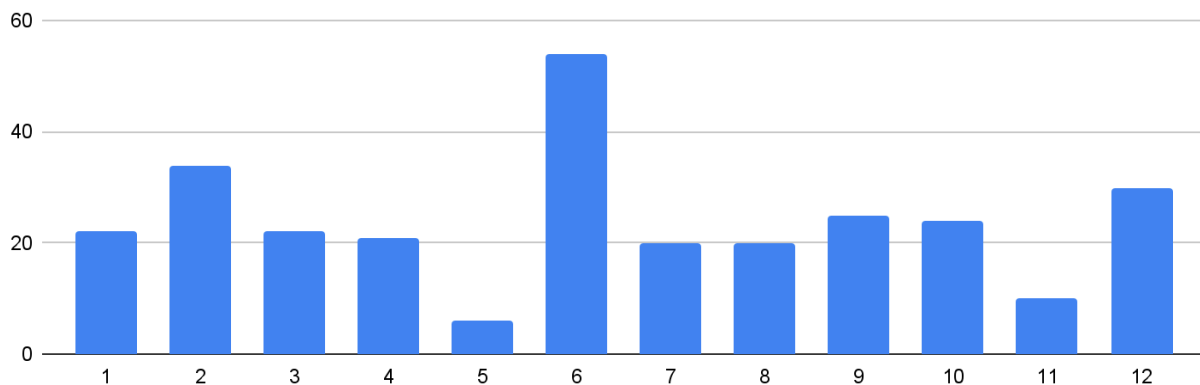


Please do not turn the page until 8:30am

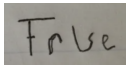
Rules:

- The exam is closed-book, closed-laptop, etc.
 - Except your own handwritten notes on a single 8.5x11in page.
- Please stop promptly at 10:20am.
- There are 288 points possible on the exam.
 - Points are distributed **unevenly** among 12 multi-part questions.



- **QUESTIONS VARY IN DIFFICULTY!**
- **GET EASY POINTS FIRST!**
- The exam is long. Be strategic with your time.
- The exam is printed double-sided, with pages numbered up to 33.

Advice:

- Read the questions carefully. Understand before you answer.
- Write down thoughts and intermediate steps.
- **Clearly** indicate your final answer.
 - **WARNING:** sneaky shenanigans like  will get negative points!
 - (Jared already tried this and it didn't work!!)
- Questions are not in order of difficulty. Try answering everything.
- If you have questions, ask.

Relax. You are here to learn.

Question 1 : Type Checking (22 points)

Which of the following TypeScript statements type check successfully? You can assume this code is inside the body of a function, not at the top level.

Code	Type checks? (T/F)
<pre>const x : bigint = 3n; const y : bigint = x + 2n;</pre>	
<pre>const x : string boolean = "hi"; if (x) { const y : string = x; }</pre>	
<pre>type T1 = {readonly a: string, readonly b: bigint number}; const x : T1 = {a: "331", b: 3n}; const y : T1 = {a: "final", b: x.b * 2};</pre>	
<pre>type T1 = {readonly a: string, readonly b: bigint number}; const x : T1 = {a: "hello", b: 2}; if (typeof x.a === "string" typeof x.b === "number") { const y : bigint = x.b; }</pre>	
<pre>const x: number = 3; const x: number = (3 * 6) + 24;</pre>	
<pre>type T1 = {readonly a: string, readonly b: bigint number}; const x : T1 = {a: "hello", b: 2}; if (typeof x.b === "number") { const y : string = x.a + "staff"; }</pre>	

(Question 1 continued)

Which of the following TypeScript statements type check successfully? You can assume this code is inside the body of a function, not at the top level.

Code	Type checks? (T/F)
<pre>const x : string number = "hello"; const y : number = x + 1;</pre>	
<pre>const x : number = 1; const y : bigint = BigInt(x) + 2n;</pre>	
<pre>const x = {a: "see", b: "you", c: 2}; const y = {a: "later", b: 3, c: "alligator"}; const z : string = x.a + y.c;</pre>	
<pre>const x : number = 4.78 const y : number = x - 3.14</pre>	
<pre>const x : [number, string] = [1.5, "hi"]; const [y, z] = x; const a : [string, bigint] = [y, BigInt(y)];</pre>	

Question 2 : Equality (34 points)

For each of the expressions below indicate whether it evaluates to true (T) or false (F).

Code	Eval (T/F)
<code>null === undefined</code>	
<code>"5" == 5</code>	
<code>const a = {x: 1, y: 2}</code> <code>const b = {x: 1, y: 2}</code> <code>a === b</code>	
<code>const a = {w: 2, x: 3}</code> <code>const b = {y: 4, z: 3}</code> <code>a.x === b.z</code>	
<code>3n === 3</code>	
<code>"hi" + "hello" === "hihello"</code>	
<code>1 == true</code>	
<code>const a = {x: "0"}</code> <code>const b = {y: 0}</code> <code>a.x === b.y</code>	
<code>"" === undefined</code>	
<code>{x: 1} === 1</code>	

(Question 2 continued)

For each of the expressions below indicate whether it evaluates to true (T) or false (F).

Code	Eval (T/F)
<pre>const c = {hi: "hello"} const a = {x: 3, y: c} const b = {x: 4, y: c} a.y === b.y</pre>	
<pre>4n == 4</pre>	
<pre>"" == false</pre>	
<pre>const a = {x: 5} const b = 5 a.x === b</pre>	
<pre>4.0 === 4</pre>	
<pre>{ } === { }</pre>	
<pre>const a : number[] = [0, 1, 2] const b : number[] = [0, 1, 2] a === b</pre>	

Question 3 : Encoding Inductive Datatypes (22 points total)**Part 1 (14 points)** Here are eight mathematical definitions of inductive datatypes:

$T1 := A \mid B \mid C(x: T1, y: T1)$ $T2 := A \mid B \mid C(x: \mathbb{Z}, y: T2)$ $T3 := A \mid B(x: S^*, y: T3, z: S^*)$ $T4 := A \mid B \mid C \mid D(x: S^*, y: T4)$	$T5 := A \mid B \mid C(x: \mathbb{Z}) \mid D(x: T5)$ $T6 := A(x: \text{bool}) \mid B(x: \text{bool})$ $T7 := A \mid B \mid C(x: \mathbb{N})$ $T8 := A(x: T8) \mid B(x: T8, y: T8)$
--	---

Match each of the TypeScript type definitions below to its corresponding math version (T1 - T8).

TypeScript Type Declaration	Corresponding Math Type (one of T1 - T8)
<pre>type _____ = {kind: "A"} {kind: "B", x: string, y: _____, z: string};</pre>	
<pre>type _____ = {kind: "A", x: boolean} {kind: "B", x: boolean}</pre>	
<pre>type _____ = {kind: "A"} {kind: "B"} {kind: "C", x: _____, y: _____};</pre>	
<pre>type _____ = {kind: "A"} {kind: "B"} {kind: "C", x: bigint, y: _____};</pre>	
<pre>type _____ = {kind: "A"} {kind: "B"}, {kind: "C", x: bigint}</pre>	
<pre>type _____ = {kind: "A", x: _____} {kind: "B", x: _____, y: _____}</pre>	
<pre>type _____ = {kind: "A"} {kind: "B"} {kind: "C", x: bigint} {kind: "D", x: _____};</pre>	

Question 3, Part 2 (8 points)

Consider the following math definitions of inductive datatypes:

$$T9 := A \mid B(z : \mathbb{Z})$$

$$T10 := C \mid D(t : T9)$$

Which of the following best describes the values that T10 represents?

Description of T10	Answer (one of A - G)
A) Binary trees with integer nodes B) Lists of integers C) Non-empty lists of integers D) Lists of lists of integers E) Non-empty lists of integers F) Lists of binary trees of integers G) None of the above	

Consider the following inductive datatype:

$$T11 := X(b : \text{boolean}, n : \mathbb{N}) \mid Y(b : \text{boolean}, n : \mathbb{N}, t : T11)$$

Which of the following best describes the values that T11 represents?

Description of T11	Answer (one of A - J)
A) Lists of natural numbers B) Lists of booleans C) Lists of (boolean, natural number) tuples D) Non-empty lists of natural numbers E) Non-empty lists of booleans F) Non-empty lists of (boolean, natural number) tuples G) Binary trees of natural numbers H) Binary trees of booleans I) Binary trees of (boolean, natural number) tuples J) None of the above	

Question 4 : Counting Inductive Datatypes (21 points)

For each of the following inductive datatypes encoded in TypeScript, how many values are there of that type?

Type	Distinct values (e.g., 0, 1, 2, ..., "infinity")
<pre>type A = { kind: "P" } { kind: "Q" }; { kind: "R" };</pre>	
<pre>type B = { kind: "S", value: B };</pre>	
<pre>type C = { kind: "CA", value: bool } { kind: "CB", value: B };</pre>	
<pre>type D = { kind: "DA", value: bool } { kind: "DC", value: C };</pre>	
<pre>type E = { kind: "X" } { kind: "Y", value: \mathbb{Z} };</pre>	
<pre>type F = { kind: "FA" } { kind: "FB", v1: F, v2: F, v3: F };</pre>	
<pre>type G = { kind: "GA", v1: [G, A] }; { kind: "GC", v1: [G, C] };</pre>	

Question 5: Pattern Matching (6 points)

Consider the this TypeScript function:

```
const u = (t: {a: [boolean, string], b: string}) : string|undefined => {
  const [m, n] = t.a;
  if (m) {
    if (n !== "") {
      return n;
    } else {
      return t.b;
    }
  }
}
```

In the Answer box below, select the correct translation (A / B / C / D) of translation to a mathematical definition using pattern matching. Recall that S is the type we use for characters (length 1 strings).

A.

func u ({a: x, b: y})	:=	y	for any $x \in S, y \in S$
func u ({a: x, b: y})	:=	x	for any $x \in S, y \in S$
func u ({a: (F, x), b: y})	:=	undefined	for any $x \in S, y \in S$

B.

func u ({a: (T, x), b: y})	:=	x	for any $x \in S$ where $x \neq ""$, $y \in S$
func u ({a: (T, ""), b: y})	:=	y	for any $x \in S, y \in S$

C.

func u ({a: (T, x), b: y})	:=	y	for any $x \in S, y \in S$
func u ({a: (F, x), b: y})	:=	x	for any $x \in S, y \in S$

D.

func u ({a: (T, x), b: y})	:=	x	for any $x \in S$ where $x \neq ""$, $y \in S$
func u ({a: (T, ""), b: y})	:=	y	for any $x \in S, y \in S$
func u ({a: (F, x), b: y})	:=	undefined	for any $x \in S, y \in S$

Answer	
--------	--

Question 6 : Specification Strength (54 points)**Part 1 (12 points)**

Consider a function `foo(n)` whose type is `bigint -> bigint`.

Here are some potential specifications for different implementations of this function.

A	<code>@requires n >= 0</code> <code>@returns -1 * n</code>
B	<code>@requires n >= 0</code> <code>@returns an integer</code>
C	<code>@requires n >= 0</code> <code>@returns an integer between 0 and 10, inclusive</code>
D	<code>@requires n >= 0</code> <code>@returns an integer between -5 and 5, inclusive</code>

For each pair of specifications below, mark whether the first is stronger than (S), weaker than (W), or incomparable to (I) the Answer columns.

Comparing	Answer (S / W / I)	Comparing	Answer (S / W / I)
A is ____ B		B is ____ C	
A is ____ C		B is ____ D	
A is ____ D		C is ____ D	

Question 6, Part 2 (12 points)

Consider a function `bar(n)` whose type is `bigint -> bigint`.

Here are some potential specifications for different implementations of this function.

A	<code>@requires n >= 0</code> <code>@returns a positive integer</code>
B	<code>@requires n > 0</code> <code>@returns a positive integer</code>
C	<code>@requires n is even</code> <code>@returns a positive integer</code>
D	<code>@requires n is an integer</code> <code>@returns a positive integer</code>

For each pair of specifications below, mark whether the first is stronger than (S), weaker than (W), or incomparable to (I) the Answer columns.

Comparing	Answer (S / W / I)	Comparing	Answer (S / W / I)
A is ____ B		B is ____ C	
A is ____ C		B is ____ D	
A is ____ D		C is ____ D	

Question 6, Part 3 (30 points)

Consider `arr.indexOf(x)`. It searches an array `arr` for a particular value `x`. Here are some potential specifications for different implementations of this function.

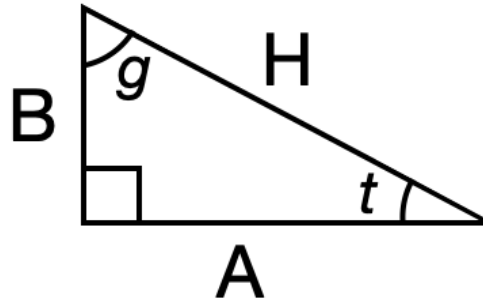
A	<code>@requires arr.length > 0 /\ arr[i] = x for some $0 \leq i < \text{arr.length}$</code> <code>@returns k such that $\text{arr}[k] = x /\ 0 \leq k < \text{arr.length}$</code>
B	<code>@requires arr.length > 0</code> <code>@throws Exception if $\text{arr}[i] \neq x$ for all $0 \leq i < \text{arr.length}$</code> <code>@returns i such that $\text{arr}[i] = x /\ 0 \leq i < \text{arr.length}$</code>
C	<code>@returns i such that $\text{arr}[i] = x /\ 0 \leq i < \text{arr.length} /\$</code> <code>$-1$ if $\text{arr}[i] \neq x$ for all $0 \leq i < \text{arr.length}$</code>
D	<code>@requires $\text{arr}[i] = x$ for some $0 \leq i < \text{arr.length}$</code> <code>@returns k such that $\text{arr}[k] = x /\ 0 \leq k < \text{arr.length} /\$</code> <code>$\text{arr}[j] = x \Rightarrow k \leq j$ for all $0 \leq j < \text{arr.length}$</code>
E	<code>@requires $\text{arr.length} \% 2 == 0 /\ \text{arr}[i] = x$ for some $0 \leq i < \text{arr.length}$</code> <code>@returns k such that $\text{arr}[k] = x /\ 0 \leq k < \text{arr.length}$</code>

For each pair of specifications below, mark whether the first is stronger than (S), weaker than (W), or incomparable to (I) the Answer columns.

Comparing	Answer (S / W / I)	Comparing	Answer (S / W / I)
A is ____ B		B is ____ D	
A is ____ C		B is ____ E	
A is ____ D		C is ____ D	
A is ____ E		C is ____ E	
B is ____ C		D is ____ E	

Question 7 : Spec and Implement ADTs (20 points total)

The next page specifies an abstract datatype (ADT), representing 2D right triangles.



Useful Facts: for a right triangle with side lengths A and B and hypotenuse length H :

- All lengths (A , B , and H) must be positive
- The triangle's side lengths are related by $A^2 + B^2 = H^2$ (Pythagorean Theorem)
- The triangle's area is $(A * B) / 2$
- Trigonometry relates angles and the ratios of their adjacent/opposite side lengths

Angle	Between Sides	cosine	sine
t	A and H	$\cos(t) = A / H$	$\sin(t) = B / H$
g	B and H	$\cos(g) = B / H$	$\sin(g) = A / H$

- Inverse trig functions **acos** and **asin** “undo” sine and cosine for angles $< \pi / 2$ (like our angles t and g since we are only considering right triangles)
 - $x = \text{acos}(\cos(x))$ and $x = \text{asin}(\sin(x))$
 - For angles $< \pi / 2$, both **acos** and **asin** return values between 0 and $\pi / 2$
- Assume angles are measured in radians (and that **sin** and **cos** work in radians)
- Assume angles are measured in radians (and that **sin** and **cos** work in radians)

Because the side lengths and angles of right triangles are strictly related, we have many choices in how to represent them concretely. However, we should be able to specify them abstractly in a way that lets clients reason about how their code uses our library while reserving freedom for ourselves to change the implementation later if needed.

In this question, we first give a specification for a right triangle library. Then we will study some different choices for how the library could be implemented.

Consider this specification for a 2D right triangle library:

```
/**
 * Represents a 2D right triangle with side A, side B, and hypotenuse H.
 * We denote a right triangle as obj = (A, B, H).
 * We refer to the angle between side A and the hypotenuse H as angleAH.
 * We refer to the angle between side B and the hypotenuse H as angleBH.
 */
export interface RightTri {
  /** @returns the length of side A */
  getSideA : () => number;

  /** @returns the length of side B */
  getSideB : () => number;

  /** @returns the length of hypotenuse H */
  getHypot : () => number;

  /** @returns the area of obj */
  area : () => number;

  /** @returns the angle between side A and hypotenuse H */
  getAngleAH : () => number;

  /** @returns the angle between side B and hypotenuse H */
  getAngleBH : () => number;
}

// factory functions below (because constructors are goofy!)

/**
 * Make a right triangle from its side lengths.
 * @requires 0 < sides.a and 0 < sides.b
 */
const ofSides = (sides : {a: number, b: number}) : RightTri => { ... }

/**
 * Make a right triangle from side A and the hypotenuse lengths.
 * @requires 0 < sideA and 0 < hyp
 */
const ofHypSideA = (hyp: number, sideA : number) : RightTri => { ... }
```

Question 7, Part 1 (10 points)

Here is one possible implementation of our right triangle library.

```
class SimpleRightTri implements RightTri {
  readonly a: number;
  readonly b: number;
  readonly area : number;

  constructor(a: number, b: number) {
    this.a = a;
    this.b = b;
    this.area = (a * b) / 2;
  }

  getSideA = () : number => this.a;

  getSideB = () : number => this.b;

  getHypot = () : number => Math.sqrt(this.a * this.a + this.b * this.b);

  area = () : number => this.area;

  getAngleAH = () : number => Math.acos(this.a / this.getHypot());

  getAngleBH = () : number => Math.acos(this.b / this.getHypot());
}

const ofSides = (sides: {a: number, b: number}) : RightTri => {
  return new SimpleRightTri(sides.a, sides.b);
};

const ofHypSideA = (hyp: number, sideA: number) : RightTri => {
  const sideB = Math.sqrt(hyp * hyp - sideA * sideA);
  return new SimpleRightTri(sideA, sideB);
};
```

On the next page we will ask you some questions about abstraction functions (AF) and representation invariants (RI) for this concrete implementation of our right triangle library specification.

Which of the following is an appropriate abstraction function (AF) for SimpleRightTri ?

Possible AF for SimpleRightTri	Answer
A) <code>obj = (this.a, this.b)</code> B) <code>obj = (this.getSideA(), this.getSideB())</code> C) <code>obj = (this.a, this.b, sqrt(this.a² + this.b²))</code> D) <code>obj = (this.a, this.b, this.h)</code> E) <code>obj = this</code> F) <code>obj = (Math.cos(this.t * this.h), this.h)</code>	

Which of the following would be the **most appropriate** representation invariant (RI) for SimpleRightTri?

Possible RI for SimpleRightTri	Answer
A) <code>0 < this.a and 0 < this.b</code> B) <code>this.a != 0 and this.b != 0</code> C) <code>this.getHypot() < this.area()</code> D) <code>this.b < 0 or this.h > 0</code> E) <code>0 < this.a and 0 < this.b and this.area = (this.a * this.b) / 2</code> F) <code>this.area = (this.a * this.b) / 2</code> G) <code>this.a < this.b</code> H) <code>this.a + this.b < this.getHypot()</code> I) <code>getAngle(this.a, this.b) === Math.PI/2</code> J) <code>console.log(`Jared loves right triangles! \${rep_inv_here}`);</code> K) No representation invariant needed	

Question 7, Part 2 (10 points)

Here is another possible implementation of our right triangle library.

```
class OtherRightTri implements RightTri {
  readonly h: number; // hypotenuse
  readonly t: number; // angle between side A and hypotenuse

  constructor(h: number, t: number) {
    this.h = h;
    this.t = t;
  }

  getSideA = () : number => this.h * Math.cos(t);

  getSideB = () : number => this.h * Math.sin(t);

  getHypot = () : number => this.h;

  area = () : number => (this.getSideA() * this.getSideB()) / 2;

  getAngleAH = () : number => this.t;

  getAngleBH = () : number => Math.PI/2 - this.t;
}

const ofSides = (sides: {a: number, b: number}) : RightTri => {
  const h = Math.sqrt(sides.a * sides.a + sides.b * sides.b);
  const t = Math.acos(sides.a / h);
  return new OtherRightTri(h, t);
};

const ofHypSideA = (hyp: number, sideA: number) : RightTri => {
  const t = Math.acos(sideA / hyp);
  return new OtherRightTri(hyp, t);
};
```

On the next page we will ask you some questions about abstraction functions (AF) and representation invariants (RI) for this concrete implementation of our right triangle library specification.

Which of the following is a valid abstraction function (AF) for OtherRightTri ?

Possible AF for OtherRightTri	Answer
A) <code>obj = (this.a, this.b)</code> B) <code>obj = (this.getSideA(), this.getSideB())</code> C) <code>obj = (this.getSideA(), this.getSideB(), this.getHypot())</code> D) <code>obj = (this.a, this.b, sqrt(this.a² + this.b²))</code> E) <code>obj = (this.a, this.b, this.h)</code> F) <code>obj = this</code> G) <code>obj = (Math.cos(this.t * this.h), Math.acos(this.h))</code>	

Which of the following would be the **most appropriate** representation invariant (RI) for SimpleRightTri?

Possible RI components for OtherRightTri	Answer
A) <code>0 < this.a</code> B) <code>this.h != 0 and this.t != 0</code> C) <code>0 < this.h and 0 < this.t < Math.PI/2</code> D) <code>this.getHypot() > this.area()</code> E) <code>0 < this.h</code> F) <code>this.a < this.b</code> G) <code>this.t < 90</code> H) <code>this.getSideA() + this.getSideB() < this.getHypot()</code> I) <code>this.t > 0</code> J) No representation invariant needed	

Question 8 : Hoare Triples (20 points)

Indicate which of the Hoare triples below are valid. Assume x and y are bigints.

Hoare Triple	Valid (T/F)
<pre> {{x > 0}} y = 3n * x; {{y ≥ 3}}</pre>	
<pre> {{0 > y and y ≥ x}} z = y * x; {{z > 0}}</pre>	
<pre> {{x ≠ 0}} if (x >= 0n) { y = -5n * x; } else { y = 5n * x; } {{y ≠ 0 and y = 5x }}</pre>	
<pre> {{{}} if (y <= 5n) { z = y + 5n; } else { z = x % 9n; } {{z ≤ 10}}</pre>	
<pre> /** @param n: an integer * @param t: an integer * @returns positive int if n > t, else negative int */ const bar = (n: bigint, t: bigint): bigint => { ... } {{x > 0 and y > 0}} if (x >= y) { x = -1n * y; } z = bar(x, y); {{z < 0}}</pre>	

Question 9 : Loop Invariants (25 points total)**Part 1 (9 points)**

At which points in a function must the loop invariant be true? Please answer true (T) or false (F) for each program point where the loop invariant must hold.

Program Point	Loop Invariant Must Hold? (T / F)
Just before the first iteration of the loop	
At the beginning of the function	
Immediately after entering the loop	
Immediately after exiting the loop	
When the function returns	
Before calling any functions from inside the loop	
At the end of the loop body	
Just before any conditionals in the loop body	
Throughout the loop body	

Question 9, Part 2 (8 points)

Below we have a sketch of a function that looks for the value 3 in an array of bigints.

The loop already has an invariant, but we will need to fill in the blanks to complete the code so that it maintains the loop invariant and guarantees the function's postcondition.

```
// @returns an index i such that A[i] = 3
//           or -1 if no such index exists
const findThrees = (A: bigint[]): number => {

  let i = /* BLANK 1 */;

  // {{ Inv: A[j] ≠ 3 for any j where 0 ≤ j ≤ i
  //           and /* BLANK 1 */ ≤ i }}
  while (i !== /* BLANK 2 */ ) {

    if (A[/* BLANK 3 */] === 3n) {

      return /* BLANK 4 */;

    }

    i = i + 1;
  }

  // {{ Post: A[j] ≠ 3 for any j where 0 ≤ j < A.length }}
  return -1;
}
```

(Question 9, Part 2 continued)

Select the option for each blank such that the function maintains the loop invariant and guarantees the post condition.

Options for Blank	Answer (A / B / C / D / E)
<pre>let i = <u>/* BLANK 1 */</u> ;</pre> <p>A) -2 B) -1 C) 0 D) 1 E) 2</p>	
<pre>while (i !== <u>/* BLANK 2 */</u>) {</pre> <p>A) A.length - 2 B) A.length - 1 C) A.length D) A.length + 1 E) A.length + 2</p>	
<pre>if (A[<u>/* BLANK 3 */</u>] === 3n) {</pre> <p>A) A[i - 2] B) A[i - 1] C) A[i] D) A[i + 1] E) A[i + 2]</p>	
<pre>return <u>/* BLANK 4 */</u> ;</pre> <p>i - 2 i - 1 i i + 1 i + 2</p>	

Question 9, Part 3 (8 points)

The following math function **del-vowels** takes an array of characters and returns it with all vowels (a, e, i, o, u, y) removed:

```

func is-vowel(c) := c = 'a' || c = 'e' || c = 'i' || c = 'o' || c = 'u' || c = 'y'

func del-vowels([]) := []
  del-vowels(A ++ [c]) := del-vowels(A)           where is-vowel(c) = T
  del-vowels(A ++ [c]) := del-vowels(A) ++ [c]     where is-vowel(c) = F

```

Here is a TypeScript implementation of **del-vowels** that removes all vowels from an array of characters (length-1 strings) *in-place* (i.e., it mutates the array). Assume the function `isVowel` returns true if its length 1 string argument is a vowel character (a, e, i, o, u, y).

```

/** Removes all vowels from a given array
 * @param str array of characters (all length-1 strings)
 * @modifies str
 * @effects str = del-vowels(str)
 */
const deleteVowels = (str: string[]) => {
  let j: number = 0;
  let i: number = 0;
  const n: number = str.length;

  // Update str, shift non-vowel chars forward to replace vowels
  // Inv1: _____
  while(j !== str.length) {
    if (!isVowel(str[j])) {
      str[i] = str[j];
      i = i + 1;
    }
    j = j + 1;
  }

  // Adjust str length since non-vowels have shifted forward
  // Inv2: _____
  while (i !== str.length) {
    str.pop(); // Removes unused indices from the end
  }
}

```

On the next page, we will identify loop invariants for **Inv1** and **Inv2**.

(Question 9, Part 3 continued)

Remember that n is `str.length`.

Select the invariant for **Inv1** that the first loop maintains.

Options for Inv1	Answer (A / B / C / D)
A) $\text{str}_0[0 \dots j-1] = \text{del-vowels}(\text{str}[0 \dots i-1])$ $\text{and } \text{str}[i \dots n-1] = \text{str}_0[i \dots n-1]$ $\text{and } i \leq j < n$	
B) $\text{str}[0 \dots i-1] = \text{del-vowels}(\text{str}_0[0 \dots j-1])$ $\text{and } \text{str}[j \dots n-1] = \text{str}_0[j \dots n-1]$ $\text{and } i \leq j < n$	
C) $\text{str}[0 \dots i-1] = \text{del-vowels}(\text{str}_0[0 \dots j-1])$ $\text{and } \text{str}_0[j \dots n-1] = \text{str}[j \dots n-1]$ $\text{and } i < j < n$	
D) $\text{str}[0 \dots n-1] = \text{del-vowels}(\text{str}_0[0 \dots n-1])$ $\text{and } \text{str}[0 \dots i-1] = \text{str}_0[0 \dots j-1]$ $\text{and } i < j \leq n$	

Select the invariant for **Inv2** that the second loop maintains.

Options for Inv2	Answer (A / B / C / D)
A) $\text{str}_0[0 \dots i-1] = \text{del-vowels}(\text{str}_0)$ $\text{and } \text{str}[i \dots n-1] = \text{str}_0[i \dots n-1]$	
B) $\text{str}[0 \dots i-1] = \text{del-vowels}(\text{str}_0)$ $\text{and } i < n$	
C) $\text{str}_0[0 \dots i-1] = \text{del-vowels}(\text{str})$ $\text{and } \text{str}[i \dots n-1] = \text{str}_0[i \dots n-1]$ $\text{and } i < n$	
D) $\text{str}[0 \dots i-1] = \text{del-vowels}(\text{str}_0)$ $\text{and } i \leq n$	

Question 10 : Strength of Assertions (24 points)

Select the strongest assertion in each set. The first row is filled in as an example. Assume all variables are of type `bigint`.

Assertions	Strongest
A. $\{\{ \text{True} \}\}$ B. $\{\{ \text{False} \}\}$ C. $\{\{ z = 42 \}\}$	B
A. $\{\{ x \geq 5n \}\}$ B. $\{\{ x > 5n \}\}$	
A. $\{\{ x = 2n \}\}$ B. $\{\{ x \% 2n = 0 \}\}$	
A. $\{\{ x > 0n \wedge y = 32n \}\}$ B. $\{\{ (x > 0n \wedge y = 32n) \vee (z = 12n) \}\}$	
A. $\{\{ x \neq 21n \}\}$ B. $\{\{ x = 21n \}\}$ C. $\{\{ x \geq 21n \}\}$	
A. $\{\{ x \geq 5n \}\}$ B. $\{\{ x \geq 0n \}\}$ C. $\{\{ x \geq -5n \}\}$	
A. $\{\{ x \neq 3n \}\}$ B. $\{\{ x = 10n \}\}$ C. $\{\{ x > 27n \}\}$	
A. $\{\{ x - 2n + y \geq 15n \}\}$ B. $\{\{ x - 2n + y \geq 15n \}\}$	
A. $\{\{ 16n \leq x \wedge x \leq 24n \}\}$ B. $\{\{ 15n \leq x \wedge x \leq 23n \}\}$ C. $\{\{ 18n \leq x \wedge x \leq 22n \}\}$	

Question 11 : Aliasing and Mutable ADTs (10 points)

Consider the following interface:

```
interface Stack {
  push(x: bigint) : void
  pop() : bigint
}
```

For the implementation below, determine whether there is a representation exposure. If there is, answer with the line number where the rep is exposed. Otherwise, just answer "No".

Stack Implementation	Rep Exposure?
<pre> 1 class StackArray implements Stack { 2 3 stack: bigint[]; 4 5 constructor(vals: bigint[]) { 6 this.stack = vals; 7 } 8 9 push(x: bigint): void { 10 this.stack.push(x); 11 } 12 13 pop(): bigint { 14 const last = this.stack.pop(); 15 if (last === undefined) { 16 throw new Error("stack is empty"); 17 } 18 return last; 19 } 20 }</pre>	

(Question 11 Continued)

Here is the same interface for Stack repeated:

```
interface Stack {
  push(x: bigint) : void
  pop() : bigint
}
```

For the implementation below, determine whether there is a representation exposure. If there is, answer with the line number where the rep is exposed. Otherwise, just answer "No".

Stack Implementation	Rep Exposure?
<pre> 1 class StackList implements Stack { 2 stack: List<bigint>; 3 constructor() { 4 this.stack = nil; 5 } 6 7 push(x: bigint): void { 8 this.stack = cons(x, this.stack); 9 } 10 11 pop(): bigint { 12 if (this.stack.kind === "nil") 13 throw new Error("stack is empty"); 14 return this.stack.hd; 15 } 16 }</pre>	

Question 12 : Subtypes (30 points)

Consider the following interface:

```
// a regular polygon is an n-sided convex shape where
//   - all sides have equal length
//   - all interior angles have equal measure
interface RegularPolygon {
  getArea: () => number;
  getPerimeter: () => number;
}
```

Are the following subclasses proper *subtypes* of RegularPolygon?

Subclass of RegularPolygon	Subtype? (T / F)
<pre>class Triangle implements RegularPolygon { s1: number; s2: number; s3: number; constructor(s1: number, s2: number, s3: number) { this.s1 = s1; this.s2 = s2; this.s3 = s3; } getArea = () : number => { const s = (this.s1 + this.s2 + this.s3) / 2; return Math.sqrt(s*(s - this.s1)*(s - this.s2)*(s - this.s3)); } getPerimeter = () : number => { return this.s1 + this.s2 + this.s3; } }</pre>	
<pre>class Square implements RegularPolygon { s: number; constructor(s: number) { this.s = s; } getArea = (): number => { return this.s * this.s; } getPerimeter = (): number => { return this.s * 4; } }</pre>	

Consider the following type

```
// a resizable polygon is an n-sided convex shape where
// - the side lengths can be changed to grow or shrink the polygon
interface ResizablePolygon {
  getArea: () => number;
  getPerimeter: () => number;
  setSides: (sides: number[]) => void;
}
```

Are the following subclasses proper *subtypes* of ResizablePolygon?

Subclass of ResizablePolygon	Subtype? (T / F)
<pre>class ResizableTriangle implements ResizablePolygon { s1: number; s2: number; s3: number; constructor(s1: number, s2: number, s3: number) { this.s1 = s1; this.s2 = s2; this.s3 = s3; } getArea = (): number => { const s = (this.s1 + this.s2 + this.s3) / 2; return Math.sqrt(s*(s - this.s1)*(s - this.s2)*(s - this.s3)); } getPerimeter = (): number => this.s1 + this.s2 + this.s3; setSides = (sides: number[]): void => { if (sides.length !== 3) { throw new Error("invalid"); } this.s1 = sides[0]; this.s2 = sides[1]; this.s3 = sides[2]; }; }</pre>	
<pre>class ResizableSquare implements ResizablePolygon { s: number; constructor(s: number) { this.s = s; } getArea = (): number => { return this.s * this.s; } getPerimeter = (): number => { return this.s * 4; } setSides = (sides: number[]): void => { this.s = sides[0]; } }</pre>	

Consider the following type:

```
// a resizable regular polygon is an n-sided convex shape where
//   - all sides have equal length
//   - all interior angles have equal measure
//   - the side length can be changed to grow or shrink the polygon
interface ResizableRegularPolygon {
  getArea: () => number;
  getPerimeter: () => number;
  setSideLength: (len: number): void;
}
```

Are the following subclasses proper *subtypes* of ResizableRegularPolygon?

Subclass of ResizableRegularPolygon	Subtype? (T / F)
<pre>class ResizableTriangle2 implements ResizableRegularPolygon { s1: number; s2: number; s3: number; constructor(s1: number, s2: number, s3: number) { this.s1 = s1; this.s2 = s2; this.s3 = s3; } getArea = (): number => { const s = (this.s1 + this.s2 + this.s3) / 2; return Math.sqrt(s*(s - this.s1)*(s - this.s2)*(s - this.s3)); } getPerimeter = (): number => this.s1 + this.s2 + this.s3; setSideLength = (len: number): void => { this.s1 = len; this.s2 = len; this.s3 = len; }; }</pre>	
<pre>class ResizableSquare2 implements ResizableRegularPolygon { s: number; constructor(s: number) { this.s = s; } getArea = (): number => { return this.s * this.s; } getPerimeter = (): number => { return this.s * 4; } setSideLength = (sideLength: number): void => { this.s = sideLength; } }</pre>	

Jared Bonus #1 (2 BONUS Points Possible)

Remember our right triangle library from Question 7. In one project, Jared implemented the `RightTri` interface using the `SimpleRightTri` class and corresponding factory functions. However, some of his users want to make a right triangle given one of the angles and the length of the hypotenuse. Implement this factory function to help Jared's users out. You have to get the code perfect to get bonus points! Make sure to use `SimpleRightTri` for this bonus!

```
/**
 * Make a SimpleRightTri from one of its angles and the hypot length.
 * ofAH(["AH", x], h) gives tri from angle x between side A and hypot h
 * ofAH(["BH", x], h) gives tri from angle x between side B and hypot h
 * @requires angle measure and hyp to be positive */
const ofAH = (angle: ["AH" | "BH"], number, hyp: number) : RightTri => {

}
}
```

Jared Bonus #2 (2 BONUS Points Possible)

Remember our right triangle library from Question 7. In another project, Jared implemented the `RightTri` interface using the `OtherRightTri` class and corresponding factory functions. However, some of his users want to make a right triangle given either of the sides and the length of the hypotenuse. Implement this factory function to help Jared's users out. You have to get the code perfect to get bonus points! Make sure to use `OtherRightTri` for this bonus!

```
/**
 * Make a SimpleRightTri from either of its sides and the hypot length.
 * ofSH(["A", x], h) gives tri with side A length x and hypot length h
 * ofSH(["B", x], h) gives tri with side B length x and hypot length h
 * @requires side length and hyp to be positive */
const ofSH = (side: ["A" | "B"], number, hyp: number) : RightTri => {

}
```


Jared Bonus #3 (1 BONUS Point Possible)

This quarter we exhausted all our Jared jokes! Unless we can get some new ones, future students in 331 may not hear about our favorite “friend of the class”.

Please give us a new Jared joke below! If you can’t come up with one, that is OK too. Instead, just share which moments of the course were your favorite.

THE END

Thanks for a great quarter!
Hope you have a fantastic Spring Break!!