Data Structure & Algorithms

Lecture 5

Recursion

Introduction

- In C++, any function can call another function.
- A function can even call itself.
- When a function call itself, it is making a recursive call.
- **Recursive Call**
  - **A function call in which the function being called is the same as the one making the call**.
- Recursion is a powerful technique that can be used in the place of iteration(looping).
- **Recursion**
  - **Recursion is a programming technique in which procedures and functions call themselves.**

Recursive Algorithm

- **Definition**
  - An algorithm that calls itself
- **Approach**
  - Solve small problem directly
  - Simplify large problem into 1 or more smaller subproblem(s) & solve recursively
  - Calculate solution from solution(s) for subproblem

Recursive Definition

- **A definition in which something is defined in terms of smaller versions of itself.**
- To do recursion we should know the followings
  - **Base Case:**
    - The case for which the solution can be stated non-recursively
    - The case for which the answer is explicitly known.
  - **General Case:**
    - The case for which the solution is expressed in smaller version of itself. Also known as recursive case.

Example 1

- We can write a function called **power** that calculates the result of raising an integer to a positive power. If X is an integer and N is a positive integer, the formula for X^N is

$$X^N = \underbrace{X * X * X * X * X * \ldots * X}_{N-times}$$

- We can also write this formula as

$$X^N = X * \underbrace{X * X * X * X * \ldots * X}_{(N-1)\_times}$$

- Also as

$$X^N = X * X * \underbrace{X * X * X * \ldots * X}_{(N-2)\_times}$$

- In fact we can write this formula as

$$X^N = X * X^{N-1}$$

Example 1(Recursive Power Function)

- Now lets suppose that X=3 and N=4

$$X^N = 3^4$$

- Now we can simplify the above equation as

$$3^4 = 3 * 3^3$$

$$3^3 = 3 * 3^2$$

$$3^2 = 3 * 3^1$$

- So the base case in this equation is

$$3^1 = 3$$

```
int Power ( int   x , int   n )
{
    if ( n == 1 )
        return x;     //Base case
    else
        return x * Power (x, n-1);
                      // recursive call
}
```

Example 2 (Recursive Factorial Function)

Factorial Function:

Given a five integer n, n factorial is defined as the product of all integers between 1 and n, including n.

So we can write factorial function mathematically as

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$$

Example 2 (Recursive Factorial Function)

- **Factorial definition**

  n! = n × n-1 × n-2 × n-3 × … × 3 × 2 × 1

  0! = 1

- **To calculate factorial of n**
  - **Base case**
    - **If n = 0, return 1**
  - **Recursive step**
    - **Calculate the factorial of n-1**
    - **Return n × (the factorial of n-1)**

Example 2 (Recursive Factorial Function)

```
int factorial(n)
   {
        if(n == 0)                          //Base Case
            return 1;
        else
            return  n * factorial(n-1);     //Recursion
   }
```

Evaluation of Factorial Example

To evaluate Factorial(3)

evaluate 3 * Factorial(2)

To evaluate Factorial(2)

evaluate 2 * Factorial(1)

To evaluate Factorial(1)

evaluate 1 * Factorial(0)

Factorial(0) is 1

Return 1

Evaluate 1 * 1

Return 1

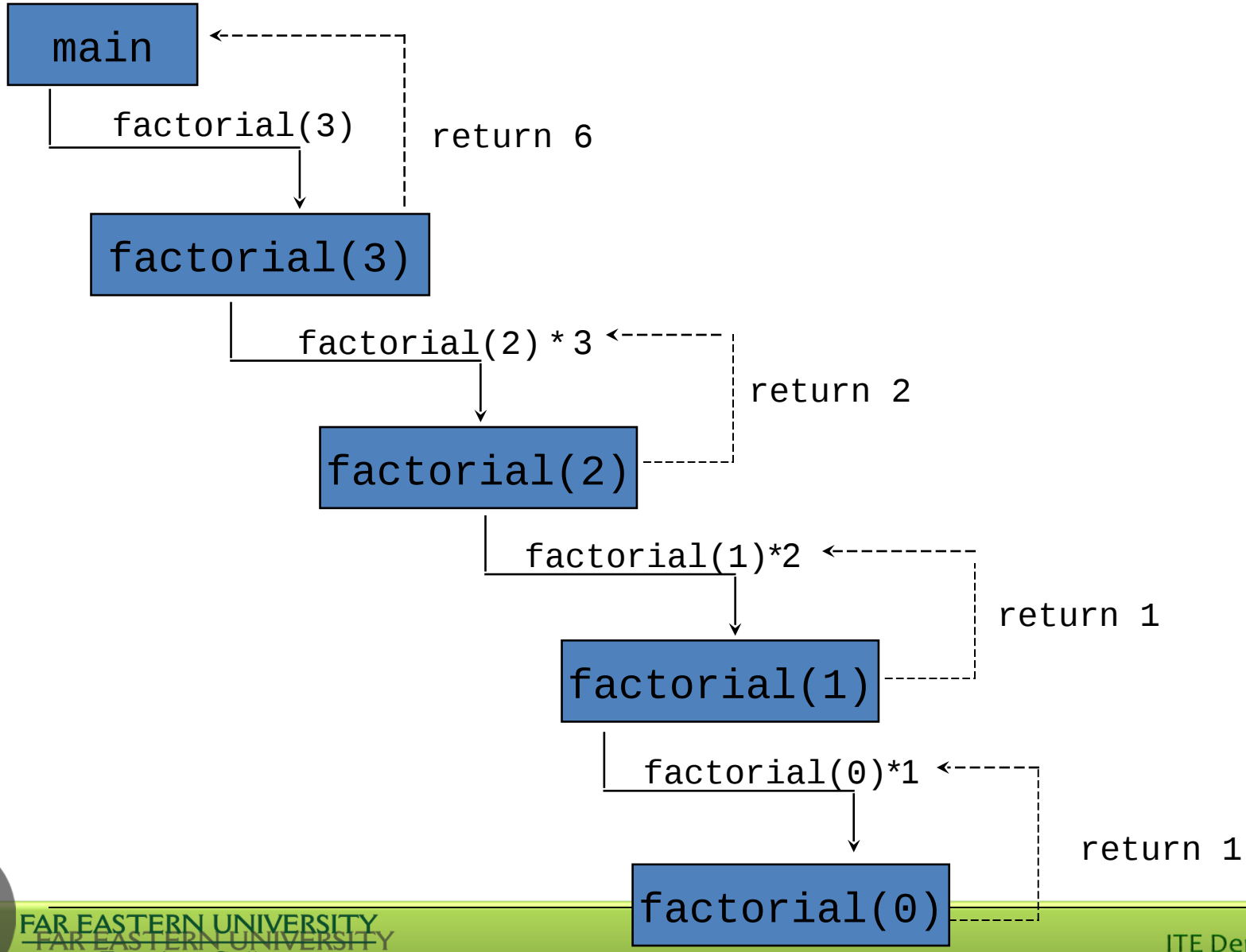
Evaluate 2 * 1

Return 2

Evaluate 3 * 2

Return 6

Recursive Programming



Rules For Recursive Function

1. In recursion, it is essential for a function to call itself, otherwise recursion will not take place.
2. Only user define function can be involved in recursion.
3. To stop the recursive function it is necessary to base the recursion on test condition and proper terminating statement such as *exit()* or *return* must be written using if() statement.
4. When a recursive function is executed, the recursive calls are not implemented instantly. All the recursive calls are pushed onto the stack until the terminating condition is not detected, the recursive calls stored in the stack are popped and executed.
5. During recursion, at each recursive call new memory is allocated to all the local variables of the recursive functions with the same name.

The Runtime Stack during Recursion

- To understand how recursion works at run time, we need to understand what happens when a function is called.
- Whenever a function is called, a block of memory is allocated to it in a run-time structure called the **stack.**
- This block of memory will contain
  - the function's **local variables,**
  - local copies of the function's **call-by-value parameters,**
  - pointers to its **call-by-reference parameters**, and
  - a **return address,** in other words where in the program the function was called from. When the function finishes, the program will continue to execute from that point.

Exercise

1. The problem of computing the sum of all the numbers between 1 and any positive integer N can be recursively defined as:

$$\sum_{i=1}^{N} +N = \sum_{i=1}^{N-1} +N + (N-1) = \sum_{i=1}^{N-2}$$

.etc=

Exercise

```
int sum(int n)
{
if(n==1)
   return n;
else
   return n + sum(n-1);
}
```

Recursion in ADT

- You can also use recursion in the data structures such as Stacks, queues, linked list, trees etc.
- **Task**
  - Write a recursive function Search. Which will search the given number in the linked list.
  - Steps involved in searching
    - **Base case**
      - If list is empty or search reached end of the list, return false
      - If number is found in the list, return true
    - **Recursive step**
      - Perform recursion for the next node in the list until you find the required element.

Tail Recursion

- A recursive function is called **tail recursive** if only one recursive call appears in the function and that recursive call is the last statement in the function body.