University of Jijel – Master AI

Faculty of Exact Sciences and Computer Science
Department of Computer Science

Project 02

Reasoning About Programs with Assertions

Project Report

Instructor: Prof. Tarek BOUTEFARA

Prepared by: Ahmed Hemimed

January 1, 2026

1 Introduction

testing is essential and is used to address software; it can only show the presence of bugs not their absence. In this work, you will explore ideas from axiomatic semantics and Hoare logic through concrete programming experiments. Instead of working with formal proofs, we will check the program correctness by annotating JavaScript programs with assertions that describe what should be true before **precondition**, during, and after execution **postcondition**

1.1 What is Hoare logic

Hoare logic (also known as Floyd–Hoare logic or Hoare rules) is a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs.[1] **Hoare triples**:

$$\{P\}\ C\ \{Q\}$$

where:

- P is the **precondition** – is a condition must be true before the programe or the work start
- C is the program or the expression.
- Q is the **postcondition** – the true result in the final (what must be true in the final)

1.2 imperative programming language:

Imperative programming is a paradigm of computer programming where the program describes steps that change the state of the computer. Unlike declarative programming, which describes "what" a program should accomplish, imperative programming explicitly tells the computer "how" to accomplish it.[2]

imperative language include:

- Variable assignments

```
var Name = "Ahmed";
console.log(`var Name = ${Name}`);
let age = 22;
console.log(`let age = ${age}`);
```

- Conditionals (if)

```
let age = 20;
if (age >= 18) {
    console.log("You are an adult");
    console.log("You can vote");
}
```

- Loops (while)

```
// Count from 1 to 5
let count = 1;
while (count <= 5) {
    count++;
}
console.log(‘final result: ${count}‘);
```

JavaScript is a straightforward programming language that helps explain how programs run and the usage of hoare logic with assertion.

1.3 assertion in Java script

The assertions in java script is a condition to check in the code to verify that a specific property holds during execution. It is used to check hypotheses like preconditions,postecondition . assertion have a direct relation to the Hoare logic, they both use to reason about program correctness.

in java script The assert() method writes a message to the console if an expression evaluates to false[3].

2 The usage Of asserions in java script

2.1 the syntax in java script :

console.assert(expression, message)

2.2 This is an example of using assertions:

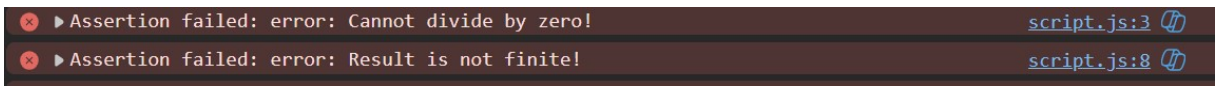
- The division by 0 in math In mathematics and programming dividing by 0 is impossible and the result is undefined. to fix that we use assersion in Java Script:
  - N / 0 = undefined
  - 0 / 0 = NaN

The programme using assertions

```

function div(a, b) {
    // pre condition P => b must not equal to 0
    console.assert(b !== 0, "error => cannot divide by 0");
    var div_result = a / b;
    // post condition Q => result must be in the final
    console.assert(isFinite(div_result), "error = > result is not
   finite");
    return div_result;
}
```

The result when we try to do 10 / 0 , and 0 / 0:



each programs is annotated By:

```
// Precondition:
// Loop invariant C:
// Postcondition:
```

these annotations are translated into executable assertions placed:

- Before the program start (precondition)
- At the beginning or end of each loop iteration (invariant C)
- After the program finish (postcondition)

3 Programs with Assertions in java script

3.1 Program 1: Sum of the First n Numbers

```
// precondition: n >= 0
// postcondition: sum == n*(n+1)/2

function sumFirstN(n) {
    console.assert(n >= 0, "Precondition: n must be non-negative");

    let sum = 0;
    let i = 0;

    // Loop invariant: sum == i*(i-1)/2 && i <= n
    while (i <= n) {
        console.assert(sum === i * (i - 1) / 2 && i <= n,
                       `Invariant violated at i=${i}, sum=${sum}`);
        sum += i;
        i++;
    }

    console.assert(sum === n * (n + 1) / 2,
                   `Postcondition failed: sum=${sum}, expected=${n*(n+1)
   /2}`);
    return sum;
}

// Test
console.log(sumFirstN(5)); // 15
```

explanation of the code and the assertion:

- **Precondition p** : $n \geq 0$ do only the sum of positive numbers.
- **Invariant C**: in each loop start, `sum` must contains only the sum of numbers from 0 to $i - 1$.
- **Postcondition Q**: in the final of the loop, `sum` must equal to $\frac{n(n+1)}{2}$.

3.2 Program 2: the max in an array

```
// precondition P => tableau.length > 0

// post condition Q => max is the maximum element in the table

function max(tableau) {

    console.assert(tableau.length >=1, "precondition p => table must
   have at least one element");

    let max =tableau[0];

    let i =1;

    // loop invariant Q => max ==Math.max(tableau[0..i-1])
    while (i < tableau.length) {
    console.assert(max==Math.max(...tableau.slice(0, i)), `error at i=${
   i}, max=${max}`);

        if (tableau[i] >=max) {
            max =tableau[i];
        }
        i++;
    }
    console.assert(max ==Math.max(...tableau),

                   `postcondition Q failed => max =${max}  expected
   value=${Math.max(...tableau)}`);
    return max;
}
// Test
console.log(max([3, 1, 4, 1, 5, 9])); // 9
```

explanation of the code and the assertion:

- **precondition P**: the array must not be empty.
- **invariant C**: we have a max from the elements they pass.
- **Postcondition**: max must be the maximum of the entire array.

3.3 Program 3: the maximum of two numbers using a conditional statement

```
function max(x,y){
// precondition: x and y must be numbers
console.assert(typeof x ==="number" && typeof y ==="number");
let max;
if (x >=y) { max = x;
}
else {max = y;
}
// post condition Q =>max >=x and max >=y
console.assert(max >=x && max >=y);
return max;
}
```

explanation of the code and the assertion:

- **precondition P**: the two param must be numbers.
- **postcondition Q**:the maximum is eqaule to one of the number and greater then the other

3.4 Program 4: Checking if a Number is Prime

```
    // precondition P =>n>1
// postcondition Q => pre is true iff n is prime

function pre(n) {
    console.assert(n > 1, "precondition P => n > 1");

    let div = 2;
    let premier = true;

    // Loop invariant C: n not divisible by any numbeer in [2, divisor
    -1]
    while (div * div <= n) {
        console.assert(!isDiv(n, 2, div - 1),
                       `Invariant violated at divisor=${div}`);
        if (n % div === 0) {
            premier = false;
            break;
        }
        div++;
    }

    console.assert(premier === (n > 1 && !isDiv(n, 2, n - 1)),
                   `Postcondition failed: prime=${premier}, n=${n}`);
    return premier;
}

function isDiv(n, a, b) {
    for (let i = a; i <= b; i++) {
        if (n % i === 0) return true;
    }
    return false;
}

console.log(pre(7));
console.log(pre(10));
```

Explanation:

- **Precondition**: We only test numbers > 1.
- **Invariant**: No integer from 2 to divisor $- 1$ divides n.
- **Postcondition**: `prime` is true if and only if n is prime.

3.5 Program 5: factorial

```
/function facto(n) {
//precondition P=> n >= 0
console.assert(n >=0 , "n must be >= 0");

let f =1;
let i =1;
while (i <=n) {
// loop invariant Q=> f = i!
console.assert(f >0);
f =f*i;
i++;
}
// postcondition Q=> f >0 or f >=n
console.assert(f >0 || f>=n);
return f;
}
console.log(facto(5)); // 120
console.log(facto(0)); // 1
```

explanation of the code and the assertion:

- **precondition P**: the number must be >=0.
- **invariant Q**: the factorial must be >0 (must increased) .
- **postcondition Q**: the facto in the final must be >0 or >=n (facto(1) = 1).

4 Observations and Challenges

4.1 Strengths of Assertion-Based Reasoning

- Makes implicit assumptions explicit.
- Helps detect logical errors early.
- Encourages rigorous thinking about loop behavior.

4.2 limitations of assertion

Runtime assertions are useful for detecting errors during execution of the codes, but they have many limitations like:

- They only check executed paths of code
- They do not constitute a complete formal proof.
- They depend on test inputs of the program
- They cannot guarantee correctness for all possible executions cases

4.3 why proving Correctness is Harder Than testing

- testing cover a finite number of cases; proofs must cover all possible cases.
- Formulating loop invariants can be complex.
- Real-world programs have complex state and side effects.

5 Conclusion

This work allowed us to put Hoare logic concepts into practice through the use of executable assertions in JavaScript. Programs with assertions, preconditions (P), invariants (C), and postconditions (Q) provide a structured approach to reasoning about the correctness of the program. Also, these techniques are useful for debugging and understanding the program's logic.

6 References

1. Wikipedia contributors. (n.d.). Hoare logic. In *Wikipedia, The Free Encyclopedia*. Retrieved December 29, 2025, from `https://en.wikipedia.org/wiki/Hoare_logic`
2. Computer Hope. (2024, September 15). Imperative programming. Retrieved from `https://www.computerhope.com/jargon/i/imp-programming.htm`
3. W3Schools. (n.d.). *console.assert()*. Retrieved December 29, 2025, from `https://www.w3schools.com/Jsref/met_console_assert.asp`