

# Recursive Functions

# Introduction

- A recursive function is a function that includes code that calls itself.

Thanks to the recursion technique many programming problems can be solved in an easier way.

- When a recursive a function is called, the computer keeps track of the various instances of the function.

In most cases this track is achieved using a call stack.

# Divide & Conquer

- “Divide & Conquer” is a computer programming technique in which a given problem is divided into sub problems of the same type.

# Recursive Functions Development

- Developing a recursive function is composed of the following four steps:

## The Programming Problem

Identifying the programming problem is the first step. It is highly important to acquire a clear understanding of the problem we cope with.

## Complexity Level

Identify the complexity level characteristic of the programming problem is important for being capable to set a general solution based on other solutions (with a lower complexity level) for the same problem.

# Recursive Function Development

## General Solution

Finding a general solution based on the solution for the same problem with a lower complexity level.

## Simplest Solution

Finding solution for the same problem(s) with lowest complexity level(s) will allow us to use the general solution for any problem of that same type with any complexity level.

# Factorial Sample

- Calculating a factorial (denoted by !) for a given integer number is done by multiplying all numbers between 1 and the given integer number.

Example

Calculating the factorial of 3 means performing the following calculation:

$$3! = 1 * 2 * 3 = 6$$

# Factorial Sample

## The Programming Problem

The programming problem is how to calculate the multiplication of all numbers between 1 and the number for which we want to calculate its factorial. Solving this problem without knowing – in advance – the exact integer number for which the calculation is required should be a very complex problem (unless we use a recursive function).

## Complexity Level

The complexity level can be  $n$ , the integer number for which  $n!$  calculation is required. The bigger  $n$  is the more difficult the problem we need to solve.

# Factorial Sample

## General Solution

Finding a general solution for  $n!$ , based on the solution for the same problem only with a lower difficulty level is very simple.

$$n! = n * (n-1)!$$

## Simplest Solution

The simplest solution is the solution for our problem, when the complexity level is the lowest possible one. When the complexity level is 0.

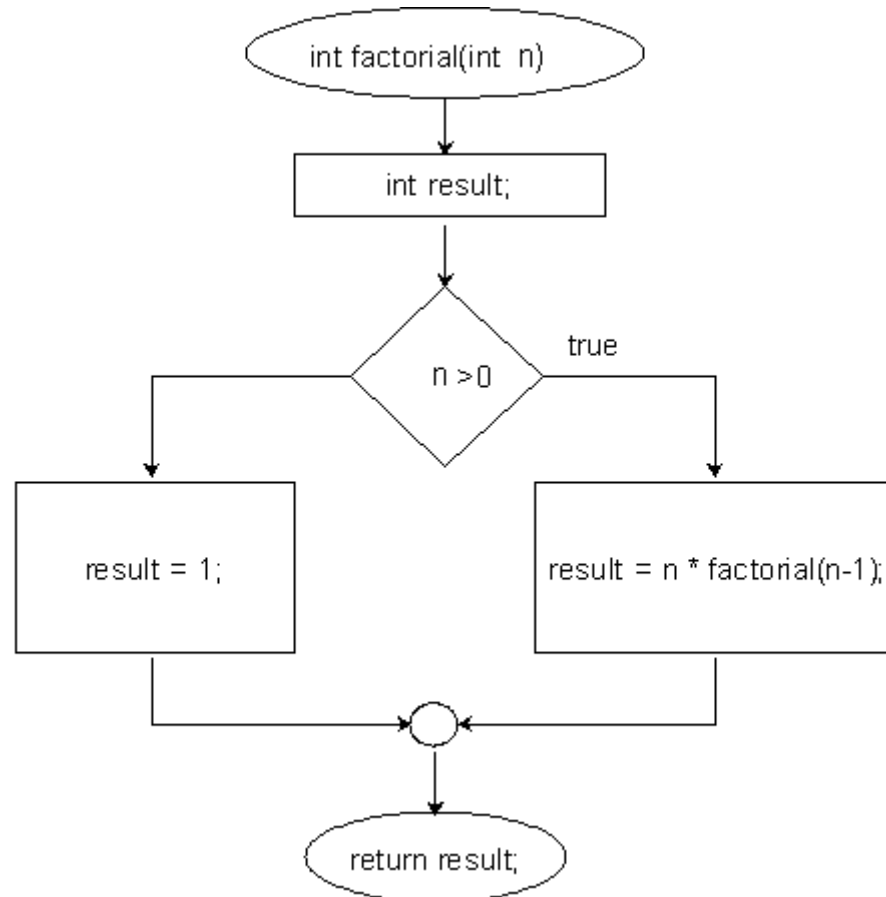
$$0! = 1$$



# Factorial Sample

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n == 0 \\ n * \text{factorial}(n-1) & \text{if } n > 0 \end{cases}$$

# Factorial Sample



# Recursive Functions

12/15/09

© 2008 Haim Mchael. All Rights Reserved.

1

## Introduction

- A recursive function is a function that includes code that calls itself.

Thanks to the recursion technique many programming problems can be solved in an easier way.

- When a recursive a function is called, the computer keeps track of the various instances of the function.

In most cases this track is achieved using a call stack.

## Divide & Conquer

- “Divide & Conquer” is a computer programming technique in which a given problem is divided into sub problems of the same type.

## Recursive Functions Development

- Developing a recursive function is composed of the following four steps:

### The Programming Problem

Identifying the programming problem is the first step. It is highly important to acquire a clear understanding of the problem we cope with.

### Complexity Level

Identify the complexity level characteristic of the programming problem is important for being capable to set a general solution based on other solutions (with a lower complexity level) for the same problem.

# Recursive Function Development

## General Solution

Finding a general solution based on the solution for the same problem with a lower complexity level.

## Simplest Solution

Finding solution for the same problem(s) with lowest complexity level(s) will allow us to use the general solution for any problem of that same type with any complexity level.

## Factorial Sample

- Calculating a factorial (denoted by !) for a given integer number is done by multiplying all numbers between 1 and the given integer number.

Example

Calculating the factorial of 3 means performing the following calculation:

$$3! = 1 * 2 * 3 = 6$$



## Factorial Sample

### The Programming Problem

The programming problem is how to calculate the multiplication of all numbers between 1 and the number for which we want to calculate its factorial. Solving this problem without knowing – in advance – the exact integer number for which the calculation is required should be a very complex problem (unless we use a recursive function).

### Complexity Level

The complexity level can be  $n$ , the integer number for which  $n!$  calculation is required. The bigger  $n$  is the more difficult the problem we need to solve.

## Factorial Sample

### General Solution

Finding a general solution for  $n!$ , based on the solution for the same problem only with a lower difficulty level is very simple.

$$n! = n * (n-1)!$$

### Simplest Solution

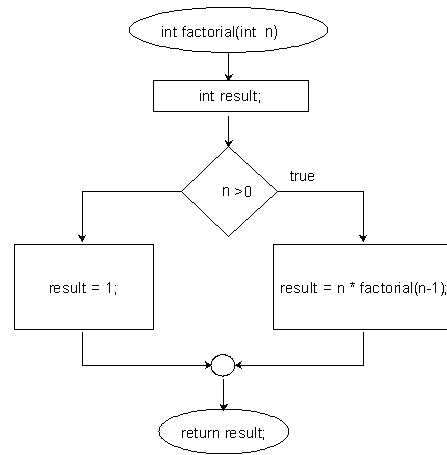
The simplest solution is the solution for our problem, when the complexity level is the lowest possible one. When the complexity level is 0.

$$0! = 1$$

## Factorial Sample

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n == 0 \\ n * \text{factorial}(n-1) & \text{if } n > 0 \end{cases}$$

## Factorial Sample



12/15/09

© 2008 Haim Michael. All Rights Reserved.

10