Assessing the Understanding of Expressions: A Qualitative Study of Notional-Machine-Based Exam Questions

Supplementary Material: Exam Questions

Contents

1 E01 2

2 E02 2

3 E03 3

4 E04 3

5 E05 4

6 E06 4

7 E07 5

8 E08 6

9 E09 7

10 E10 & E11 8

11 E12 8

1 E01

Assume the following variables:

```
int i;
double d;
String s;
```

Write an expression tree for the following expression. Then annotate each node of the tree with the type of the value it produces.

```
(d > i/2) ? ("Hi" + '!').length() : new int[s.length()].length + i
```

Note: If you are not entirely sure how to draw the tree, come up with what you think is the best way, explain your way (write a brief note), and then draw it your way.

2 E02

Gegeben sei der folgende Java Code:

```
public class A {
  public String m(int[] ar, int ix) {
    return "ar[ix]=" + ((null!=ar && ix<ar.length) ? (ar[ix] + "") : "no");
  }
}
```

(a) Zeichnen Sie den Ausdrucksbaum des Ausdrucks im obigen return-Statement. Markieren sie die Wurzel des Baums mit einem Sternchen.

(b) Geben Sie, über jedem Knoten, den Typ der vom Knoten produzierten Werte an. Für den Literal null können Sie als Type "NullType" hinschreiben.

(c) Geben Sie, neben jedem Knoten, den Wert an, der dieser Knoten produzieren wird, falls der Ausdruck ausgewertet wird wenn `ar = new int[] {2, 4, 6}; ix =2;` ist

Stellen Sie sicher, dass Sie über jedem Knoten dessen Typ angeben. Und stellen Sie sicher, dass Sie neben jedem Knoten, der ausgewertet wird, dessen Typ Wert angeben.

3 E03

Give the following methods:

```
public static boolean gt(int a, int b) {
  return a > b;
}

public static String compare(int a, int b) {
  return gt(a, b) ? "a > b" : (a < b ? "a < b" : "a == b"); // <-----
}
```

(a) **Draw an expression tree** for the expression in the return statement of the `compare` method.

(b) Annotate each node with a **type** (the type of value it produces).

(c) Assume that someone calls `compare(1, 2)`. Annotate each node with the **value** it produces in that situation. If a node does not get evaluated, do NOT annotate it with a value, but annotate it with a dash (–).

4 E04

Given this class:

```
public class Demo {

  private static Demo[] id(Demo[] arg) {
    return arg;
  }

  public String toString() {
    return "S";
  }

  public static String run() {
    int i = 0;
    Demo[] a = new Demo[] { new Demo() };
    String s = "a[i] = " + (a!=null ? id(a)[i].toString() + "0" : "X");
    return s;
  }
}
```

Draw the expression tree of the expression to the right of `String s =`.
Indicate which node is the **root** (with a star).
For each node indicate its **type**. As the type of the value `null`, write `NULL`.
For each node indicate its **value**. Use `@1`, `@2`, `@3`, ... to represent reference values (like we did in class), except for the null reference, which you represent as the `null` literal, and for strings, which you represent as a `String` literal.

5 E05

Gegeben sei der folgende Java Code:

```
public class A {
  public String m(boolean bedingung) {
    return "bedingung=" + (bedingung ? "true" : "false");
  }
}
```

(a) Zeichnen Sie den Ausdrucksbaum des Ausdrucks im obigen return-Statement. Markieren sie die Wurzel des Baums mit einem Sternchen.

(b) Geben Sie, über jedem Knoten, den Typ der vom Knoten produzierten Werte an.

(c) Geben Sie, neben jedem Knoten, den Wert an, der dieser Knoten produzieren wird, falls der Ausdruck ausgewertet wird wenn `bedingung = true;` ist

Stellen Sie sicher, dass Sie über jedem Knoten dessen Typ angeben. Und stellen Sie sicher, dass Sie neben jedem Knoten, der ausgewertet wird, dessen Wert angeben.

6 E06

Gegeben sei der folgende Java Code:

```
public class A {
  public String m(int[] a, int i) {
    return "a[i] = " + ((a!=null && a.length>i) ? ("" + a[i]) : "nothing");
  }
}
```

(a) Zeichnen Sie den Ausdrucksbaum des Ausdrucks im obigen return-Statement.

(b) Geben Sie, über jedem Knoten, den **Typ** der vom Knoten produzierten Werte an. Für den Literal `null` können Sie als Typ "NullType" hinschreiben.

(c) Geben Sie, neben jedem Knoten, den **Wert** an, der dieser Knoten produzieren wird, falls der Ausdruck ausgewertet wird wenn `a = new int[] {1, 2, 3}; i = 2;` ist.

Stellen Sie sicher, dass Sie über **jedem** Knoten dessen Typ angeben. Und stellen Sie sicher, dass Sie neben **jedem** Knoten, der ausgewertet wird, dessen Typ angeben.

7 E07

```
public class Runner {
  public static float go(long v) {
    return new Runner(v).a().b(c(d));
  }

  private long f;

  public Runner(long v) {
    this.f = v;
  }

  public Runner a() {
    f = -f;
    return this;
  }

  public float b(int param) {
    return f + param;
  }

  public static int c(double arg) {
    return (int)arg;
  }

  private static double d = 0;
}
```

Draw the expression tree, and **annotate each node with its type**, for the expression in the method `go()` above: `new Runner(v).a().b(c(d))`

8 E08

Given this class:

```
public class Demo {

  private static String id(String arg) {
    return arg;
  }

  public String toString() {
    return "D";
  }

  public static String run() {
    int i = 0;
    Demo[] a = new Demo[] { new Demo() };
    String s = "a[i] = " + (a==null ? "X" : id(a[i].toString())) + '+' + 0;
    return s;
  }
}
```

Draw the expression tree of the expression to the right of `Strubg s =`.

Indicate which node is the **root** (with a star).

For each node indicate its **type**. As the type of the value `null`, write `NULL`.

For each node indicate its **value**. Use `@1`, `@2`, `@3`, ... to represent reference values (like we did in class), except for the null reference, which you represent as the `null` literal, and for strings, which you represent as a `String` literal.

9 E09

```
public class Maker {

  public String publish(String a, String b) {
    System.out.println("pub");
    System.out.println(a);
    return "done";
  }

  public String make(String thing) {
    System.out.println("making");
    return "made " + thing;
  }

  public void run() {
    String s = publish(make("this"), make("that"));
    System.out.println("complete");
  }
}
```

(a) For the expression to the right of the equals sign (=) in the `run` method, draw the expression tree (nodes, edges, **and a star on the root node**). For each node, including the root, draw its type, and draw its value.

(b) What gets printed on the console after executing the `run` method?

10 E10 & E11

Given the following code:

```
public interface Seq {
  public int len();
}
public record Cons(String v, Seq rest) implements Seq {
  public int len() {
    return 1 + rest.len();
  }
}
public record Empty() implements Seq {
  public int len() {
    return 0;
  }
}
public class Demo {
  public static int run() {
    return new Cons("A", new Empty()).len();
  }
}
```

Draw the expression trees of the following expressions.
Annotate each node with its **type**!

(a) Expression in body of `Cons.len`

(b) Expression in body of `Demo.run`

11 E12

Given the following methods:

```
public static int mul(int a, int b) {
  return a * b;
}

public static int twice(int i) {
  return mul(i, 2);
}

public static String nameFor(int h) {
  return h > 0 ? "larger than zero" : "oops";
}
```

Assume that `height` is of type `int`. Draw the expression tree for:

```
height >= 0 ? nameFor(twice(height)) : "height < 0"
```

For each node in your tree, show the **type** above the node (but do *not* show the value above the node). Indicate the **root** with a star.