# Recursion Day 2 — Worked Solutions

*APCS MC Practice — Answer Key with Traces*

## Problem 6: check(String s)

```
public boolean check(String s) {
    return s.length() >= 2 &&
           (s.charAt(0) == s.charAt(1) || check(s.substring(1)));
}
```

This checks if the first two characters are equal, OR recursively checks the rest of the string (starting from index 1). It returns true if ANY two consecutive characters are the same.

**Answer: (B) s contains two or more of the same characters in a row**

→ check("abc") → 'a'=='b'? No → check("bc") → 'b'=='c'? No → check("c")
→ check("c"): length < 2, returns false
→ check("abbc") → 'a'=='b'? No → check("bbc") → 'b'=='b'? Yes! returns true

---

## Problem 9: mystery(38)

```
public void mystery(int n) {
    if (n > 2)
        mystery(n % 3);
    System.out.print((n / 3) + " ");
}
```

Recurses with n % 3 while n > 2, then prints n / 3 on the way back.

→ mystery(38): 38 > 2, call mystery(38 % 3 = 2)
→ mystery(2): 2 > 2? No. Print 2/3 = 0. Done.
→ Back to mystery(38): print 38/3 = 12

**Answer: (A) 0 12**

---

## Problem 10: processLine("ABCD", 0)

```
public void processLine(String str, int pos) {
    if (pos < str.length()) {
        for (int i=0; i<=pos; i++)
            System.out.print(str.substring(pos, pos+1));
        processLine(str, pos + 1);
        for (int i=0; i<=pos; i++)
            System.out.print(str.substring(pos, pos+1));
    }
}
```

At each position: print char (pos+1) times, recurse, print char (pos+1) times again.

**Answer:** **(C) ABBCCCDDDDDDDCCCBBA**

```
→ pos=0: print A (1 time) → A
→   pos=1: print B (2 times) → BB
→     pos=2: print C (3 times) → CCC
→       pos=3: print D (4 times) → DDDD
→         pos=4: base case (4 < 4 is false)
→       pos=3: print D (4 times) → DDDD
→     pos=2: print C (3 times) → CCC
→   pos=1: print B (2 times) → BB
→ pos=0: print A (1 time) → A
```

Full output: A BB CCC DDDD DDDD CCC BB A = ABBCCCDDDDDDDCCCBBA

---

## Problem 15: sum(arr, 3) where arr = {2, 4, 6}

```
int sum(int arr[], int n) {
    if (n == 0) return 0;
    else {
        int smallResult = sum(arr, n - 1);
        return smallResult + arr[n - 1];
    }
}
```

This sums the first n elements of the array.

**Answer:** **12**

```
→ sum(arr, 3) = sum(arr, 2) + arr[2] = sum(arr, 2) + 6
→ sum(arr, 2) = sum(arr, 1) + arr[1] = sum(arr, 1) + 4
→ sum(arr, 1) = sum(arr, 0) + arr[0] = 0 + 2 = 2
→ Unwind: 2, then 2+4=6, then 6+6=12
```

---

## Problem 17: mystery6

```
public void mystery6(int x, int y) {
    if (y == 1) {
        System.out.print(x);
    } else {
        System.out.print(x * y + ", ");
        mystery6(x, y - 1);
        System.out.print(", " + x * y);
    }
}
```

Prints x*y before recursing, x*y after recursing. Mirrors around the base case.

**mystery6(4, 1) =** **4**

```
→ y == 1, print 4
```

**mystery6(8, 2) =** **16, 8, 16**

```
→ y=2: print 8*2=16, call mystery6(8,1), print 16
→ y=1: print 8
```

→ Output: 16, 8, 16

**mystery6(3, 4) = `12, 9, 6, 3, 6, 9, 12`**

→ y=4: print 12, recurse, print 12
→ y=3: print 9, recurse, print 9
→ y=2: print 6, recurse, print 6
→ y=1: print 3
→ Output: 12, 9, 6, 3, 6, 9, 12

---

## Problem 20: mystery9

```
public void mystery9(int x) {
    if (x < 10) {
        System.out.print(x);
    } else {
        int y = x % 10;
        System.out.print(y);
        mystery9(x / 10);
        System.out.print(y);
    }
}
```

Prints last digit, recurses on remaining digits, prints last digit again. Creates a palindrome.

**mystery9(7) = `7`**

→ 7 < 10, print 7

**mystery9(38) = `838`**

→ x=38: y=8, print 8, call mystery9(3), print 8
→ x=3: print 3
→ Output: 838

**mystery9(194) = `49194`**

→ x=194: y=4, print 4, call mystery9(19), print 4
→ x=19: y=9, print 9, call mystery9(1), print 9
→ x=1: print 1
→ Output: 4 9 1 9 4 = 49194

---

## Problem 21: splat("*")

```
public static void splat(String s) {
    if (s.length() < 8)
        splat(s + s);
    System.out.println(s);
}
```

Doubles the string until length >= 8, then prints on the way back up.

**Answer: `(E)`**

→ splat("*"): len=1 < 8, call splat("**")
→ splat("**"): len=2 < 8, call splat("****")
→ splat("****"): len=4 < 8, call splat("********")

```
→ splat("********"): len=8, not < 8, print "********"
→ Return: print "****"
→ Return: print "**"
→ Return: print "*"
```

Output: ********, ****, **, * (each on its own line)

---

## Problem 22: cheer(0)

```
public void cheer(int i) {
    if (i != 8) {
        i = i + 2;
        cheer(i);
        System.out.print(i + " ");
    } else {
        System.out.print("who do we appreciate!");
    }
}
```

Current behavior: increments i, recurses, then prints. Prints happen on the way BACK, so output is reversed: "who do we appreciate! 8 6 4 2"

Desired: "2 4 6 8 who do we appreciate!"

Fix: swap the recursive call and print so we print BEFORE recursing.

**Answer: (D) swap line 4 and line 5**

---

## Problem 25: disarray(a, 7)

```
public void disarray(int[] a, int n) {
    if (n > 1) {
        disarray(a, n - 1);
        a[n - 1] += a[n - 2];
    }
}
```

Recurses to n=1 first, then on the way back, adds previous element to current.

Initial: {1, 3, 4, 7, 9, 11, 13}

**Answer: (A) {1, 4, 8, 15, 24, 35, 48}**

```
→ Recurse down to n=1 (base case, do nothing)
→ n=2: a[1] += a[0] → a[1] = 3+1 = 4. Array: {1,4,4,7,9,11,13}
→ n=3: a[2] += a[1] → a[2] = 4+4 = 8. Array: {1,4,8,7,9,11,13}
→ n=4: a[3] += a[2] → a[3] = 7+8 = 15. Array: {1,4,8,15,9,11,13}
→ n=5: a[4] += a[3] → a[4] = 9+15 = 24. Array: {1,4,8,15,24,11,13}
→ n=6: a[5] += a[4] → a[5] = 11+24 = 35. Array: {1,4,8,15,24,35,13}
→ n=7: a[6] += a[5] → a[6] = 13+35 = 48. Array: {1,4,8,15,24,35,48}
```

---

## Problem 27: tricky(7, 3)

```
public int tricky(int x, int y) {
    if (y == 2) return x;
    else return tricky(x, y-1) + x;
}
```

This computes x * (y - 1). Each recursive call adds x, starting from y=2.

**Answer: `14`**

→ tricky(7, 3) = tricky(7, 2) + 7
→ tricky(7, 2) = 7 (base case)
→ Result: 7 + 7 = 14

---

## Problem 28: mystery(7, 3)

```
public int mystery(int a, int b) {
    if (a < b) return 5;
    else return b + mystery(a-1, b+1);
}
```

**Answer: `17`**

→ mystery(7,3): 7 >= 3, return 3 + mystery(6,4)
→ mystery(6,4): 6 >= 4, return 4 + mystery(5,5)
→ mystery(5,5): 5 >= 5, return 5 + mystery(4,6)
→ mystery(4,6): 4 < 6, return 5
→ Unwind: 5 + 5 = 10, 4 + 10 = 14, 3 + 14 = 17

---

## Problem 29: printString("stressed")

```
public static void printString(String s) {
    if (s.length() > 0) {
        printString(s.substring(1));
        System.out.println(s.substring(0, 1));
    }
}
```

Recurses first, prints first character after. Prints in REVERSE order.

**Answer: `d e s s e r t s (each on new line)`**

→ Recurses down to empty string, then prints on way back
→ "stressed" reversed = "desserts"

Each character prints on its own line: d, e, s, s, e, r, t, s

---

## Problem 32: printArray(a, 0)

```
public static void printArray(String[] a, int k) {
    if (k < a.length) {
        printArray(a, k+1);
```

```
            System.out.print(a[k]);
        }
    }
```

Array is {"a", "b", "c", "d"}. Recurses first, prints after → reverse order.

**Answer: (E) dcba**

→ printArray(a,0) → printArray(a,1) → printArray(a,2) → printArray(a,3) → printArray(a,4)
→ k=4: 4 < 4 is false, return
→ k=3: print a[3] = "d"
→ k=2: print a[2] = "c"
→ k=1: print a[1] = "b"
→ k=0: print a[0] = "a"
→ Output: dcba

---

## Problem 39: getSomething(val)

```
public int getSomething(int value) {
    if (value < 2) return 0;
    else return 1 + getSomething(value - 2);
}
```

Subtracts 2 each time and counts. This is integer division by 2: val / 2.

**Answer: (D) val / 2**

→ getSomething(7) = 1 + gS(5) = 1 + 1 + gS(3) = 1 + 1 + 1 + gS(1) = 3 + 0 = 3
→ 7 / 2 = 3 ✓
→ getSomething(8) = 1+1+1+1+gS(0) = 4. 8/2 = 4 ✓

---

## Problem 41: change(29)

```
public void change(int value) {
    if (value < 5)
        System.out.print(value % 5);
    else {
        System.out.print(value % 5);
        change(value / 5);
    }
}
```

Prints value % 5 then recurses with value / 5. Converts to base 5, least significant digit first.

→ change(29): print 29%5=4, call change(29/5=5)
→ change(5): print 5%5=0, call change(5/5=1)
→ change(1): 1 < 5, print 1%5=1
→ Output: 4, 0, 1 = 401

**Answer: (E) 401**

---

## Problem 43: doSomething(4)

```
public void doSomething(int value) {
    if (0 < value && value < 10) {
        doSomething(value - 1);
        doSomething(value + 1);
        System.out.print(" " + value);
    }
}
```

This branches: first recurses with value-1, then value+1, then prints.

Key insight: The branches create mutual recursion that never terminates.

**Answer: (E) Nothing will be printed due to infinite recursion**

```
→ doSomething(4) calls doSomething(3)
→ doSomething(3) calls doSomething(2)
→ doSomething(2) calls doSomething(1)
→ doSomething(1) calls doSomething(0) — stops (0 < value is false)
→ doSomething(1) then calls doSomething(2) — but we're back to 2!
→ doSomething(2) calls doSomething(1) again...
```

The calls doSomething(1) ↔ doSomething(2) bounce back and forth forever.

---

## Problem 45: getIt(0)

```
public int getIt(int index) {
    if (index == list.length - 1)
        return list[index];
    else {
        int target = getIt(index + 1);
        if (target < list[index])
            return target;
        else
            return list[index];
    }
}
```

Recursively gets result from rest of array, then returns the smaller of (that result) or (current element). Finds minimum.

**Answer: (A) The smallest value in list**

---

## Problem 51: fun(3, 6)

```
public int fun(int x, int y) {
    if (y == 2) return x;
    else return fun(x, y-1) + x;
}
```

Same as Q27. Computes x * (y - 1).

**Answer:** **(D) 15**

→ fun(3,6) = fun(3,5) + 3 = fun(3,4) + 3 + 3 = fun(3,3) + 9 = fun(3,2) + 12
→ fun(3,2) = 3
→ Total: 3 + 12 = 15

Or: 3 × (6-1) = 3 × 5 = 15

---

## Problem 55: minVal completion

```
public static int minVal(int[] a, int n) {
    if (n == 1)
        return <missing code 1>;
    int min = minVal(a, n-1);
    if (min < a[n-1])
        return <missing code 2>;
    else
        return <missing code 3>;
}
```

When n=1, return the first element: a[0]

If the recursive min is smaller than a[n-1], return min

Otherwise return a[n-1] (the current element is smaller)

**Answer:** **(E) a[0], min, a[n-1]**

→ missing code 1: a[0] (base case returns first element)
→ missing code 2: min (recursive result was smaller)
→ missing code 3: a[n-1] (current element is smaller or equal)