BINUS UNIVERSITY

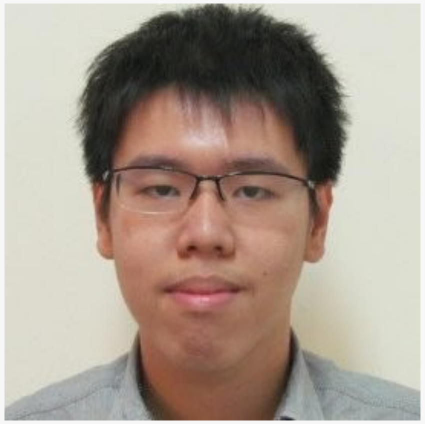
People
Innovation
Excellence

Subject : ALGORITHM AND PROGRAMMING
Year : 2022

Recursion (T)

People
Innovation
Excellence

Introduction



- **Alexander, S.Kom., M.T.I**
  - **S1 Binus University (2010-2014)**
    - Teknik Informatika
  - **S2 Binus University (2014-2016)**
    - Magister Teknik Informatika
  - **Professional Experience**
    - **IT Automation Consultant (2016-2021)**
      - Telco, Bank, F&B
    - **Engineering Manager (2021-current)**
      - Data Engineering & Data Science

People
Innovation
Excellence

Recursive Definition

- **Recursive** is a function call inside a certain function calling itself
- Recursive Function, suitable for recursive problem
- Example :

  Factorial (n) or n! defined as follows :

  n! = 1, for n = 0;

  *n! = n * (n-1)!, for n > 0*

  4! = 4 * 3!

  3! = 3 * 2!

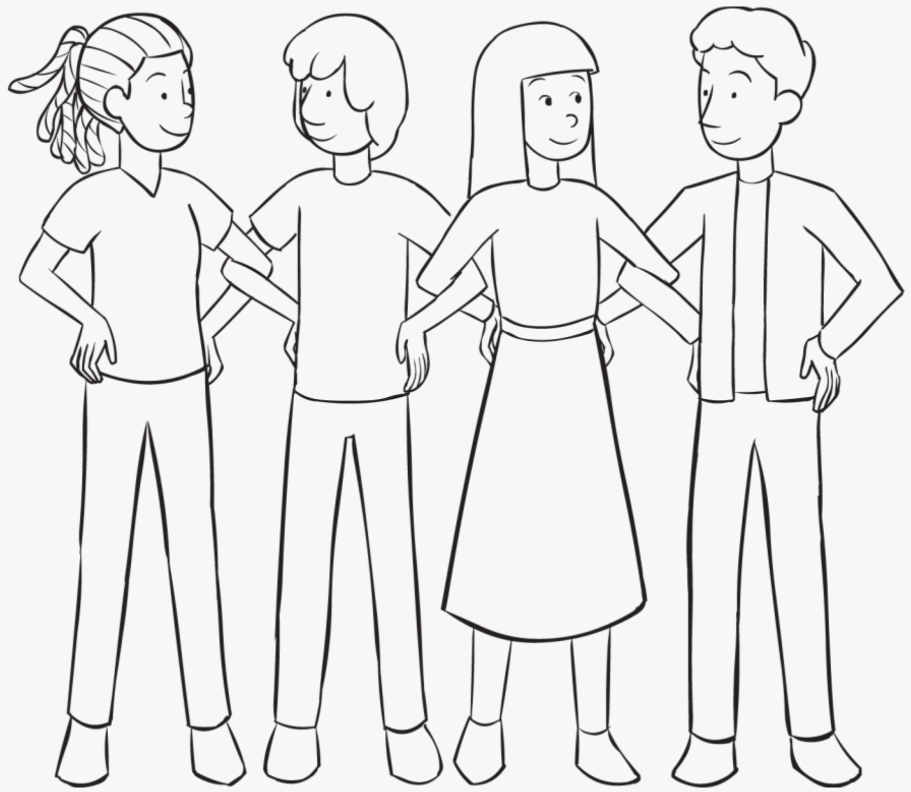
  2! = 2 * 1!

  1! = 1* 0!

  0! = 1

  Trace back : 4! = 1*2*3*4 = 24

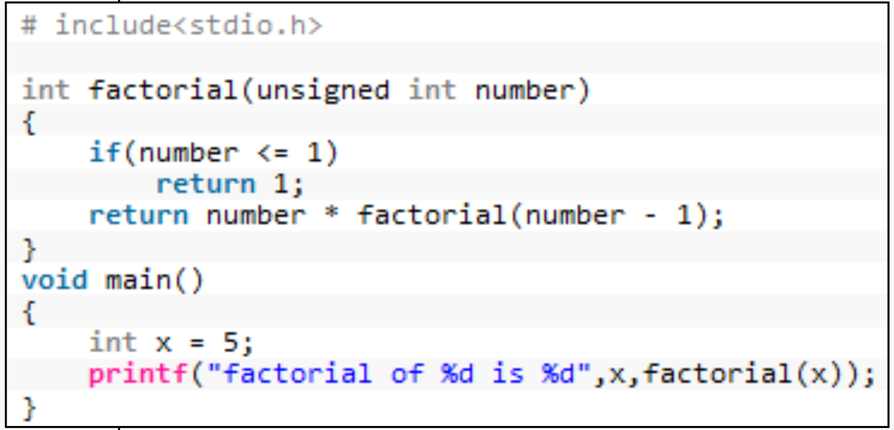
How Recursive Works?



Recursive Definition

Example: (5 factorial)

5!
(5 * 4!)
(5 * (4 *3!))
(5 * (4 * (3 * 2!)))
(5 * (4 * (3 * (2 * 1!))))
(5 * (4 * (3 * (2 * (1 * 0!)))))
(5 * (4 * (3 * (2 * (1 * 1)))))
(5 * (4 * (3 * (2 * 1))))
(5 * (4 * (3 * 2)))
(5 * (4 * 6))
(5 * 24)
120



```c
# include<stdio.h>

int factorial(unsigned int number)
{
    if(number <= 1)
        return 1;
    return number * factorial(number - 1);
}
void main()
{
    int x = 5;
    printf("factorial of %d is %d",x,factorial(x));
}
```

People
Innovation
Excellence

People
Innovation
Excellence

Recursive Function

Recursive Function has two components:

- **Base case**:
  return value(constant) without calling next recursive call.
- **Reduction step**:
  sequence of input value converging to the base case.

Example: (Factorial function)

- **Base case :** n = 0
- **Reduction step:** f(n) = n * f(n-1)

People
Innovation
Excellence

Iterative vs Recursive

Example: (Iterative vs Recursive)

- **Factorial - Recursive**

```
long factor (int n)
{
     if(n==0) return (1);
      else return(n * factor(n-1));
}
```

- **Factorial - Iterative**

```
long factor(int n) {
     long i, fac = 1;
     for(i=1; i<=n; i++)  fac *= i;
     return (fac);
}
```

People
Innovation
Excellence

Recursive

Recursive Drawback

Although recursive code more concise it needs:

- More memory consumption – as stack memory is needed
- Takes longer time, should traverse through all recursive call using stack

Recursive Best Practice

Generally, use recursive solution if:

- Difficult to solve iteratively.
- Efficiency using recursive has been reached
- Efficiency is less important in comparison with readability
- Memory efficiency and execution time are not the main concern

Consider carefully speed and efficiency using iterative approach, rather than nice logical design using recursive

People
Innovation
Excellence

Program Example Using Recursive

Fibonacci Number

sequence: 0, 1, 1, 2, 3, 5, 8, 13 ...

Relation between the number define recursively as follows:

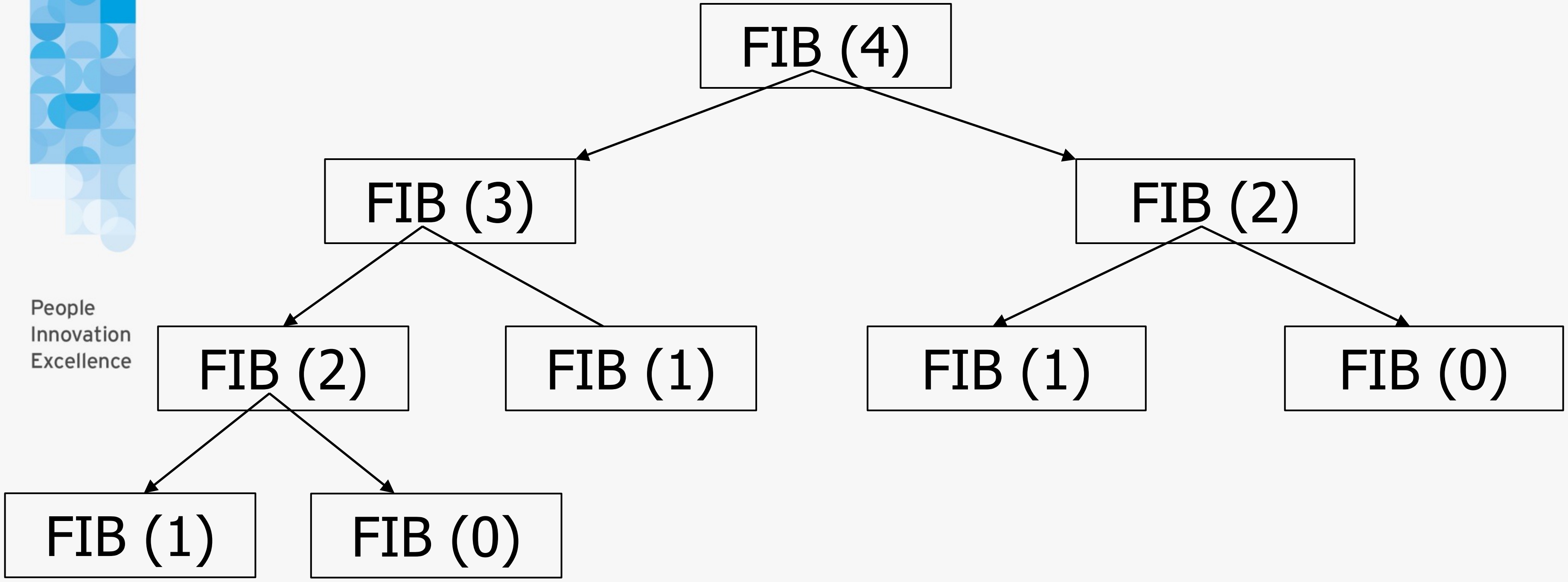
Fib(N) = N	if N = 0 or 1
Fib(N) = Fib(N-2) + Fib(N-1)	if N >= 2

```
int Fib(int n) {
    int f;
    if(n==0) f = 0;
        else if(n==1) f = 1;
            else f = Fib(n-2) + Fib(n-1);
    return f;
}
```

Program Example Using Recursive

Fibonacci Number

Fibonacci illustration N=4



People
Innovation
Excellence

References

- Paul Deitel & Harvey Deitel. (2016). C how to program : with an introduction to C++. 08. Pearson Education. Hoboken. ISBN: 9780133976892. Chapter 5
- Functions in C: http://aelinik.free.fr/c/ch15.htm

People
Innovation
Excellence

END