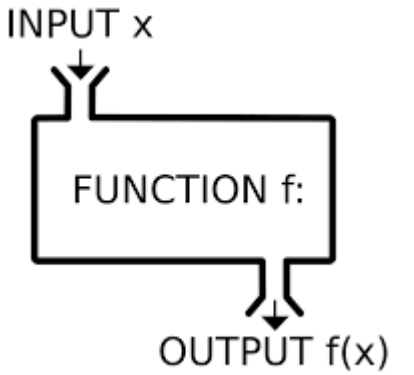
Δομημένος προγραμματισμός

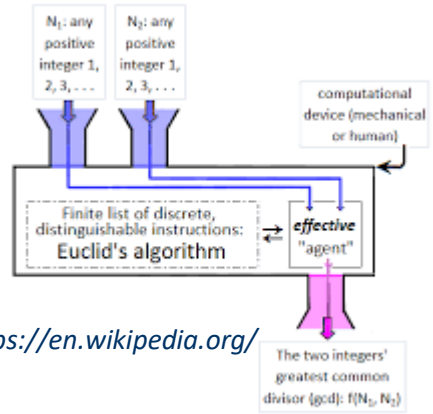


https://commons.wikimedia.org/

Οι συναρτήσεις στη C

Περίγραμμα της ύλης που θα καλυφθεί

- Οι συναρτήσεις στη C, γενικά
- Ορισμός μιας συνάρτησης
- Κλήση συνάρτησης
- Δήλωση συνάρτησης
- Μεταβίβαση δεδομένων (κατ' αξία και κατ' αναφορά)
- Εμβέλεια μεταβλητών
- Κατηγορίες μνήμης
- Οι πίνακες ως ορίσματα συναρτήσεων
- Αναδρομικότητα



https://en.wikipedia.org/

https://pixabay.com/

Οι συναρτήσεις – τι είναι

- Πρόκειται για *ανεξάρτητα τμήματα* ενός προγράμματος (υπο-προγράμματα) που επιτελούν συγκεκριμένες εργασίες.
- Καλούνται από το κυρίως πρόγραμμα ή από άλλες συναρτήσεις (μπορούν να καλέσουν ακόμα και τον εαυτό τους) και παράγουν, όπου είναι σκόπιμο, αποτελέσματα με χρήση δεδομένων.
- Η χρήση συναρτήσεων μας δίνει τη δυνατότητα να επιμερίσουμε ένα πολύπλοκο πρόγραμμα σε μικρότερα *αυτόνομα τμήματα κώδικα* (*modules*), που αναλαμβάνουν να λύσουν μεμονωμένα επί μέρους πρόβλήματα.
- Οι συναρτήσεις είναι δομικά στοιχεία της C (τα προγράμματα της είναι και αποτελούνται από συναρτήσεις).
- Όλα τα προγράμματα που έχουμε δει έως τώρα περιέχουν τουλάχιστον μια συνάρτηση: την *main()* που είναι η κύρια συνάρτηση σε ένα πρόγραμμα.

Οι συναρτήσεις – που βρίσκονται

- Υπάρχουν έτοιμες (προ-μεταγλωττισμένες ή όχι) που περιλαμβάνονται σε βιβλιοθήκες, οι οποίες ενσωματώνονται με κατάλληλες εντολές στο κυρίως πρόγραμμα (*#include*).
- Μέσα στο κυρίως πρόγραμμα όπου δηλώνονται και συντάσσονται σαν *ξεχωριστές ενότητες* (δημιουργούνται από τους προγραμματιστές).
- Επίσης, μπορεί να βρίσκονται και σε αρχεία διαφορετικά από αυτό του κυρίως προγράμματος.
- Σε κάθε περίπτωση είναι μέρος του κυρίως προγράμματος και μεταγλωττίζονται μαζί με αυτό (ή ενσωματώνονται στο εκτελέσιμο, αν είναι ήδη μεταγλωττισμένες).
- Η κλήση τους γίνεται με το όνομα τους, που είναι *μοναδικό*.
- Είναι δυνατόν να κληθούν πολλές φορές και με διαφορετικά δεδομένα.

Οι συναρτήσεις – πλεονεκτήματα

- Τα επιμέρους (υπο) προβλήματα διαχειρίζονται πιο εύκολα (μικρότερα προβλήματα --> πιο εύκολα προβλήματα).
- Ευανάγνωστα προγράμματα.
- Αποφεύγουμε την επανάληψη κώδικα.
- Μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις που δημιουργούμε και σε άλλα προγράμματα (επαναχρησιμοποίηση).
- *Αφαίρεση* (κρύβουν λεπτομέρειες που δεν είναι πάντα απαραίτητες).
- Αν διαγραφούν από τον κώδικα, απενεργοποιείται μόνο η συγκεκριμένη λειτουργία που κάνουν.
- Μπορούν να μεταγλωττιστούν ανεξάρτητα.
- Μεγαλύτερη ευκολία στον έλεγχο και την διόρθωση του κώδικα.
- Μειωμένο κόστος ανάπτυξης.
- Η συντήρηση (μετατροπές/βελτιώσεις) των προγραμμάτων καθίσταται πιο εύκολη.
- Εφαρμογή του *δομημένου προγραμματισμού*.

Συναρτήσεις - χρήση

- Κάθε συνάρτηση μπορεί να δεχθεί δεδομένα (εισόδου) και να επιστρέψει αποτελέσματα (δη. δεδομένα εξόδου) στο πρόγραμμα που την καλεί.

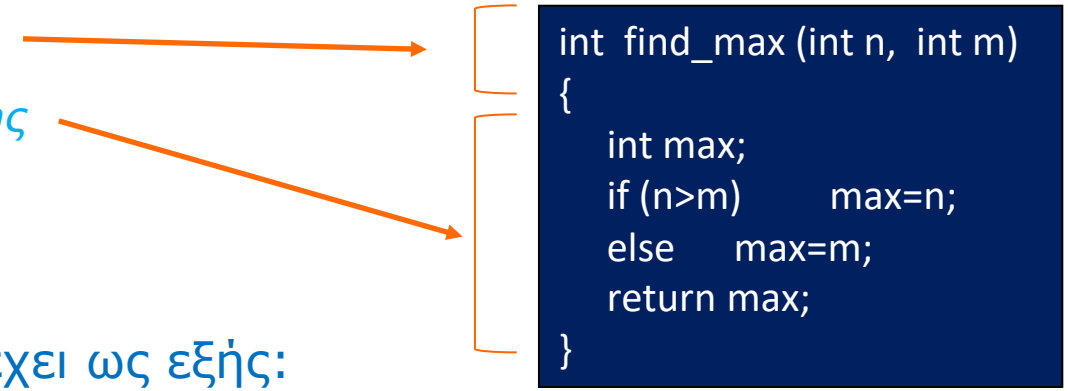
- Για την ανταλλαγή των δεδομένων χρησιμοποιούνται (όχι πάντα) *παράμετροι* -μεταβλητές ή διευθύνσεις αυτών)

- Υπάρχουν διάφορα είδη συναρτήσεων:
  - *Αυτές που δεν χρησιμοποιούν παραμέτρους και δεν επιστρέφουν κάτι.*
  - *Αυτές που δεν χρησιμοποιούν παραμέτρους και επιστρέφουν κάτι.*
  - *Αυτές που χρησιμοποιούν παραμέτρους και δεν επιστρέφουν κάτι.*
  - *Αυτές που χρησιμοποιούν παραμέτρους και επιστρέφουν κάτι.*

- Χαρακτηριστικά:
  - *Η είσοδος δεδομένων γίνεται από ένα μόνο σημείο.*
  - *Όταν καλείται μια συνάρτηση σταματά η εκτέλεση του κυρίως προγράμματος (αναμονή) και εκτελείται μόνο η συνάρτηση.*
  - *Όταν ολοκληρωθεί η εκτέλεση της συνάρτησης, ο έλεγχος επιστρέφει στο πρόγραμμα που έκανε την κλήση της*.

Ορισμός μιας συνάρτησης στη C

- Η γενική δομή ορισμού μιας συνάρτησης σε C περιλαμβάνει:
  - *την επικεφαλίδα και*
  - *το σώμα της συνάρτησης*



- Η γενική δομή ορισμού έχει ως εξής:

τύπος_επιστροφής όνομα_συνάρτησης (τυπικοί παράμετροι)

{

Δηλώσεις μεταβλητών; — *Τοπικές μεταβλητές*

Εντολές συνάρτησης; — *Εντολές που επιτελούν συγκεκριμένη εργασία*

return <εκφραση> ; — *Τιμή που επιστρέφει η συνάρτηση*

}

Ο ορισμός μιας συνάρτησης ΔΕΝ μπορεί να γίνει μέσα σε άλλη συνάρτηση. Μπορεί να γίνει πριν ή μετά τη *main()*.

Ορισμός μιας συνάρτησης στη C

τύπος_επιστροφής όνομα_συνάρτησης (τυπικοί παράμετροι)

- Ο *τυπος_επιστροφής* δηλώνει τον τύπο της τιμής (το πολύ μια) που επιστρέφει η συνάρτηση με την εντολή *return*.

  Αν η συνάρτηση δεν επιστρέφει κάποια τιμή, τότε ο τύπος επιστροφής είναι τύπου void.

  Το όνομα_συνάρτησης είναι μοναδικό.

- Οι *τυπικοί παράμετροι* (μεταβλητές) είναι ουσιαστικά ο μηχανισμός επικοινωνίας της συνάρτησης με τη συνάρτηση (η main ή κάποια άλλη) που την καλεί. Είναι μια λίστα της μορφής:

  τυπος παραμετρος-1, τύπος παράμετρος-2, … τυπος παραμετρος-n

  Αν δεν υπάρχουν παράμετροι χρησιμοποιείται ο τύπος void.

- Παράδειγμα ορισμού:

Το σώμα της συνάρτησης

- Πρόκειται για κώδικα (εντολές, συναρτήσεις) που περιέχεται σε άγκιστρα και περιλαμβάνει δηλώσεις μεταβλητών και εντολές.
- Εκτελείται όταν κληθεί η συνάρτηση. Μόνο ο κώδικας της *main()* εκτελείται αυτόματα.
- Οι μεταβλητές που δηλώνονται ισχύουν μόνο μέσα στο σώμα της συνάρτησης. Το ίδιο ισχύει και για τις παραμέτρους.
- Η εκτέλεση τερματίζεται με εντολές *return*, οι οποίες προκαλούν άμεση έξοδο από τη συνάρτηση.
- Αν δεν υπάρχει εντολή *return*, η έξοδος προκαλείται όταν εκτελεστεί η τελευταία εντολή.

```
int  find_max (int n,  int m)
{
    int max;
    if (n>m)        max=n;
    else      max=m;
    return max;   // επιστροφή τιμής
}
```

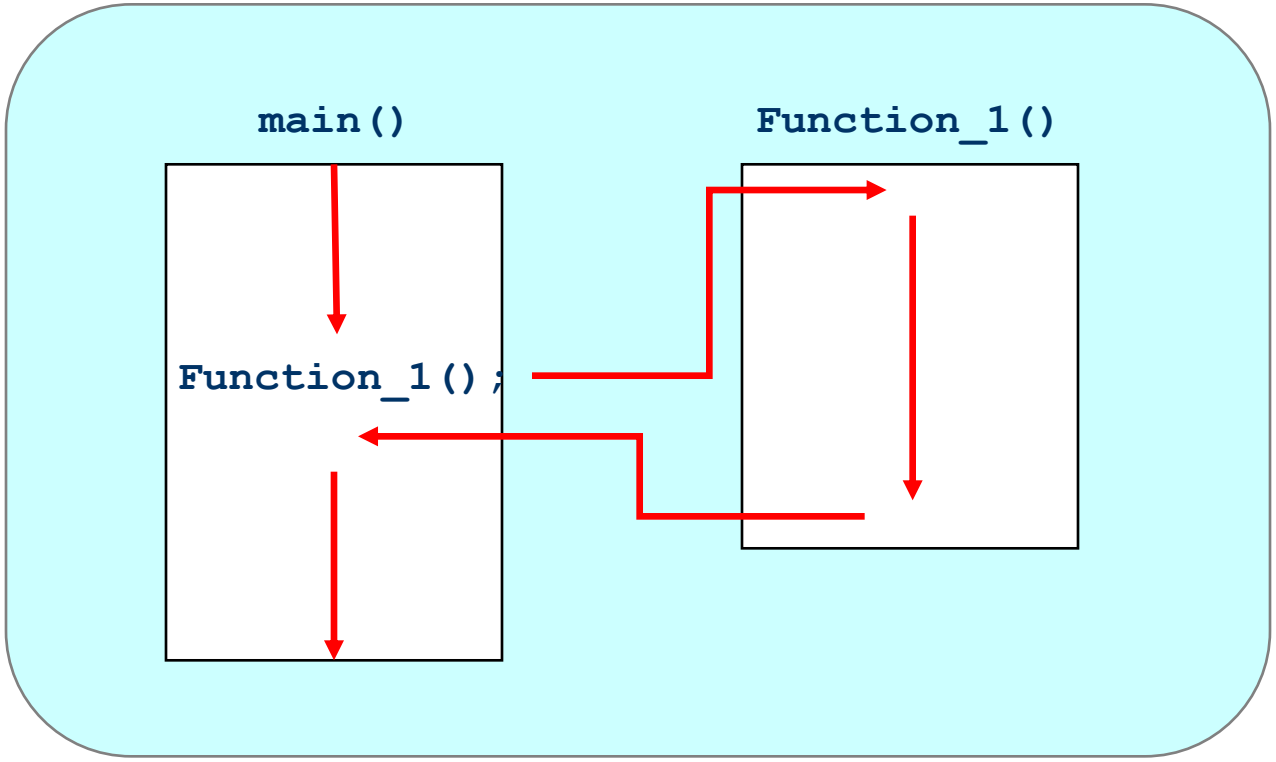
Κλήση μιας συνάρτησης

- Με τη κλήση μιας συνάρτησης, ο έλεγχος του προγράμματος μεταφέρεται στο *σώμα* της συνάρτησης.
- Όταν ολοκληρωθεί η εκτέλεση της συνάρτησης, ο έλεγχος του προγράμματος επιστρέφει στη συνάρτηση που έκανε τη κλήση.
- Μια συνάρτηση καλείται με το όνομα της και τις τιμές που δίνουμε στις μεταβλητές των τυπικών παραμέτρων.
- Αυτές οι τιμές ονομάζονται *ορίσματα* (πραγματικοί παράμετροι) και πρέπει να είναι του ίδιου τύπου με τις μεταβλητές των τυπικών παραμέτρων.
- Όταν καλείται μια συνάρτηση πρέπει να είναι γνωστός ο τύπος επιστροφής:

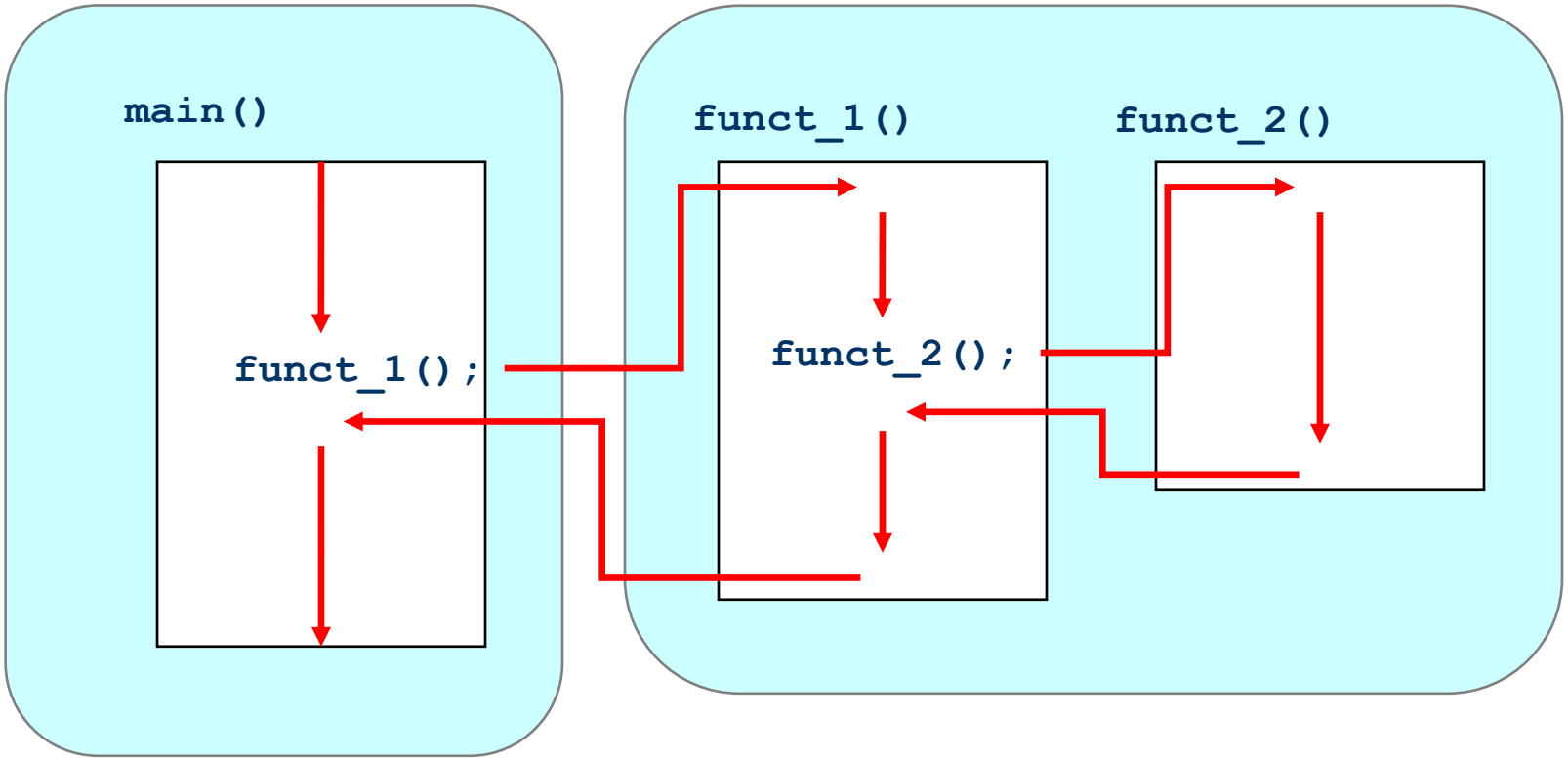
max: ακέραιος
Τύπος επιστροφής:int

```
max = find_max (a, b);

max = find_max (5, 6);
```

Ροή προγράμματος όταν καλείται μια συνάρτηση



Ροή προγράμματος με κλήση συναρτήσεων



Παράδειγμα ορισμού και κλήσης συνάρτησης

```
#include <stdio.h>
int find_max (int n,  int m) { // Ορισμός της συνάρτησης find_max()
    int max;  //τοπική  μεταβλητή
     if(n>m)  max=n;
     else       max=m;
     return max;   // επιστροφή τιμής
}
main()   // κυρίως πρόγραμμα
{
    int a,b, max;
    printf ("a:"); scanf("%d",&a);
    printf ("b:"); scanf("%d",&b);
    max= find_max (a,b);  // κλήση της συνάρτησης find_max()
    printf("Max of %d and %d is %d\n", a, b, max);
}
```

εμβέλεια μέσα στη συνάρτηση

Μεταβλητές της main().
max διαφορετική από αυτή της συνάρτησης

Δήλωση μιας συνάρτησης (πρωτότυπο/αρχέτυπο)

- Απαιτείται αν πρόκειται να χρησιμοποιήσουμε μια συνάρτηση *πριν* τον ορισμό της.
- Αυτό συμβαίνει όταν ο ορισμός γίνεται μετά τη συνάρτηση *main()* ή βρίσκεται σε διαφορετικό αρχείο ή η συνάρτηση είναι ήδη μεταγλωττισμένη.
- Η δήλωση του πρωτοτύπου μιας συνάρτησης προσδιορίζει τον τύπο επιστροφής της συνάρτησης, το όνομα της και τους τύπους των παραμέτρων της:

```
int  find_max (int, int) ;
```

Τελειώνει με το ;

Δεν χρειάζονται ονόματα παραμέτρων

Δεν υπάρχει σώμα!

- Μια καλή πρακτική είναι οι δηλώσεις των συναρτήσεων να γίνονται πριν την *main()*. Αν βρίσκονται σε άλλο αρχείο θα πρέπει να συμπεριλαμβάνονται στο πρόγραμμα με την οδηγία *#include*.

Τα πρωτότυπα συναρτήσεων

- Δεν είναι απαραίτητα, ούτε υποχρεωτικά αλλά είναι πολύ χρήσιμα.
- Οι πρώτες εκδόσεις της C δεν τα υποστήριζαν.
- Ένα πρόγραμμα χωρίς πρωτότυπα δεν θεωρείται συντακτικά λανθασμένο.
- Ένα πρωτότυπο συνάρτησης δηλώνει:
  - *το τύπο της τιμής που επιστρέφει η συνάρτηση,*
  - *τον αριθμό παραμέτρων της συνάρτησης και*
  - *τους τύπους δεδομένων των παραμέτρων της συνάρτησης.*
- Όλα αυτά είναι πολύ χρήσιμα στη μεταγλώττιση, διότι
  - *δίνουν χρήσιμη πληροφόρηση στο compiler, με αποτέλεσμα ν' αποφεύγονται πιθανά λάθη.*
- Όταν δεν υπάρχουν παράμετροι στα πρωτότυπα πρέπει να χρησιμοποιείται ο τύπος *void*.

Παράδειγμα δήλωσης, κλήσης και ορισμού συνάρτησης

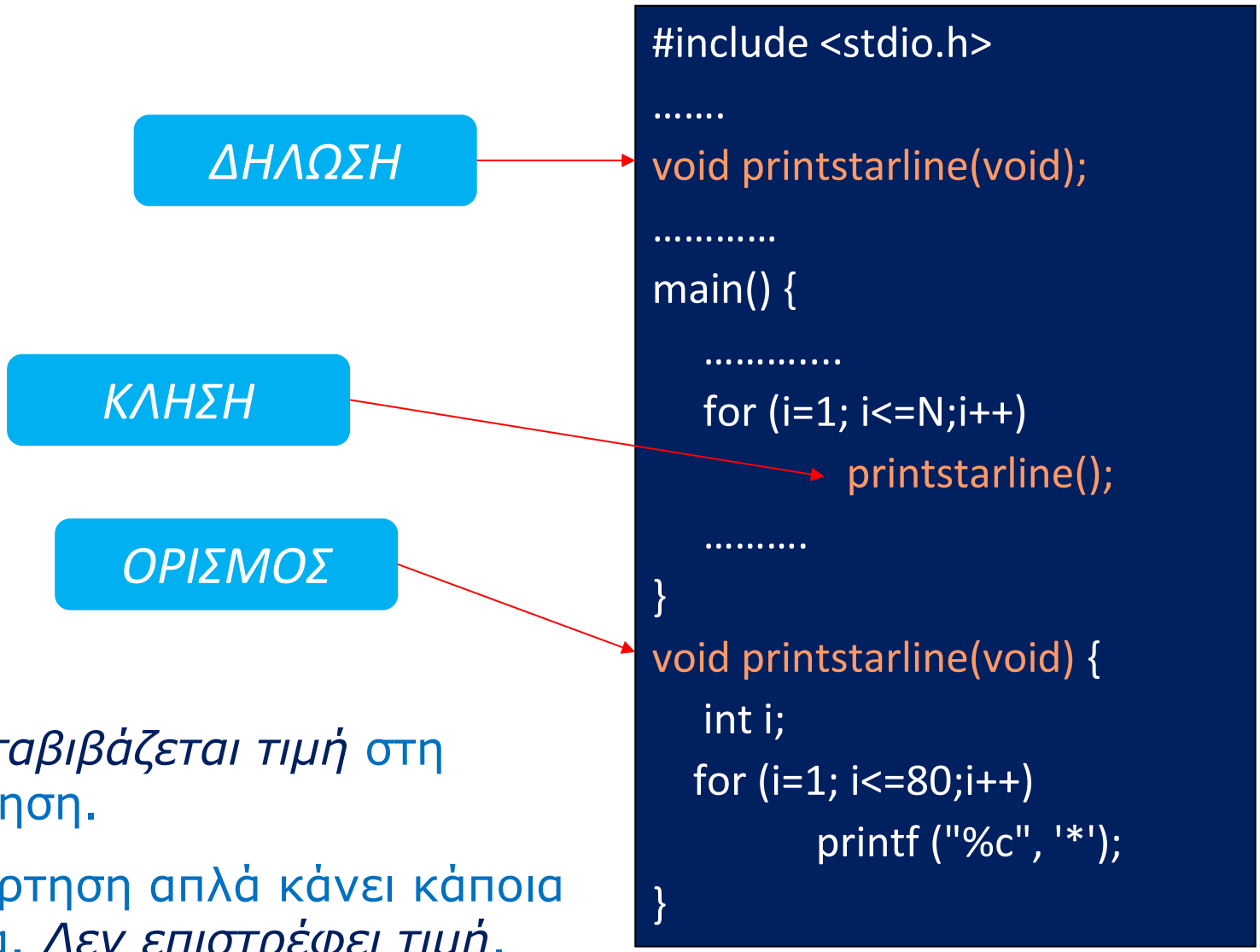
```
#include <stdio.h>
int find_max(int, int); //Δήλωση της συνάρτησης - πρωτότυπο
main() {
        int a,b, max;
        printf ("a:"); scanf("%d",&a);
        printf ("b:"); scanf("%d",&b);
        max= find_max(a,b); // Κλήση της συνάρτησης
        printf("Max of %d and %d is %d\n", a, b, max);
}

int find_max (int n, int m)  {   //Ορισμός συνάρτησης
         int max;
         if(n>m)   max=n;
         else   max=m;
         return max;
}
```

printf("Max of %d and %d is %d\n", a, b, find_max(a,b));

Αλλά και:

```
int find_max (int n, int m) {
        if(n>m) return n;
        else    return m;
}
```

Συναρτήσεις ΧΩΡΙΣ παραμέτρους και ΧΩΡΙΣ επιστροφή



- *Δεν μεταβιβάζεται τιμή* στη συνάρτηση.
- Η συνάρτηση απλά κάνει κάποια εργασία. *Δεν επιστρέφει τιμή*.

Συναρτήσεις ΧΩΡΙΣ παραμέτρους ΜΕ επιστροφή

ΔΗΛΩΣΗ

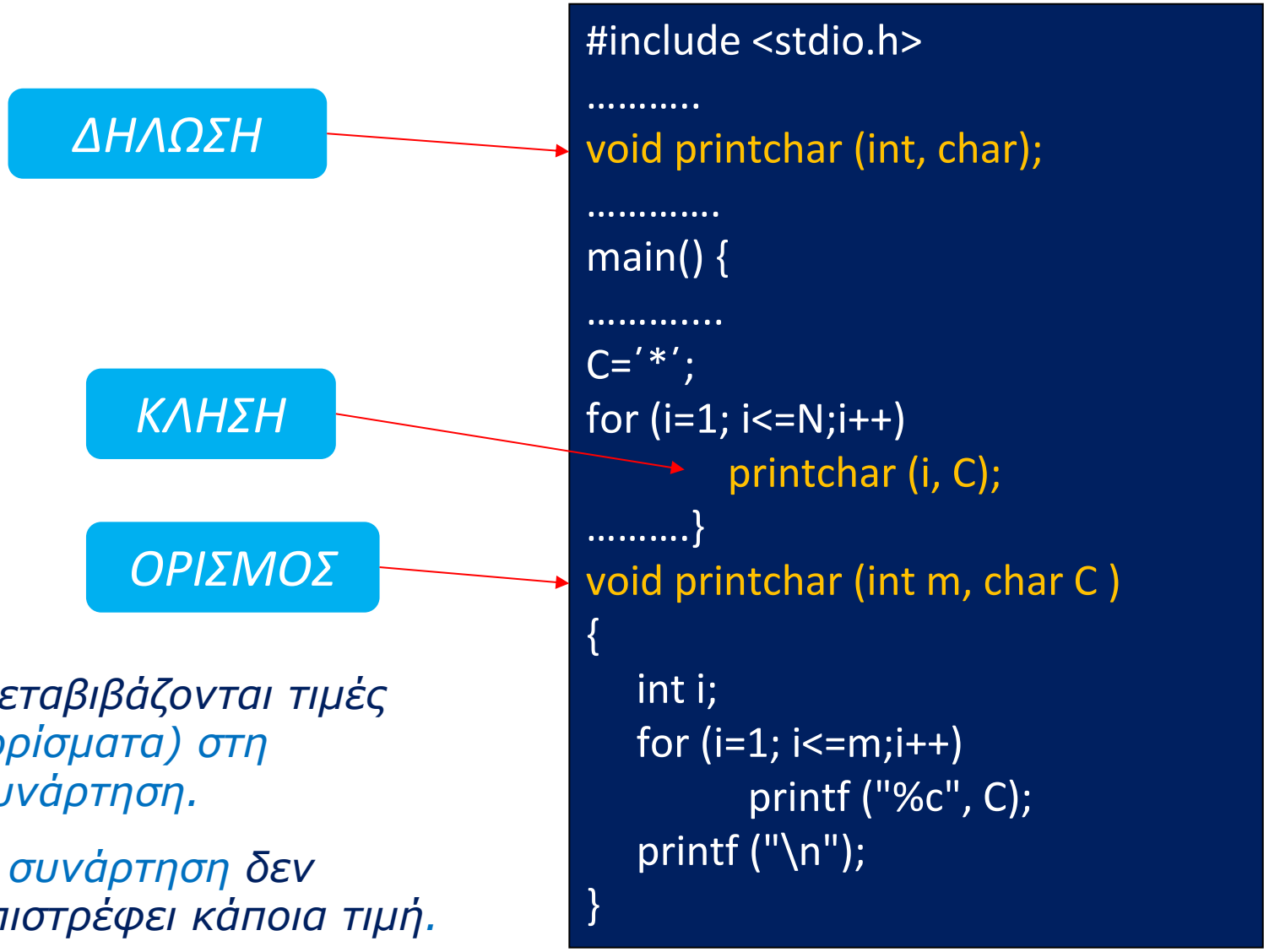
ΚΛΗΣΗ

ΟΡΙΣΜΟΣ

```
float findsum(void);

main() {
            ……..
            athroisma = findsum();
            MO = athroisma/7;
            ……..
}
float findsum(void) {
     int i;
     float a, sum=0;
     printf ("You should give 7 numbers\n");
     for (i=1; i<=7;i++) {
          printf("Please give %d number:",i);
          scanf("%f", &a);
          sum=sum+a;
      }
     return sum;
}
```

- *Δεν μεταβιβάζεται τιμή στη συνάρτηση.*
- Η συνάρτηση *επιστρέφει κάποια τιμή.*

Συναρτήσεις ΜΕ παραμέτρους ΧΩΡΙΣ επιστροφή



- *Μεταβιβάζονται τιμές (ορίσματα) στη συνάρτηση.*
- *Η συνάρτηση δεν επιστρέφει κάποια τιμή.*

Συναρτήσεις ΜΕ παραμέτρους και ΜΕ επιστροφή

ΔΗΛΩΣΗ

ΚΛΗΣΗ

ΟΡΙΣΜΟΣ

```
#include <stdio.h>
#define MAXN 12
int compute_factorial (int);
main()
{
  int i;
  for (i=1 ; i <= MAXN ; i++)
      printf("%2d! = %d\n", i, compute_factorial(i));
}
int compute_factorial (int n)
{
  int i, factorial=1;
  for (i=1; i<=n; i++)
     factorial *= i;
  return factorial;
}
```

- *Μεταβιβάζονται τιμές* (ορίσματα) στη συνάρτηση.
- Η συνάρτηση *επιστρέφει κάποια τιμή.*

Μεταβίβαση δεδομένων (κατ' αξία και κατ' αναφορά)

Παράμετροι: οι μεταβλητές που εμφανίζονται στη δήλωση και στον ορισμό της συνάρτησης.

Ορίσματα: οι τιμές που περιέχονται στην κλήση της συνάρτησης.

- Μεταβίβαση δεδομένων *κατ' αξία* (by value), όπου:
  - *αντιγράφονται οι τιμές των ορισμάτων στις παραμέτρους που είναι τοπικές μεταβλητές της συνάρτησης και τυχόν αλλαγές δεν επηρεάζουν στις μεταβλητές της συνάρτησης που κάνει τη κλήση.*
- Μεταβίβαση δεδομένων *κατ' αναφορά* (by reference), όπου:
  - *δεν μεταβιβάζονται οι αξίες, αλλά οι διευθύνσεις των μεταβλητών που περιέχουν τις τιμές, έτσι η συνάρτηση έχει πρόσβαση στις διευθύνσεις των μεταβλητών και μπορεί να κάνει αλλαγές στις τιμές.*
- Καλώντας κάποια συνάρτηση κατ' αναφορά έχουμε τη δυνατότητα ν' αλλάξουμε πολλές τιμές μεταβλητών (άρα και να έχουμε επιστροφή πολλών τιμών).

Παράδειγμα κλήσεων κατ' αξία και αναφορά

```
#include <stdio.h>
void swap (int, int);
main()
{
    int x = 1, y = 2;
    printf("Before: x=%d, y=%d.\n", x, y);
    swap (x, y);  // αξίες - τιμές
    printf("After: x=%d, y=%d. \n", x, y);
}
void  swap (int u, int v)
{
     int temp;
     temp = u;
     u = v;
     v = temp;
}
```

ΔΕΝ ΑΛΛΑΖΟΥΝ ΟΙ ΤΙΜΕΣ ΤΩΝ ΜΕΤΑΒΛΗΤΩΝ

```
#include <stdio.h>
void swap (int*, int*);
main()
{
    int x = 1, y = 2;
    printf("Before: x=%d, y=%d.\n", x, y);
    swap(&x, &y); //διευθύνσεις
    printf("After: x=%d, y=%d. \n", x, y);
}
void swap(int *u, int *v)
{
     int temp;
     temp = *u;
     *u = *v;
     *v = temp;
}
```

ΑΛΛΑΖΟΥΝ ΟΙ ΤΙΜΕΣ ΤΩΝ ΜΕΤΑΒΛΗΤΩΝ

Εμβέλεια μεταβλητών

- Εμβέλεια μιας μεταβλητής είναι το τμήμα του προγράμματος που η μεταβλητή ισχύει (μπορεί ν' αξιοποιηθεί).
- Αναφορικά με την εμβέλεια τους οι μεταβλητές κατηγοριοποιούνται σε:
  - *Τοπικές* (local) μεταβλητές και
  - *Καθολικές* (global) μεταβλητές που δηλώνονται εκτός συναρτήσεων και έχουν εμβέλεια σε όλο το πρόγραμμα.
- Η κατηγοριοποίηση αυτή εξαρτάται από το σημείο που δηλώνεται κάποια μεταβλητή:
  - *Οι τοπικές μεταβλητές δηλώνονται μέσα σε μια συνάρτηση (στο σώμα της) και έχουν εμβέλεια μόνο μέσα σε αυτή.*
  - *Οι καθολικές μεταβλητές δηλώνονται εκτός συναρτήσεων (συνήθως στη αρχή του αρχείου του κώδικα) και έχουν εμβέλεια σε όλο το πρόγραμμα.*

Παρατηρήσεις στη εμβέλεια των μεταβλητών - I

- Τοπικές μεταβλητές μπορούν να δηλωθούν και σε κάποιο τμήμα του κώδικα

  (δηλαδή, σε block που αρχίζει και τελειώνει με ***άγκιστρα****: { …. }).*

- Η τοπική μεταβλητή έχει εμβέλεια μόνο μέσα στο συγκεκριμένο τμήμα που έχει δηλωθεί.

- Η απόδοση αρχικών τιμών σε τοπικές μεταβλητές είναι μέριμνα του προγραμματιστή

  (δεν αποδίδονται αυτόματα αρχικές τιμές).

- Σε μια συνάρτηση εκτός των μεταβλητών που δηλώνονται μέσα στο σώμα της συνάρτησης, τοπικές μεταβλητές είναι και οι παράμετροι στον ορισμό της συνάρτησης.

- Οι παράμετροι δεν χρειάζονται αρχική τιμή, διότι παίρνουν τιμές όταν καλείται η συνάρτηση (από τις τιμές των ορισμάτων).

- Οι τοπικές μεταβλητές μπορούν να έχουν το ίδιο όνομα σε διαφορετικές συναρτήσεις. Είναι όμως διαφορετικές μεταβλητές.

Παρατηρήσεις στη εμβέλεια των μεταβλητών - ΙΙ

- Οι καθολικές μεταβλητές έχουν εμβέλεια σε όλο το πρόγραμμα και κάθε συνάρτηση του προγράμματος μπορεί να τις χρησιμοποιήσει.
- Στις καθολικές μεταβλητές *αποδίδεται αυτόματα η τιμή 0* (εκτός αν ο προγραμματιστής της αναθέσει κάποια άλλη αρχική τιμή).
- Οι τοπικές και οι καθολικές μεταβλητές μπορούν να έχουν το ίδιο όνομα, αλλά δεν παύουν να είναι διαφορετικές.
- Σε περίπτωση που έχουμε μια τοπική και μια καθολική μεταβλητή με το ίδιο όνομα, στο τμήμα (εμβέλεια) που έχει δηλωθεί η τοπική μεταβλητή, αυτή υπερισχύει της καθολικής.
- Η εμβέλεια μια καθολικής μεταβλητής ξεκινά από το σημείο που δηλώνεται αυτή και όλες οι συναρτήσεις που ορίζονται μετά από τη δήλωση της μπορούν να τη χρησιμοποιήσουν.

Παράδειγμα με καθολική μεταβλητή

```
#include <stdio.h>
int find_max(int, int);
main()
{
    int a,b, max;
    printf ("a:"); scanf("%d",&a);
    printf ("b:"); scanf("%d",&b);
    max= find_max(a,b);
    printf("Max is: %d\n",  max);
}
int find_max (int n, int m) {
        int max;
        if(n>m)   max=n;
        else   max=m;
        return max;
}
```

Όλες οι μεταβλητές είναι τοπικές

2 διαφορετικές max

```
#include <stdio.h>
int max;
void find_max(int, int);
main()
{
   int a,b;
   printf ("a:");scanf("%d",&a);
   printf ("b:");scanf("%d",&b);
   find_max(a,b);
   printf("Max is:%d\n",max);
}
void find_max(int n, int m)  {
        if(n>m)  max=n;
        else  max=m;
}
```

max: καθολική

a, b : τοπικές

n, m : τοπικές

Δίνουμε τιμές στη καθολική max

Χωρίς return!

Μεταβλητές static (τοπικές)

- Το χαρακτηριστικό γνώρισμα μιας *static τοπικής μεταβλητής* είναι ότι μπορεί να διατηρήσει τη τιμή της σε διαδοχικές κλήσεις της συνάρτησης.

static int i;

- Αυτό είναι χρήσιμο όταν θέλουμε να διατηρήσουμε τη τιμή μιας τοπικής μεταβλητής, ανάμεσα σε κλήσεις της συνάρτησης.

- Η *static* τοπική μεταβλητή παίρνει τιμή όταν καλείται για πρώτη φορά η συνάρτηση και διατηρεί τη τελευταία της τιμή στις επόμενες κλήσεις της συνάρτησης.

- Κρατά τη ίδια θέση μνήμης κατά τη διάρκεια της εκτέλεσης του προγράμματος.

- Ισχύει όμως μόνο μέσα στη εμβέλειας της (στο τμήμα που έχει δηλωθεί).

Παράδειγμα με static μεταβλητή

```
#include <stdio.h>
#define N 5
void func(void);
main() {
   int i;
   for (i=1; i<=N; i++)
                  func();
          }
void func()  {
          static int x = 1;
          int i;
          for (i=1;i<=x;i++)
                        printf("%c", '*');
          x = x + 1;
          printf("\n");
}
```

Δήλωση

Κλήση

Ορισμός

x: static

Η τιμή της **x** διατηρείται και στη επόμενη κλήση!

Αποτέλεσμα εκτέλεσης:

```
*
**
***
****
*****
```

Μεταβλητές extern (καθολικές)

- Χρησιμοποιούνται όταν η εμβέλεια μιας καθολικής μεταβλητής πρέπει να περιλαμβάνει και άλλα αρχεία εκτός του αρχείου του προγράμματος (ένα πρόγραμμα μπορεί ν' αποτελείται από πολλά αρχεία).

- Ουσιαστικά δηλώνεται ξανά μια καθολική μεταβλητή, η οποία έχει οριστεί σε κάποιο άλλο αρχείο του προγράμματος.

- Βρίσκει εφαρμογή σε μεγάλα προγράμματα που εκτείνονται σε πολλά αρχεία.

- Η εντολή:

  extern int global;

  ενημερώνει το μεταγλωττιστή ότι η μεταβλητή global *έχει οριστεί σε άλλο αρχείο του προγράμματος.*

- Όλα τα αρχεία του προγράμματος στα οποία δηλώνεται η εξωτερική (extern) μεταβλητή μπορούν να την επεξεργαστούν.

Έτοιμες (προ-μεταγλωττισμένες) συναρτήσεις της C

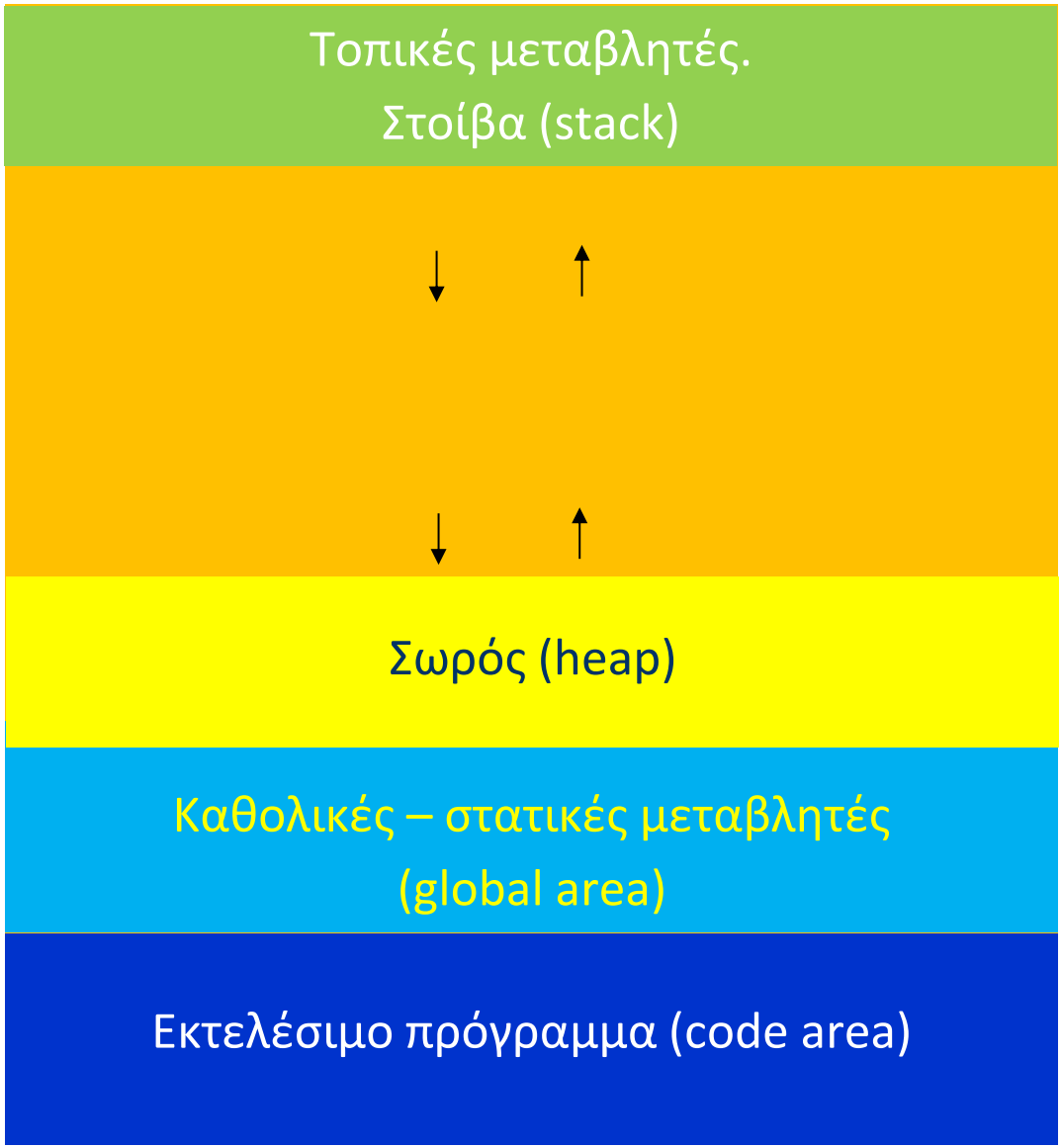
- Περιλαμβάνονται σε πρότυπες βιβλιοθήκες της C που δηλώνονται με την οδηγία *#include* (header files)
- Τα *header files*, περιέχουν πρότυπα συναρτήσεων και δηλώσεις σταθερών και τύπων δεδομένων για κάποια συγκεκριμένη βιβλιοθήκη.
- Γνωστές βιβλιοθήκες έτοιμων συναρτήσεων:

```
<stdio.h>  // συναρτήσεις εισόδου/ εξόδου
<math.h>  // μαθηματικές συναρτήσεις
<string.h>  // συναρτήσεις για χειρισμό συμβολοσειρών
<stdlib.h>  // μετατροπής αριθμών, διαχείρισης μνήμης κ.α
<time.h>    // διαχείρισης ημερομηνιών και ώρας
<ctype.h>  //  διαχείρισης χαρακτήρων
<limits.h>  //  σταθερές που ορίζουν όρια των ακεραίων
<float.h>    // σταθερές που ορίζουν όρια πραγματικών (double)
```

Κατηγορίες μνήμης εκτελέσιμου προγράμματος

- Στις *καθολικές* και *στατικές* μεταβλητές, οι χώροι μνήμης δεσμεύονται κατά την διάρκεια της μεταγλώττισης.
- Οι *τοπικές μεταβλητές* δεσμεύονται προσωρινά, όσο διαρκεί η εκτέλεση της συνάρτησης που τις περιέχει.
- Όταν εκτελείται ένα πρόγραμμα η μνήμη που καταλαμβάνει χωρίζεται τυπικά σε *4 μέρη*:
  - *Στη μνήμη που καταλαμβάνει ο εκτελέσιμος κώδικας (code area).*
  - *Στη μνήμη για τις καθολικές και στατικές μεταβλητές (global area).*
  - *Στη μνήμη όπου αποθηκεύονται οι τοπικές μεταβλητές και οι παράμετροι συναρτήσεων. Ονομάζεται «στοίβα» (stack).*
  - *Στη μνήμη για τις μεταβλητές που δεσμεύονται με δυναμική διαχείριση. Πρόκειται για ελεύθερη μνήμη και ονομάζεται «σωρός» (heap).*

Οι κατηγορίες μνήμης



Τι γίνεται όταν καλείται μια συνάρτηση

- Όταν γίνεται η κλήση μιας συνάρτησης, δεσμεύεται μνήμη για τις μεταβλητές που δηλώνονται σαν παράμετροι της συνάρτησης, καθώς επίσης και για τις μεταβλητές που δηλώνονται στο σώμα της (μνήμη στοίβα -stack).
- Η αποδέσμευση αυτής της μνήμης γίνεται αυτόματα όταν ολοκληρωθεί η εκτέλεση της συνάρτησης.
- Η κλήση μιας συνάρτησης που δέχεται παραμέτρους, σημαίνει ότι στη συνάρτηση μεταβιβάζονται τιμές μέσω των ορισμάτων της.
- Πότε αναφερόμαστε σε *παραμέτρους* και πότε σε *ορίσματα*:
  - *Παράμετροι: αφορούν στις μεταβλητές που εμφανίζονται στη δήλωση και στον ορισμό της συνάρτησης.*
  - *Ορίσματα: αφορούν στις τιμές (ή περιεχόμενα μεταβλητών ή αποτελέσματα πράξεων/εκφράσεων) που χρησιμοποιούνται κατά τη κλήση της συνάρτησης.[πραγματικοί παράμετροι]*

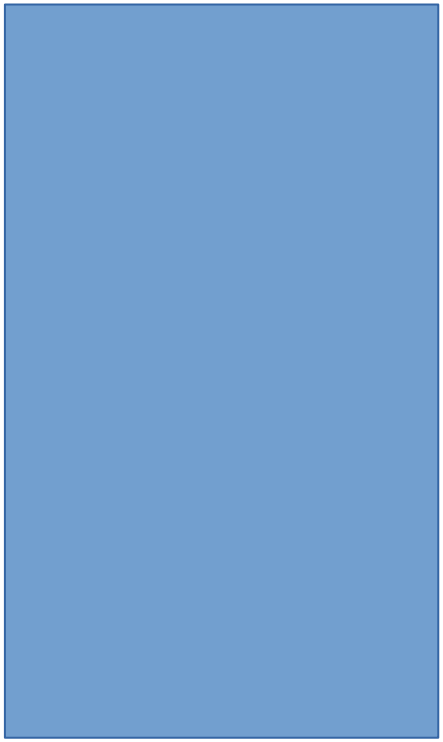
Τι γίνεται στη στοίβα όταν καλούνται συναρτήσεις

```
#include <stdio.h>
#include <math.h>
float calc2 (float z) {
        return (2*z+3);
}
  float calc1(float z) {
  float u;
  u= calc2(z);
  return pow(u,2);
}
main () {
  float x,y;
  printf("doste X:");
  scanf ("%f", &x);
  y=calc1(x);
  printf("Y=%f", y);
}
```

x <--2

y <--....

Πριν κληθεί η συνάρτηση, η στοίβα είναι άδεια



ΣΤΟΙΒΑ (stack)

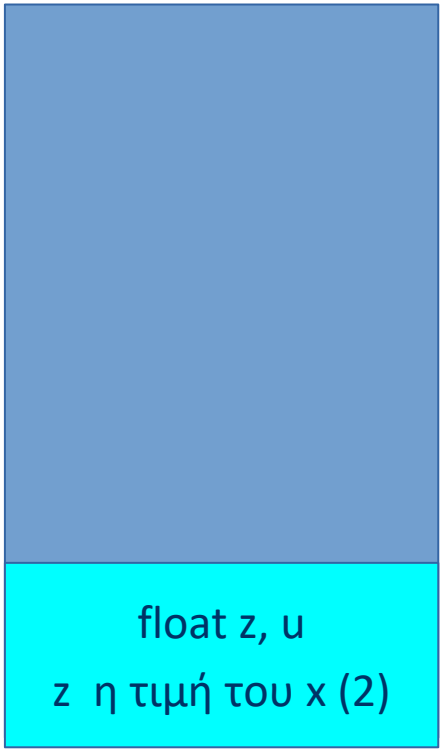
Τι γίνεται στη στοίβα όταν καλούνται συναρτήσεις

```
#include <stdio.h>
#include <math.h>
float calc2 (float z) {
        return (2*z+3);
}
   float calc1(float z) {
   float u;
   u= calc2(z);
   return pow(u,2);
}
main () {
   float x,y;
   printf("doste X:");
   scanf ("%f", &x);
   y=calc1(x);
   printf("Y=%f", y);
}
```

x <--2

y <--....

Με τη κλήση της συνάρτησης calc1 δεσμεύεται χώρος στη στοίβα



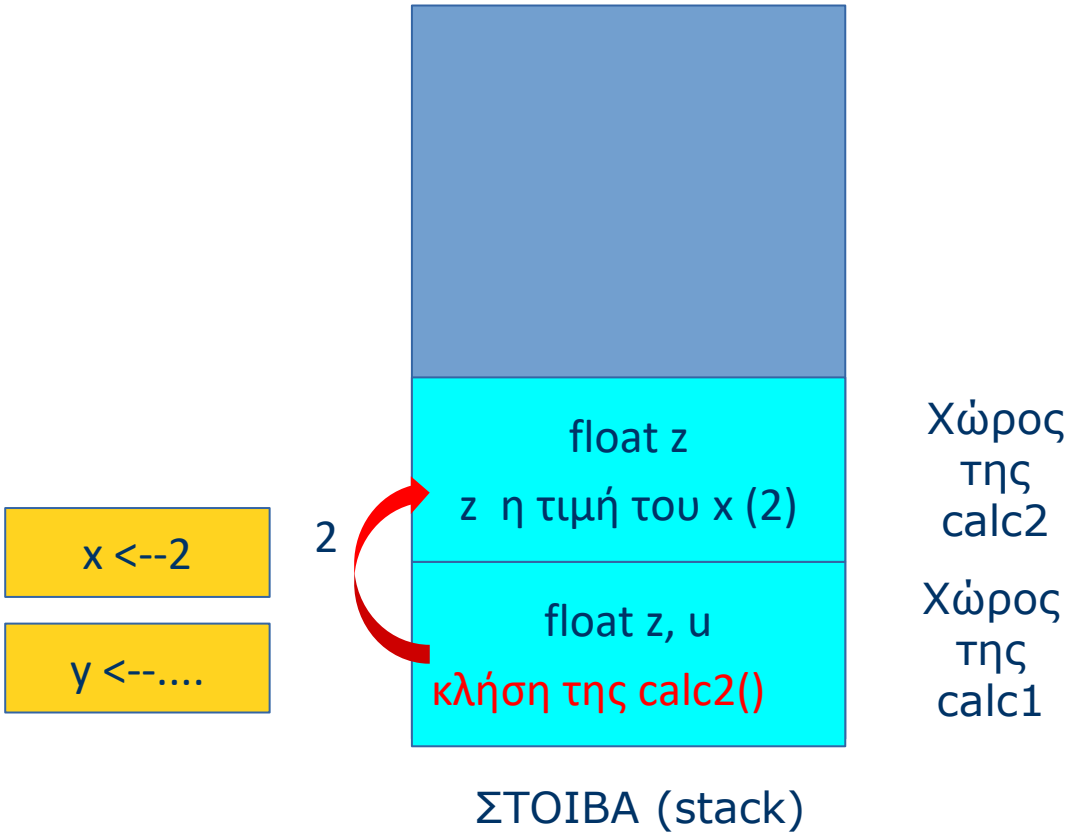
Χώρος της calc1

ΣΤΟΙΒΑ (stack)

Τι γίνεται στη στοίβα όταν καλούνται συναρτήσεις

```
#include <stdio.h>
#include <math.h>
float calc2 (float z) {
        return (2*z+3);
}
  float calc1(float z) {
  float u;
  u= calc2(z);
  return pow(u,2);
}
main () {
  float x,y;
  printf("doste X:");
  scanf ("%f", &x);
  y=calc1(x);
  printf("Y=%f", y);
}
```

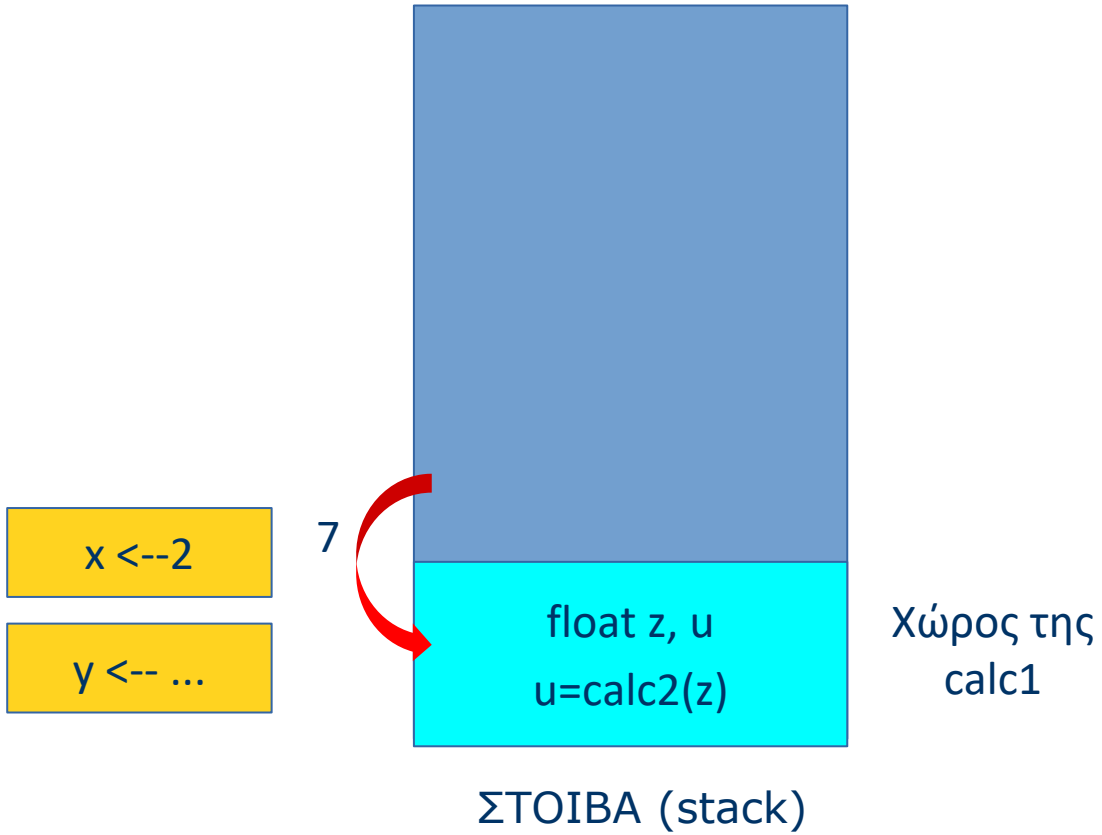
Με τη κλήση της συνάρτησης calc2 από την cacl1, δεσμεύεται επιπλέον χώρος στη στοίβα



Τι γίνεται στη στοίβα όταν καλούνται συναρτήσεις

```
#include <stdio.h>
#include <math.h>
float calc2 (float z) {
        return (2*z+3);
}
  float calc1(float z) {
  float u;
  u= calc2(z);
  return pow(u,2);
}
main () {
  float x,y;
  printf("doste X:");
  scanf ("%f", &x);
  y=calc1(x);
  printf("Y=%f", y);
}
```

Όταν ολοκληρωθεί η εκτέλεση της calc2, επιστρέφεται η τιμή της στη calc1 και απελευθερώνεται ο χώρος τη calc2



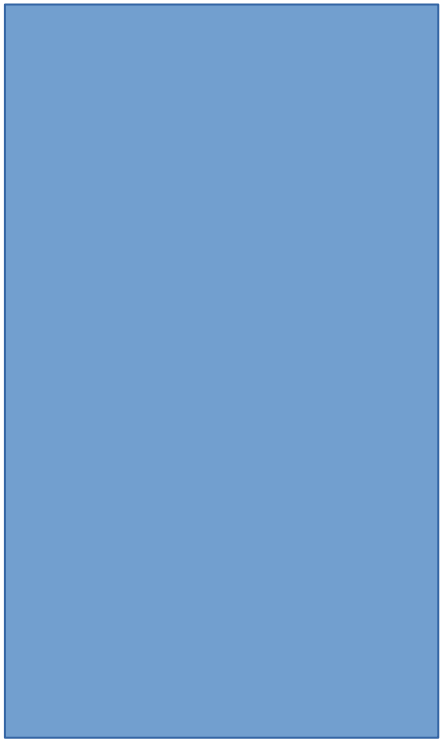
Τι γίνεται στη στοίβα όταν καλούνται συναρτήσεις

```
#include <stdio.h>
#include <math.h>
float calc2 (float z) {
        return (2*z+3);
}
  float calc1(float z) {
  float u;
  u= calc2(z);
  return pow(u,2);
}
main () {
  float x,y;
  printf("doste X:");
  scanf ("%f", &x);
  y=calc1(x);
  printf("Y=%f", y);
}
```

Όταν ολοκληρωθεί η εκτέλεση της calc1 επιστρέφεται η τιμή της στη main() και απελευθερώνεται ο χώρος της calc1

x <--2

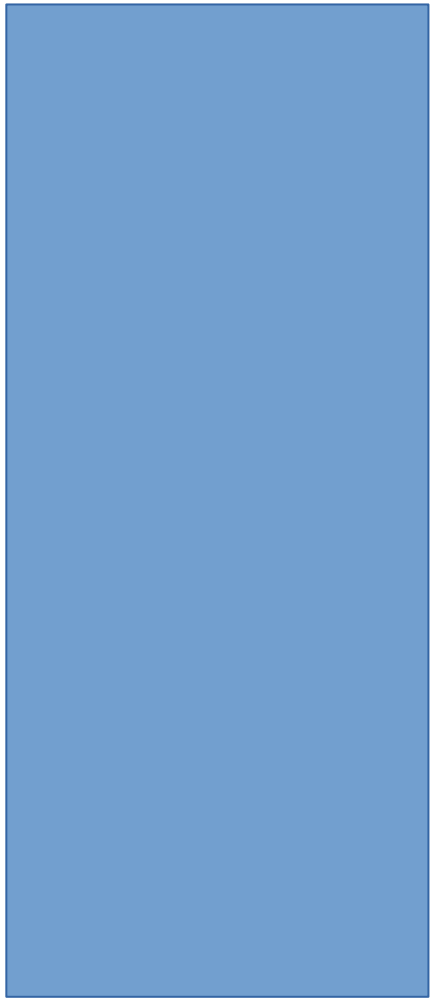
y <-- 49



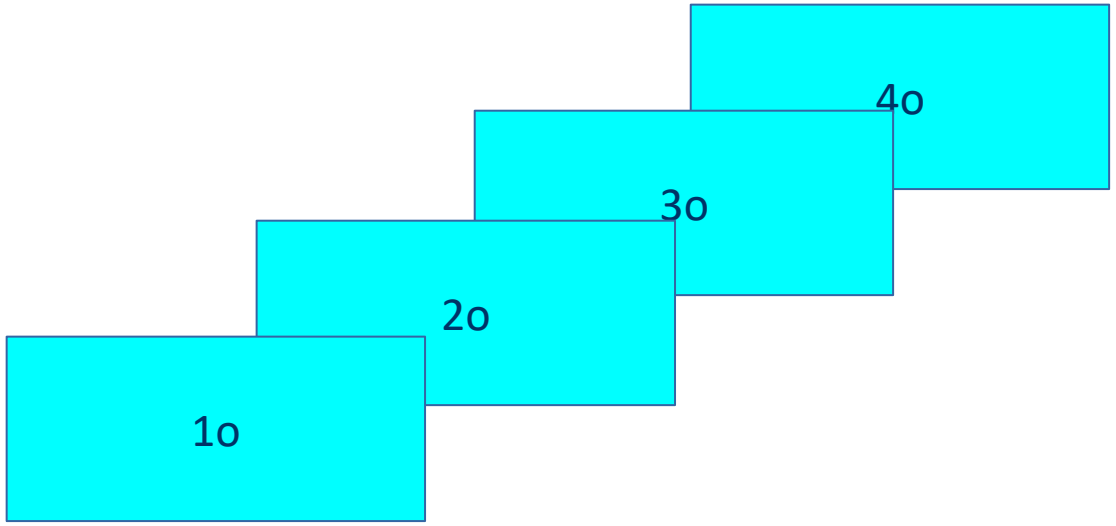
ΣΤΟΙΒΑ (stack)

Η στοίβα είναι άδεια

Η στοίβα ως μηχανισμός LIFO

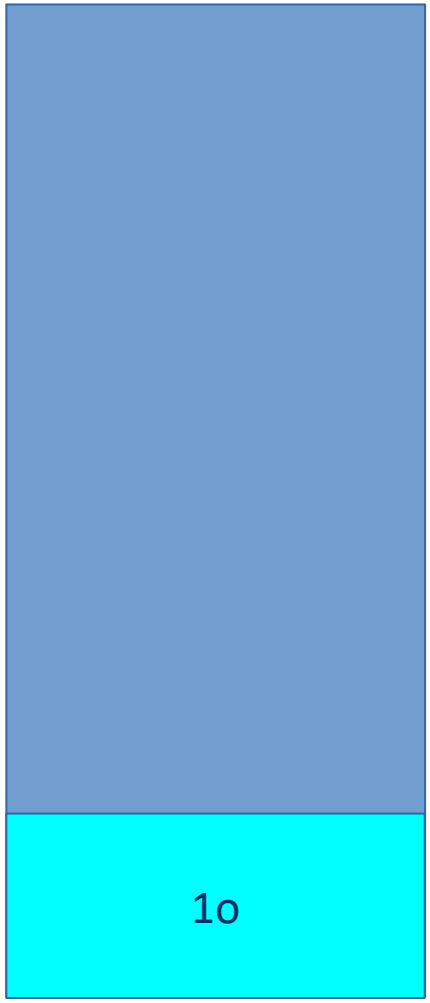


Η στοίβα είναι άδεια

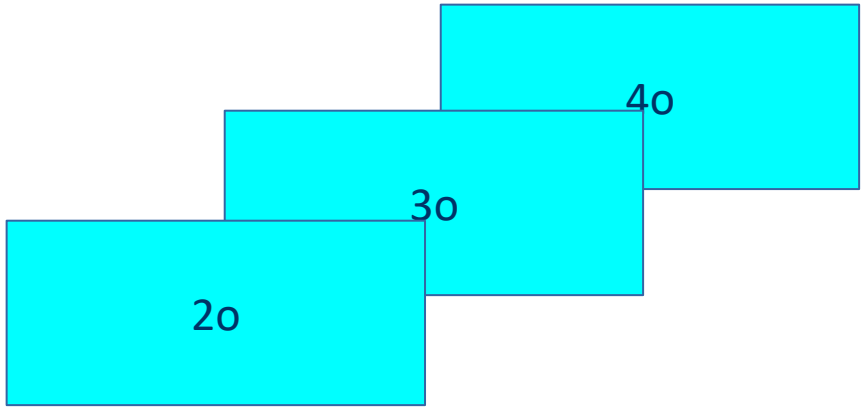


ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



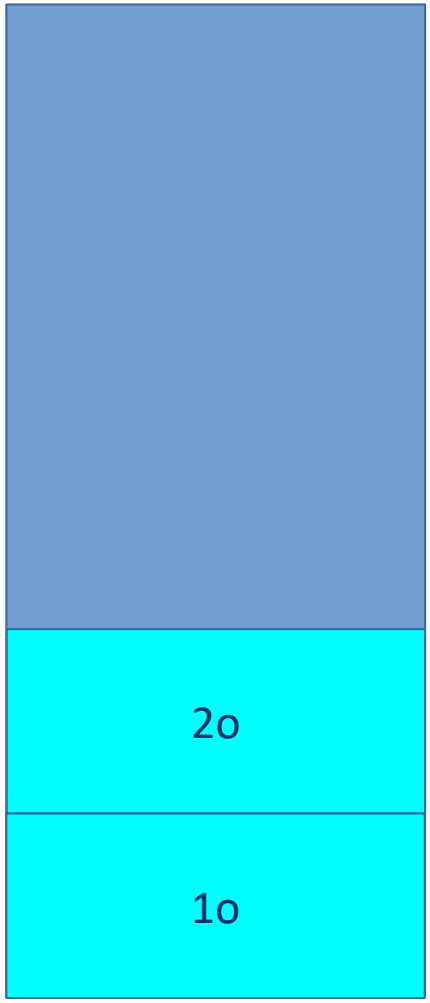
1 στοιχείο στη στοίβα



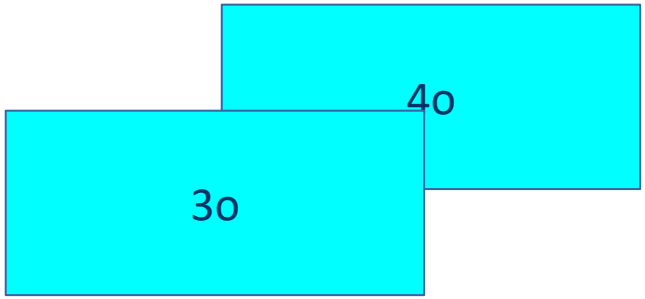
Ώθηση (push)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



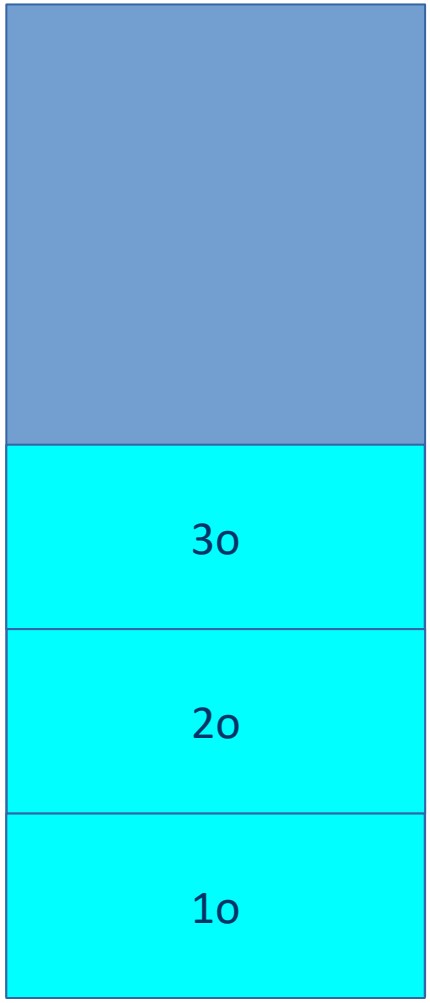
2 στοιχεία στη στοίβα



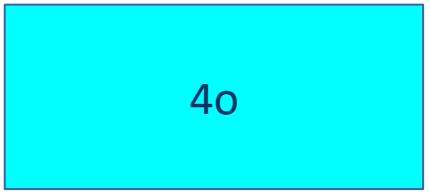
Ώθηση (push)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



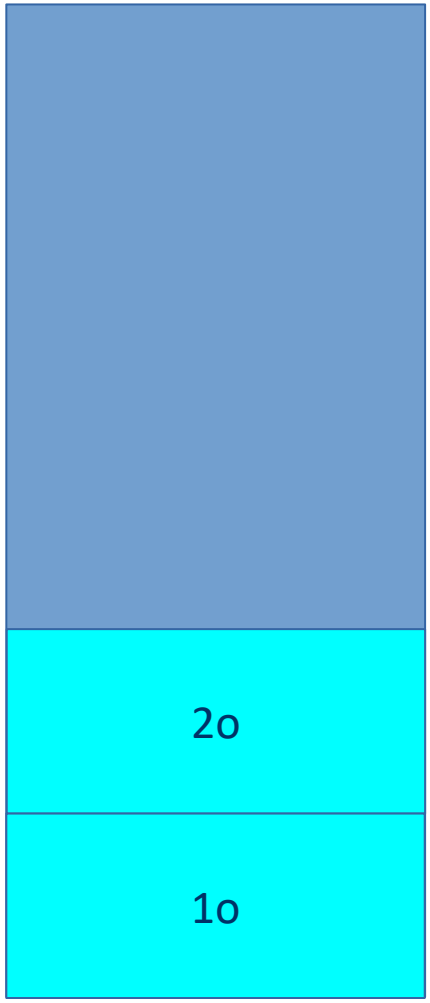
3 στοιχεία στη στοίβα



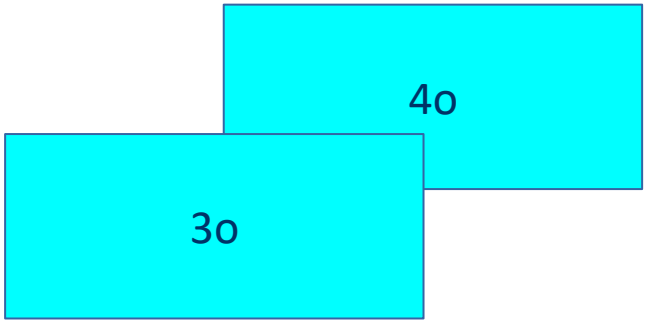
Ώθηση (push)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



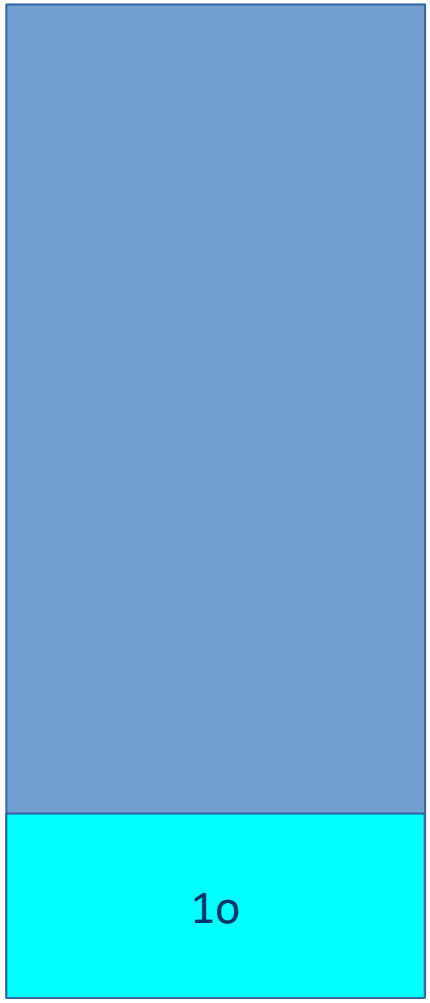
2 στοιχεία στη στοίβα



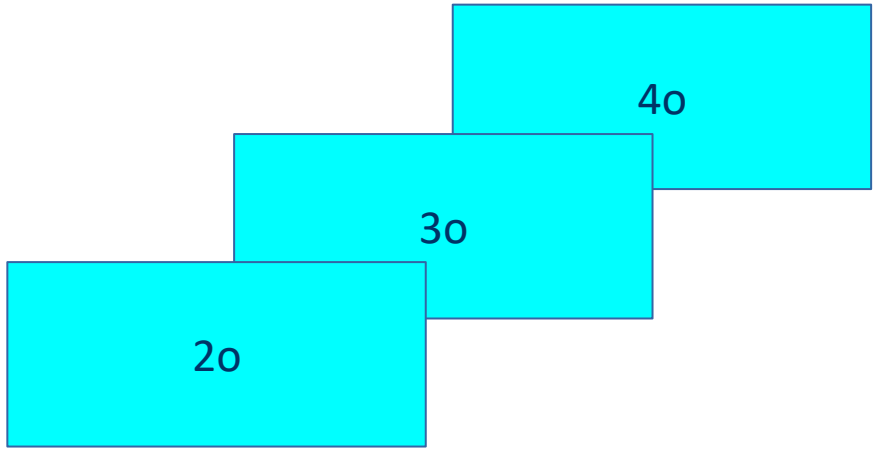
απώθηση
(pop)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



1 στοιχείο στη στοίβα



απώθηση
(pop)

ΣΤΟΙΒΑ (stack)

Ένα περίεργος τρόπος

...για να τυπώσετε όλους τους ζυγούς μέχρι το 30!

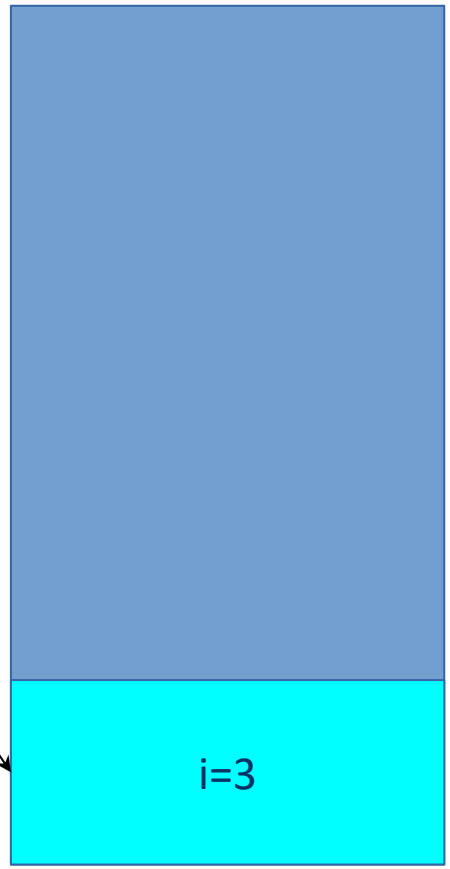
Αν αντί για 30 χρησιμοποιήσουμε 31;

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
     int n = 30;
     f1(n);
}
void f1(int i) {
     if (i)  f2(i-1);    //αληθής όσο i>0
      printf("%d ", i);
}
void f2(int i) {
        if (i)  f1(i-1);   //αληθής όσο i>0
}
```

Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
      if (i)  f1(i-1);
}
```

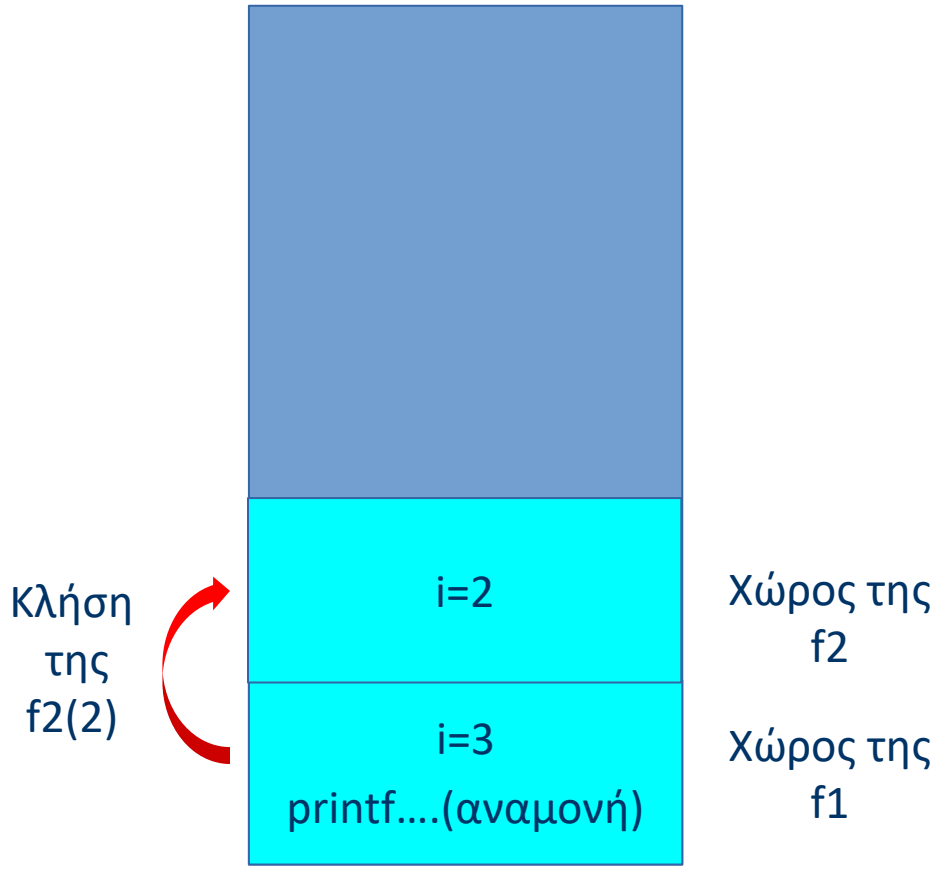
Κλήση της f1(3)



Χώρος της f1

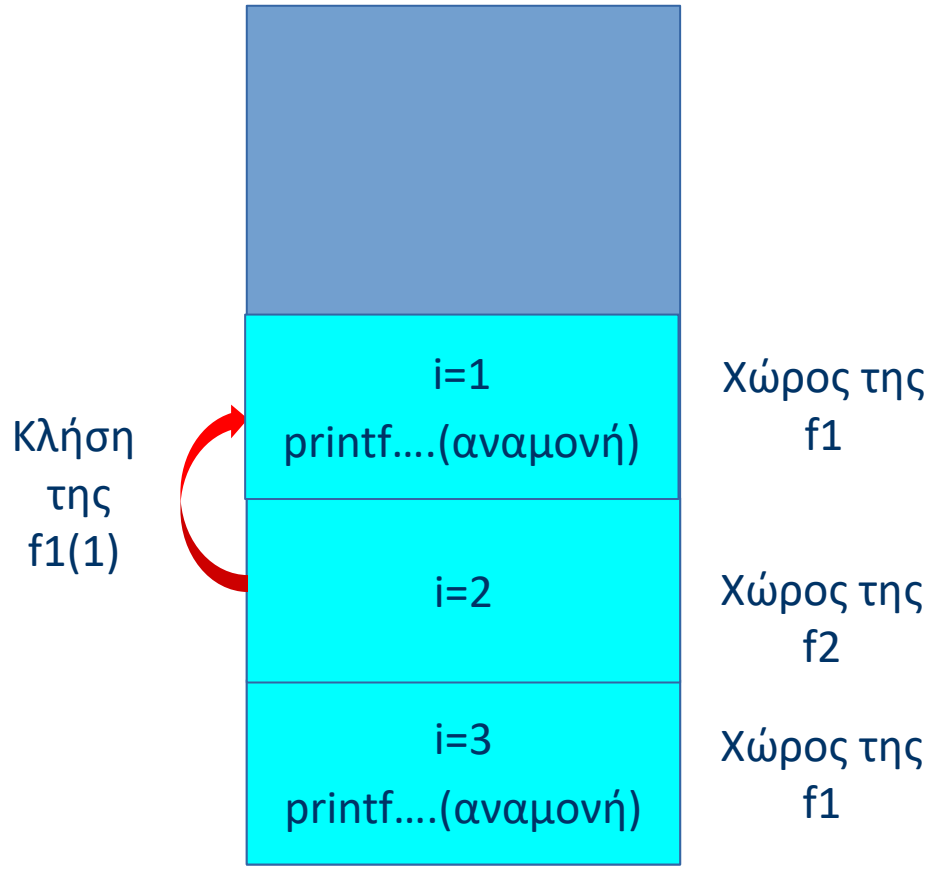
Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
      if (i)  f1(i-1);
}
```



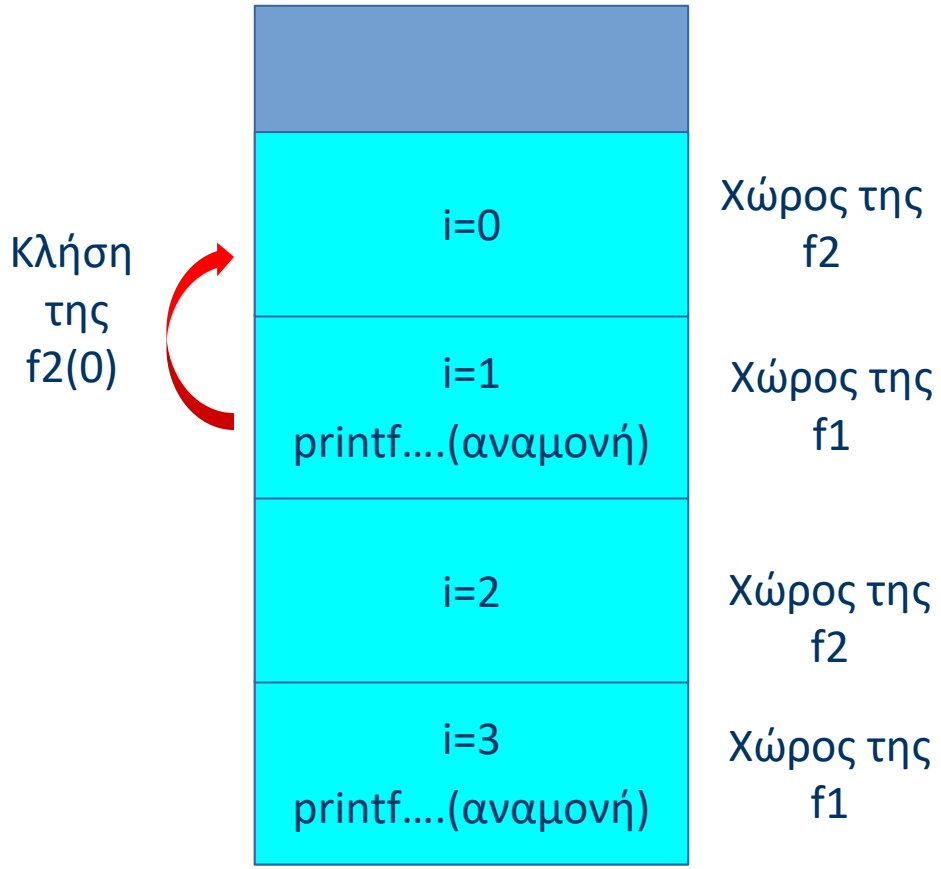
Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
        if (i)  f1(i-1);
}
```



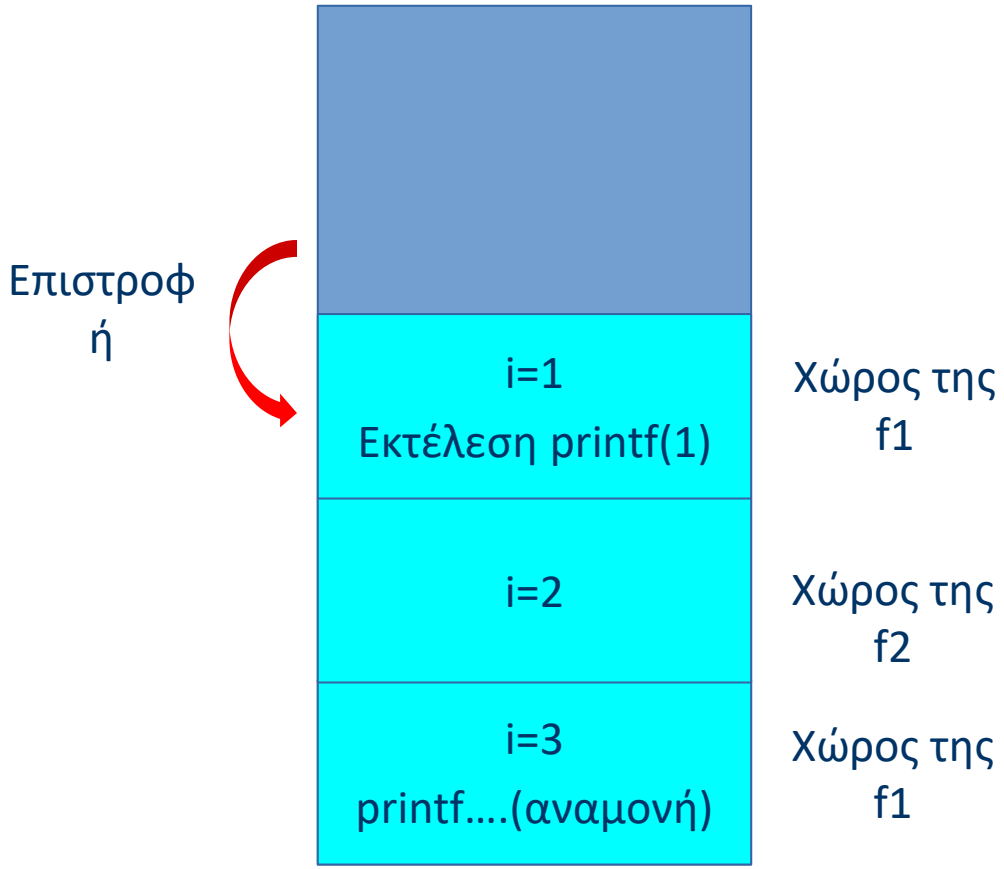
Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
      if (i)  f1(i-1);
}
```



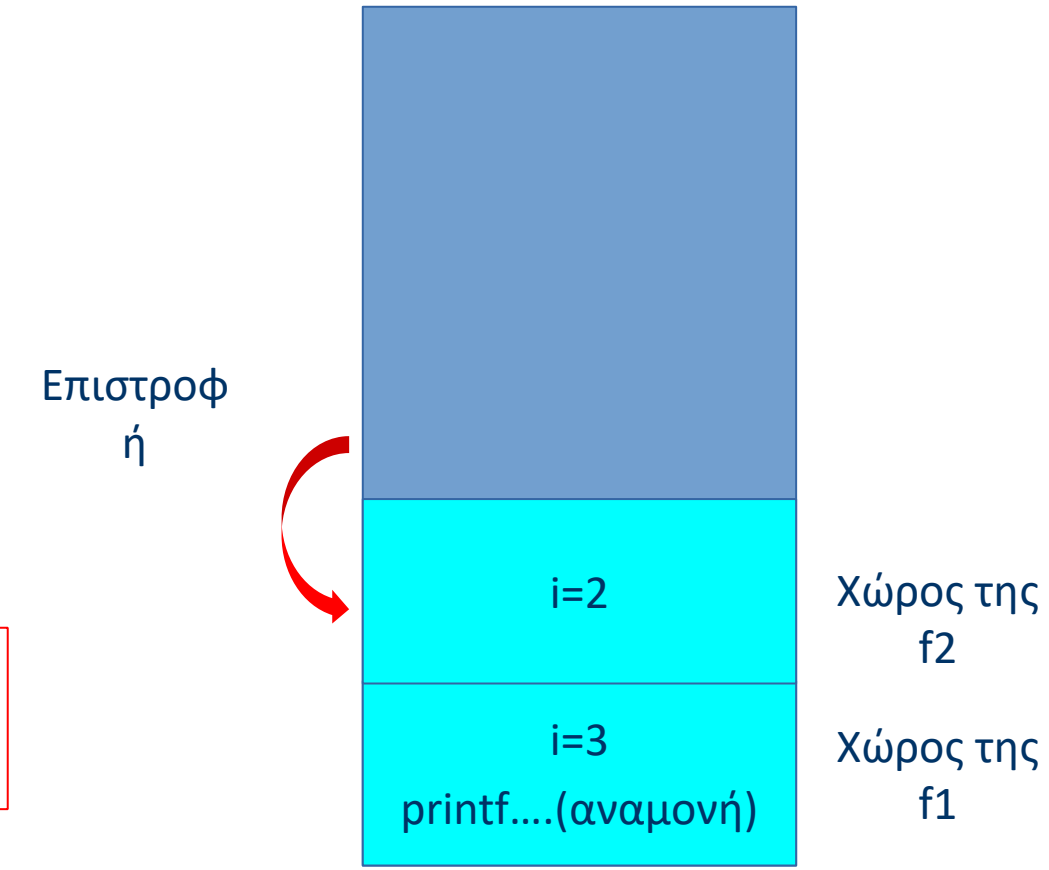
Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
        if (i)  f1(i-1);
}
```



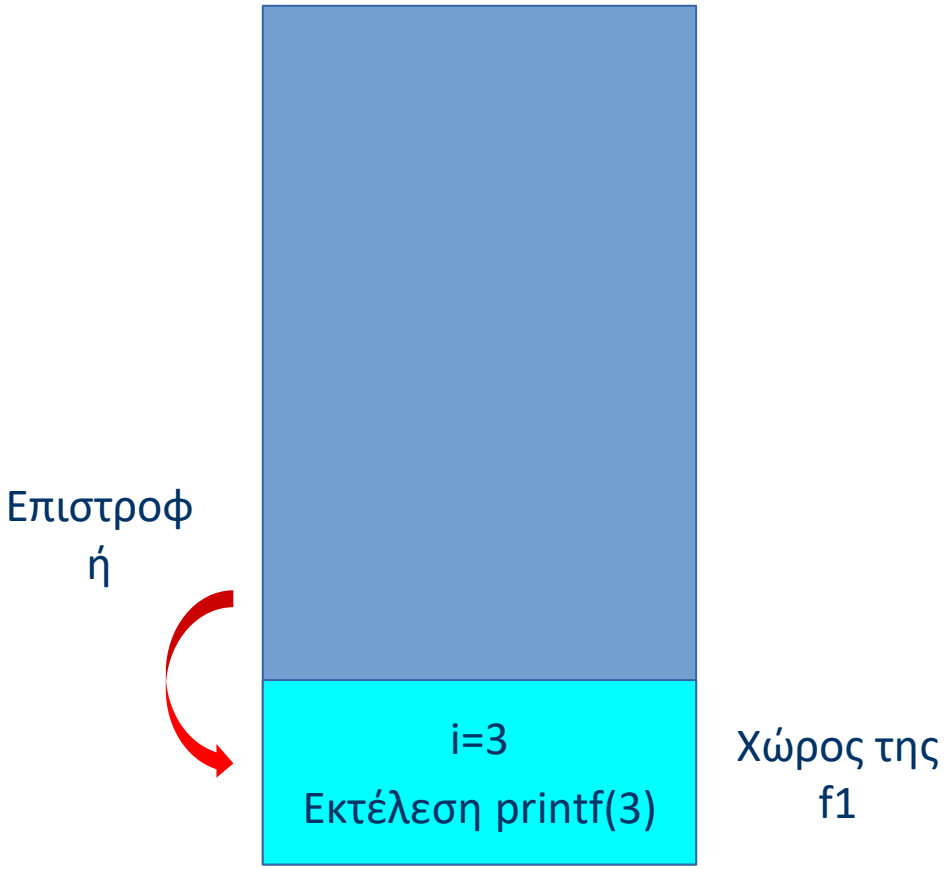
Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
        if (i)  f1(i-1);
}
```



Ο περίεργος τρόπος με n=3

```
#include <stdio.h>
void f1 (int i);
void f2 (int i);
int main(void) {
    int n = 3;
    f1(n);
}
void f1(int i) {
    if (i)  f2(i-1);
    printf("%d ", i);
}
void f2(int i) {
      if (i)  f1(i-1);
}
```



Μονοδιάστατοι πίνακες ως ορίσματα συναρτήσεων

- Κατά την κλήση μιας συνάρτησης με όρισμα έναν μονοδιάστατο πίνακα (διάνυσμα), γράφουμε μόνο το όνομα του πίνακα.
- Στη πραγματικότητα μεταβιβάζεται η διεύθυνση τη 1ης θέσης του πίνακα και συνεπώς έχουμε μεταβίβαση δεδομένων (δηλ. των στοιχείων του πίνακα) *κατ' αναφορά* (call-by-reference).

  Θυμηθείτε: το όνομα του πίνακα είναι και δείκτης στο 1ο στοιχείο ([0]) του πίνακα.

- Δηλαδή, η τυπική παράμετρος της συνάρτησης πρέπει να είναι ένας δείκτης ίδιου τύπου με αυτόν του πίνακα.
- Με αυτόν τον τρόπο η συνάρτηση:
  - *έχει άμεση πρόσβαση στη μνήμη του πίνακα και*
  - *μπορεί να αλλάξει τα περιεχόμενα του πίνακα.*
- Όμως όταν ένα στοιχείο του πίνακα είναι όρισμα, τότε έχουμε μεταβίβαση *κατ' αξία* (call-by-value).

printf(”%d”, myArray[3]);

Μεταβίβαση διανύσματος σε συνάρτηση

Δίνει τιμές στο πίνακα.

Τι τιμές θα δοθούν?

1 2 3 4 5 6 7 8 9 10

Αν θέλαμε να δώσουμε τυχαίες τιμές (πχ 0-10)?

```
void fill(int a[], int size) {
  int i;
  srand(time(NULL));
  for (i=0;i<10;i++) {
      a[i]=rand()%11;
  }
}
```

```
#include <stdio.h>
 void fill(int a[], int size);
 main() {
     int i, c[10] = {0};
     printf("printing before call\n");
     for (i = 0; i < 10; i++) {
          printf("%2d, %3d\n", i, c[i]);
     }
     fill(c, 10);
     printf("\nprinting after call\n");
     for (i = 0; i < 10; i++) {
          printf("%2d, %3d\n", i, c[i]);
     }
}
void fill(int a[], int size) {
    while(size) {
          a[size-1] = size--;
    }
}
```

Τα διανύσματα ως τυπικοί παράμετροι σε συναρτήσεις

```
#include <stdio.h>
void f1 (int A[5]);
void f2 (int A[]);
void f3 (int *A);
int main(void)
{
   int B[5] = {1,2,3,4,5};
   f1(B); printf("\n");
   f2(B); printf("\n");
   f3(B); printf("\n");
}
```

Και οι 3 συναρτήσεις τυπώνουν ένα μονοδιάστατο πίνακα

```
void f1(int A[5]) {
    int i;
    for (i=0;i<5;i++)
          printf("%d ", A[i]);
}
```

```
void f2(int A[]) {
    int i;
   for (i=0;i<5;i++)
         printf("%d ", A[i]);
}
```

```
void f3(int *A)  {
   int i;
   for (i=0;i<5;i++)
         printf("%d ", A[i]);
}
```

Φυσικά αντί για A[i] μπορούμε και *(A+i)

Πίνακες 2 διαστάσεων ως ορίσματα συναρτήσεων

- Στη δήλωση μιας συνάρτησης που έχει σαν παράμετρο πίνακα 2 διαστάσεων γράφουμε το όνομα του πίνακα με τις μέγιστες τιμές των διαστάσεων του.
- Το μέγεθος (μέγιστη τιμή) της πρώτης διάστασης (αριθμός γραμμών) μπορεί να μη δηλώνεται, απαιτείται όμως πάντα η δήλωση της τιμής της 2ης διάστασης.

void function(pin[][5]);

- Στους πολυδιάστατους πίνακες, πρέπει πάντα να δηλώνεται η μεγαλύτερη τιμή κάθε διάστασης, εκτός της πρώτης διάσταση.
- Όταν γίνεται κλήση μιας συνάρτησης που έχει σαν παράμετρο έναν πίνακα, γράφουμε μόνο το όνομα του πίνακα, χωρίς τις αγκύλες:

function (pin);

Πίνακες 2 διαστάσεων και συναρτήσεις

Απόδοση τιμών

```
#include <stdio.h>
#define SIZE 100
void ins_m (int A[][100], int r, int c);
void print_m (int (*A)[100], int r, int c);
main () {
    int mat[SIZE][SIZE];
    int row, column;
    printf("Input rows: ");
    scanf("%d", &row);
    printf("Input columns: ");
    scanf  ("%d", &column);
    printf ("Input data: \n");
    ins_m(mat, row, column);
    printf ("You entered: \n");
    print_m(mat, row, column);
}
```

```
void ins_m (int A[][100], int r, int c){
   int i, j;
   for (i = 0; i < r; i++){
      for (j = 0; j < c; j++){
         printf ("Row %d column %d: ", i+1,
j+1);
         scanf  ("%d", &A[i][j]);
       }
    }
```

```
void print_m (int (*A)[100], int r, int c){
   int i, j;
   for(i=0; i<r; i++){
      for(j=0; j<c; j++){
         printf("%d ", A[i][j]);
       }
      printf("\n");
    }
}
```

Εύρεση min & max διαγώνιου πίνακα NxN

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 5
void ins_m (int A[][N]);
void print_m (int (*A)[N]);
void findmm(int A[][N], int *min, int *max);
main() {
   int i,j, pin[N][N], min, max;
   ins_m (pin);
   print_m (pin);
   findmm(pin, &min, &max);
   printf("\n Min diagoniou:%d\n",min);
   printf("\n Max diagoniou:%d\n",max);
}
```

```
void ins_m (int A[][N]){
   int i, j;
   srand(time(NULL));
   for (i = 0; i < N; i++)
         for (j=0; j<N; j++) A[i][j]=rand()%100;
}
void print_m (int (*A)[N]){
   int i, j;
   for(i=0; i<N; i++){
        for(j=0; j<N; j++) {
              printf("%3d ", A[i][j]);
        }
        printf("\n");
     }
  }
```

```
void findmm(int A[][N], int *min, int *max){
   int i;
   *min=*max=A[0][0];
    for (i=0;i<N; i++) {
          if (*min>A[i][i]) *min=A[i][i];
          if (*max<A[i][i]) *max=A[i][i];
    }
}
```

Η συνάρτηση findmm αλλάζει το περιεχόμενο των min και max

Αναδρομή

- Στη C μια συνάρτηση μπορεί να καλεί τον εαυτό της.
- Σε αυτή τη περίπτωση υπάρχει ένα μόνο αντίγραφο της συνάρτησης.
- Όταν καλείται μια συνάρτηση δεσμεύεται χώρος στη μνήμη (stack) για την αποθήκευση των παραμέτρων της.
- Όταν λοιπόν μια συνάρτηση καλεί τον εαυτό της, αυτή εκτελείται με ένα νέο σύνολο παραμέτρων.
- Ο κώδικας της συνάρτησης παραμένει ο ίδιος.
- Η κατ' επανάληψη κλήση μιας συνάρτησης προϋποθέτει ότι στο κώδικα της συνάρτησης υπάρχουν εντολές που *προκαλούν τη παύση της επαναλαμβανόμενης κλήσης*.
- Αν δεν υπάρχει αυτή η «υπό όρους» αυτό-κλήση της συνάρτησης, υπάρχει κίνδυνος να αναλωθεί όλος ο διαθέσιμος χώρος.

Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

Χωρίς αναδρομή

```
#include <stdio.h>
int factorial(int n);
main()
{
    int i=3, F;
    F = factorial(i);
    printf("Factorial of %d is %d\n",i,F);
    getchar();
}
int factorial(int n) {
    int product, i;
    product = 1;
    for (i = 1; i <= n; i++) {
            product *= i;
    }
    return (product);
}
```

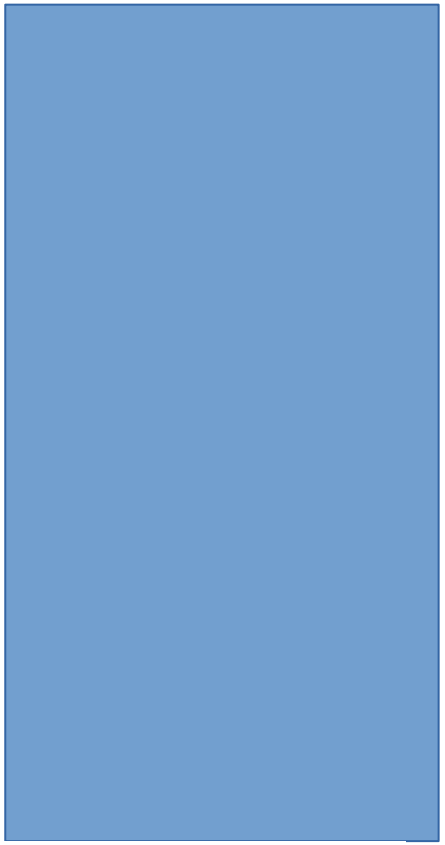
Με αναδρομή

```
#include <stdio.h>
int factorial(int n);

main()
{
    int i=5, F;
    F = factorial(i);
    printf("Factorial of %d is %d\n",i,F);
    getchar();
}
int factorial(int n) {
    int product;
    if (n==1) return 1;
    product=n*factorial(n-1);
    return (product);
}
```

Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```

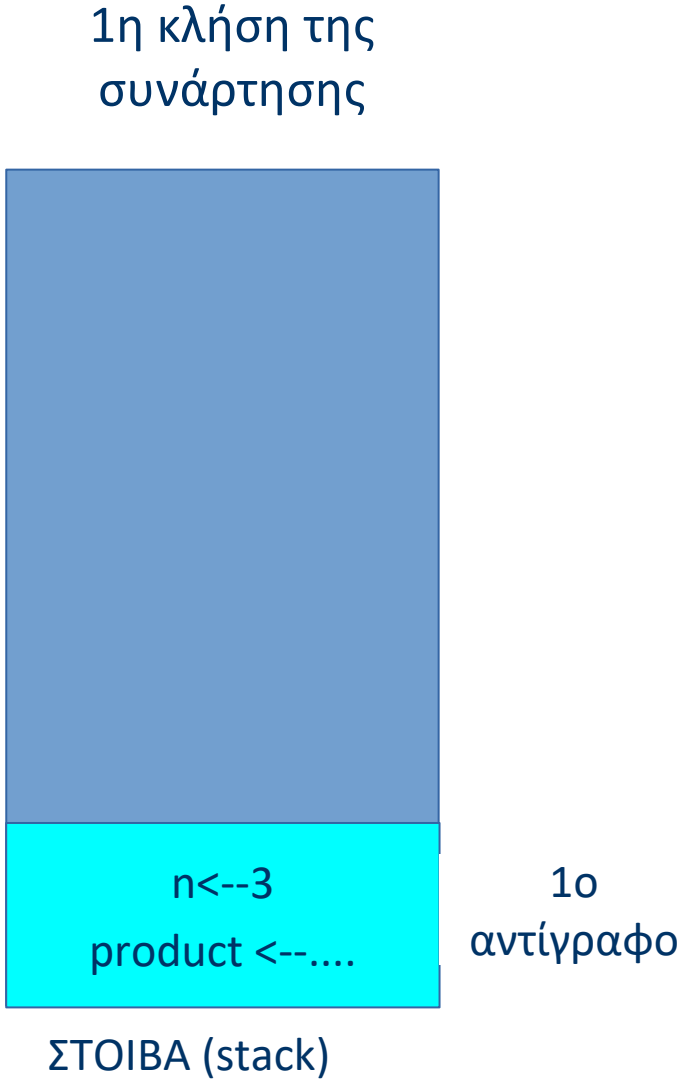
Πριν κληθεί η συνάρτηση, η στοίβα είναι άδεια



ΣΤΟΙΒΑ (stack)

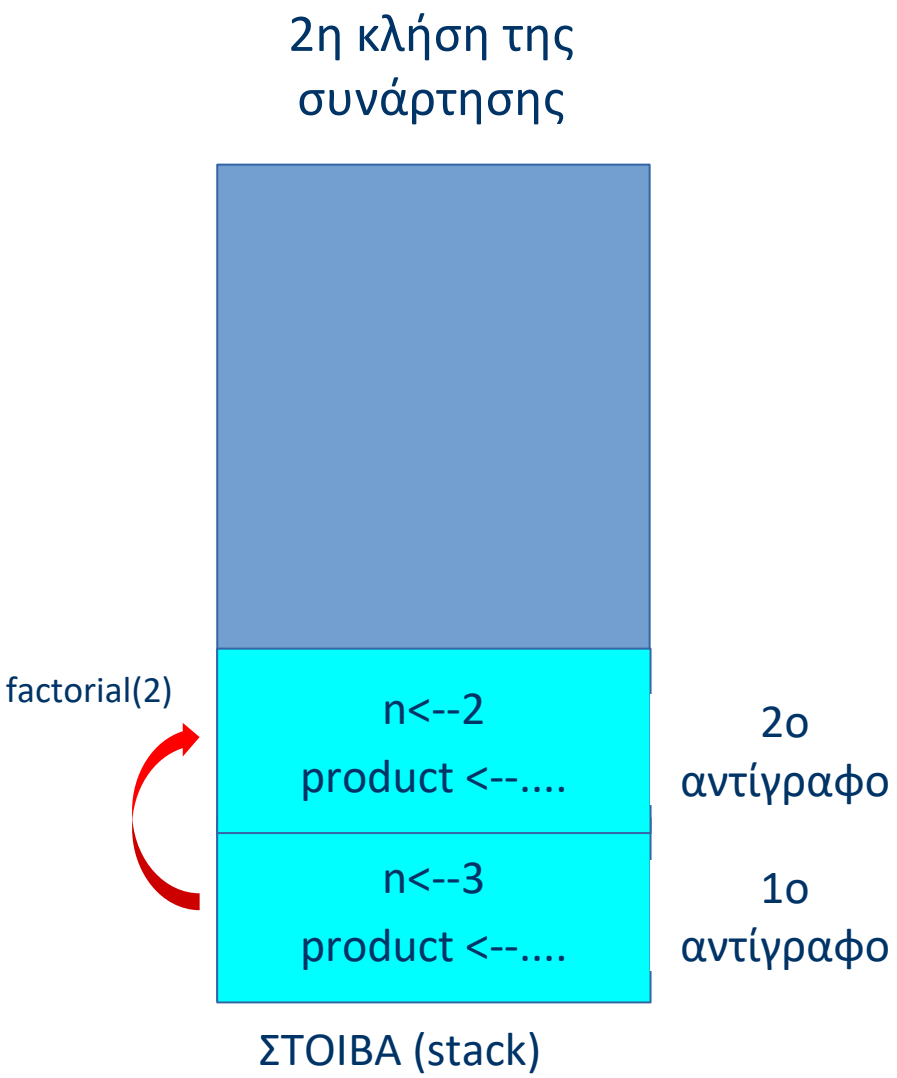
Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```



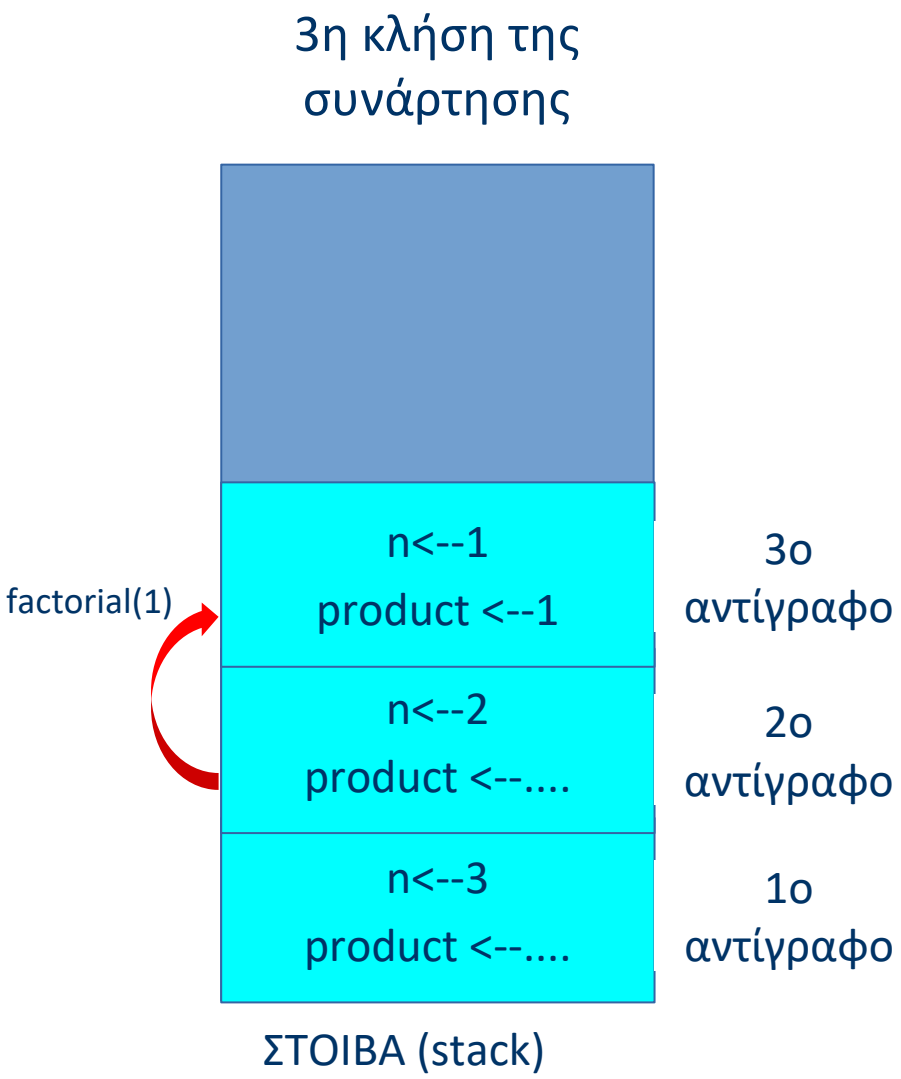
Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```



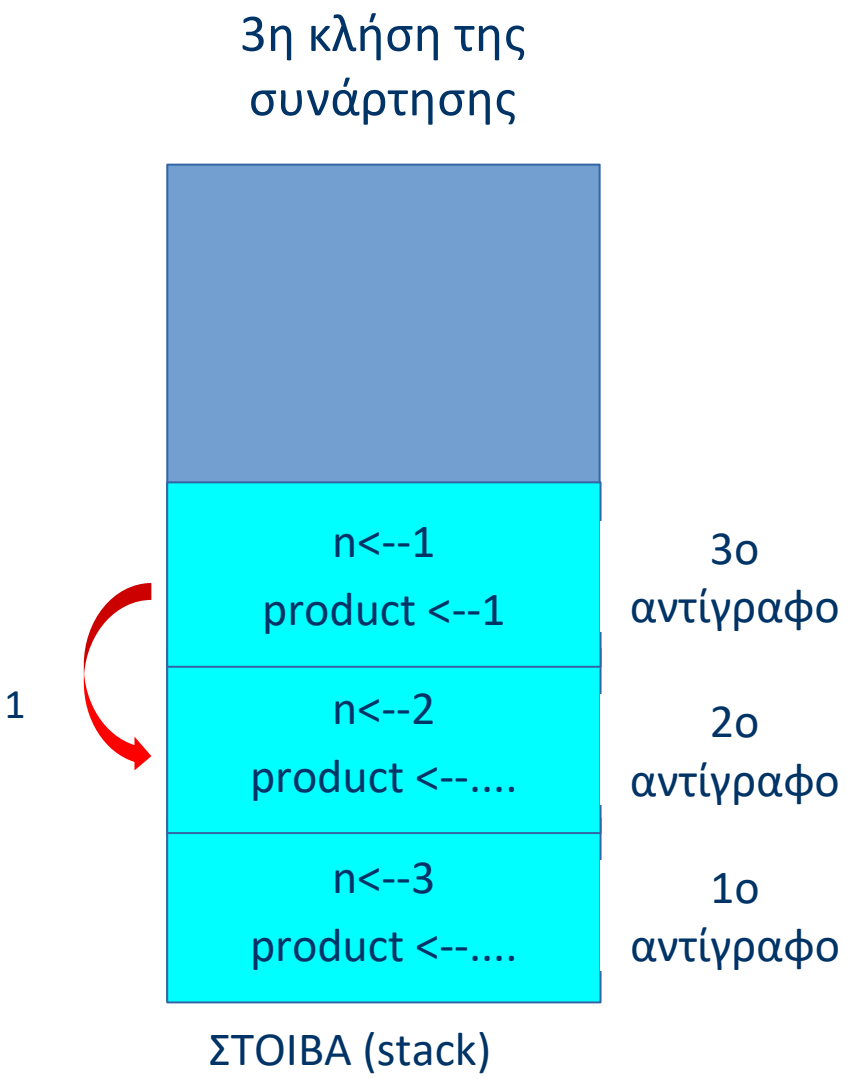
Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```



Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

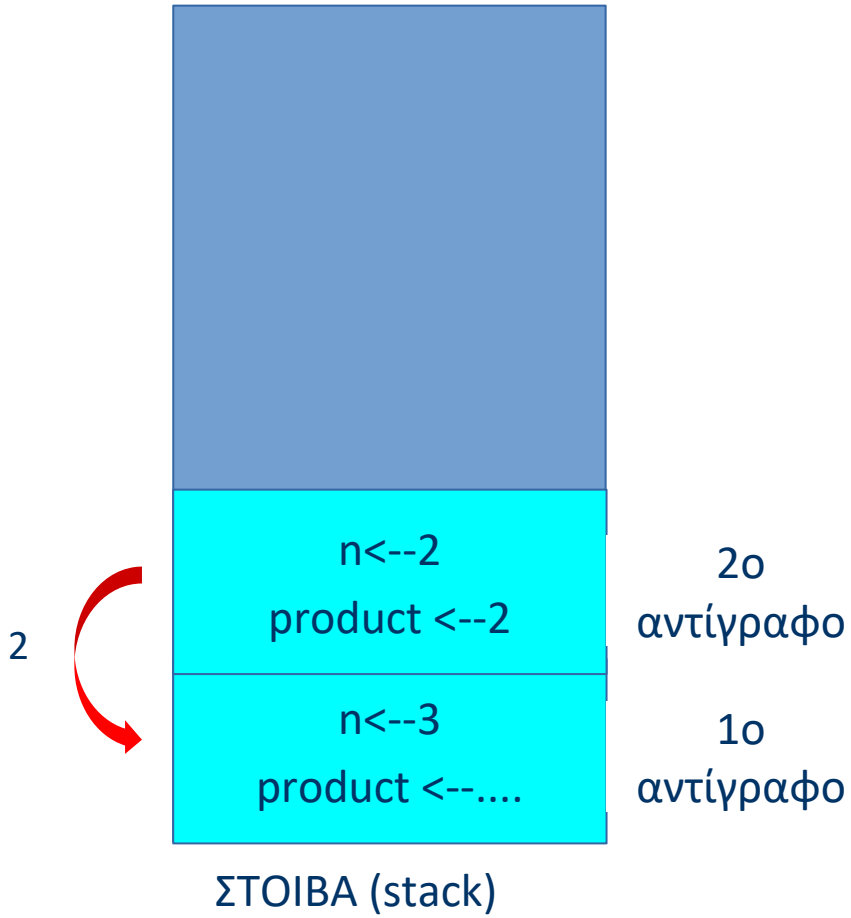
```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```



Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```

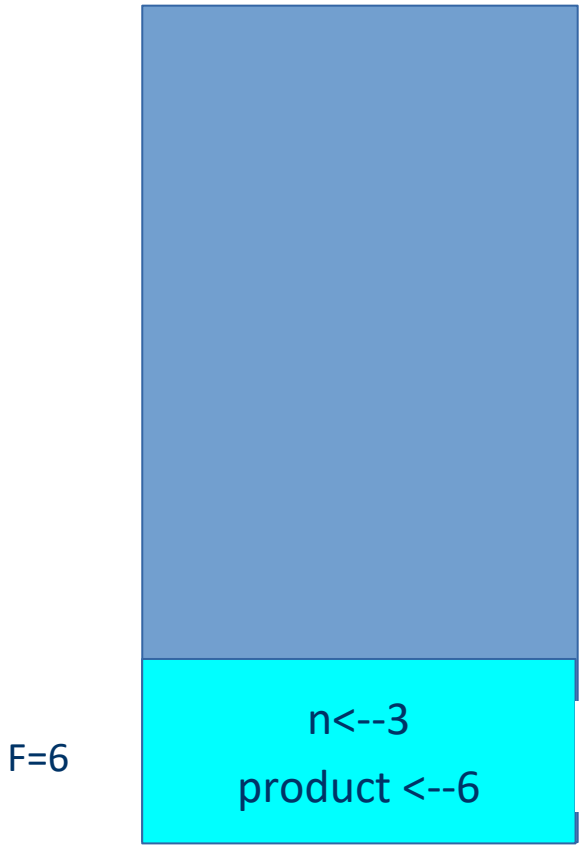
Αποδεσμεύεται η μνήμη του 3ου αντίγραφου



Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```

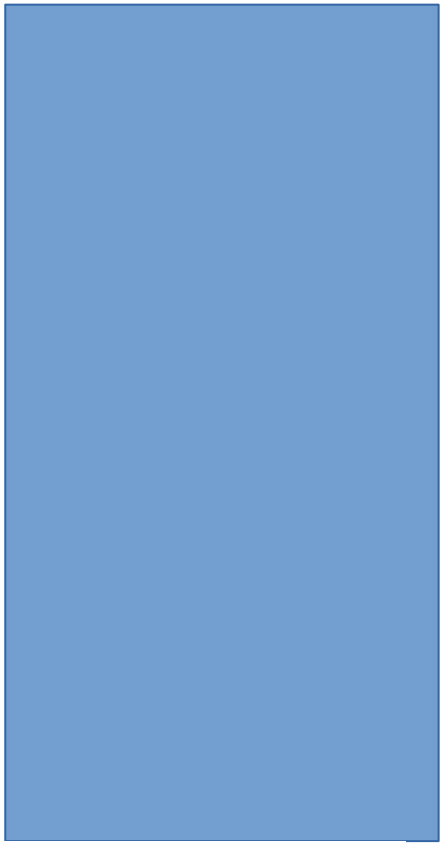
Αποδεσμεύεται η μνήμη του 2ου αντίγραφου



Παράδειγμα με αναδρομή: υπολογισμός παραγοντικού

```
#include <stdio.h>
int factorial(int n);

main()
{
   int i=3, F;
   F = factorial(i);
   printf("Factorial of %d is %d\n",i,F);
   getchar();
}
int factorial(int n) {
   int product;
   if (n==1) return 1;
   product=n*factorial(n-1);
   return (product);
}
```

Αποδεσμεύεται η μνήμη του 3ου αντίγραφου



ΣΤΟΙΒΑ (stack)

Άλλο ένα παράδειγμα με αναδρομή

```
#include <stdio.h>
void recurse(int i);

int main(void)
{
    int n = 0;
    recurse(n);
}
void recurse(int i) {
  if (i<10){
        recurse(i+1);
        printf("%d ", i);
  }
}
```

Αποτέλεσμα εκτέλεσης:

9 8 7 6 5 4 3 2 1 0

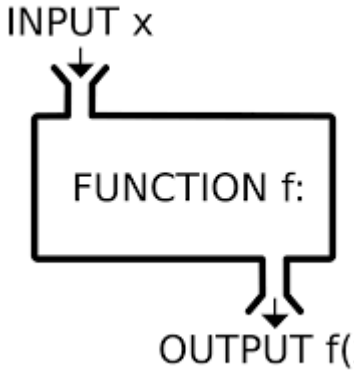
Τι θα δείξει αν αλλάξει η συνάρτηση ως εξής:

```
void recurse(int i)
{
    if (i<10){
        printf("%d ", i);
        recurse(i+1);
  }
}
```

0 1 2 3 4 5 6 7 8 9

Περισσότερη εξάσκηση

- Λύστε τη σειρά των ασκήσεων που έχουν αναρτηθεί στο δικτυακό τόπο του μαθήματος.



https://commons.wikimedia.org/