L8 - Recursion

Friday, August 21, 2020 9:50 AM

Recursion is a Design Strategy/Paradigm/Technique for algorithm design.

What is Recursion?

Recur --- re-occur

Recursive Formulation/Recursive Decomposition

To conceive/decompose a problem in terms of simpler problem (**of the same nature as that of the original problem**) and some trivial activities.

In other words, in the solution to the problem, the same problem re-occurs, many times, on a smaller size (simpler problem(s)) until the result is achieved.

Iterative Solution

Fact(1) = 1
Fact(2) = 2 * 1
Fact(3) = 3 * 2 * 1
Fact(4) = 4 * 3 * 2 * 1
Fact(5) = 5 * 4 * 3 * 2 * 1
…
Fact(n) = n * n-1 * n-2 * … * 3 *2 *1

Iterative_Factorial(n)

1. let factorial = 1
2. for i = 2 to n
3.   factorial = factorial * i
4. return factorial

Fact(1) = 1
Fact(2) = 2 * Fact(1)
Fact(3) = 3 * Fact(2)
Fact(4) = 4 * Fact(3)
Fact(5) = 5 * Fact(4)
…

Let the ~~recursive~~/cyclical/circular formulation be
Fact(n) = n * Fact(n-1) **eq. 1**

for n = 5

Fact(5) = 5 * *Fact*(4)

= 5 * $\underline{4 * \mathit{Fact}(3)}$

= 5 * 4 * $\underline{3 * \mathit{Fact}(2)}$

= 5 * 4 * 3 * $\underline{2 * \mathit{Fact}(1)}$

= 5 * 4 * 3 * 2 * $\underline{1 * \mathrm{Fact}(0)}$

This will go on and on… It must stop.

Therefore, the correct Recursive Formulation/Decomposition is

$$\mathit{Fact(n)} = \begin{cases} 1 & if\ n = 1 \qquad Basis\ step \\ n * Fact(n-1) & if\ n > 1; \quad Recursive\ step \end{cases}$$

The *Recursive Step* can be thought of as a mechanism for generating simpler problems of the same kind as that of the original problem.

Fact(5) = 5 * $\underline{\mathit{Fact}(4)}$
= 5 * $\underline{4 * \mathit{Fact}(3)}$
= 5 * 4 * $\underline{3 * \mathit{Fact}(2)}$
= 5 * 4 * 3 * $\underline{2 * \mathit{Fact}(1)}$
= 5 * 4 * 3 * 2 * $\underline{1}$

Recursive_Factorial(n)
1. If n = 1
2. return 1
3. return n * Recursive_Factorial(n-1)

Problem: Calculate the Sum of a list of numbers.

Iterative Solution:

Sum(Arr)

1. let sumResult = 0
2. for i = 1 to Arr.length
3.   sumResult = sumResult + Arr[i]
4. return sumResult

Recursive Formulation/Decomposition

2 5 3 7 9 12 34 23 11 20
2 5 3 7 9 12 34 23 11 plus 20
2 5 3 7 9 12 34 23 plus 11 plus 20
2 5 3 7 9 12 34 plus 23 plus 11 plus 20

and so on …

Sum(n) = $\begin{cases} 0; & if\ n = 0; \quad Basis\ step \\ Sum(n-1) + Sum[n]; \quad (lastNumber); & if\ n > 0; \quad Recursive\ step \end{cases}$

Recursive_Sum(Arr, n)

1. if n = 0   // if n = 1
2.   return 0   //   return Arr[n]
3. return Recursive_Sum(Arr, n-1) + Arr[n]

Multiplication

4 * 5 (a*b b>=0)

4 + 4 + 4 + 4 + 4 = 4 * 5
4 + $\underline{4 + 4 + 4 + 4}$ = 4 + 4 * 4
4 + 4 + $\underline{4 + 4 + 4}$ = 4 + 4 + 4 * 3
…

Multiply(a, b) = $\begin{cases} 0; & if\ b = 0; \quad Basis\ step \\ a + Multiply(a, b-1); & if\ b > 0; \quad Recursive\ step \end{cases}$

Recursive-Multiply(a, b)

1. if b = 0
2.   return 0
3. return a + Recursive-Multiply(a, b-1)

Print a list of Elements

Display the list of elements/numbers in reverse order

$$ReversePrint(n) = \begin{cases} display\ n^{th}(first)\ element; & if\ n = 1 \\ display\ n^{th}\ element\ and\ then\ ReversePrint(n-1); & if\ n > 1 \end{cases}$$

Print(Arr,n)

1. If n == 1
2.   display Arr[n]
3.   return
4. display Arr[n]
5. Print(Arr,n-1)

2 15 7

Print(Arr,3) -> print(Arr, 2) -> print(Arr, 1)
7 15 2