Unit 9: Recursion

Objectives

- Understand and learn to use recursion

Recursion

- A definition is recursive when it is defining something in terms of the definition itself.
- Example: The factorial function
  - "The factorial of a number is obtained multiplying the number by the factorial of the number minus 1"

$$n! = \begin{cases} n * (n-1)! & \text{If } n>0 \\ 1 & \text{If } n==0 \end{cases}$$

Recursion

- A function or script is recursive when a call to itself is included in its code. The call can be direct:

```
function functionA()
    functionA();
end
```

or indirect:

```
function functionA()
  functionB();
end
```

```
function functionB()
    functionA();
end
```

- A recursive function includes of 2 different execution paths
  - One invoking the own function (recursion).
  - The other indicating the stop condition without invoking the function.

Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```

Test.m

```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```

factorial.m

Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```


```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```

FUNCTION WORKSPACE

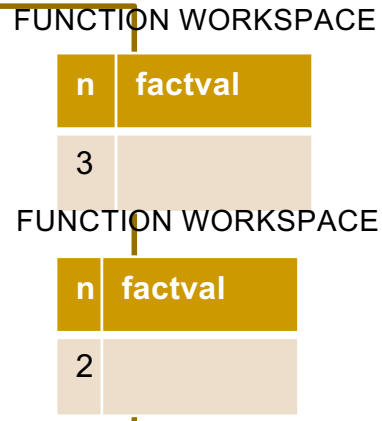
n	factval
3	

Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```


```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```



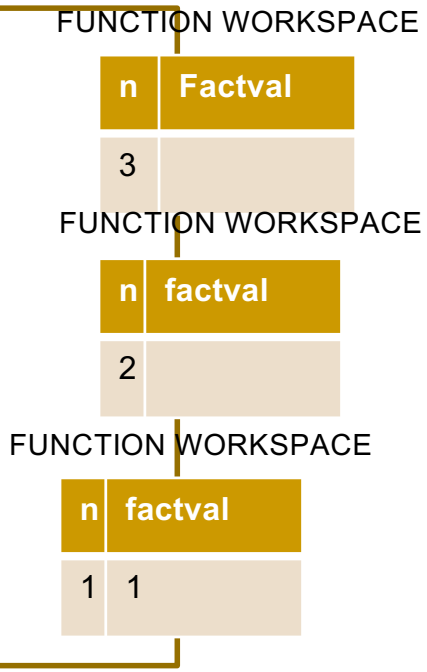
Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```

3

```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```



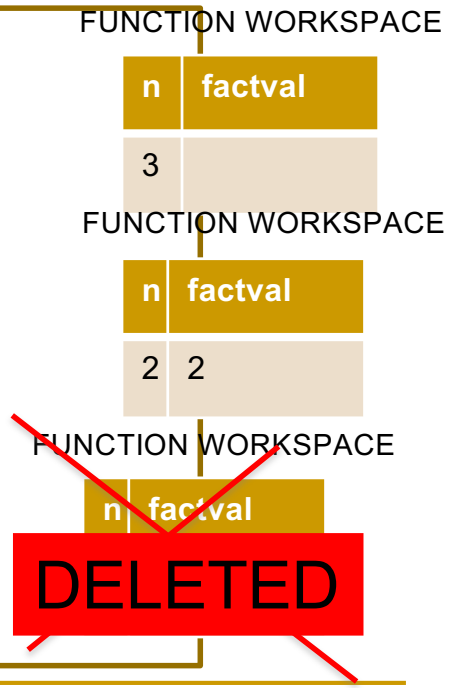
Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```

3

```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```



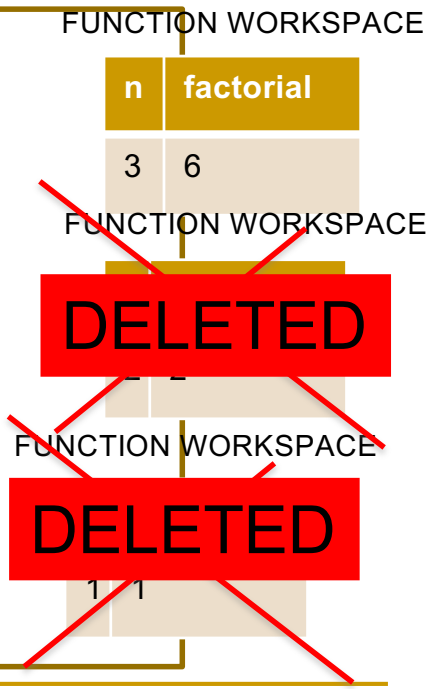
Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```

3

```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```

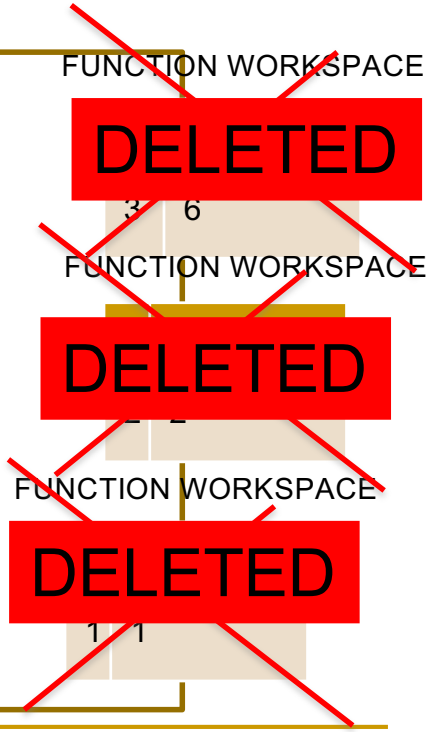


Recursion Example

```
numberVal = input('Introduce a number');
rdo = factorial(numberVal);
fprintf('The factorial of that number is %d', rdo);
```


```
function [factVal]= factorial(n)
% Function to compute the factorial of a number

            if (n >1)
               factVal = n* factorial (n-1));
            else
               factVal = 1;
            end;
end
```



Fundamental rules of recursion

- *Base cases.* You must **always** have some base case where the function can do its job without making a recursive call. **Always specify a base case otherwise indefinite recursive will occur.**

- *Always make progress.* For the cases that are to be solved recursively, the recursive call must always be to a case that makes progress toward a base case.

Exercise

- Write a function which computes the power of a number using recursion. The function receives the number and its power. It only handles positive power numbers.

Exercise

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;

end
```

Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;

end
```

1st call - FUNCTION WORKSPACE

num	vpow	rdo
2	3	

Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;


end
```

1st call - FUNCTION WORKSPACE

num	vpow	rdo
2	3	

2nd call

num	vpow	rdo
2	2	

Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;

end
```

1st call - FUNCTION WORKSPACE

num	vpow	rdo
2	3	

2nd call

num	vpow	rdo
2	2	

3rd call

num	vpow	rdo
2	1	

Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;

end
```

1st call - FUNCTION WORKSPACE

num	vpow	rdo
2	3	

2nd call

num	vpow	rdo
2	2	

3rd call

num	vpow	rdo
2	1	

4rd call

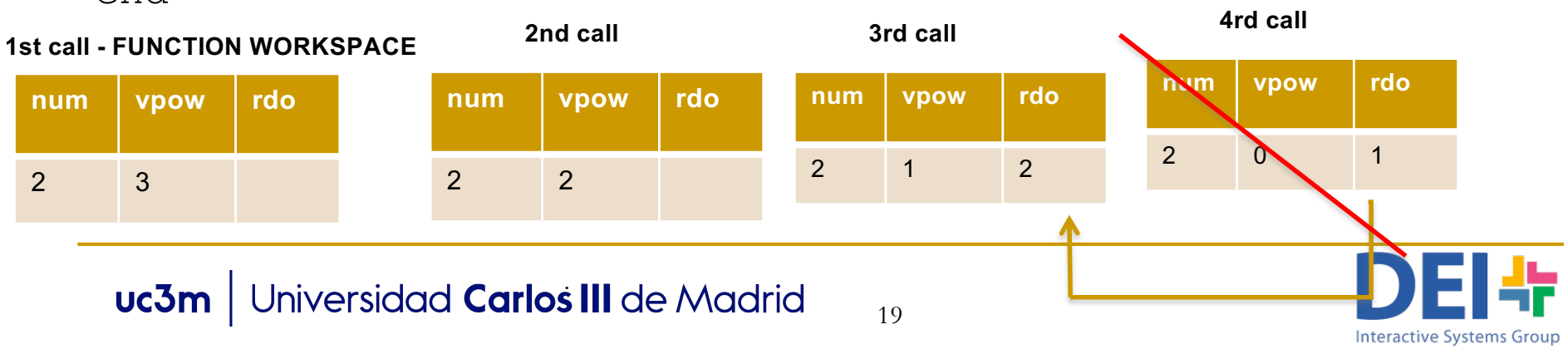
num	vpow	rdo
2	0	1

Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;


end
```

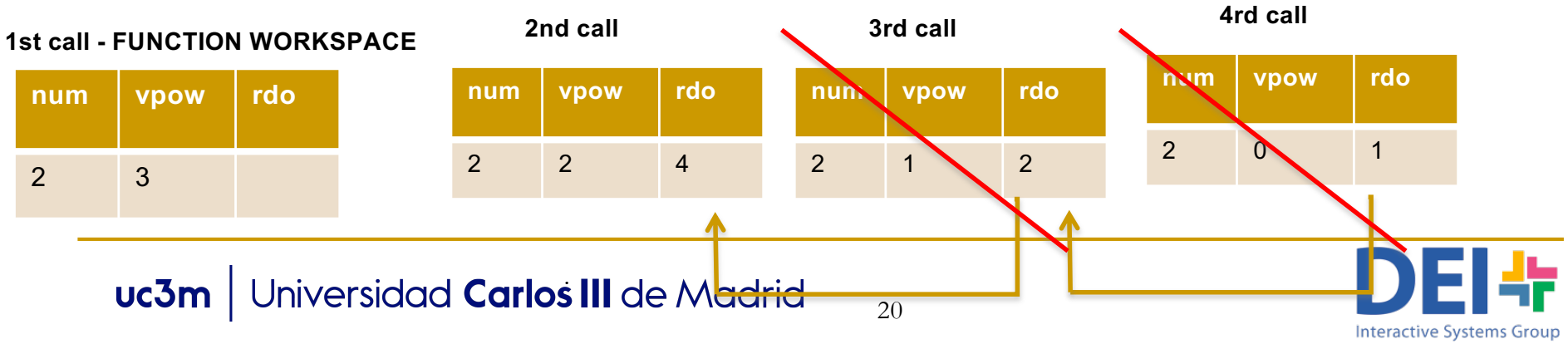


Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;


end
```

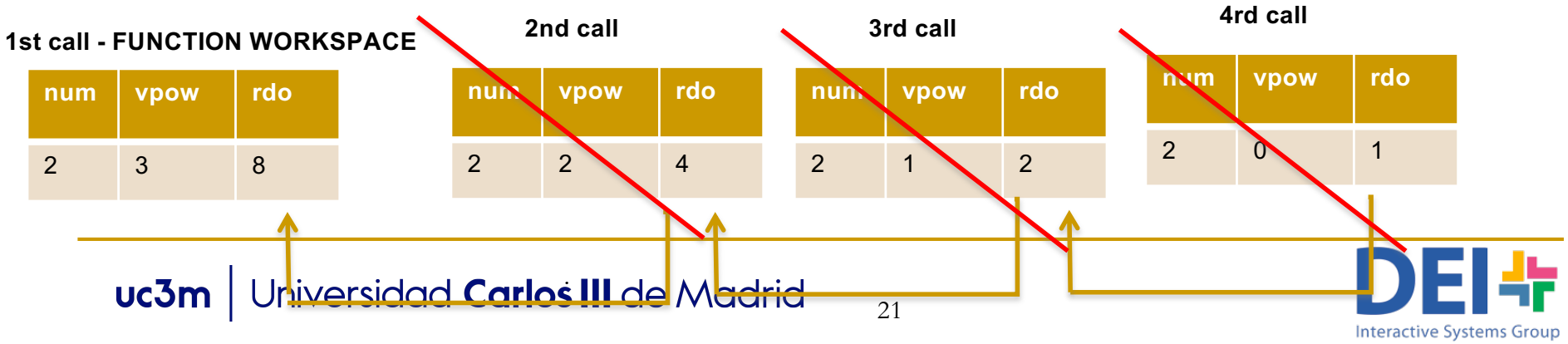


Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;

end
```



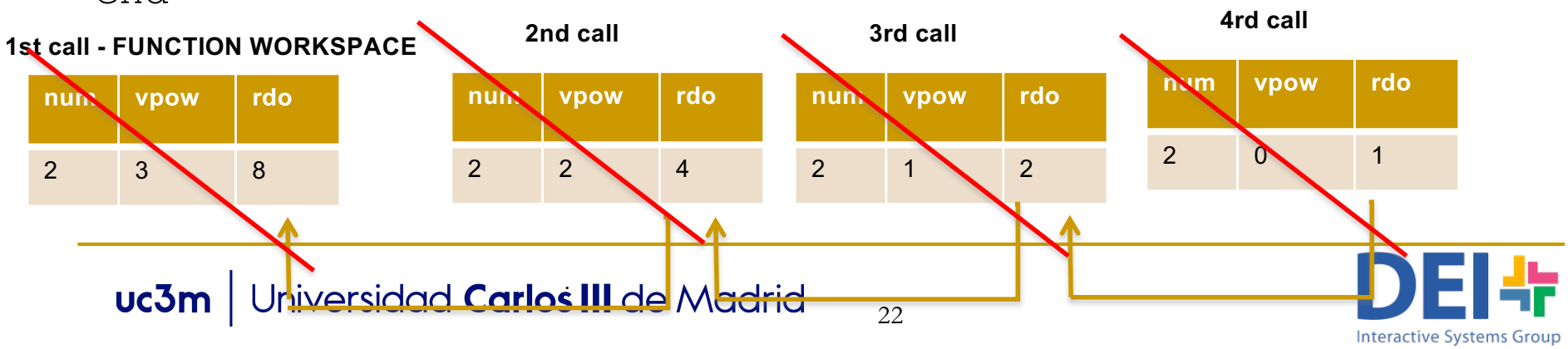
Exercise

LET'S SAY WE CALL THIS FUNCTION LIKE THIS:
var = myPower(2,3)

THE VALUE OF VAR WILL BE 8

```
function [rdo] = myPower(num, vpow)
 % Function myPpower. Returns the power of a number
 %
 if vpow == 0
   rdo = 1;
 else
   rdo = num * myPower(num, vpow-1);
 end;


end
```



Exercise

- Write a function which computes the Nth number of the Fibonacci sequence, where each number is computed by summing the two preceding and starting with 0 and 1: 0,1,1,2,3,5,8,13,…

Example

```
function [fibNum] = fibonacci(pos)
 % Function fibonacci. Returns the fibonacci number
 % correspondent to the position received.
 switch pos
    case 1
        fibNum = 0;
    case 2
        fibNum = 1;
    otherwise
        fibNum = fibonacci(pos-1) + fibonacci(pos-2);
 end;

end
```

Recursion Example

```
processNumbers();
disp('End');
```

Test.m

What does this program do?

```
function processNumbers ()
%
   numberVal = input('\n Introduce a number');
   if (numberVal ~= 0)
      processNumbers();
      fprintf('\n %d',numberVal);
   end;
end
```

processNumbers.m

Recursion Example

```
processNumbers();
disp('End');
```

Test.m

What does this program do?

```
function processNumbers ()
%
   numberVal = input('\n Introduce a number');
   if (numberVal ~= 0)
      processNumbers();
      fprintf('\n %d',numberVal);
   end;
end
```

processNumbers.m

It prints the numbers the user introduces in reverse order until he/she introduces a 0

Exercise

- Write a function ‘reverseVector’ which receives a vector and using recursion returns a vector containing the elements in reverse order.

Exercise

```
function [ revVect ] = reverseVector( vect )
 % Function reverseVectore receives a vector and returns a vector
 % containing the same elements in reverse order

 maxLength =length(vect);
 if maxLength == 1
     revVect = vect;
 else
     revVect = [vect(maxLength) reverseVector(vect(1:maxLength-1))];
 end;
end
```

Exercise

- Write a function ‘sumVector’ which receives a vector and using recursion returns the sum of its elements.

Exercise

```
function [ outSum ] = sumElements ( varVect )
 % Function sumElements returns the sum of the vector elements
  outSum = 0;
  if not(isempty(varVect))
     if length(varVect) == 1
      outSum = varVect(1);
     else
      outSum = varVect(length(varVect)) +
               sumElements(varVect(1:length(varVect)-1)));
     end;
 end;

end
```