Διάλεξη 20: Αναδρομή (Recursion)

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Η έννοια της αναδρομής
- Μη-αναδρομικός / Αναδρομικός Ορισμός Συναρτήσεων
- Παραδείγματα Ανάδρομης
- Αφαίρεση της Αναδρομής

Διδάσκων: Παναγιώτης Ανδρέου

Μη αναδρομικές συναρτήσεις

- Προτού δούμε τι είναι αναδρομή θεωρήστε το πρόβλημα εύρεσης του παραγοντικού κάποιου αριθμού (factorial).

  0! = 1 , **1!** = 1 **2!** = 1x2 = 2 **3!** = 1x2x3 = 6

  4! = 1x2x3x4=24 **5!** = 1x2x3x4x5 = 120

  Ζητούμενο: *Να υλοποιήσουμε την factorial(int n) η οποία μας επιστρέφει το παραγοντικό κάποιου θετικού ακεραίου n.*

  Λύση

```
int factorial(int n) {
        int i, result=1;
        for (i=1; i<=n; i++)  {
                result *= i;
        }
        return result;
}
```

Αναδρομή

- Βασική έννοια στα Μαθηματικά και στην Πληροφορική.
- Στην πληροφορική η αναδρομή χρησιμοποιείται σαν *τεχνική προγραμματισμού* και σαν *μέθοδος σχεδιασμού αλγορίθμων*.
- Στον προγραμματισμό η αναδρομή εμφανίζεται με την **κλήση ενός υποπρογράμματος από τον εαυτό του**. Ένα αναδρομικό υποπρόγραμμα αποτελείται από:
  - ένα **βήμα τερματισμού**, όπου ορίζεται η εκτέλεση του υποπρογράμματος για κάποιες “μικρές ”τιμές των παραμέτρων του, και
  - ένα **αναδρομικό βήμα**, κατά το οποίο η εκτέλεση του υποπρογράμματος ορίζεται ως συνδυασμός κλήσεων του υποπρογράμματος σε άλλες “μικρότερες” τιμές των παραμέτρων.

Παράδειγμα 1: Παραγοντικός Αριθμός με Αναδρομή

- Ας ορίσουμε τώρα τον αναδρομικό ορισμό της factorial.

  0! = 1 , **1!** = 1 **2!** = 1x2 = 2 **3!** = 1x2x3 = 6

  4! = 1x2x3x4=24 **5!** = 1x2x3x4x5 = 120

- Ας ορίσουμε τώρα τον αναδρομικό ορισμό της factorial.

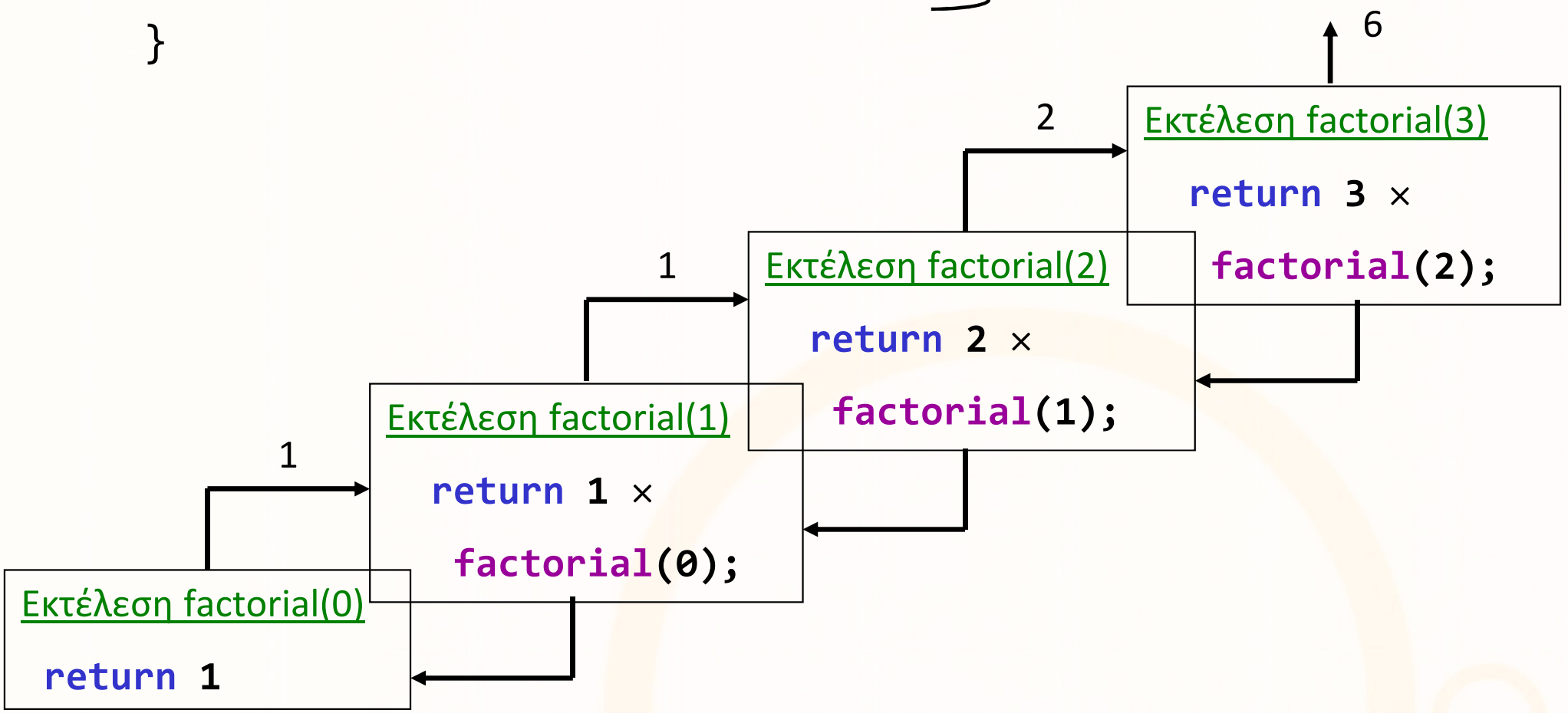
- **Αναδρομικός Ορισμός συνάρτησης <factorial>**
  - Βήμα Τερματισμού: **0! = 1**
  - Αναδρομικό Βήμα: **n! = n x (n-1)!**

Παραγοντικός Αριθμός με Αναδρομή

```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n x factorial(n-1);
        }
```

Βήμα Τερματισμού

Αναδρομικό Βήμα



Υλοποίηση αναδρομής

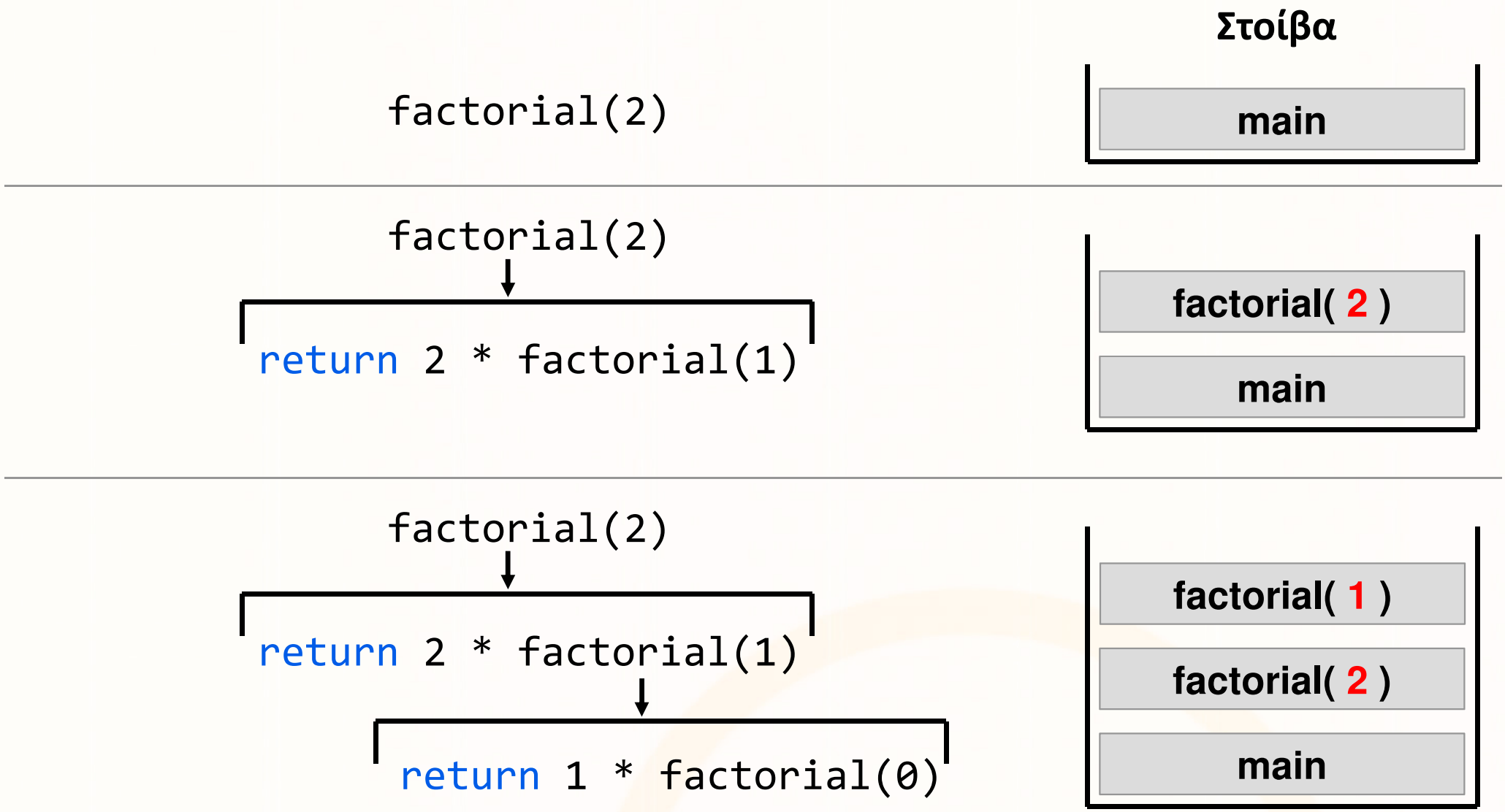
- Σε κάθε κλήση οποιασδήποτε συνάρτησης ένα σύνολο από λέξεις **(stack frame) φυλάσσεται σε μια στοίβα (την στοίβα του προγράμματος)**, από όπου μπορεί να ανασυρθεί.

- Όταν μια συνάρτηση διακόψει την εκτέλεσή της με την κλήση μιας άλλης συνάρτησης ***οι παράμετροι της συνάρτησης***, *η* ***διεύθυνση επιστροφής*** και *οι* ***τοπικές μεταβλητές*** της **καλούσας συνάρτησης** φυλάσσονται μέσα στη στοίβα του προγράμματος.

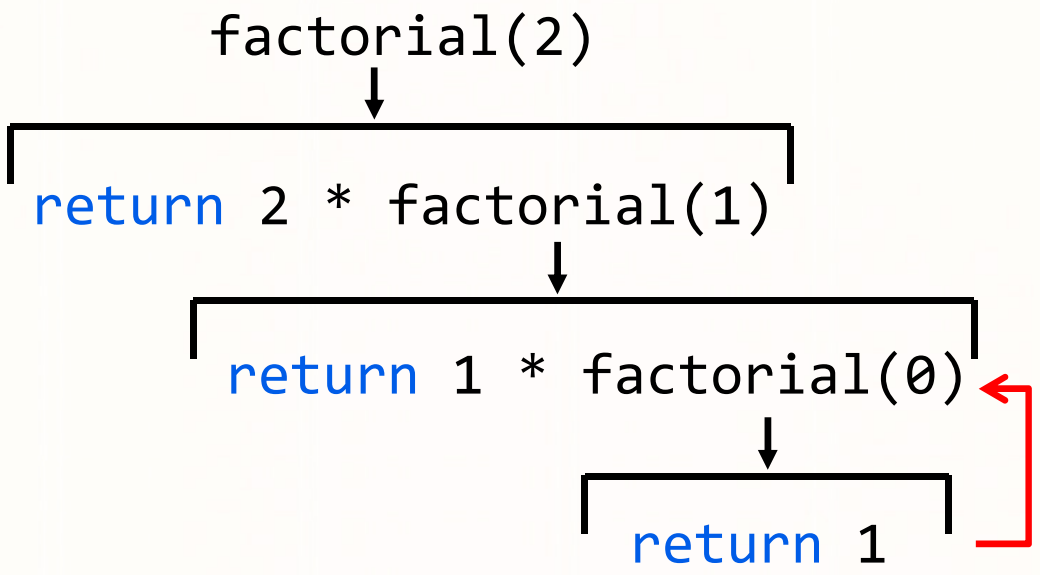
- Έτσι όταν η κληθείσα συνάρτηση τερματίσει το περιβάλλον την καλούσας συνάρτησης **ανασύρεται από τη στοίβα** για να συνεχιστεί κανονικά η εκτέλεσή της.

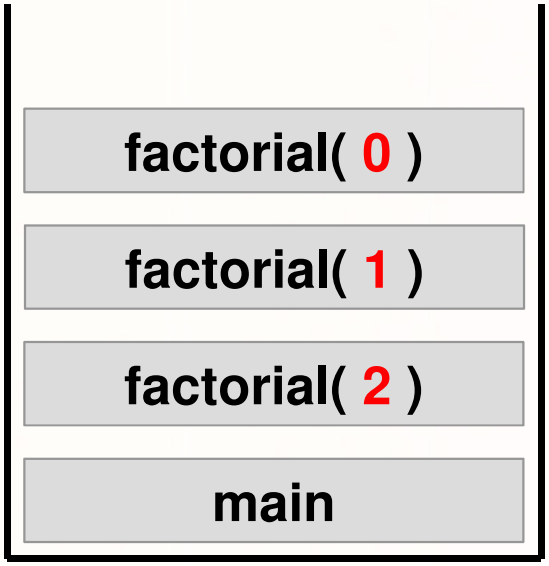
- Αφού κάθε κλήση μιας διαδικασίας εκτελείται στο δικό της περιβάλλον, είναι επιτρεπτή και **η κλήση συναρτήσεων από τον εαυτό** τους (αναδρομή).

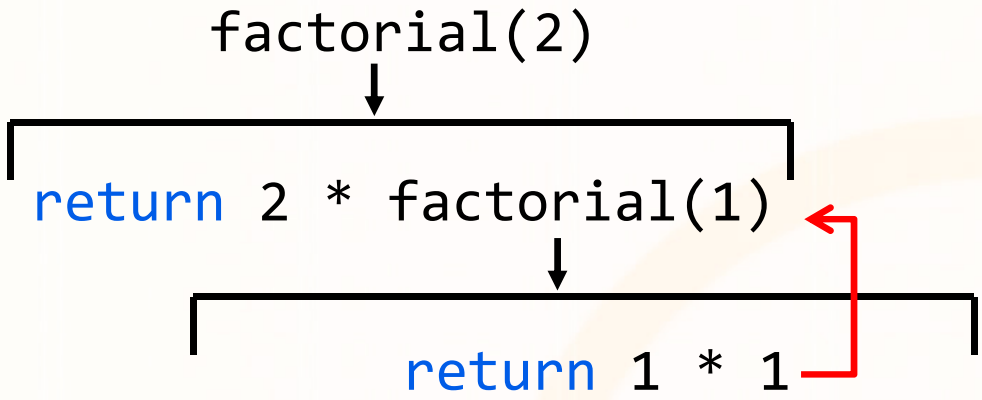
Αναδρομή και Διαχείριση Μνήμης

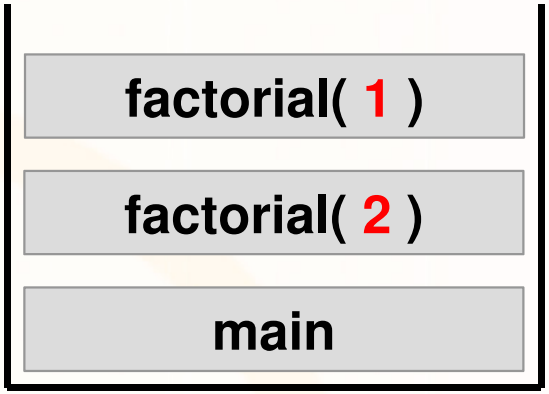


Αναδρομή και Διαχείριση Μνήμης

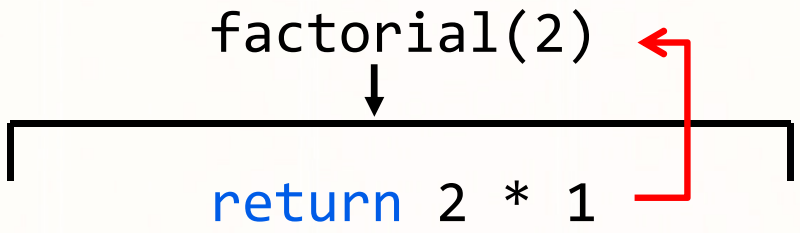


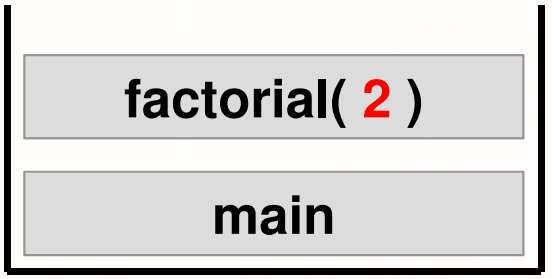






Αναδρομή και Διαχείριση Μνήμης





2

Στοίβα

Παράδειγμα 2: Δύναμη Αριθμού με Αναδρομή

2) Δύναμη (Power)

$a^0 = 1$

$a^n = a^{n-1}.a \quad (n>=1)$

```
int mpower(int a, int n) {
        if (n==0) return 1;
        return a*mpower(a, n-1);
}
```

Παράδειγμα

mpower(2,3) => 2*mpower(2,2)
= 2*2*mpower(2,1)
= 2*2*2*mpower(2,0)
= 2*2*2*1= 8

Παράδειγμα 3: Fibonacci Numbers

3) Αριθμοί Fibonacci (Leonardo of Pisa – 1202μΧ)

Χρησιμοποιήθηκαν για να εκφράσουν την αύξηση κουνελιών!

- Στον μήνα 0 έχουμε 0 ζεύγη , και στον μήνα 1 έχουμε 1 ζεύγος (η Γένεσης!)
- Το ζεύγος γονιμοποιείται μετά τον πρώτο μήνα (δηλ. στον δεύτερο).
- Κάθε μήνα, Κάθε ζεύγος παράγει ένα νέο ζεύγος
- Έστω ότι είμαστε στον μήνα **n** και έχουμε ένα πληθυσμό **F(n) ζευγών**. Αυτή την στιγμή μόνο κουνέλια που ήταν ζωντανά την στιγμή **n-2** παράγουν ένα νέο ζεύγος.
- Επομένως **F(n-2)** ζευγάρια προστίθεται στον παρόν πληθυσμό των F(n-1).
- Ο ολικός πληθυσμός την στιγμή F(n) είναι επομένως **F(n) = F(n-1) + F(n-2)**

0,1,1,2,3,5,8,13,21,34,55,89,….

Παράδειγμα 3: Fibonacci Numbers (συν.)

- Μαθηματικός Ορισμός

$$F_n = F(n) = \begin{cases} 0 & n = 0 \\ 0 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

- Κώδικας 1

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

- Κώδικας 2 ☺

```
int fibonacci(int n) {
  return (n<=1)? n : Fibonacci(n-1) + Fibonacci(n-2);
}
```

Παράδειγμα 4: Παλίνδρομα

4) Παλίνδρομα (Palindrome)

abac false

abba **true**

aba **true**

```
boolean isPalindrome (String s) {
  if (s.length()<=1)
    return true;

  else if ( s.charAt(0)!=s.charAt(s.length()-1) )
    return false;

  else
    return isPalindrome( s.substring(1, s.length()-1) );
}
```

Παράδειγμα 4: Παλίνδρομα (συν.)

- Η προηγούμενη μέθοδος παρουσιάζει **πρόβλημα αποδοτικότητας** διότι δημιουργεί ένα καινούριο String κάθε φορά που καλείται αναδρομικά

```
return isPalindrome( s.substring(1, s.length()-1) );
```

- Μερικές φορές είναι καλύτερα να χρησιμοποιηθεί μία βοηθητική συνάρτηση για να αποφύγουμε τέτοια προβλήματα

```
boolean isPalindrome (String s) {
  return isPalindrome(s, 0, s.length()-1);
}

boolean isPalindrome (String s, int low, int high) {
  if (high<=low) return true;
  else if ( s.charAt(low)!=s.charAt(high) )
    return false;
  else
    return isPalindrome(s, low+1, high-1 );
}
```

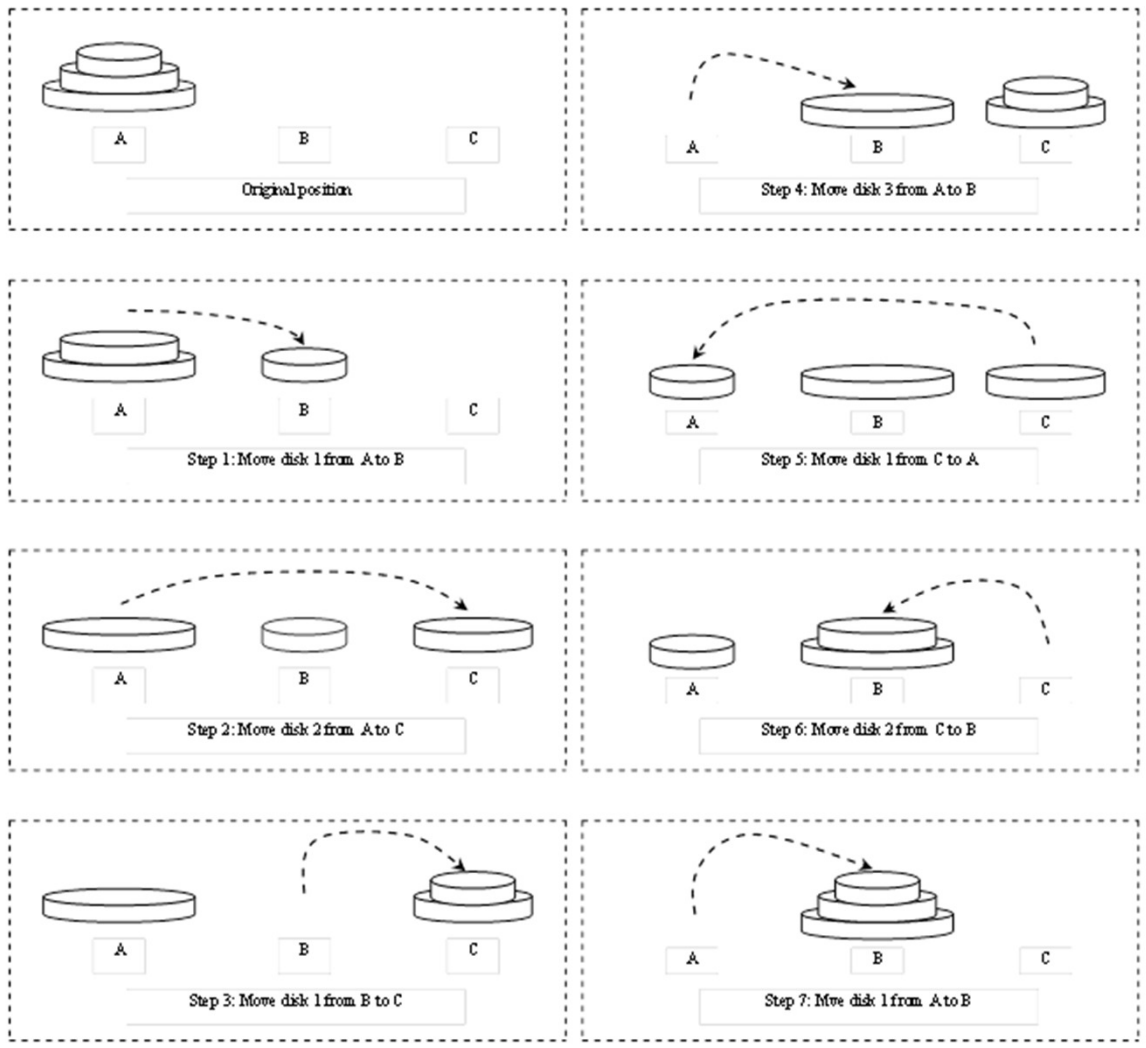
Υπερφόρτωση (Overloading)

Παράδειγμα 5: Μέγεθος Φακέλου (Directory Size)

```
import java.io.File;
import java.util.Scanner;

public class DirectorySize {
  public static void main(String[] args) {
    // Prompt the user to enter a directory or a file
    System.out.print("Enter a directory or a file: ");
    Scanner input = new Scanner(System.in);
    String directory = input.nextLine();

    // Display the size
    System.out.println(getSize(new File(directory)) + " bytes");
  }

  public static long getSize(File file) {
    long size = 0; // Store the total size of all files

    if (file.isDirectory()) {
      File[] files = file.listFiles(); // All files and subdirectories
      for (int i = 0; i < files.length; i++) {
        size += getSize(files[i]); // Recursive call
      }
    }
    else { // Base case
      size += file.length();
    }
    return size;
  }
}
```

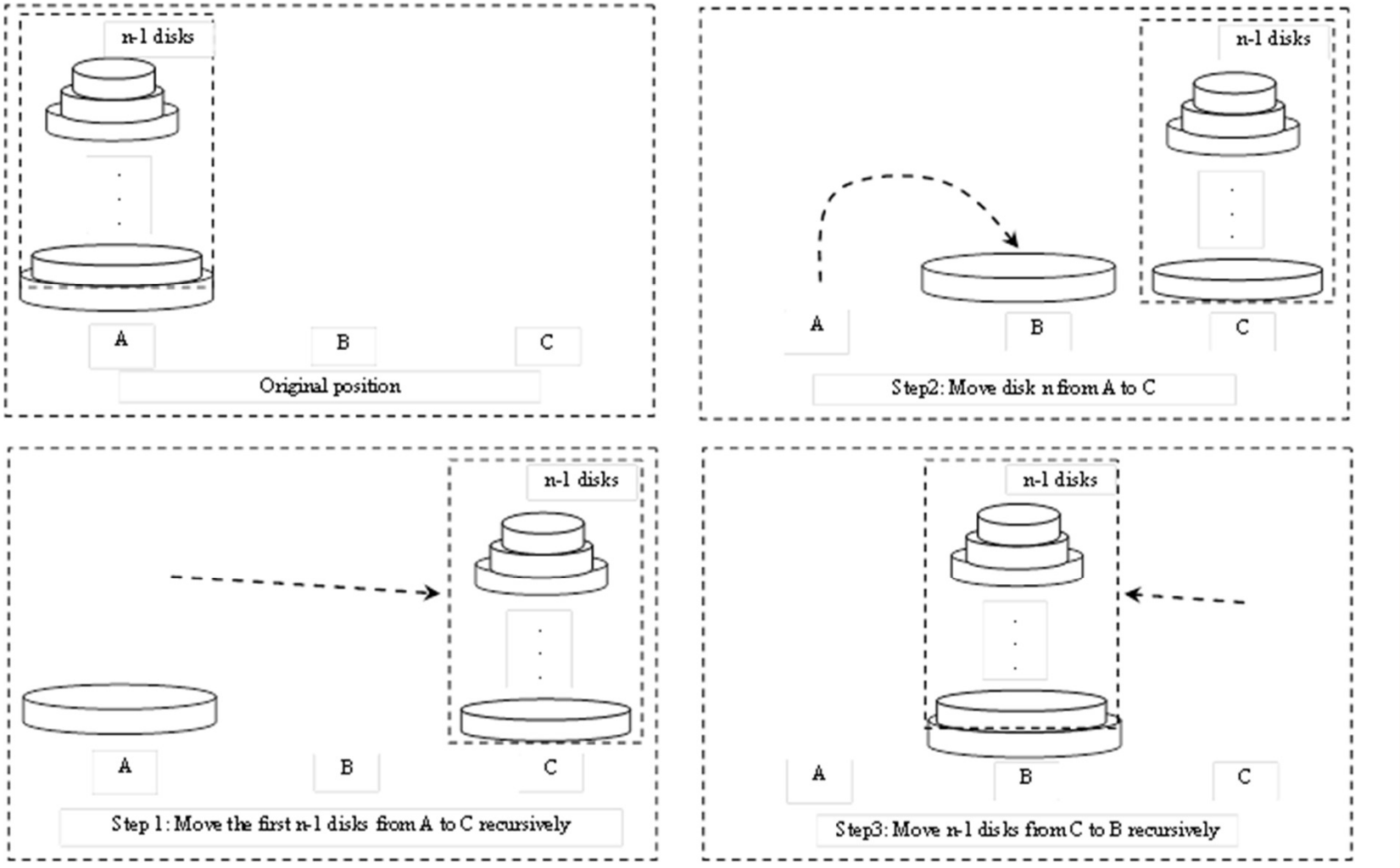
Παράδειγμα 6: Πύργοι του Hanoi

- Υπάρχουν n δίσκοι με αριθμούς 1, 2, 3, . . ., *n*, και τρεις πύργοι με ονόματα A, B, και C.
- Κανένας δίσκος δεν μπορεί να είναι πάνω σε μικρότερο δίσκο ανά πάσα στιγμή.
- Αρχικά, όλοι οι δίσκοι τοποθετούνται στον πύργο A.
- Μόνο ένας δίσκος μπορεί να μετακινηθεί ανά πάσα στιγμή και πρέπει να είναι ο πιο ψηλός δίσκος σε ένα πύργο



Παράδειγμα 6: Πύργοι του Hanoi (συν.)

- Το πρόβλημα των πύργων του Hanoi μπορεί να διασπαστεί σε 3 υπο-προβλήματα



Παράδειγμα 6: Πύργοι του Hanoi (συν.)

```
import java.util.Scanner;
public class TowersOfHanoi {

  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter number of disks: ");
    int n = input.nextInt();
    // Find the solution recursively
    System.out.println("The moves are:");
    moveDisks(n, 'A', 'B', 'C');
  }

  /** The method for finding the solution to move n disks
      from fromTower to toTower with auxTower */
  public static void moveDisks(int n, char fromTower,
      char toTower, char auxTower) {
    if (n == 1) // Stopping condition
      System.out.println("Move disk " + n + " from " +
        fromTower + " to " + toTower);
    else {
      moveDisks(n - 1, fromTower, auxTower, toTower);
      System.out.println("Move disk " + n + " from " +
        fromTower + " to " + toTower);
      moveDisks(n - 1, auxTower, toTower, fromTower);
    }
  }
}
```

Αφαίρεση της Αναδρομής

- Η χρήση της αναδρομής επιτρέπει την επίλυση πολύπλοκων προβλημάτων με **άμεσο και σαφή τρόπο**. Συχνά όμως **υστερεί από άποψη αποδοτικότητας.**

- Η **αφαίρεση της αναδρομής από μια συνάρτηση**, δηλαδή, η μετατροπή της σε επαναληπτική συνάρτηση χωρίς αναδρομή, είναι δυνατή (κάτω από κάποιες συνθήκες).

- Συχνά προϋποθέτει τη χρήση κάποιων βοηθητικών δομών (π.χ. στοίβα ή ουρά.

- Επίσης στις περισσότερες περιπτώσεις μπορούμε να επιλύσουμε μια αναδρομική εξίσωση και να βρούμε την λύση της σε κλειστή μορφή (με την χρήση Αναδρομικών Σχέσεων – Recurrence Relations).

  Π.χ. η λύση της εξίσωσης Fibonacci είναι :

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$