មហាវិទ្យាល័យវិស្វកម្ម
FACULTY OF ENGINEERING

Data Structure & Algorithm II

Lecture 1
Recursion and Applied Recursion

Chhoeum Vantha, Ph.D.
Telecom & Electronic Engineering

Content

- Recursion
  - Definitions
  - Recursive Characteristic
  - Recursive Structure
  - Recursive function notable comments
  - Type of Recursion
  - Infinite recursion
  - Requirements to design a recursive function
- Applied Recursion

Definitions

- Process of solving a problem by reducing it to smaller versions of itself
- Example: factorial problem
  - 5!
    - 5 x 4 x 3 x 2 x 1 =120
  - If n is a nonnegative
    - Factorial of n (n!) defined as follows:

$$0! = 1$$

$$n! = n \times (n-1)! \quad \text{if} \quad n > 0$$

Recursive Characteristic

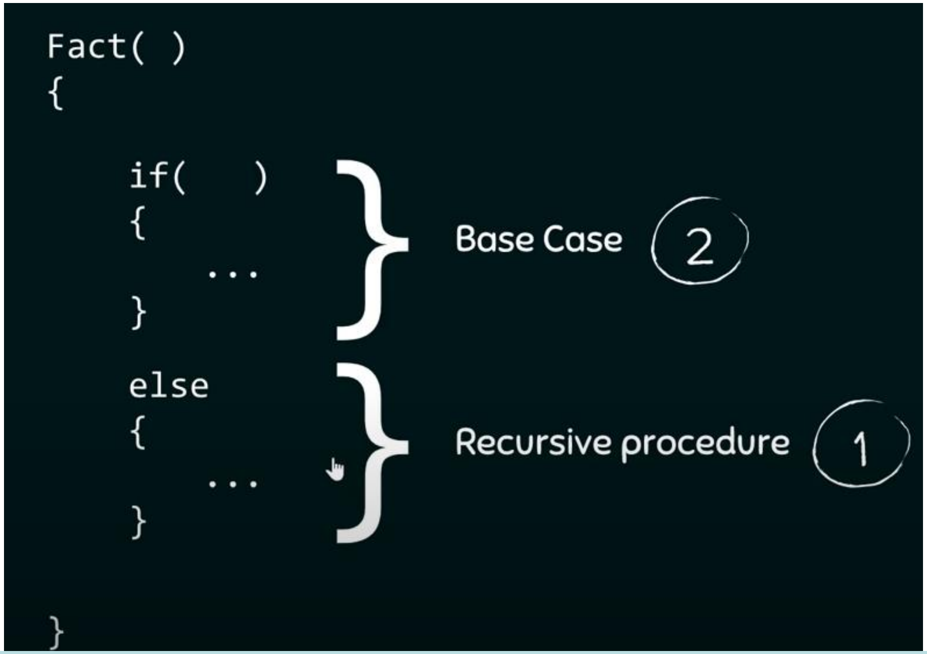
- Direct solution
  - Right side of the equation contains no factorial notation
- Recursive definition
  - A definition in which something is defined in terms of a smaller version of itself
- Base case
  - Case for which the solution is obtained directly
- General case/ Recursion procedure
  - Case for which the solution is obtained indirectly using recursion

Recursive Structure

- Syntax

1 Divide the problem into smaller sub-problems.

2 Specify the base condition to stop the recursion.

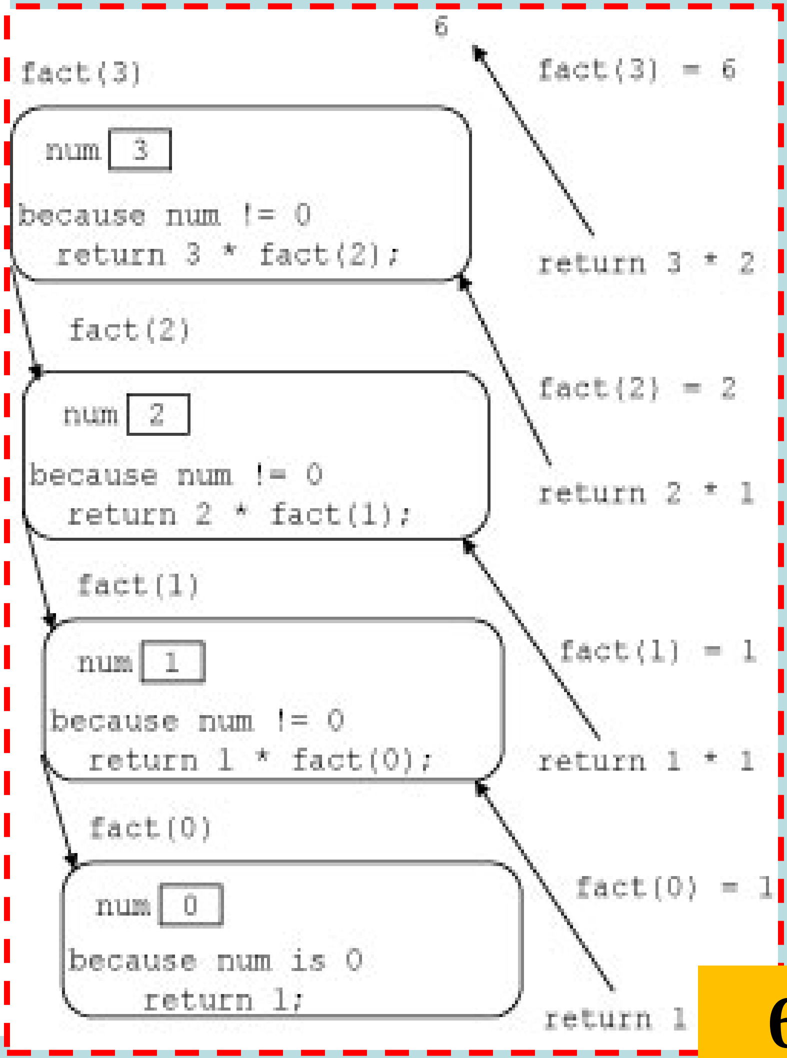


Recursive Structure

- **Example:** Recursive function implementing the factorial function

```
int fact(int num)
{
    if (num == 0)
        return 1;
    else
        return num * fact(num - 1);
}
```

- Tree step in execution of **fact(3)**



Recursive Structure

- Recursion insight gained from factorial problem
  - Every recursive definition must **have one (or more) base cases**
  - **General case** must eventually reduce to a base case
  - Base case **stops** recursion
- Recursive algorithm
  - Finds problem solution by **reducing a problem** to smaller versions of itself
- Recursive function
  - Function that **calls itself**

Recursive function notable comments

- Recursive function has an unlimited number of copies of itself (logically)
- Every call to a recursive function has its own
  - Code, set of parameters, local variables
- After completing a particular recursive call
  - Control goes back to the calling environment (previous call)
  - Current (recursive) call must execute completely before control goes back to the previous call
  - Execution in the previous call begins from the point immediately following the recursive call

Type of Recursion

- Direct recursion
  - Calls itself
- Indirectly recursive function
  - Calls another function, eventually results in original function call
  - Requires same analysis as direct recursion
  - Base cases must be identified, appropriate solutions to them provided
  - Tracing can be tedious
- Tail recursive function
  - Last statement executed: the recursive call
- None-Tail recursive function

Type of Recursion

- Directly recursive function

Direct recursion

A function is called direct recursive if it calls the same function again.

Structure of Direct recursion:

```
fun() {
    //some code

    fun();

    //some code
}
```

Type of Recursion

- Indirectly recursive function

A function (let say fun) is called indirect recursive if it calls another function (let say fun2) and then fun2 calls fun directly or indirectly.

Structure of Indirect recursion:

```
fun() {                         fun2() {
    //some code                     //some code

    fun2();                         fun();

    //some code                     //some code
}                               }
```

Infinite recursion

- Occurs if every recursive call results in another recursive call
- Executes forever (in theory)
- Call requirements for recursive functions
  - System memory for local variables and formal parameters
  - Saving information for transfer back to right caller
- Finite system memory leads to
  - Execution until system runs out of memory
  - Abnormal termination of infinite recursive function

Requirements to design a recursive function

- Understand problem requirements
- Determine limiting conditions
- Identify base cases, providing direct solution to each base case
- Identify general cases, providing solution to each general case in terms of smaller versions of itself

Applied Recursion: function

Look at the code distribution at the right side:

a) Predict the output of the program

b) What is base case and General case?

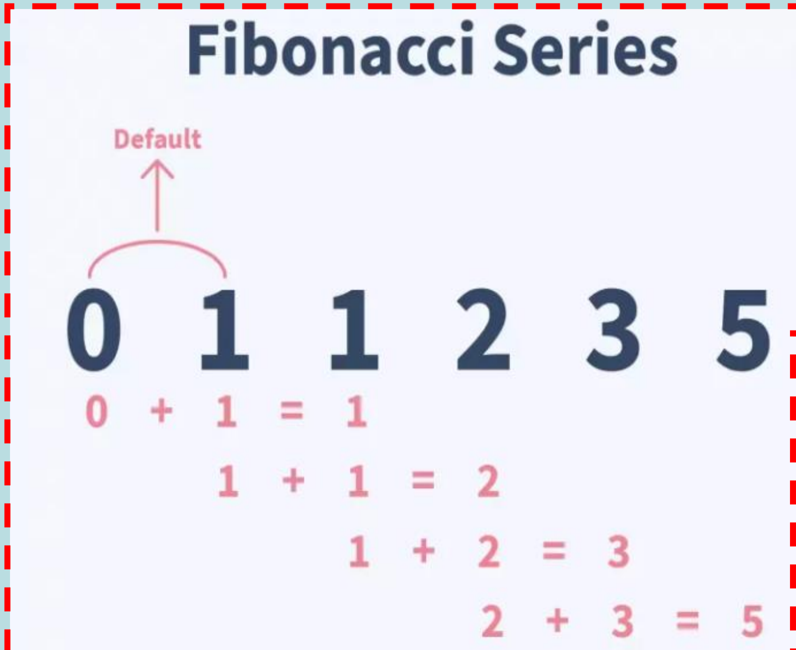
c) Show all recursion tree steps for the execution of **fun (5)**

```
#include <iostream>
using namespace std;
void fun(int x)
{
    if (x > 0) {
        cout << x << " ";
        fun(x - 1);
    }
}
int main()
{
    system ("cls");
    fun(5);
    return 0;
}
```

Applied Recursion: Fibonacci Number

- Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34 . . .
- Given first two numbers (a1 and a2)
  - nth number an, n >= 3, of sequence given by: $a_n = a_{n-1} + a_{n-2}$
- Recursive function: rFibNum
  - Determines desired Fibonacci number
  - Parameters: three numbers representing first two numbers of the Fibonacci sequence and a number n, the desired nth Fibonacci number
  - Returns the nth Fibonacci number in the sequence

Applied Recursion: Fibonacci Number



$$fibonacci(0) = 0$$
$$fibonacci(1) = 1$$
$$fibonacci(2) = (0 + 1) = 1$$
$$fibonacci(3) = (1 + 1) = 2$$
$$fibonacci(4) = (1 + 2) = 3$$
$$fibonacci(5) = (2 + 3) = 5$$
$$fibonacci(6) = (3 + 5) = 8$$

Applied Recursion: Fibonacci Number

- Third Fibonacci number
  - Sum of first two Fibonacci numbers
- Fourth Fibonacci number in a sequence
  - Sum of second and third Fibonacci numbers
- Calculating fourth Fibonacci number
  - Add second Fibonacci number and third Fibonacci number

$$F(n)=F(n-1)+F(n-2)$$

- Fibonacci series is a sequence of numbers in which each number is the sum of previous two numbers.
- Where F(n) denotes the nth term of the Fibonacci series

Applied Recursion: Fibonacci Number

- Recursive algorithm
  - Calculates nth Fibonacci number
    - a denotes first Fibonacci number
    - b denotes second Fibonacci number
    - n denotes nth Fibonacci number

$$rFibNum(a, b, n) = \begin{cases} a & \text{if } n = 1 \\ b & \text{if } n = 2 \\ rFibNum(a, b, n-1) + & \\ rFibNum(a, b, n-2) & \text{if } n > 2. \end{cases}$$

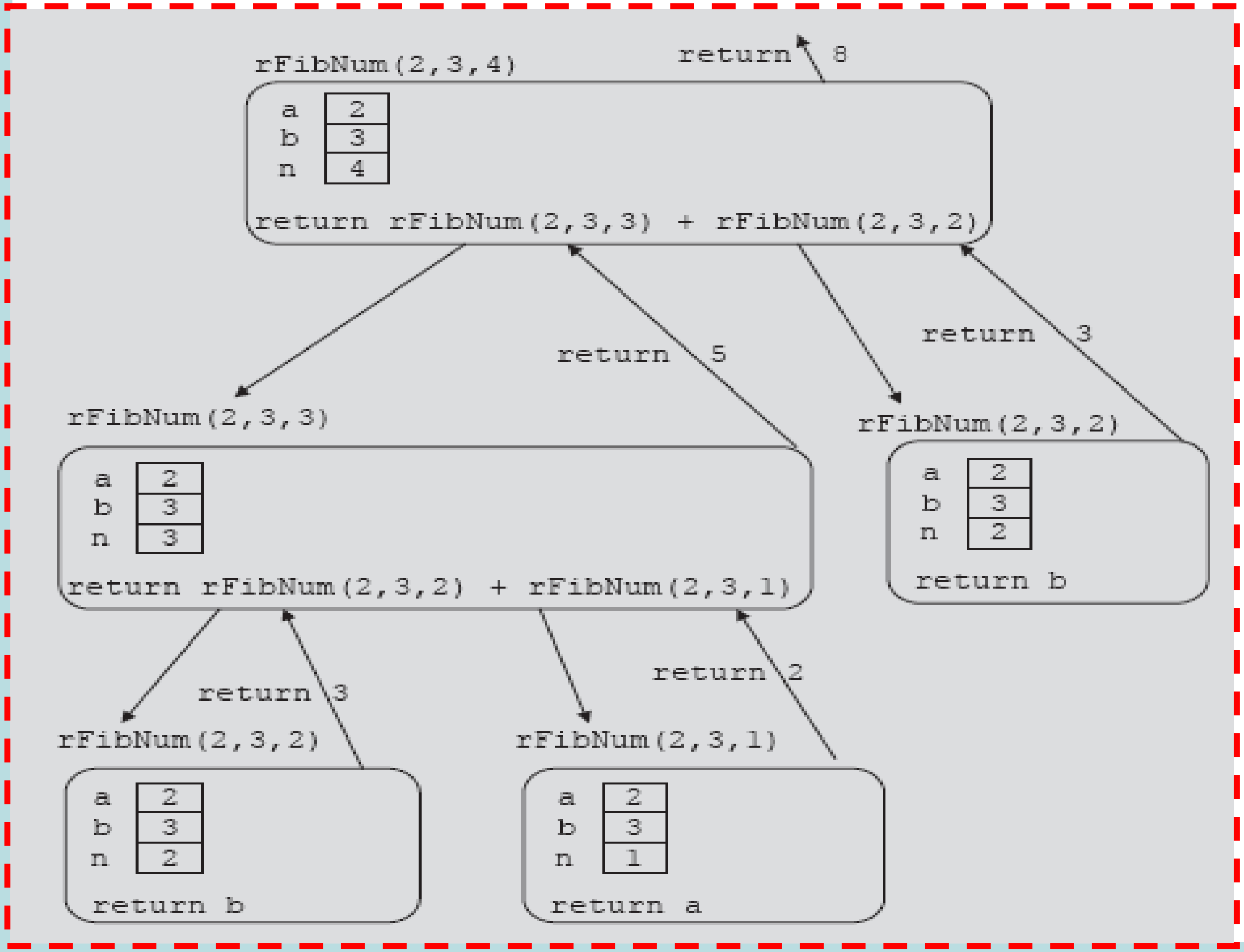
Applied Recursion: Fibonacci Number

- Recursive function implementing algorithm
- Trace code execution

```
int rFibNum(int a, int b, int n)
{
    if (n == 1)
        return a;
    else if (n == 2)
        return b;
    else
        return rFibNum(a, b, n - 1) + rFibNum(a, b, n - 2);
}
```

Applied Recursion: Fibonacci Number

- The tree step in execution of **rFibNum(2, 3, 4)**



W1 – Lab 1

Exercise

1. Create Power Function in C++ with **recursion and no- recursion** to calculate the power of numbers

power(a, n) = a^n

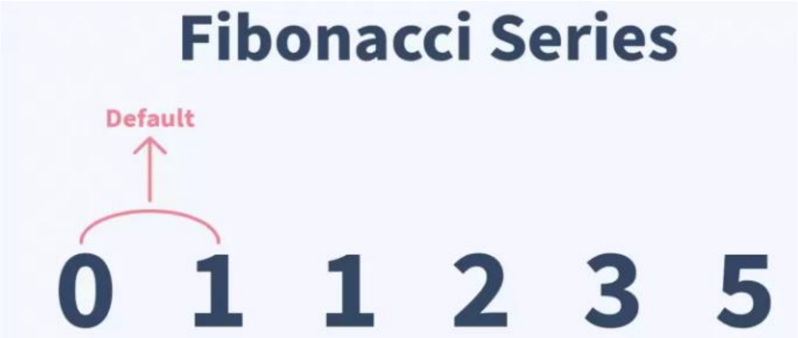
2. Reverse A Number Using **recursion and no- recursion** In C++

Original No. 379

Reverse No. 973

Exercise 1

3. Print Fibonacci series in C++ using recursion



4. Fibonacci in C++ using recursion

$$F(n)=F(n-1)+F(n-2)$$

Preparation Reading/Research

1. What is Recursion means in Data Structure and Algorithm?
2. What are the characteristic of Recursion?
3. Give a few examples of the application of Recursion

=> Write down your answer on your note!

Thanks!