Array Algorithms

Lecture 8

Announcements

- Topics Page
- PS2 - Test Cases due Tomorrow Night
- Quiz 2 Thursday
- PS2 - Implementations due Monday 10/1 at 11:59pm
  - Warning: Expect a long queue and wait Monday night in OH. Recommendation: Finish early and get extra credit!

Challenge Question #0

In the main function, what is **printed**?

```
export let main = async () => {
   let input = [1, 2, 2, 3];
   let result = bar(input, 2);
   print(result);
};

let bar = (a: number[], n: number): boolean => {
   for (let i = 0; i < a.length; i++) {
      if (a[i] === n) {
         return false;
      }
   }
   return true;
};
```

Operation Assignment Operators

- Consider the following assignment statements:

```
i = i + 10;
s = s + "!";
```

- Increasing a variable, concatenating to a variable, and so on, are so common that there are built-in shorthand operators:

Operator	Syntax	Example	Equivalent To
Addition Assignment	+=	`i += 10;`	`i = i + 10;`
Subtraction Assignment	-=	`i -= 10;`	`i = i - 10;`
Multiplication Assignment	*=	`i *= 10;`	`i = i * 10;`
Division Assignment	/=	`i /= 10;`	`i = i / 10;`
Remainder Assignment	%=	`i %= 10;`	`i = i % 10;`
Concatenation Assignment	+=	`s += "!!!";`	`s = s + "!!!";`

```
export let main = async () => {
    let input = [1, 2, 3];
    let result = foo(input);
    print(result);
};

let foo = (a: number[]): string[] => {
    let b: string[] = [];
    for (let i = 0; i < a.length; i++) {
        b[i] = repeatB(i);
    }
    return b;
};

let repeatB = (n: number): string => {
    let s = "";
    for (let i = 0; i < n + 1; i++) {
        s += "b";
    }
    return s;
};
```

Challenge Question #1

Q1. In the main function, what is the variable result's **data type**?

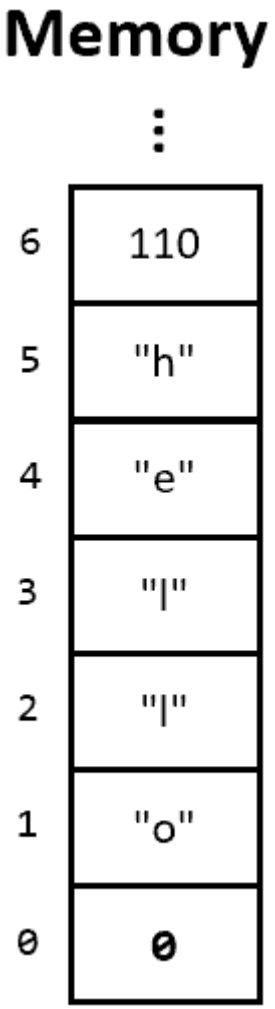
Q2. When main is executed, what is **printed**?

Challenge Question #2: What is printed?

```
let s = "abc";
print(s[1]);
print(s.length);
```

Strings are Arrays of Characters

- In the first video of the semester, your computer's memory was introduced with a diagram to the left.
- Notice cells 1-5 store individual characters... *not strings.*
- A string is an array of single characters underneath the hood.
  - We haven't needed to worry over this detail thanks to ***data abstraction!***
- We can "poke through" the abstraction!
  - Access individual characters with **`stringName[index]`**
  - Access the length of a string with **`stringName.length`**



Promotion Filter - Hands-on

- In "The Promotion" Season 6, Episode 3 of The Office, Michael and Jim use some seemingly arbitrary criteria to decide on promotions.

- Today we'll do that with the COMP110 UTA staff...

1. **Read 00-filter-app.ts - especially `filterByNamePredicate`**
   - **What is this function doing?**

2. **Your Job:** Update the **`namePredicate`** function to **return true** when the first character of the **`name`** parameter is "K", **return false** otherwise.

- Check-in on PollEv.com/compunc when you have narrowed it down to 3 UTAs...

Pattern: Nesting **if-then** in an **else** Pattern

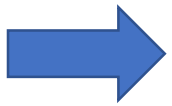
- It is commonly useful to nest additional if-then-else statements inside of subsequent else-blocks

- Why? It allows us to choose one next step from many possible options.
  - "If *this* then do X, otherwise if *that* do Y, *otherwise* do Z."

```
if (response === 0) {
    print("Very doubtful");
} else {
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
}
```

This is so common and useful, we tend to use simpler syntax for it…

```
if (response === 0) {
    print("Very doubtful");
} else {
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
}
```

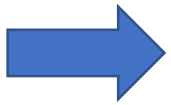


```
if (response === 0) {
    print("Very doubtful");
} else
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
```

1. First we remove the curly braces surrounding the if-then that is nested inside of the else-block.

This is so common and useful, we tend to use simpler syntax for it…

```
if (response === 0) {
    print("Very doubtful");
} else
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
```

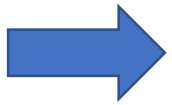


```
if (response === 0) {
    print("Very doubtful");
} else if (response === 1) {
    print("Ask again later");
} else {
    print("It is certain");
}
```

2. Then we clean up the spacing.

Using the **else-if** pattern is a change of *style* only. These two listings of code have the ***exact same logic***.

```
if (response === 0) {
    print("Very doubtful");
} else {
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
}
```



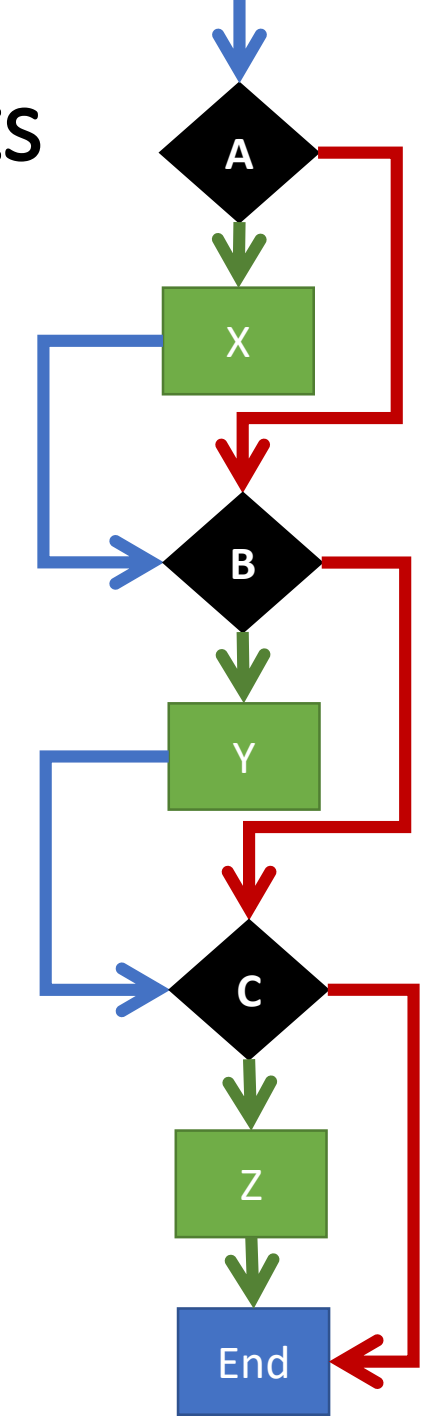
```
if (response === 0) {
    print("Very doubtful");
} else if (response === 1) {
    print("Ask again later");
} else {
    print("It is certain");
}
```

Notice the code is visually simpler and cleaner by using else-if.

Many, independent `if-then-else` statements

- When two or more if-then-else statements are *not* nested, they are independent statements of one another.
- Each `boolean` test expression will be evaluated.
- Notice in the diagram that there is a path through *every* block X, Y, Z.

```
if (testA()) {
  print("X");
}

if (testB()) {
  print("Y");
}

if (testC()) {
  print("Z");
}

print("End");
```



Tracing through `else-if` statements

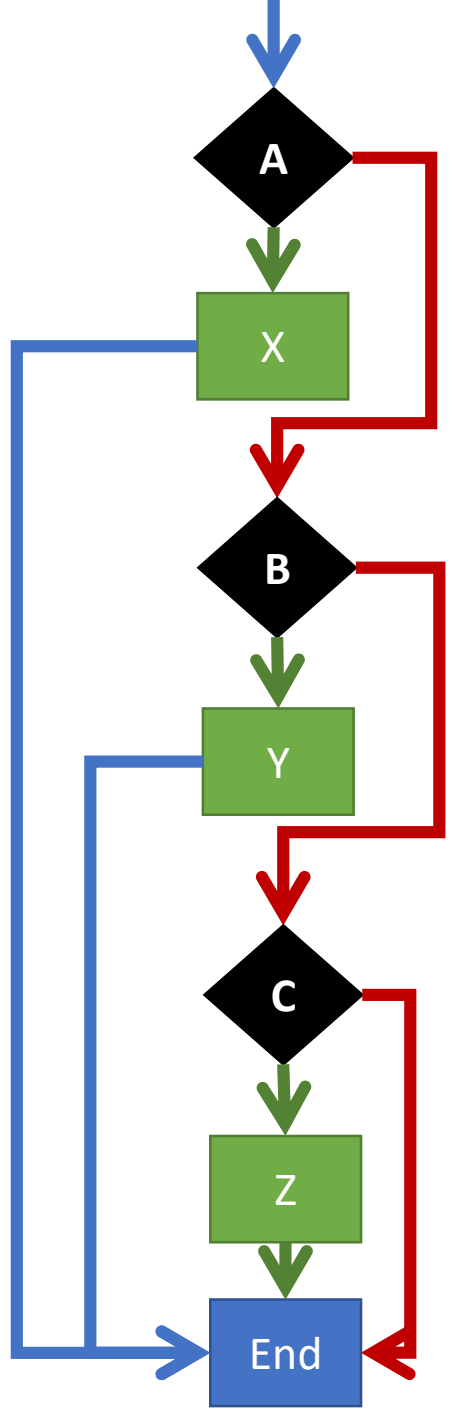
- The previous slide does not apply to else-if statements *because…*
  - An else-if is a nested if-then
  - It is nested in the else-block

- Each `boolean` test expression will be evaluated **<u>until one evaluates to true</u>**. The rest are then skipped.

- Notice in the diagram that there is a path through *only one* outcome X, Y, Z.

- Useful when there are many possible next steps but you only want to choose one.

```
if (testA()) {

    print("X");

} else if(testB()) {

    print("Y");

} else if(testC()) {

    print("Z");

}

print("End");
```



Playing with Graphical Procedures

- Today we'll introduce a simple graphics library called Turtle Graphics
  - It's a style of teaching introductory computer science that dates back to 1967!

- We have a number of procedures available to us to guide an invisible "turtle" on the screen who is dragging around a marker…

  `forward(n: number): void` – Moves the turtle forward by **`n`** pixels
  `left(rad: number): void` – Turns the turtle left by **`rad`** in radians
  `right(rad: number): void` – Turns the turtle right by **`rad`** in radians

- You can import these functions by:
  - **`import { forward, left, right } from "introcs/turtle";`**

Hands-on: Draw a Parametric Star

- Open 01-star-app.ts

- At TODO #0 draw a parametric "star" by:
  - Writing a **`for`** loop that iterates **`points`** parameter number of times
    - Don't forget to increment your counter variable inside the loop!
  - Moves **`forward`** by the **`diameter`** parameter amount
  - Turns **`left`** by the **`angle`** parameter amount

- At TODO #1 call the star function ("procedure") with the following parameters:
  - `5` for the points argument
  - `100` for the diameter argument
  - `4 / 5 * Math.PI` for the angle argument

- Done? Check-in on PollEv.com/compunc. Try playing around with different arguments!

```
export let main = async () => {

    // TODO #1 - Call the star procedure
    star(5, 100, 4 / 5 * Math.PI);

};

let star = (points: number, diameter: number, angle: number): void => {
    // TODO #0 - Draw a star!
    for (let i = 0; i < points; i++) {
        forward(diameter);
        left(angle);
    }
};
```

Follow Along: Placing Stars

- Open 02-starry-night-app.ts

- Let's add a procedure to place a randomly generated star at some x, y coordinate

- The moveTo procedure jumps the turtle to some x, y coordinate on the screen without drawing a line

```
moveTo(x, y);
let points = 5;
let diameter = 100;
let angle = 4 / 5 * Math.PI;
star(points, diameter, angle);
```