MACQUARIE University

Department of Computing

COMP125 Fundamentals of Computer Science
Workshop - Recursion

Learning outcomes

By the end of this session, you will have learnt about recursions.

1. **Recursion trace**

   Consider the following recursive function definition,

```
int foo(int a) {
        int result;
        if(a == 2) {
                result = 2;
        }
        else {
                result = a + foo(a / 2);
        }
        return result;
}
```

   What is the value of variable `status` if the method call is,

```
        int status = foo(8);
```

   Also, draw a memory diagram explaining the variables for each method call and the interaction between methods.

   Solution:

```
foo(8) = 8 + foo(4)
foo(4) = 4 + foo(2)
foo(2) = 2
Therefore,
foo(4) = 4 + 2 = 6
foo(8) = 8 + 6 = 14
status = 14
```

2. **Debugging recursive methods**

   The following method attempts to compute the factorial of integer n. What is wrong with the method?

```
int factorial(int n) {
        return n * factorial(n - 1);
        if(n == 0)
                return 0;
}
```

Solution:

a. Termination conditions should be BEFORE recursive call.

b. Termination condition is incorrect. Should return 1 if $n \leq 1$ otherwise factorial will become 0.

```
int factorial(int n) {
        if(n <= 1)
                return 1;
        return n * factorial(n - 1);
}
```

Give an example of a value, that, if passed to the method `foo`, results in `StackOverflowError` (method calls itself indefinitely).

Solution:

```
foo(24) = 24 + foo(12)
foo(12) = 12 + foo(6)
foo(6) = 6 + foo(3)
foo(3) = 3 + foo(1)
foo(1) = 1 + foo(0)
foo(0) = 0 + foo(0)
...
```

3. **Some more recursive trace**

Consider the following recursive method definition,

```
int foo(int a) {
        if(a <= 0)
                return 0;
        if(a % 2 == 0)
                return foo(a/2);
        else
                return 1 + foo(a/2);
}
```

What is the value of variable `result` if the method call is,

```
        int result = foo(59);
```

Solution:

```
foo(59) = 1 + foo(29)
foo(29) = 1 + foo(14)
foo(14) = foo(7)
foo(7) = 1 + foo(3)
foo(3) = 1 + foo(1)
foo(1) = 1 + foo(0)
foo(0) = 0
Therefore, foo(59) = 5
```

4. **Writing a recursive method**

   a. Write a recursive method, that when passed an integer, returns the number of even digits in that integer. Return 0 if the integer is 0.

Solution:

```
int nEvenDigits(int n) {
        if(n == 0)
                return 0;
        if(n%2 == 0)
                return 1 + nEvenDigits(n/10);
        else
                return nEvenDigits(n/10);
}
```

   b. Write a recursive method, that when passed an integer n, return the sum of squares of the first n positive integers ($1^2 + 2^2 + ... + n^2$).

Solution:

```
int sumSquares(int n) {
        if(n <= 0)
                return 0;
        return n*n + sumSquares(n - 1);
}
```

5. **Writing a recursive method dealing with text**

   Write a recursive method, that when passed a String, returns the number of digits in the String.

   Solution:

```java
int nDigits(String s) {
        if(s == null || s.length() == 0)
                return 0;
        if(s.charAt(0) >= '0' && s.charAt(0) <= '9')
                return 1 + nDigits(s.substring(1));
                return nDigits(s.substring(1));
}
```

6. **Counting recursive method calls**

   How many calls are made to `gcd` if the original call is `gcd(30, 72`?

```java
int gcd(int a, int b) {
        if(b == 0)
                return a;
        return gcd(b, a%b);
```

   Solution:

```
gcd(30, 72) = gd(72, 30)
gcd(72, 30) = gcd(30, 12)
gcd(30, 12) = gcd(12, 6)
gcd(12, 6) = gcd(6, 0)
gcd(6, 0) = 6

a total of 5 method calls
```

7. **(Tracing slightly more complex recursive methods)**

   Consider the definition of the following recursive method,

```
public static void displayBrackets(int n) {
        if(n == 0)
                return;
        System.out.print("{");
        for(int i=0; i < n - 2; i++) {
                displayBrackets(n - 1);
                System.out.print(", ");
        }
        displayBrackets(n - 1);
        System.out.print("}");
}
```

   What is the output of the following statement?

```
displayBrackets(3);
```

8. **(Time permitting) Defining recursive methods**

   I have made up a sequence called a *tribonacci* sequence. The first three numbers of this sequence are 1, 2 and 3, and every subsequent number in this sequence is the sum of the previous **three** numbers. Thus, the sequence is $1,2,3,6,11,20,37,68,\ldots$. Write a method to compute the n^{th} *tribonacci* number. Assuming the 1^{st} number is 1.

   Solution:

```
int tribonacci(int n) {
        if(n < 4)
                return n;
        return tribonacci(n - 1) + tribonacci(n - 2) + tribonacci(n - 3);
}
```

9. **(Time-permitting otherwise take-home task) Counting recursive method calls**

   How many calls are made to `tribonacci` if the original call is `tribonacci(5)`?

   Solution:

```
t(5) = t(4) + t(3) + t(2)
t(4) = t(3) + t(2) + (t1)
t(3) called twice returns 3 each time
t(2) called twice returns 2 each time
t(1) called once returns 1
So termination level calls are 5, and in addition t(4) and t(5) once each.
So a total of 7 method calls.
```

10. **(Time-permitting otherwise take-home task) Writing a recursive method**

Write a recursive method that displays an hour-glass pattern. For example, it displays the following pattern for $n = 6$. You MAY add more parameters to the method if required.

```
***********
 *********
  *******
   *****
    ***
     *
     *
    ***
   *****
  *******
 *********
***********
```

And it displays the following pattern for $n = 7$.

```
*************
 ***********
  *********
   *******
    *****
     ***
      *
      *
     ***
    *****
   *******
  *********
 ***********
*************
```

```java
public static void displayHourGlass(int n, int spaces) {
        if(n==0)
                return;
        String line = "";
        for(int i=1; i<= spaces; i++)
                line = line + "␣";
        for(int i=1; i<= 2*n-1; i++)
                line = line + "*";
        System.out.println(line);
        displayHourGlass(n-1, spaces+1);
        System.out.println(line);
}
```

11. **CHALLENGING (Time-permitting otherwise take-home task)**

Write a method that when passed a String containing letters of the English alphabet (you may assume that each letter occurs only once), returns an array of Strings containing all combinations of the letters from the input String.

```java
public static String[] getCombinations(String s)
```

For example, if `s = "abcd"`, the method should return the String

```
{
 "abcd", "abdc", "acbd", "acdb", "adbc", "adcb",
 "bacd", "badc", "bcad", "bcda", "bdac", "bdca",
 "cabd", "cadb", "cbad", "cbda", "cdab", "cdba",
 "dabc", "dacb", "dbac", "dbca", "dcab", "dcba"
}
```

```java
public static String[] getCombinations(String s) {
        if(s.length() == 1)
                return new String[]{s};
        int f1 = factorial(s.length());
        int f2 = factorial(s.length() - 1);
        String[] result = new String[f1];
        int counter = 0;
        for(int i=0; i < s.length(); i++) {
                String[] sub = getCombinations(s.substring(0,i)+s.substring(i+1));
                for(int k=0; k < f2; k++) {
                        result[counter++] = s.charAt(i) + sub[k];
                }
        }
        return result;
}
```