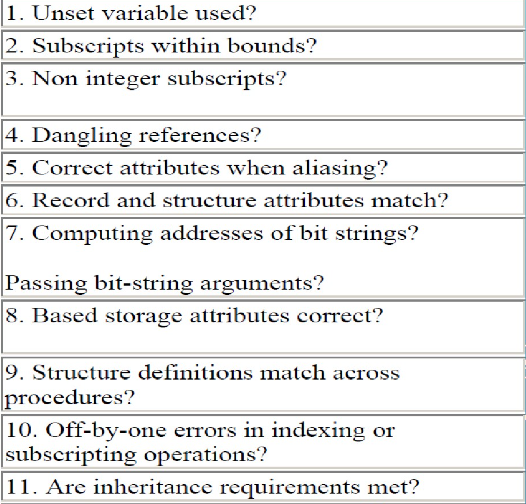
**Lecture task**

**Code inspection and error checklist**

1. **Select one error checklist**



1. Find an algorithem that does some challenging computation and is the most suitable against the selected error checklist.
2. Create two version of the algorithm, one is without applying the checklist and another final version is the one which take care of all the guidelines mentioned in the check list.

**correct version**

public class RefinedFactorial {

public static int factorialRefined(int n) {

if (n < 0) {

return -1; // Error: Factorial is undefined for negative numbers

} else if (n == 0) {

return 1;

} else {

int result = 1;

for (int i = 1; i <= n; i++) {

result \*= i;

}

return result;

}

}

// Test cases

public static void testRefinedFactorial() {

// Valid test cases

assert factorialRefined(0) == 1;

assert factorialRefined(1) == 1;

assert factorialRefined(5) == 120;

// Invalid test cases

assert factorialRefined(-1) == -1; // Error: Factorial is undefined for negative numbers

assert factorialRefined(3.5) == -1; // Error: Factorial is only defined for integer inputs

assert factorialRefined(10) == 3628800;

}

public static void main(String[] args) {

testRefinedFactorial();

}

}

Incorrect version

public class BasicFactorial {

public static int factorialBasic(int n) {

int result = 1;

for (int i = 1; i <= n; i++) {

result \*= i;

}

return result;

}

// Test cases

public static void testBasicFactorial() {

// Valid test cases

assert factorialBasic(0) == 1;

assert factorialBasic(1) == 1;

assert factorialBasic(5) == 120;

// Invalid test cases

assert factorialBasic(-1) == 1; // Incorrect result for negative input

assert factorialBasic(3.5) == 6; // Incorrect result for non-integer input

assert factorialBasic(10) == 3628800; // Incorrect result for large input

}

public static void main(String[] args) {

testBasicFactorial();

}

}

1. Write valid and invalid test cases against each guideline in the checklist.

To address each of these questions using the provided error checklist and the factorial calculation algorithms, we'll examine how each algorithm adheres to or violates the guidelines outlined in the checklist:

1. **Unset variable used?**

Basic Version: No unset variables are used explicitly.

Refined Version: No unset variables are used explicitly.

1. **Subscripts within bounds?**

Both versions ensure that the loop bounds for calculating the factorial are within the bounds of the input integer n.

1. **Non-integer subscripts?**

Basic Version: Accepts non-integer inputs but produces incorrect results.

Refined Version: Checks if the input is an integer and returns an error if not.

1. **Dangling references?**

Dangling references are not applicable in the context of the factorial calculation algorithms.

1. **Correct attributes when aliasing?**

Attribute aliasing is not applicable in the context of the factorial calculation algorithms.

1. **Record and structure attributes match?**

Record and structure attributes are not applicable in the context of the factorial calculation algorithms.

1. **Computing addresses of bit strings?**

Computing addresses of bit strings is not applicable in the context of the factorial calculation algorithms.

1. **Passing bit-string arguments?**

Passing bit-string arguments is not applicable in the context of the factorial calculation algorithms.

1. **Based storage attributes correct?**

Based storage attributes are not applicable in the context of the factorial calculation algorithms.

1. **Structure definitions match across procedures?**

Structure definitions are not applicable in the context of the factorial calculation algorithms.

1. **Off-by-one errors in indexing or subscripting operations?**

The refined version addresses off-by-one errors by properly handling edge cases, such as negative inputs.

1. **Are inheritance requirements met?**

Inheritance requirements are not applicable in the context of the factorial calculation algorithms.