

Massachusetts Institute of Technology
6.170 Laboratory in Software Engineering

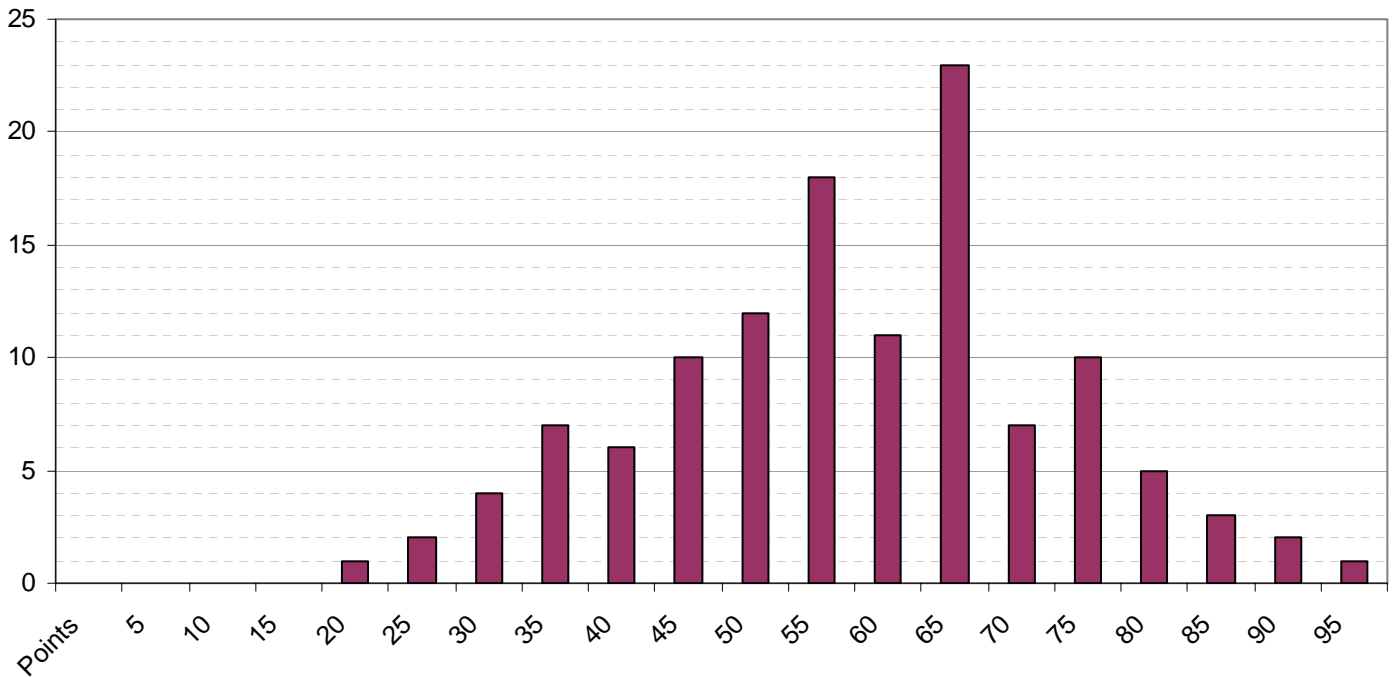
Spring 2006

Quiz 1

Friday, March 3, 2006

Number of students	123
Mean	60.53
Median	61
Mode	68
Standard Deviation	14.99

Histogram



Massachusetts Institute of Technology
6.170 Laboratory in Software Engineering

Spring 2006

Quiz 1

Friday, March 3, 2006

Name: Ben Bitdiddle

Athena username: bit

Section (circle one):

- 1: Matthew Tschantz (10am) 2: Vincent Yeung (11am) 3: C. Scott Ananian (12pm)
4: Philip Guo (1pm) 5: Tucker Sylvestro (3pm) 6: Natan Cliffer (3pm)

This quiz is closed book, closed notes. You have 50 minutes to complete it. It contains 26 questions in 17 pages (including this one), totaling 100 points. The last two pages contain duplicate material from multi-page questions. You should tear-off those pages for your reference. Before you start, please check your copy to make sure it is complete. Turn in pages 1-17, together, when you are finished.

Write your initials and section number on the top of ALL pages.

Please write neatly; we cannot give credit for what we cannot read.

Good luck!

Question(s)	Grading Scheme										Total	Max				
1 to 3											3	X 2 =	6	of 6		
4 to 20	4, 5	6	7, 8	9, 10	11, 12	13.. 15	16, 17	18, 19	20			X 4 =	68	of 68		
	2	1	2	2	2	3	2	2	1							
21													4	of 4		
22													4	of 4		
23													3	of 3		
24													4	of 4		
25												8 - 2 X	0	=	8	of 8
26													3	of 3		
											100		of 100			

True/False Questions

Question 1.

Consider the following code:

```
class A {  
    Number foo(Number bar);  
}  
class B {  
    Integer foo(Number bar);  
}
```

T / F B is a Java subtype of A.

This question asks about Java subtypes; thus, B would need to extend A.

Question 2.

T / F Methods automatically generated are called Factory Methods

Factory Methods are methods that create objects and are used to abstract over constructors for classes (see Effective Java). Automatically generated methods are not a topic of this course.

Question 3.

T / F S is a true subtype of T iff an object of T can be substituted for an object of S.

The reverse is true.

Multiple Choice Questions (circle all that apply, no partial grade):

The following code is used in Problems 4 and 5.
The code is also repeated at end of this exam.

Suppose we create the following classes `Account` and `OverdraftAccount` to represent bank accounts. As was the case in lecture, an `OverdraftAccount` is a subtype of a general `Account`.

```
/** Account represents a mutable bank account. */
class Account {
    double balance;

    /** @effects constructs a new Account with
     *     balance = initialBalance */
    Account(double initialBalance) {
        this.balance = initialBalance;
    }

    /** Returns a textual description of the type of this account.
     *     @return the String "Account" */
    String getAccountType() {
        return "Account";
    }

    /** @effects prints to standard output the type of this account
     *     followed by the balance. */
    void printBalance() {
        System.out.println(this.getAccountType() + ": $" + balance);
    }
}

/** OverdraftAccount represents a mutable bank account with a credit limit.*/
class OverdraftAccount extends Account {
    double creditLimit;

    /** @effects constructs a new OverdraftAccount with
     *     balance = initialBalance,
     *     creditLimit = initialCreditLimit */
    OverdraftAccount(double initialBalance, double initialCreditLimit) {
        super(initialBalance);
        this.creditLimit = initialCreditLimit;
    }

    /** Returns a textual description of the type of this account.
     *     @return the String "OverdraftAccount" */
    String getAccountType() {
        return "OverdraftAccount";
    }
}
```

Question 4. (circle all that apply, no partial grade)

The method `getAccountType()` is invoked in the code for `printBalance()`. What is the compile-time type of the receiver of this method invocation?

- (A) `Object`
- (B) `Account`
- (C) `OverdraftAccount`
- (D) The answer varies depending on how `printBalance()` is invoked.
- (E) None of the above

The receiver of the method invocation is implicitly "this", which, in the context of the `Account` class in which the `printBalance()` resides, has a compile-time type of `Account`. Note that an object has only one compile-time type; thus, circling both `Object` and `Account` is incorrect, even though `Account` is a subtype of `Object`.

Question 5. (circle all that apply, no partial grade)

Suppose we now wish to print the balance information for an account with the following code:

```
Account myAccount = new OverdraftAccount(5, 10);  
myAccount.printBalance();
```

Which of the following statements are true?

- (A) The printed output is the string "Account"
- (B) The printed output is the string "OverdraftAccount"
- (C) The printed output is the string "Account: \$5"
- (D) The printed output is the string "OverdraftAccount: \$5"
- (E) There will be a compilation error because `OverdraftAccount`, the run-time type of `myAccount`, does not define the `printBalance()` method.

`OverdraftAccount.getAccountType()` is called from `printBalance()` as `myAccount` has a runtime type of `OverdraftAccount`.

Question 6. (circle all that apply, no partial grade)

Consider the following code:

```
/** An immutable class with a person's physical shape */
class PhysicalShape {
    int weight, height;

    /** @effects constructs an object with the given Weight and height */
    PhysicalShape(int weight, int height) {
        this.weight = weight;
        this.height = height;
    }

    /** @returns the weight of the person */
    int getWeight() { return weight; }
}

/** A mutable class with my physical shape */
class MyPhysicalShape extends PhysicalShape {

    /** @effects constructs an object with the given Weight and height */
    MyPhysicalShape(int weight, int height) {
        super(weight, height);
    }

    /** @modifies this
     * @effects the weight is increased by 10 */
    void vacation( ) {
        weight += 10;
    }

    /** @modifies this
     * @effects the weight is reduced by 1 */
    void diet( ) {
        weight -= 1;
    }
}
```

which of the following statements are commonly true?

- (A) PhysicalShape is a Java subtype of MyPhysicalShape
- (B) PhysicalShape is a true subtype of MyPhysicalShape
- (C) MyPhysicalShape is a Java subtype of PhysicalShape
- (D) MyPhysicalShape is a true subtype of PhysicalShape
- (E) MyPhysicalShape and PhysicalShape does not have any subtype relationship

MyPhysicalShape is not a true subtype of PhysicalShape, because MyPhysicalShape is mutable, and PhysicalShape specifies that it is immutable.

Question 7. (circle all that apply, no partial grade)

which of these statements about black-box tests are commonly true?

- (A) Black-box tests generally aim to provide as much coverage of the lines of code in a module as possible, because coverage is usually positively-correlated with test quality.
- (B) Black-box tests can be used to find aliasing errors that occur when two different formal parameters of a method both refer to the same object.
- (C) Black-box tests can be effective at detecting representation exposure.
- (D) In practice, it is often difficult to re-use the same black-box test when crucial parts of the implementation change, even when the specification remains the same.
- (E) Black-box tests are able to partition the input space into finer (smaller) partitions than glass-box tests.

Black-box tests can find aliasing errors, especially in the inputs to the black-box region of code. They can even find aliasing errors elsewhere, but they are not guaranteed to find all such errors. Likewise, good black-box tests can detect representation exposure.

Question 8. (circle all that apply, no partial grade)

Which of these statements about testing are commonly true?

- (A) For a particular 'for' loop in the program, a black-box test should strive to test all possible numbers of iterations.
- ~~(B) For a particular 'for' loop in the program, a glass-box test should strive to test all possible numbers of iterations.~~
- (C) A complete set of Black-box tests makes Glass-box tests unnecessary.
- (D) A test is called a Black-box test when the test is written without look at the source code and a test is called a Glass-box test when it is written without looking at the specification.
- (E) For the people who did not write the implementation, Black-box tests are easier to write than glass-box tests.

Question 9. (circle all that apply, no partial grade)

Which of the following statements are true about specifications?

- (A) They allow implementations to be changed as long as the new version still fulfills the original specification.
- (B) They should only be used by a client when the source code is not available.
- (C) They should document the implementation details of methods and classes.
- (D) They should map every legal input to exactly one output.
- (E) Since specifications are easier to write than code, it is a good idea to keep the code simple even if it makes the interface more complicated.

A: True. This is the goal of decoupling specifications from implementations.

B: False. The client should write code which obeys the specifications, even when the code is available. It is poor practice to depend on the functionality or behavior of a particular implementation that's not promised by the spec.

C: False. The specification should document the abstract datatype, its domain, range, and possibly its role in the system. Specifications are not implementation docs.

D: False. Certain inputs can lead to multiple or non-deterministic output and still follow the specification. Ex: random number generators.

E: False. Complicated, or at least overly complicated, interfaces yield code that is difficult for someone to use correctly. In addition, writing a specification to make it easy to write one particular implementation will typically make it difficult to write other implementations.

Question 10. (circle all that apply, no partial grade)

In his guest lecture, Gilad Bracha described the important considerations followed by the maintainers of Java standard to be

- (A) Incorporating multiple inheritance in to the language
- (B) Following the philosophy of “Do no harm”
- (C) Making sure all the new features in C# get included
- (D) Separating the platform from the language
- (E) Make it extensible with Aspect Oriented Programming (AOP)

(circle all that apply, no partial grade)

Which of the choices below make this sentence true?

Strengthening a specification _____

- (A) provides more guarantees for clients.
- (B) can be achieved by making the @requires clause easier for clients to satisfy.
- (C) can be achieved by making the @requires clause harder for clients to satisfy.
- (D) may expand the number of <input, output> pairs in the transition relation denoted by the specification.
- (E) can be achieved by putting more in the @modifies clause

Question 11. (circle all that apply, no partial grade)

Let S be a specification written by a customer, and let P be a program written by a contractor.

Which of these statements is commonly true?

- (A) If P does not satisfy S, then P is incorrect with respect to the customer's desires.
- (B) If P satisfies S, then P is a correct program.
- (C) If the customer is satisfied with S, then P is a correct program.
- (D) Suppose S' is a weaker specification than S. If P satisfies S, then P might not satisfy S'.
- (E) It is often useful to provide a customer with P before S because the customer can tune his own components around the details of P before being locked down to a particular specification S.

Question 12. (circle all that apply, no partial grade)

The _____ forms the input to the Abstraction Function:

- (A) Abstract Value
- (B) Concrete Representation
- (C) Specification
- (D) Abstract Data Type
- (E) Test data

Question 13. (circle all that apply, no partial grade)

The _____ forms the output of the Abstraction Function.

- (A) Abstract Value
- (B) Concrete Representation
- (C) Specification
- (D) Abstract Data Type
- (E) Test results

Question 14. (circle all that apply, no partial grade)

The _____ defines the _____ of the Abstraction Function:

- (A) Specification ... range
- (B) Specification ... domain
- (C) Representation Invariant ... range
- (D) Representation Invariant ... domain
- (E) None of the above

A is true because the specification states what abstract values a class may represent. The Abstraction Function must map (well-formed) concrete representations to one of these abstract values. D is true because a concrete representation must pass the Representation Invariant to be well-formed and the Abstraction Function only takes well-formed concrete representations as input.

The following code is used in Problems 16 and 17.

Given the code below:

```
class A {
    public void print() { System.out.println("hi"); }
}

class B extends A {
    public void print() { System.out.println("bye"); }
}
```

What does each of the following code samples do:

Question 15. (circle all that apply, no partial grade)

```
A a = new B();
a.print();
```

- (A) Prints nothing
- (B) Prints "hi"
- (C) Prints "bye"
- (D) The code does not compile, the compiler returns a type error.
- (E) The code throws a runtime exception

The print() method of the run-time type, B, is called.

Question 16. (circle all that apply, no partial grade)

```
B b = new A();
b.print();
```

- (A) Prints nothing
- (B) Prints "hi"
- (C) Prints "bye"
- (D) The code does not compile, the compiler returns a type error.
- (E) The code throws a runtime exception

The compiler will not allow you to set an object reference to an instance whose runtime type is a Java supertype of the variable's compile time type.

Question 17. (no partial grade)

Ben Bitdiddle writes the following implementation for the given specification:

```
/** Returns the (integer) square root of the given number n.
 * @requires there exists some integer x such that x*x == n
 * @returns the integer x, if n>=0
 * @throws ArithmeticException if n<0 */
int sqrt(int n) {
    if (n<0) throw new ArithmeticException();
    if (n < 100) return cached_sqrt[n];
    for (int i=10; i<=n; i++)
        if (i*i == n) return i;
    return -1;
}
```

Alyssa P. Hacker is assigned to write tests for this method and wrote the following tests.

```
assertEquals(10, sqrt(100))
```

This test is most likely created as

- (A) A Blackbox test
- (B) A Glassbox test
- (C) A performance test
- (D) Neither a Blackbox nor a Glassbox nor a performance test

*The test is *most* likely a glass box test, because it tests the output for the border of a subdomain that is specifically handled differently in the code, but not in the specification.*

Question 18. (circle all that apply, no partial grade)

Which of the following sentence(s) on exceptions are commonly true?

- (A) Use an exception when the cost of checking is prohibitive
- ~~(B) Failure to catch exceptions or declare them as thrown violates modularity~~
- (C) It is good practice to use unchecked exceptions to mean failure
- (D) The class RuntimeException and its subtypes represent unchecked exceptions
- (E) If there is no enclosing try block within the procedure, the exception always terminates the program.

A. Don't use an exception when the cost of checking is prohibitive.

To throw an exception, you have to do the checking.

E. There could be an enclosing try block within a procedure that called the procedure, etc.

Question 19. (circle all that apply, no partial grade)

Given the following (partial) Daikon output:

```
ps2.GeoFeature.addSegment(ps2.GeoSegment):::EXIT
this.length >= 0.999997425
gs.heading >= 0.0
size(this.segments[]) one of { 1, 2 }
this.length < ps2.GeoPoint.MILES_PER_DEGREE_LATITUDE
this.length < ps2.GeoPoint.MILES_PER_DEGREE_LONGITUDE
ps2.GeoPoint.MIN_LATITUDE < gs.p1.latitude
ps2.GeoPoint.MIN_LATITUDE < gs.p1.longitude
```

Which of the following statements can be made?

- (A) In every possible execution of the program, `this.length` must be non-zero.
- (B) In the test cases Daikon saw, GeoFeatures were never more than 2 segments long.
- (C) Daikon's output indicates a bug: we should be able to add points where `gs.p1.latitude == GeoPoint.MIN_LATITUDE`
- (D) Daikon's output indicates that the test suite tested a sufficient number of different values of `this.length`
- (E) None of these statements are valid

Short Answer Questions

The following code is used in Problems 21, 22 and 23.

```
1  /* An immutable class for representing a point in n-dimensional space.
2   * @specfield dimension the number of dimensions of the space the point is in
3   * @specfield position the position in n-dimensional space of the point */
4  class PointND {
5      private List<Double> components;
6
7      /** Creates a new PointND with dimension = the length of components
8       * and having the position corresponding to the ordered list:
9       * (components0, components1, ..., components(dimension-1))
10     @Requires components != null */
11     public PointND(List<Double> components) {
12         this.components = components;
13     }
14
15     /* @returns the dimension of this */
16     public int getDimension() {
17         return components.size();
18     }
19
20     //... more methods are omitted for the sake of brevity
21 }
```

The bug is that the constructor exposes the rep. The client could later alter the components list directly and cause PointND's rep to change behind its back.

Either 11 or 12 was accepted as an answer, so long as the student made it clear (either with a note or in their answer to Q22) that they understood that the bug was rep exposure.

Question 20.

There is a bug in this program. Identify the line number of the bug.

12

Question 21.

Write a substitution for that line that removes the bug (the substitution may take multiple lines)

Code to make a duplicate
of components

Question 22.

Change the specification of the buggy method so that the original implementation is correct. Fill in the boxes for only the changed requirements.

@requires

component is immutable

@modifies

@throws

@effects

@returns

The correct answer is a weaker specification that makes the original implementation valid.. Since the main bug in the original code is rep exposure, weakening the specification by requiring the component to be immutable (note that this is not the same as final) is the simplest modification to the specification.

Question 23.

Consider the following Stack implementation.

```

/** Implementation of a first-in, last-out stack. */
// Abstraction function:
// AF(R) = sequence of elements theArray[0]..theArray[topOfStack+1],
//       where theArray[0] is at the bottom of the stack and
//       theArray[topOfStack+1] is at the top of the stack
// Rep invariant:
//   theArray != null
//   topOfStack >= -1
//   topOfStack <= size(theArray[])-1
public class Stack {
    private T[] theArray;
    private int topOfStack;
    ... many operations omitted ...
    /** ... you will fill this in... */
    public void checkRep() {
        // assert throws AssertionError if the condition does not hold
        assert topOfStack >= -1;
        // theArray.length throws NullPointerException if theArray == null
        assert topOfStack <= theArray.length-1;
    }
}

```

Write a specification for the `checkRep` procedure using ALL five 6.170 specification tags requires, modifies, throws, effects and returns. (Do not leave any boxes blank)

@requires

none

@modifies

none

@throws

none

@effects

none

@returns

none

Remember that a method specification is for a client of a class to read; thus, it must not reference private fields. Therefore, any answer that says things like:

*@requires theArray != null
or
@effects throws a Exception
if theArray == null.*

are incorrect.

One should not include a throws clause because one can not reasonably describe when the exceptions would be throw without referring to private fields.

**The following information is used in Problems 25 and 26.
The specifications are also repeated at the end of the exam.**

Ben Bitdiddle joins the Megasensible Corporation's famous Flunk spreadsheet group. Ben's first assignment is to refactor the Flunk source base to eliminate duplicate code that had accumulated over the years. Ben finds 4 implementations for the function that copies the first n elements from List `src` to List `dest`.

```
void partialcopy(int n, List<Integer> dest, List<Integer> src)
```

Fortunately for Ben, all the implementations have specifications written in 6.170 style.

Specification A

@requires: $n > 0$

@modifies: `dest`

@throws: `ArrayOutOfBoundsException` if `src.length < n`

@effects: for $i=1..n$ `dest[i]post = src[i]pre`

@returns: nothing

Specification B

@requires: $n > 0$

@modifies: `src`, `dest`

@throws: `ArrayOutOfBoundsException` if `src.length < n`

@effects: for $i=1..n$ `dest[i]post = src[i]pre`

@returns: nothing

Specification C

@requires: $n > 0$

@modifies: `dest`

@throws: nothing

@effects: for $i=1..min(n, src.length)$ `dest[i]post = src[i]pre` and for $i=src.length+1..n$ `dest[i]post = 0`

@returns: nothing

Specification D

@requires: $n > 0$ and `src.length >= n`

@modifies: `dest`

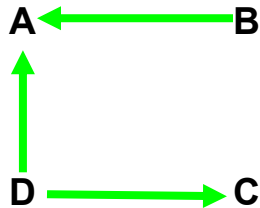
@throws: nothing

@effects: for $i=1..n$ `dest[i]post = src[i]pre`

@returns: nothing

Question 24.

In the following diagram, for all pairs of specifications, draw an arrow from X to Y ($X \rightarrow Y$) if and only if Y is stronger than X.



Common incorrect edges:

$D \rightarrow B$: B and D are incomparable because B modifies more but requires less.

$A \rightarrow C$: A and C are incomparable because their postconditions are incomparable: when $n > \text{src.length}$, A guarantees to throw an exception, while C guarantees to set extra elements to 0. A client expecting one of the two specifications will not be satisfied by an implementation of the other.

Question 25.

According to the specifications, which method(s) should Ben choose to replace all the others?

A,C

The correct answer is A and C. These are the only types of the four for which there is no stronger type. The answer "A or C" is incorrect because you cannot replace A with C or vice-versa, as they are incomparable specifications

End of Quiz!

Make sure that you have written your initials
and section number on the top of *ALL* pages.

Turn in pages 1 to 17.

Tear-off this page for Reference

The following code is used in Problems 4 and 5.

```
/** Account represents a mutable bank account. */
class Account {
    double balance;

    /** @effects constructs a new Account with
     *     balance = initialBalance */
    Account(double initialBalance) {
        this.balance = initialBalance;
    }

    /** Returns a textual description of the type of this account.
     *     @return the String "Account" */
    String getAccountType() {
        return "Account";
    }

    /** @effects prints to standard output the type of this account
     *     followed by the balance. */
    void printBalance() {
        System.out.println(this.getAccountType() + ": $" + balance);
    }
}

/** OverdraftAccount represents a mutable bank account with a credit limit.*/
class OverdraftAccount extends Account {
    double creditLimit;

    /** @effects constructs a new OverdraftAccount with
     *     balance = initialBalance,
     *     creditLimit = initialCreditLimit */
    OverdraftAccount(double initialBalance, double initialCreditLimit) {
        super(initialBalance);
        this.creditLimit = initialCreditLimit;
    }

    /** Returns a textual description of the type of this account.
     *     @return the String "OverdraftAccount" */
    String getAccountType() {
        return "OverdraftAccount";
    }
}
```

Tear-off this page for Reference

The following specifications are used in Problems 25 and 26.

Specification A

@requires: $n > 0$

@modifies: dest

@throws: ArrayOutOfBoundsException if $\text{src.length} < n$

@effects: for $i=1..n$ $\text{dest}[i]_{\text{post}} = \text{src}[i]_{\text{pre}}$

@returns: nothing

Specification B

@requires: $n > 0$

@modifies: src, dest

@throws: ArrayOutOfBoundsException if $\text{src.length} < n$

@effects: for $i=1..n$ $\text{dest}[i]_{\text{post}} = \text{src}[i]_{\text{pre}}$

@returns: nothing

Specification C

@requires: $n > 0$

@modifies: dest

@throws: nothing

@effects: for $i=1..\min(n, \text{src.length})$ $\text{dest}[i]_{\text{post}} = \text{src}[i]_{\text{pre}}$ and for $i=\text{src.length}+1..n$ $\text{dest}[i]_{\text{post}} = 0$

@returns: nothing

Specification D

@requires: $n > 0$ and $\text{src.length} \geq n$

@modifies: dest

@throws: nothing

@effects: for $i=1..n$ $\text{dest}[i]_{\text{post}} = \text{src}[i]_{\text{pre}}$

@returns: nothing