Αναδρομή

Σε αυτήν την (βοηθητική) ενότητα θα μελετηθούν τα εξής :

Η έννοια της αναδρομής

Υλοποίηση και αποδοτικότητα

Αφαίρεση της αναδρομής

Αναδρομή

- Βασική έννοια στα Μαθηματικά και στην Πληροφορική.
- Παράδειγμα: αναδρομικός ορισμός του παραγοντικού ενός μη αρνητικού ακεραίου:

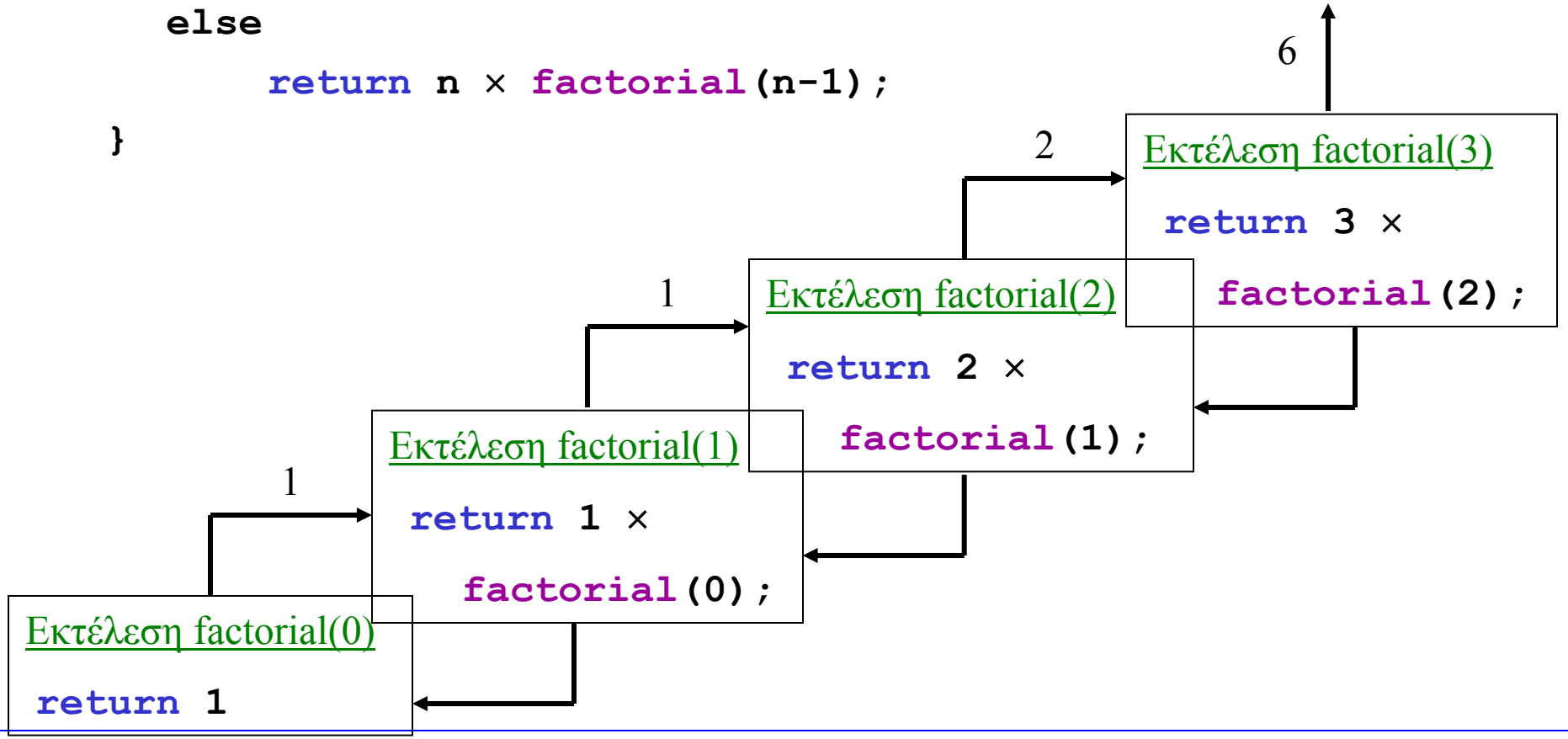
  $0! = 1$

  $n! = n \times (n-1)!$, $n > 0$

- Στην πληροφορική η αναδρομή χρησιμοποιείται σαν εργαλείο για τον *ορισμό ενός ΑΤΔ*, σαν *μέθοδος σχεδιασμού αλγορίθμων* και σαν *τεχνική προγραμματισμού*.
- Στον προγραμματισμό η αναδρομή εμφανίζεται με την κλήση ενός υποπρογράμματος από τον εαυτό του. Ένα αναδρομικό υποπρόγραμμα αποτελείται από:
  - ένα βήμα διακοπής, 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 αναδρομικής διαδικασίας

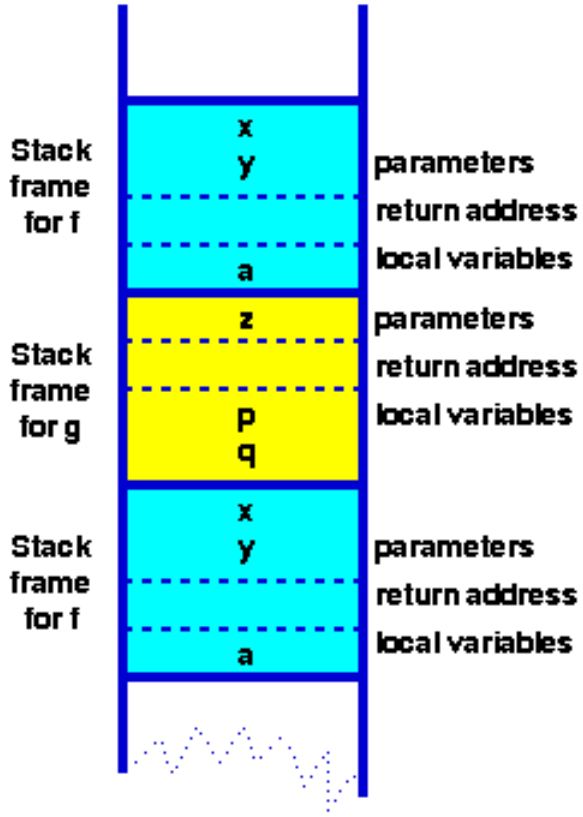
```
int factorial ( int n ) {
   if n == 0
        return 1;
   else
        return n × factorial(n-1);
}
```



Υλοποίηση αναδρομής

- Σε κάθε κλήση οποιασδήποτε συνάρτησης ένα σύνολο από λέξεις (stack frame) φυλάσσεται σε μια στοίβα, 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).

Υλοποίηση αναδρομής




```
int f(int x, int y) {
  int a;
  if ( term_cond ) return ...;
  a = .....;
  return g(a);
}

int g(int z) {
  int p,q;
  p = ...; q = ...;
  return f(p,q);
}
```

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).

Αναδρομή σε δένδρα

Να γράψετε αναδρομική διαδικασία η οποία να τυπώνει τα στοιχεία ενός ΔΔΑ σε αύξουσα σειρά.

- Προφανώς ο αλγόριθμος έχει ως εξής:

  Αν το δένδρο δεν είναι κενό, τύπωσε

  1. αναδρομικά τα στοιχεία του αριστερού υποδένδρου,
  2. το στοιχείο της ρίζας και
  3. αναδρομικά, τα στοιχεία του δεξιού υποδένδρου.

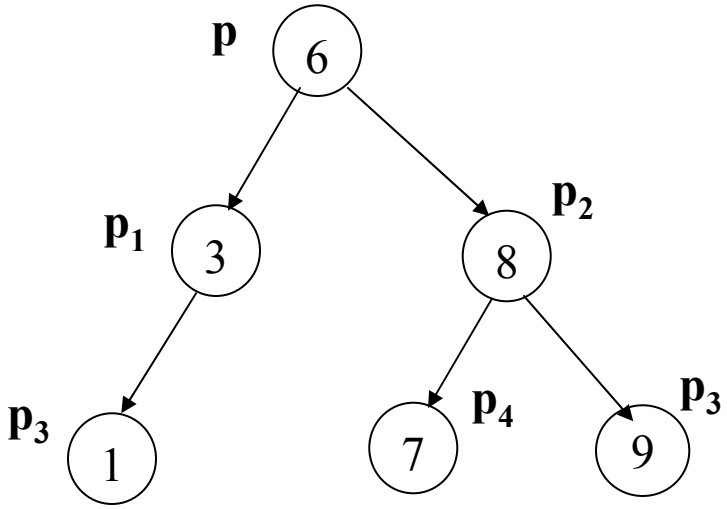
- Χρησιμοποιούμε τη δομή

```
typedef struct NODE{
   int              val;
   struct NODE      *left;
   struct NODE      *right;
} node;
```

και υποθέτουμε πως ένα δένδρο είναι υλοποιημένο ως δείκτης στη ρίζα, δηλαδή, έχει τύπο *node.

Η διαδικασία

```
Increasing(Node *q){
   if (q != null)
        Increasing(q -> left);
        print q -> val;
        Increasing(q -> right);
   }
```



Ορθότητα

Η ορθότητα της διαδικασίας αποδεικνύεται με επαγωγή.

Έστω Π(n) η πρόταση: Η διαδικασία Increasing(p), όπου p δείκτης σε δένδρο με n κόμβους, τυπώνει τα στοιχεία του δένδρου σε αύξουσα σειρά.

- Βασική Περίπτωση: Για n = 0, p = NULL, και η διαδικασία τερματίζει.
- Υπόθεση της Επαγωγής: Ας υποθέσουμε ότι η πρόταση είναι ορθή για κάθε n<k.
- Βήμα της επαγωγής: Για n = k > 0 η διαδικασία εκτελεί

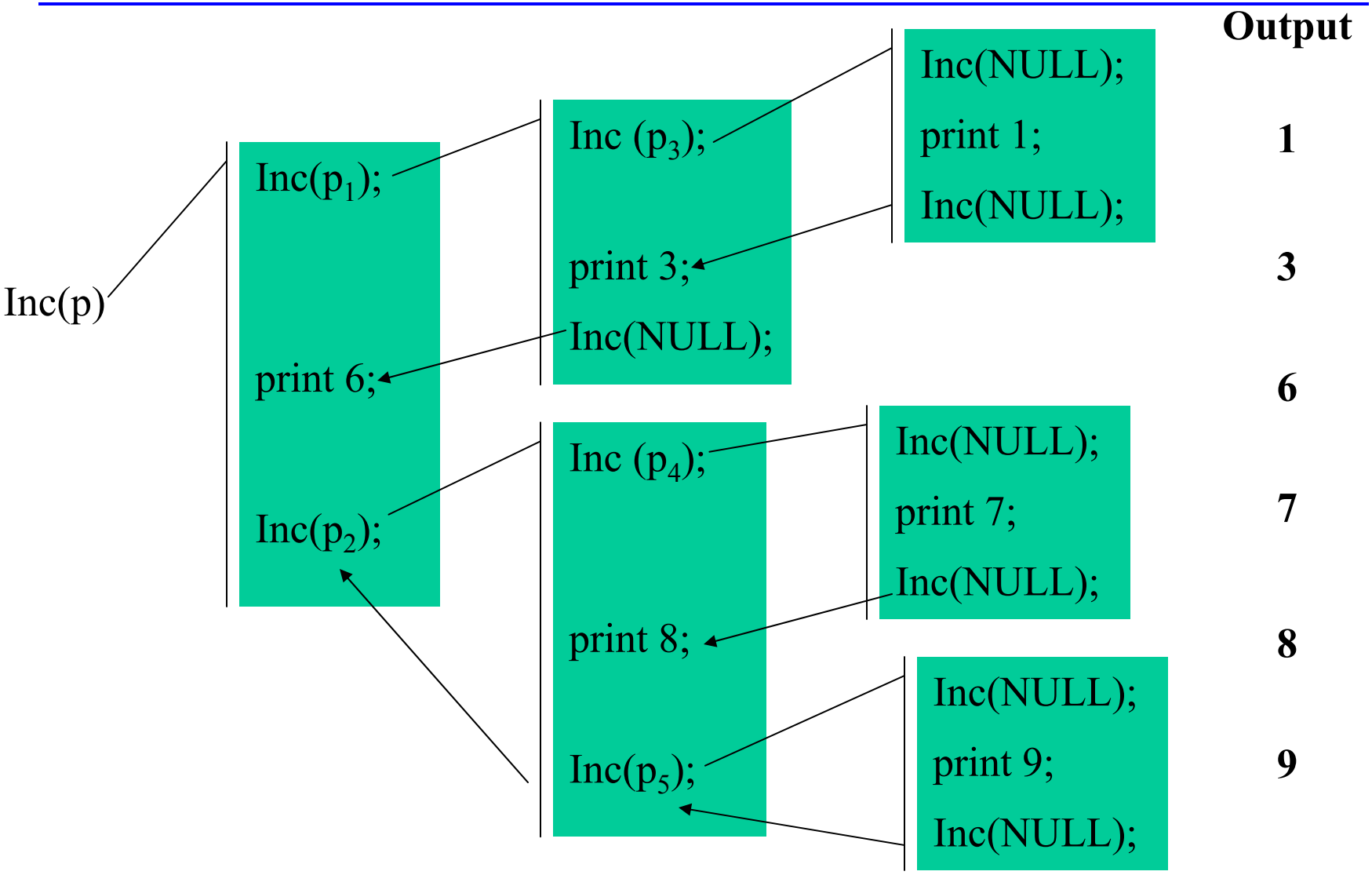
  `Increasing(p->left); print p->val; Increasing(p->right)`

  Προφανώς τα δένδρα που δείχνονται από τους δείκτες **`p->left`** και **`p->right`** έχουν λιγότερους από k κόμβους. Επομένως, από την υπόθεση της επαγωγής, οι `Increasing(p->left)` και `Increasing(p->right)` θα τυπώσουν τα στοιχεία τους σε αύξουσα σειρά. Αφού επίσης, ισχύει ότι

  στοιχεία του δένδρου p->left < p->val < στοιχεία του δένδρου p->right

  το ζητούμενο έπεται.

Παράδειγμα της εκτέλεσης



Αφαίρεση της αναδρομής

Να γράψετε μη-αναδρομική διαδικασία η οποία να τυπώνει τα στοιχεία ενός ΔΔΑ σε αύξουσα σειρά. Για να το πετύχετε μπορείτε να χρησιμοποιήσετε στοίβες.

- Η βασική ιδέα είναι η εξής:

  Εφόσον το δένδρο δεν είναι κενό προχωρούμε προς τα αριστερά φυλάγοντας τους κόμβους από τους οποίους περνούμε σε μία στοίβα.

  Όταν δεν μπορούμε να προχωρήσουμε άλλο ανασύρουμε τον κόμβο κορυφής της στοίβας, τυπώνουμε το στοιχείο του, και επαναλαμβάνουμε την ίδια διαδικασία στο δεξιό του παιδί.

- Υποθέτουμε την ύπαρξη υλοποίησης στοίβας και συγκεκριμένα των πράξεων, `MakeEmpty`, `IsEmpty`, `Pop` και `Push`.

Η μη-αναδρομική διαδικασία

```
Increasing2(node *p){
   stack S;
   MakeEmpty(S);
   while (p != NULL OR !IsEmpty(S))
       if (p != NULL)
              Push(p,S);
              p = p ->left;
       else
              p = Pop(S);
              print p->val;
              p = p->right;
}
```