________________________ $\}\}$

$\{\{\ Q_3:$ ______________________________ $\}\}$

```
      A[j] = A[i];
      j = j + 1;
  }
  i = i + 1;
}
```

$\{\{\ P_4:$ ______________________________ $\}\}$

$\{\{\ \text{Post}:\ A[..j-1] = \text{remove}(A, y)\ \}\}$

```
return j;
```

# Task 3 – Rally the Loops

```
int i = 0;
int j = 0;
```

$\{\{\ P_1 :$ ______________________________ $\}\}$

$\{\{\ \text{Inv}: \ A[..\,j-1] = \text{remove}(A[..\,i-1], y) \text{ and } A[i\,..] = A_0[i\,..]\ \}\}$

**a)** Fill in $P_1$ using forward reasoning and then prove that $P_1$ implies Inv.

$P_1$ should say that $i = 0$ and $j = 0$ (and $A = A_0$ if you want).

The second fact of Inv holds since

$$
\begin{aligned}
A[i\,..] &= A[0\,..] && \text{since } i = 0 \\
&= A \\
&= A_0 && \text{since } A = A_0 \\
&= A_0[0\,..] \\
&= A_0[i\,..] && \text{since } i = 0
\end{aligned}
$$

# Task 3 – Rally the Loops

```
int i = 0;
int j = 0;
```

$\{\{\, P_1 : \underline{\hspace{12cm}} \,\}\}$

$\{\{\, \text{Inv} : \; A[..\, j-1] = \text{remove}(A[..\, i-1], y) \text{ and } A[i\, ..] = A_0[i\, ..] \,\}\}$

**a)** Fill in $P_1$ using forward reasoning and then prove that $P_1$ implies Inv.

$P_1$ should say that $i = 0$ and $j = 0$ (and $A = A_0$ if you want).

The first fact of Inv holds since

$$
\begin{aligned}
A[..\, j-1] &= A[..\, 0-1] && \text{since } j = 0 \\
&= \text{nil} \\
&= \text{remove}(\text{nil}, y) && \text{def of remove} \\
&= \text{remove}(A[..\, -1], y) \\
&= \text{remove}(A[..\, i-1], y) && \text{since } i = 0
\end{aligned}
$$

# Task 3 – Rally the Loops

$\{\{\, P_4 : \underline{\hspace{10cm}} \,\}\}$
$\{\{\, \textsf{Post}: \; A[..\, j-1] = \textsf{remove}(A, y) \,\}\}$

```
return j;
```

**b)** Fill in $P_4$ using forward reasoning and then prove that $P_4$ implies the postcondition.

$P_4$ should say $A[..\, j-1] = \textsf{remove}(A[..\, i-1], y)$ and $A[i\, ..] = A_0[i\, ..]$ and $i = \textsf{len}(A)$. This gives us the postcondition since

$$
\begin{aligned}
A[..\, j-1] &= \textsf{remove}(A[..\, i-1], y) && \text{since } A[..\, j-1] = \textsf{remove}(A[..\, i-1], y) \\
&= \textsf{remove}(A[..\, \textsf{len}(A)-1], y) && \text{since } i = \textsf{len}(A) \\
&= \textsf{remove}(A, y)
\end{aligned}
$$

# Task 3 – Rally the Loops

```
if (A[i] == y) {
    {{ P_2: ______________________________ }}
    {{ Q_2: ______________________________ }}
```

**c)** Fill in $P_2$ using forward reasoning and $Q_2$ using backward. Then, prove that $P_2$ implies $Q_2$.

$P_2$ should say $A[..\, j-1] = \text{remove}(A[..\, i-1], y)$, $A[i\, ..] = A_0[i\, ..]$, and $A[i] = y$.
$Q_2$ should say $A[..\, j-1] = \text{remove}(A[..\, i], y)$ and $A[i+1\, ..] = A_0[i+1\, ..]$.

The second part of $Q_2$ is implied by the second fact from $P_2$ (since $A[i+1\, ..]$ is a sublist of $A[i\, ..]$). The first fact follows since

$$
\begin{aligned}
A[..\, j-1] &= \text{remove}(A[..\, i-1], y) \\
&= \text{remove}(A[..\, i-1] \mathbin{+\!\!+} [A[i]], y) \quad \text{Lemma 2 (since } y = A[i]\text{)} \\
&= \text{remove}(A[..\, i], y)
\end{aligned}
$$

# Task 3 – Rally the Loops

```
} else {
    {{ P3: ________________________________ }}
    {{ Q3: ________________________________ }}
```

**d)** Fill in $P_3$ using forward reasoning and $Q_3$ using backward. Then, prove that $P_3$ implies $Q_3$.

$P_3$ should say $A[..\,j-1] = \text{remove}(A[..\,i-1], y)$, $A[i\,..] = A_0[i\,..]$, and $A[i] \neq y$.
$Q_3$ should say $A[..\,j-1] + [A[i]] = \text{remove}(A[..\,i], y)$ and $A[i+1\,..] = A_0[i+1\,..]$.

The second part of $Q_3$ is implied by the second fact from $P_3$ (since $A[i+1\,..]$ is a sublist of $A[i\,..]$). The first fact follows since

$$
\begin{aligned}
& A[..\,j-1] + [A[i]] \\
&= \text{remove}(A[..\,i-1], y) + [A[i]] && \text{since } A[..\,j-1] = \text{remove}(A[..\,i-1], y) \\
&= \text{remove}(A[..\,i-1] + [A[i]], y) && \text{Lemma 2 (since } y \neq A[i]) \\
&= \text{remove}(A[..\,i], y)
\end{aligned}
$$