# Unit 2 – ICS4U0 – Methods and Recursion

Sample Test – March 4, 2024

Name: Solutions

| Total | % | Knowledge | Communication | Thinking | Application |
|---|---|---|---|---|---|
| | | | | | |
| (98) | % | (25) | (23) | (24) | (26) |

## Knowledge

1. Trace the code, using the memory diagram/tracing chart provided. /7

***Riddle:* What has a neck but no head?**

*Answer key:* 0=i, 1=e, 2=t, 3=n, 4=u, 5=l, 6=o, 7=s, 8=r, 9=b, 10=h, 11=a, 12=m

```
int s = 2;
int a [] = {8, 6, 4, 2, 1, 0};
a [5] = 1;
a [4] = 8;
a [s] --;
a [2] = a [3];
a [4] = s + 3;
a [0] ++;
```

| [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|
| 8 | 6 | 4 | 2 | 1 | 0 |
| 8 | 6 | 4 | 2 | 1 | 1 |
| 8 | 6 | 4 | 2 | 8 | 1 |
| 8 | 6 | 3 | 2 | 8 | 1 |
| 8 | 6 | 2 | 2 | 8 | 1 |
| 8 | 6 | 2 | 2 | 5 | 1 |
| 9 | 6 | 2 | 2 | 5 | 1 |

Riddle Answer: ....bottle.....

2. What does ORATE stand for? /5

| | | | | |
|---|---|---|---|---|
| Organization | Reusability | Abstraction | Testing | Extensibility |

3. Using the method signatures, call the methods. /8

(a) Signature: `public void maple (double height)`

Call:

```
double height = IO.inputDouble("How tall? ");
maple(height);
```

(b) Signature: `public double danger(int alligator)`

Call:

```
int alligator = IO.inputInt("How many alligators? ");
double ans = danger(alligator);
```

(c) Signature: `public void okapi()`

Call:

```
okapi();
```

(d) Signature: `public int turkeySize(String food)`

Call:

```
String food = IO.inputString("What type of food? ");
int answer = turkeySize(food);
```

4. Enter the return type and the parameter type of the following String methods. /5

| | | Return Type | Parameter Type |
|---|---|---|---|
| (a) | charAt | char | int |
| (b) | substring | String | int |
| (c) | length | int | / |
| (d) | indexOf | int | char |
| (e) | equals | boolean | String |

## Communication

5. Consider this recursive method, identify the following: /9

```
public int mandlebrot (int n) {
  if (n <= 2)
    return 10;
  else
    return mandlebrot (n - 2) + 20;
}
```

Base Case

Recursive Case

| n | |
|---|---|
| 0 | 10 |
| 1 | 10 |
| 2 | 10 |
| 3 | 30 |
| 4 | 30 |
| 5 | 50 |

| | | |
|---|---|---|
| (a) | The return type | int |
| (b) | The method name | mandlebrot |
| (c) | The parameter name | n |
| (d) | The parameter type | int |
| (e) | Circle **and** label the base case and recursive case | ☑ |

Evaluate the following:

(f) `mandlebrot(1).` 10

(g) `mandlebrot(4).` 30

(h) `mandlebrot(0).` 10

6. Provide the phrase or number required. /10

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 5 | 8 | 13 |

| | | |
|---|---|---|
| 8 | (a) | The 6th Fibonacci number. (The 1st is 1 and the 2nd is also 1) |
| Sea shell coil proportions | (b) | Where does the Fibonacci sequence appear in nature? |
| 24 | (c) | The answer to 4! (four factorial). |
| probability calcs | (d) | What are factorials used for in math? |
| testing | (e) | White and black box verification have fewer cases. (ORATE) |
| extensibility | (f) | By putting code in one place, changes are easier. (ORATE) |
| abstraction | (g) | To use methods, understanding is not needed. (ORATE) |
| parameter | (h) | The name of the input to a method. |
| stackoverflow error | (i) | The equivalent of an infinite loop in recursion. |
| base case | (j) | The equivalent of a loop stopping condition in recursion. |

7. Which of the ORATE principles applied best to the **Tic Tac Toe** program? (2 points, one sentence each) /2

Reusability refers to the fact that instead of cutting and pasting, we make a method and call it, thus reusing the code more easily. [In TicTacToe, the grid has 9 buttons which can be coded in one method and be reused for all 9. 1|2|3 / 4|5|6 / 7|8|9] Note that this section is very specific to TicTacToe.

8. Why are if statements important to recursion? (2 points, one sentence each) /2

Key words.

If statements are used to make decisions. In recursion, they allow use to choose between the base case (stopping) and the recursive case (repeating). Both are needed to make recursion work and the code must use ifs to choose which to run and when.

## Thinking

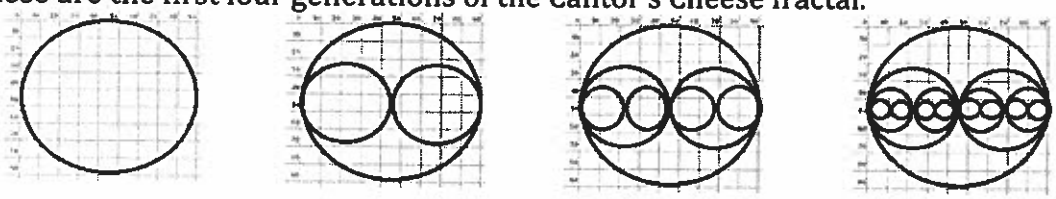
9. Fill in the chart to trace the following method call `System.out.println(avatar(5));` /5

```
public int avatar (int n) {
    if (n < 2)
        return 5;
    else
        return n + avatar (n - 2);
}
```

work isn't required but might get part marks

| avatar(0) | avatar(1) | avatar(2) | avatar(3) | avatar(4) | avatar(5) |
|---|---|---|---|---|---|
| 5 | 5 | =5+2<br>7 | = 5+3<br>8 | =7+4<br>11 | 8+5<br>13 |

10. These are the first four generations of the Cantor's Cheese fractal.



(a) How many circles are drawn in each generation of Cantor's Cheese? /4

| Generation | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| New Circles | 0 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
| Total Circles | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |

(b) Write a recursive method to find the total number of circles after n generations. /5

public (method start word) int (return type (output)) circle (method name (whatever)) ( int (parameter type) n (parameter name (input)) ) {

must match

```
if (n <= 1)
    return 1;
else
    return circle (n - 1) * 2 + 1
```

* Be careful to match method name + parameter name

11. An anagram game asks users to figure out an appropriate anagram for the word "listen". /5


(a) Fill in the memory diagram for the start of the game (listen):

| [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|
| L | I | S | T | E | N |

(b) Fill in the memory diagram for the winning condition of the game: (silent)

| [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|
| S | I | L | E | N | T |

(c) Declare the array for the start of the game.

char a[] = {'L', 'I', 'S', 'T', 'E', 'N'};

(d) Fill in the code to check for a win:

```
public boolean win ()
{   int count = 0;
    if (__a__ [__0__] == '__S__'))
        count++;
    if (__a__ [__1__] == '__I__'))
        count++;
    if (__a__ [__2__] == '__L__'))
        count++;
    if (__a__ [__3__] == '__E__'))
        count++;
    if (__a__ [__4__] == '__N__'))
        count++;
    if (__a__ [__5__] == '__T__'))
        count++;

    if (count == 6)
        return true;
    else
        return false;
}
```

12. Circle and correct 5 errors. The code is used to change a square in the Pizza Party game. /5

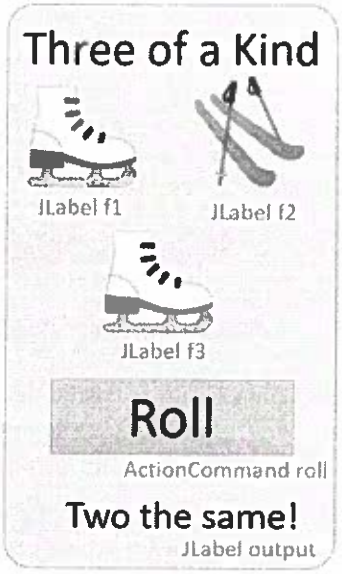
```
public int rollsmall (JLabel square)
{   void
    int n = (int) (Math.random () * 6);

    if (n == 1)
        square.setIcon (createImageIcon ("small_tomato.png"));
    else if (n == 2)
        square.setIcon (createImageIcon ("small_pineapple.png"));
    else if (n == 3)
        square.setIcon (createImageIcon ("small_pepper.png"));
    else if (n == 4)
        square.setIcon (createImageIcon ("small_mushroom.png"));
    else if (n == 5)
        square.setIcon (createImageIcon ("small_bacon.png"));
    else
        sq.setIcon (createImageIcon ("small_cheese.png"));
}   square
```

# Application

13. Fill in the method to change images in the "Three of a Kind" app. Then, call it in ActionPerformed. /6

(a) Fill in the setPic method:

```
public void setpic(JLabel t, int num) {
    if (num == 0 )
        t.setIcon(createImageIcon(" ski.png "));
    else if (num == 1 )
        t.setIcon(createImageIcon(" scarf.png "));
    else if (num == 2 )
        t.setIcon(createImageIcon(" skate.png "));
    else
        t.setIcon(createImageIcon(" gloves.png "));
}
```




The picture names, the corresponding random numbers.


(b) Call the setPic method for each picture in actionPerformed.

```
public void actionPerformed(ActionEvent e) {
//random number for each
  int r1 = (int) Math.random()*4;
  int r2 = (int) Math.random()*4;
  int r3 = (int) Math.random()*4;

//three method calls:
  setPic ( f1 , r1 );
  setPic ( f2 , r2 );
  setPic ( f3 , r3 );
}
```

14. Write a recursive method for the following sequence: 9 12 15 18 21 24 27 30 33 36 ... /6
For example, seq(1) returns 9 and seq(3) returns 15.

1 2 3 4

```
public int seq ( int n ){
    if (n <= 1)
        return 9;
    else
        return seq (n - 1) + 3
```

method start word | return type (output) | method name (whatever) | parameter type | parameter name (input)

must match.
Be careful to match method name & parameter name

15. Consider this win method from a tic tac toe game.

```
public void win () {
  if (b [0] [0] != 'b' && b [0] [0] == b [0] [1] && b [0] [1] == b [0] [2]) {
    String w = b [0] [0] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Horizontal Win", JOptionPane.ERROR_MESSAGE);
    button [0] [0].setEnabled (false);
    button [0] [1].setEnabled (false);
    button [0] [2].setEnabled (false);
  }
  if (b [1] [0] != 'b' && b [1] [0] == b [1] [1] && b [1] [1] == b [1] [2]){
    String w = b [1] [0] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Horizontal Win", JOptionPane.ERROR_MESSAGE);
    button [1] [0].setEnabled (false);
    button [1] [1].setEnabled (false);
    button [1] [2].setEnabled (false);
  }
  if (b [2] [0] != 'b' && b [2] [0] == b [2] [1] && b [2] [1] == b [2] [2]){
    String w = b [2] [0] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Horizontal Win", JOptionPane.ERROR_MESSAGE);
    button [2] [0].setEnabled (false);
    button [2] [1].setEnabled (false);
    button [2] [2].setEnabled (false);
  }
  if (b [0] [0] != 'b' && b [0] [0] == b [1] [0] && b [1] [0] == b [2] [0]){
    String w = b [0] [0] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Vertical Win", JOptionPane.ERROR_MESSAGE);
    button [0] [0].setEnabled (false);
    button [1] [0].setEnabled (false);
    button [2] [0].setEnabled (false);
  }
  if (b [0] [1] != 'b' && b [0] [1] == b [1] [1] && b [1] [1] == b [2] [1]) {
    String w = b [0] [1] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Vertical Win", JOptionPane.ERROR_MESSAGE);
    button [0] [1].setEnabled (false);
    button [1] [1].setEnabled (false);
    button [2] [1].setEnabled (false);
  }
  if (b [0] [2] != 'b' && b [0] [2] == b [1] [2] && b [1] [2] == b [2] [2]){
    String w = b [0] [2] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Vertical Win", JOptionPane.ERROR_MESSAGE);
    button [0] [2].setEnabled (false);
    button [1] [2].setEnabled (false);
    button [2] [2].setEnabled (false);
  }
  if (b [0] [0] != 'b' && b [0] [0] == b [1] [1] && b [1] [1] == b [2] [2]) {
    String w = b [0] [0] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Diagonal Win", JOptionPane.ERROR_MESSAGE);
    button [0] [0].setEnabled (false);
    button [1] [1].setEnabled (false);
    button [2] [2].setEnabled (false);
  }
  if (b [0] [2] != 'b' && b [0] [2] == b [1] [1] && b [1] [1] == b [2] [0]) {
    String w = b [0] [2] + " has won!!";
    JOptionPane.showMessageDialog (null, w, "Diagonal Win", JOptionPane.ERROR_MESSAGE);
    button [0] [2].setEnabled (false);
    button [1] [1].setEnabled (false);
    button [2] [0].setEnabled (false);
  }
}
```

(a) Circle 8 sets of repeated code in the before section. /2

(b) Rewrite the win method to call the oneLine method (coded in part c). /6

```
public void win(){
  oneLine(0, 0, 0, 1, 0, 2, "Horizontal Win");
  oneLine(1, 0, 1, 1, 1, 2, "Horizontal Win");
  oneLine(2, 0, 2, 1, 2, 2, "Horizontal Win");
  oneLine(0, 0, 1, 0, 2, 0, "Vertical Win");
  oneLine(0, 1, 1, 1, 2, 1, "Vertical Win");
  oneLine(0, 2, 1, 2, 2, 2, "Vertical Win");
  oneLine(0, 0, 1, 1, 2, 2, "Diagonal Win");
  oneLine(0, 2, 1, 1, 2, 0, "Diagonal Win");
}
```

(c) Write the oneLine method using the parameters given. /6

```
public void oneLine(int x1, int y1, int x2, int y2, int x3, int y3, String dir)
{
if (b[x1][y1] != 'b' &&
  b[x1][y1]==b[x2][y2] && b[x2][y2]==b[x3][y3])
{
  String w = b[x1][y1] + " has won!!";
  JOptionPane.showMessageDialog(null, w,
    dir, JOptionPane.ERROR_MESSAGE);
  button[x1][y1].setEnabled(false);
  button[x2][y2].setEnabled(false);
  button[x3][y3].setEnabled(false);
}
}
```