Section 19.2 - Recursion

Layer 6: High-Order Language

Syllabus Content Section 19: Computational Thinking and Problem-Solving

S19.2.1 Show understanding of recursion

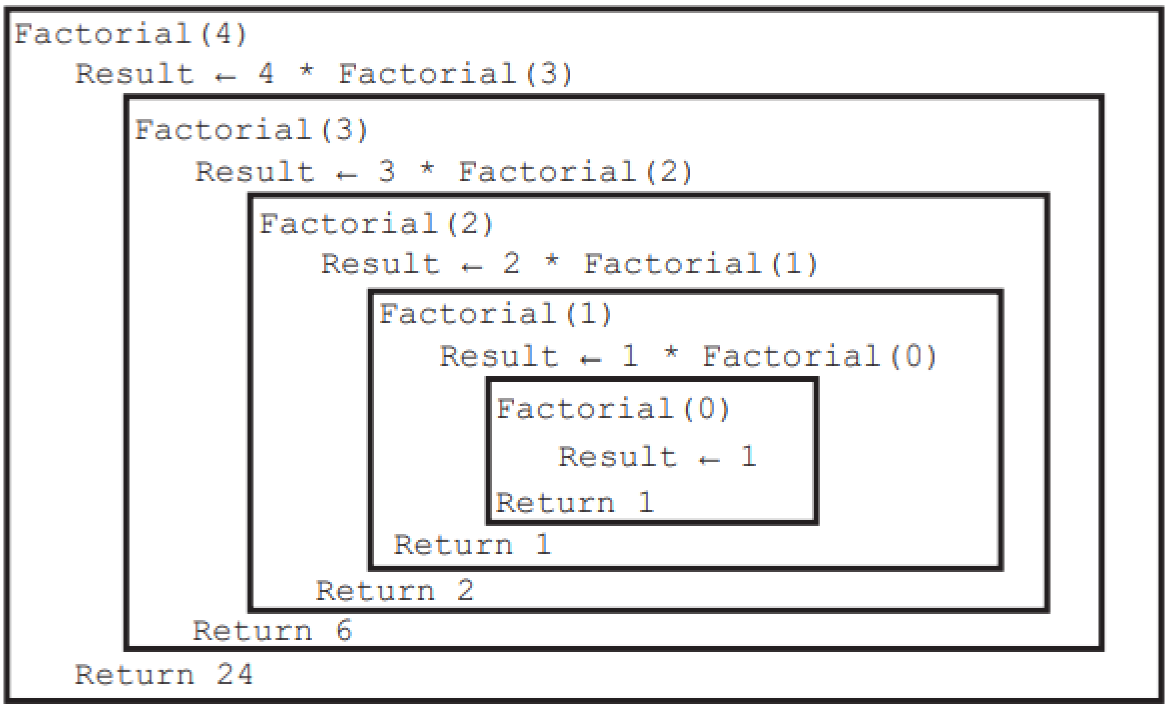
- Essential features of recursion.
- How recursion is expressed in a programming language.
- Write and trace recursive algorithms
- When the use of recursion is beneficial

`Recursive routine`: a function or procedure defined in terms of itself
`Base case`: an explicit solution to a recursive function
`General case`: a definition of a recursive function in terms of itself

> Tracing a recursive function

```
FUNCTION Factorial(n : INTEGER) RETURNS INTEGER
        IF n = 0 THEN
                Result ← 1
        ELSE
                Result ← n * Factorial(n - 1)
        ENDIF
        RETURN Result
ENDFUNCTION
```

call number	Procedure call	n=0	Result	Return Value
1	Factorial(4)	FALSE	4 * Factorial(3)	
2	Factorial(3)	FALSE	3 * Factorial(2)	
3	Factorial(2)	FALSE	2 * Factorial(1)	
4	Factorial(1)	FALSE	1 * Factorial(0)	
5	Factorial(0)	TRUE	1	1
(4)	Factorial(1)	FALSE	1 * 1	1
(3)	Factorial(2)	FALSE	2 * 1	2
(2)	Factorial(3)	FALSE	3 * 2	6
(1)	Factorial(4)	FALSE	4 * 6	24



```
Factorial(4)
    Result ← 4 * Factorial(3)
        Factorial(3)
            Result ← 3 * Factorial(2)
                Factorial(2)
                    Result ← 2 * Factorial(1)
                        Factorial(1)
                            Result ← 1 * Factorial(0)
                                Factorial(0)
                                    Result ← 1
                                Return 1
                        Return 1
                    Return 2
            Return 6
    Return 24
```

- Benifit
  - More elegant code / less lines of code needed
  - Easier for mathematical operations
- Drawback
  - If lots of repeated called then takes a lot of processor effort (overhead)
  - If lots of repeated called then takes a lot of memory effort (also called overhead)

S19.2.2 Show awareness of what a compiler has to do to translate recursive programming code

- Use of stacks and unwinding

`Unwinding` = When your functions reaches the base case and now works back up and gives you the answers / values
`Stacks` = Data Structure. Push and Pop data from it

Object code pushes a return address ad value of local variables (winding)
The object code pops the return address and local variables (unwinding)

> Example

call number	Procedure call	n=0	Result	Return Value	
1	Factorial(4)	FALSE	4 * Factorial(3)		winding
2	Factorial(3)	FALSE	3 * Factorial(2)		winding
3	Factorial(2)	FALSE	2 * Factorial(1)		winding
4	Factorial(1)	FALSE	1 * Factorial(0)		winding
5	Factorial(0)	TRUE	1	1	Base Case

call number	Procedure call	n=0	Result	Return Value	
(4)	Factorial(1)	FALSE	1 * 1	1	unwinding
(3)	Factorial(2)	FALSE	2 * 1	2	unwinding
(2)	Factorial(3)	FALSE	3 * 2	6	unwinding
(1)	Factorial(4)	FALSE	4 * 6	24	unwinding