Συναρτήσεις

Ορισμός συναρτήσεων

```
<τύπος> <όνομα> (<τυπικές παράμετροι>) {
   /* δήλωση μεταβλητών */
   /* εντολές ελέγχου/επεξεργασίας */
}
```

- Μια συνάρτηση ορίζεται δίνοντας (α) τον τύπο του αποτελέσματος που επιστρέφει (`void` αν δεν επιστρέφει τιμή), (β) το όνομα της, (γ) την λίστα με τις «τυπικές» παραμέτρους της, και (δ) το σώμα της.
- Τα (α), (β), (γ) αποτελούν την **επικεφαλίδα** και το (δ) τον **κώδικα** / **σώμα** (body) της συνάρτησης.
- Μια συνάρτηση μπορεί να **δηλωθεί** (ξεχωριστά) μέσω της επικεφαλίδας της, με την υλοποίηση της να δίνεται σε παρακάτω σημείο του κώδικα (ή ακόμα και σε διαφορετικό αρχείο – βλέπε αργότερα).

Επιστροφή αποτελέσματος

- Η επιστροφή αποτελέσματος μιας συνάρτησης γίνεται με την εντολή `return(<τιμή>)` – αν αυτή δεν υπάρχει ή χρησιμοποιηθεί χωρίς κάποια τιμή, τότε η συνάρτηση επιστρέφει μια τυχαία τιμή.
- Όταν εκτελεσθεί η εντολή `return` **τερματίζεται** αυτόματα και η εκτέλεση της συνάρτησης.
- Η εντολή `return` μπορεί να υπάρχει σε πολλά σημεία του σώματος της συνάρτησης – χρειάζεται προσοχή έτσι ώστε να επιστέφεται το επιθυμητό αποτέλεσμα σε κάθε περίπτωση.
- Η κυρίως συνάρτηση `main` επιστρέφει την τιμή της στο περιβάλλον εκτέλεσης (λειτουργικό σύστημα).

Κλήση συνάρτησης

- Η κλήση μιας συνάρτησης πραγματοποιείται δίνοντας το όνομα της συνάρτησης, και σε παρενθέσεις μια τιμή για κάθε μια από τις τυπικές παραμέτρους της, χρησιμοποιώντας το ‘,’ ως διαχωριστικό.
- Αν μια συνάρτηση επιστρέφει αποτέλεσμα, τότε η κλήση της συνάρτησης αποτελεί **έκφραση αποτίμισης** που δίνει τιμή του αντίστοιχου τύπου.
- Μια κλήση συνάρτησης μπορεί να χρησιμοποιηθεί για την ανάθεση τιμής σε μεταβλητή ή/και σαν τμήμα έκφρασης σε συνδυασμό με κατάλληλο τελεστή (που μπορεί να χρησιμοποιηθεί για τιμές του τύπου που επιστρέφει η συνάρτηση).

```c
/* υπολογισμός μέγιστης τιμής 2 ακεραίων */

#include <stdio.h>

int max2(int x, int y) {
  if (x > y) { return(x); }
  else { return(y); }
}

int main(int argc, char *argv[]) {
  int a,b,c;

  printf("enter 2 int: ");
  scanf("%d %d", &a, &b);
  c = max2(a,b);
  printf("%d\n", c);

}
```

```c
/* υπολογισμός μέγιστης τιμής 2 ακεραίων */

#include <stdio.h>

int max2(int x, int y) {
  if (x > y) { return(x); }
  return(y);
}

int main(int argc, char *argv[]) {
  int a,b,c;

  printf("enter 2 int: ");
  scanf("%d %d", &a, &b);
  c = max2(a,b);
  printf("%d\n", c);

}
```

```
/* υπολογισμός μέγιστης τιμής 2 ακεραίων */

#include <stdio.h>

int max2(int x, int y) {
  int z;

  if (x > y) { z = x; }
  else { z = y; }

  return(z);
}

int main(int argc, char *argv[]) {
  int a,b,c;

  printf("enter 2 int: ");
  scanf("%d %d", &a, &b);
  c = max2(a,b);
  printf("%d\n", c);

}
```

Παράμετροι συνάρτησης

- Οι παράμετροι που ορίζονται κατά την υλοποίηση μιας συνάρτησης ονομάζονται **τυπικές παράμετροι**.
- Τα ονόματα τους είναι συμβολικά, έτσι ώστε ο κώδικας της συνάρτησης να μπορεί να επεξεργαστεί τις τιμές που περνιούνται στην κλήση, και ο τύπος τους απλά προσδιορίζει τον **τύπο** των **τιμών** που πρέπει να δοθούν σαν παράμετροι κατά την κλήση.
- Οι τιμές που δίνονται όταν καλείται μια συνάρτηση, για κάθε μια από τις τυπικές παραμέτρους της ονομάζοται **πραγματικές παράμετροι**.
- Για κάθε κλήση, η ίδια συνάρτηση μπορεί να δέχεται **διαφορετικές** πραγματικές παραμέτρους για τις **ίδιες** τυπικές παραμέτρους.

Πέρασμα παραμέτρων καθ’ αποτίμηση

- Για κάθε τυπική παράμετρο τύπου Τ μιας συνάρτησης μπορεί κατά την κλήση να δοθεί σαν πραγματική παράμετρος μια **οποιαδήποτε** τιμή τύπου Τ.
- Αν σαν παράμετρος κλήσης, αντί για συγκεκριμένη τιμή, δοθεί μια έκφραση, τότε αυτή θα **αποτιμηθεί** και σαν πραγματική παράμετρος της κλήσης θα περαστεί το **αποτέλεσμα** της έκφρασης.
- Πιθανές παράμετροι κλήσης για τυπική παράμετρο Τ:
  – κυριολεκτικό τύπου Τ
  – μεταβλητή τύπου Τ
  – έκφραση που αποτιμάται σε τιμή Τ
- Σημείωση: σαν παράμετρος κλήσης μιας συνάρτησης μπορεί να δοθεί μια κλήση συνάρτησης τύπου Τ.

```
/* υπολογισμός μέγιστης τιμής 2 ακεραίων */

#include <stdio.h>

int max2(int x, int y) {
  if (x > y) { return(x); }
  else { return(y); }
}

int main(int argc, char *argv[]) {
  int a,b,c;

  printf("enter 2 int: ");
  scanf("%d %d", &a, &b);
  c = max2(a,b);
  printf("%d\n", c);

}
```

τυπικές παράμετροι, τα **ονόματα** των οποίων χρησιμοποιούνται για να γίνεται αναφορά στις τιμές που θα περαστούν **όταν** κληθεί η συγκεκριμένη συνάρτηση

επιστρεφόμενη **τιμή**

πραγματικές παράμετροι, που είναι οι **τιμές** που περνιούνται στην κλήση συνάρτησης

αποθήκευση **τιμής** που επιστρέφεται για τις τιμές που περάστηκαν ως παράμετροι

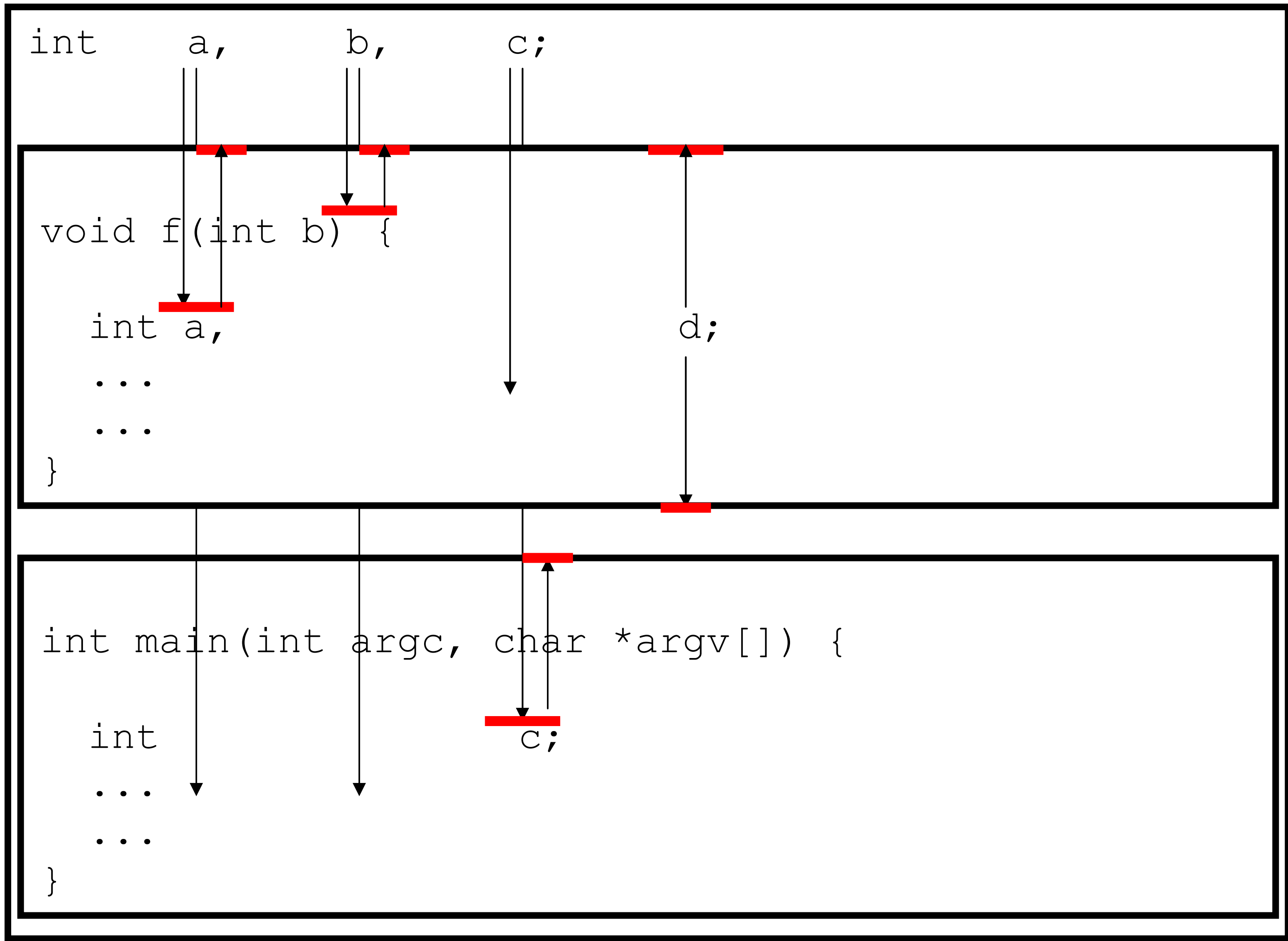
```
#include <stdio.h>

int max2(int x, int y) {
  if (x > y) { return(x); }
  else { return(y); }
}

int main(int argc, char* argv[]) {
  int a,b,c;

  printf("enter 3 int: ");
  scanf("%d %d %d", &a, &b, &c);

  printf("%d\n", max2(a,b));

  printf("%d\n", max2(a+b,25);

  printf("%d\n", max2(a, max2(c,b)) );

}
```

Παράμετροι και τοπικές μεταβλητές

- Κάθε συνάρτηση μπορεί να δηλώνει νέες **δικές της** (τοπικές) μεταβλητές που χρησιμοποιεί για τους **δικούς της** σκοπούς (επιθυμητή επεξεργασία).
- Οι τυπικές παράμετροι αντιστοιχούν σε **ειδικές τοπικές μεταβλητές** που χρησιμοποιούνται για την αποθήκευση (και πρόσβαση) των **πραγματικών** παραμέτρων κατά την κλήση της συνάρτησης.
- <u>Σημείωση</u>: στις αρχικές εκδόσεις της γλώσσας C, η δήλωση των τυπικών παραμέτρων μιας συνάρτησης γινόταν (σχεδόν) όπως για τις τοπικές μεταβλητές.
- Οι τυπικές παράμετροι δεν μπορεί να έχουν το ίδιο όνομα με τοπικές μεταβλητές ούτε το αντίστροφο (διαφορετικά δεν θα υπήρχε τρόπος αναφοράς σε αυτές μέσα από τον κώδικα της συνάρτησης).

Εμβέλεια μεταβλητών

- Οι **τοπικές** μεταβλητές (και τυπικές παράμετροι) ορίζονται στα πλαίσια μιας συνάρτησης και είναι προσπελάσιμες (ορατές) **μόνο** από τον κώδικα της.
- Οι **καθολικές** μεταβλητές ορίζονται στην αρχή του κειμένου του προγράμματος έξω από τις συναρτήσεις (και έξω από την `main`) και είναι προσπελάσιμες (ορατές) μέσα από κάθε συνάρτηση.
- Αν μια τοπική μεταβλητή (ή τυπική παράμετρος) μιας συνάρτησης έχει το ίδιο όνομα με μια καθολική μεταβλητή, τότε **αποκρύπτει** την καθολική μεταβλητή και την καθιστά μη προσπελάσιμη για τον κώδικα της συνάρτησης.


int a, b, c;
void f(int b) {
int a, d;
...
...
}
int main(int argc, char *argv[]) {
int c;
...
...
}

```c
#include <stdio.h>

int a=0,b=0,c=0;

void f(int b) {
  int a,d;
  a=b--; c=a*b; d=c-1;
  printf("f: a=%d, b=%d, c=%d, d=%d\n",a,b,c,d);
}

int main(int argc, char *argv[]) {
  int c=1,d=1;
  c=a+b; b=b+1;
  printf("main: a=%d, b=%d, c=%d, d=%d\n",a,b,c,d);
  f(c);
  printf("main: a=%d, b=%d, c=%d, d=%d\n",a,b,c,d);
  c=a+b; b=b+1;
  f(a);
  printf("main: a=%d, b=%d, c=%d, d=%d\n",a,b,c,d);
}
```

Συναρτήσεις και καθολικές μεταβλητές

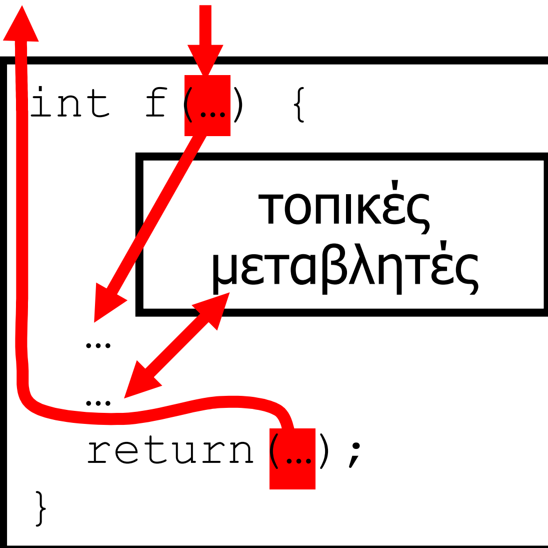
- Η αλλαγή μιας καθολικής μεταβλητής μέσα από μια συνάρτηση συνιστά μια (κλασική) **παρενέργεια**.
- Αυτή η αλλαγή δεν μπορεί να εντοπιστεί χωρίς να διαβάσουμε τον κώδικα της συνάρτησης.
- Φυσικά οι καθολικές μεταβλητές υπάρχουν ακριβώς για αυτό το λόγο, δηλαδή για να επιτρέπουν την επικοινωνία ανάμεσα σε διαφορετικές συναρτήσεις.
- Αυτή η λύση πρέπει να επιλέγεται με **σύνεση** (και να τεκμηριώνεται κατάλληλα, π.χ. σύντομο σχόλιο), όταν το επιθυμητό αποτέλεσμα δεν μπορεί να επιτευχθεί (με απλό τρόπο) με πέρασμα κατάλληλων παραμέτρων και επιστροφή αποτελεσμάτων.

προγραμματισμός με παρενέργειες

```
void f() {
   …
   …
}
```

καθολικές μεταβλητές

προγραμματισμός χωρίς παρενέργειες



```
int f(…) {
   …
   …
   return(…);
}
```

τοπικές μεταβλητές

Διάρκεια ζωής μεταβλητών

- Οι καθολικές μεταβλητές είναι **μόνιμες**, δηλαδή υφίστανται και κρατάνε τις τιμές τους **καθ' όλη** την διάρκεια της εκτέλεσης του προγράμματος.
- Οι τοπικές μεταβλητές είναι **προσωρινές**, δηλαδή υφίστανται και κρατάνε τις τιμές τους **μόνο** όσο κρατά η εκάστοτε εκτέλεση της συνάρτησης.
- **Εξαίρεση:** τοπικές μεταβλητές με τον προσδιορισμό `static` είναι **μόνιμες**, δηλαδή κρατούν την τιμή τους ανάμεσα στις εκτελέσεις της συνάρτησης.
- Οι `static` τοπικές μεταβλητές πρέπει πάντα να αρχικοποιούνται – η εντολή αρχικοποίησης εκτελείται μια φορά όταν η συνάρτηση κληθεί για πρώτη φορά.

```
#include <stdio.h>

void f() {

  static int a=0;

  a++;
  printf("f: a=%d\n",a);
}

int main(int argc, char *argv[]) {

  f();

  f();

  f();

}
```

Εκτέλεση συνάρτησης και δέσμευση μνήμης τοπικών μεταβλητών

Εκτέλεση συνάρτησης

- Όταν μια συνάρτηση A καλεί μια άλλη συνάρτηση B, η εκτέλεση του κώδικα της «**καλούσας**» συνάρτησης σταματά μέχρι να ολοκληρωθεί η εκτέλεση του κώδικα της «**κληθείσας**» συνάρτησης:

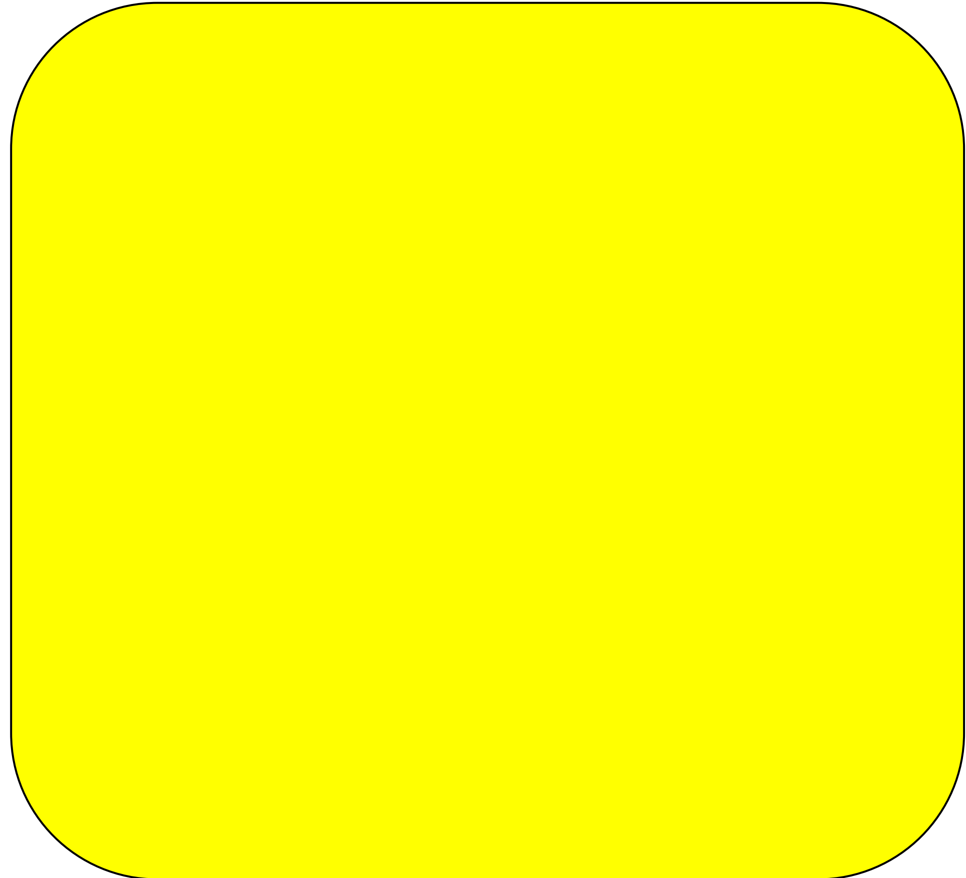
1. Η εκτέλεση της συνάρτησης A σταματά στο σημείο όπου γίνεται η κλήση της συνάρτησης B.
2. Αρχικοποιούνται οι παράμετροι και τοπικές μεταβλητές της συνάρτησης B.
3. Αρχίζει η εκτέλεση του κώδικα της συνάρτησης B (που μπορεί να καλέσει και άλλες συναρτήσεις).
4. Όταν τερματίζεται η εκτέλεση της συνάρτησης B, η εκτέλεση συνεχίζεται στην συνάρτηση A με την αμέσως επόμενη εντολή, μετά την κλήση της B.

Πλαίσιο εκτέλεσης συνάρτησης

- Ο **ίδιος** κώδικας της συνάρτησης εκτελείται κάθε φορά σε ένα **διαφορετικό** πλαίσιο εκτέλεσης.
- Το πλαίσιο εκτέλεσης **δημιουργείται** (εκ νέου) πριν αρχίσει η εκτέλεση του κώδικα της συνάρτησης και **καταστρέφεται** όταν ολοκληρωθεί η εκτέλεση της.
- Το πλαίσιο εκτέλεσης χρησιμεύει για την αποθήκευση των τοπικών μεταβλητών και των πραγματικών παραμέτρων για την **συγκεκριμένη** κλήση.
- Για κάθε κλήση, δημιουργείται ένα **καινούργιο** και **ξεχωριστό** πλαίσιο εκτέλεσης, που **δεν** έχει σχέση με προηγούμενα πλαίσια εκτέλεσης.

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση main

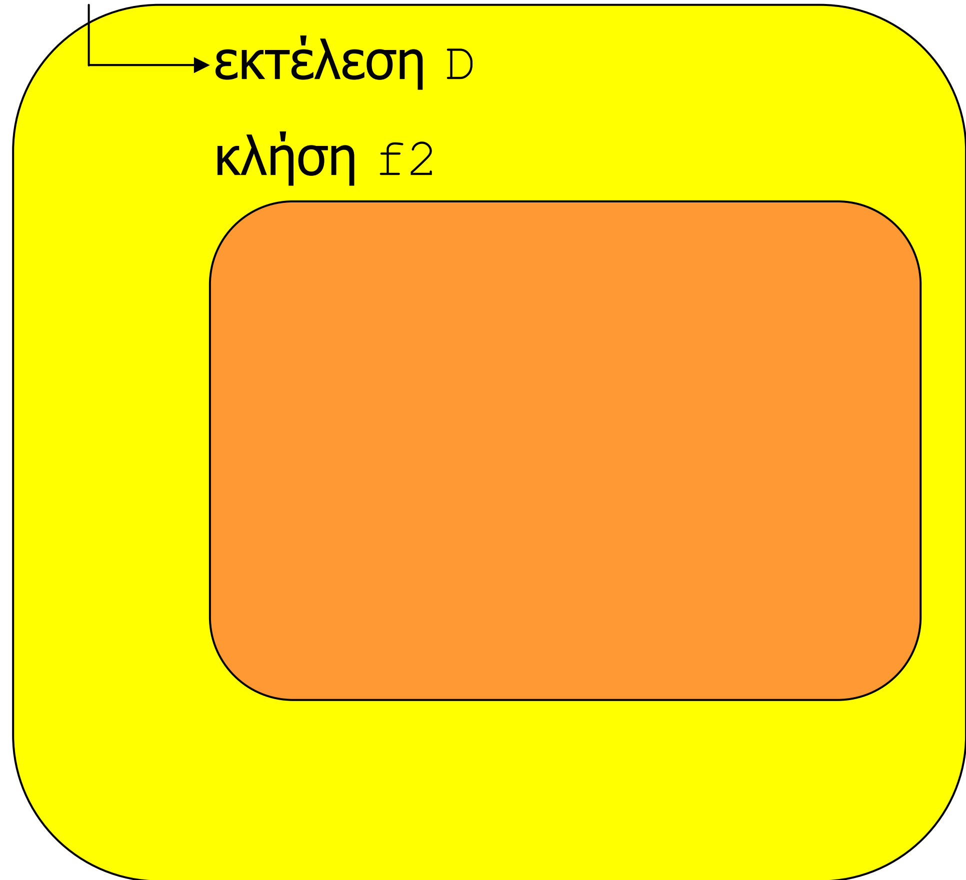


```
void f1(…) {
 <A>
}

void foo2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση `main`

εκτέλεση `D`

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```



```
void f1(…) {
 <A>
}

void foo2(…) {
  <B>
  f1(…);
  <C>
}

int main(…){
  <D>
  f2(…);
  <E>
}
```

κλήση main

εκτέλεση D

κλήση f2

εκτέλεση B

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση main

εκτέλεση D

κλήση f2

εκτέλεση B

κλήση f1

```
void f1(…) {
  <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση `main`

εκτέλεση `D`

κλήση `f2`

εκτέλεση `B`

κλήση `f1`

εκτέλεση `A`

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση main

εκτέλεση D

κλήση f2

εκτέλεση B

κλήση f1

εκτέλεση A

επιστροφή

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση main

εκτέλεση D

κλήση f2

εκτέλεση B

κλήση f1

εκτέλεση A

επιστροφή

εκτέλεση C

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```

κλήση main

εκτέλεση D

κλήση f2

εκτέλεση B

κλήση f1

εκτέλεση A

επιστροφή

εκτέλεση C

επιστροφή

```
void f1(…) {
 <A>
}

void f2(…) {
  <B>
  f1(…);
  <C>
}

int main(…){
  <D>
  f2(…);
  <E>
}
```

κλήση main

εκτέλεση D

κλήση f2

εκτέλεση B

κλήση f1

εκτέλεση A

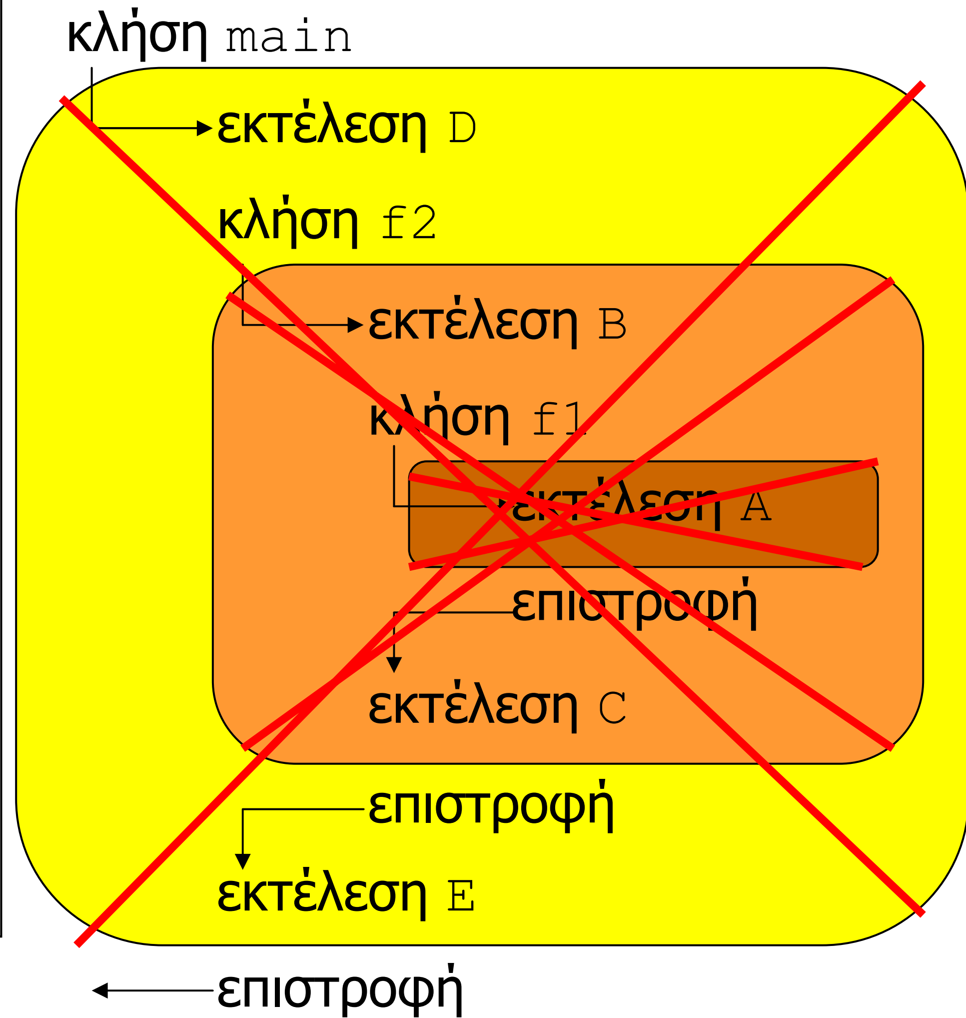
επιστροφή

εκτέλεση C

επιστροφή

εκτέλεση E

```
void f1(…) {
 <A>
}

void f2(…) {
   <B>
   f1(…);
   <C>
}

int main(…){
   <D>
   f2(…);
   <E>
}
```



Δέσμευση μνήμης μεταβλητών

- Η μνήμη των **μόνιμων** (καθολικών και τοπικών) μεταβλητών είναι **στατική**, και δεσμεύεται για όλη την διάρκεια της εκτέλεσης του προγράμματος.
- Η μνήμη των **προσωρινών** (τοπικών) μεταβλητών μιας συνάρτησης είναι **δυναμική**, και **δεσμεύεται / αποδεσμεύεται μαζί με το αντίστοιχο πλαίσιο εκτέλεσης**.
- Η διαχείριση της τοπικής μνήμης των συναρτήσεων γίνεται μέσω του μηχανισμού της **στοίβας**.
- Υποστηρίζεται η **εναλλάξ** ή/και **αλυσιδωτή** εκτέλεση συναρτήσεων με τους **λιγότερους** δυνατούς πόρους και την **μεγαλύτερη** δυνατή ταχύτητα εκτέλεσης.

Στοίβα

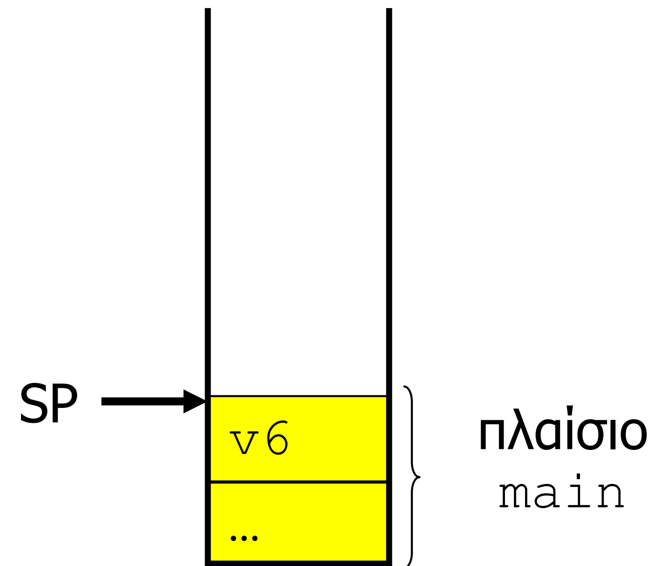
- Δεσμεύεται ένα (μεγάλο) συνεχόμενο τμήμα μνήμης, που χρησιμοποιείται σύμφωνα με την λογική της **στοίβας** (LIFO queue) για την αποθήκευση των τιμών των παραμέτρων και τοπικών μεταβλητών (και καταχωρητών της CPU) των συναρτήσεων.
- Το όριο της μνήμης της στοίβας που χρησιμοποιείται ανά πάσα στιγμή υποδεικνύεται από ένα ειδικό δείκτη (που διαχειρίζεται το περιβάλλον εκτέλεσης) που ονομάζεται stack pointer.
- Κάθε φορά που γίνεται μια νέα κλήση και κάθε φορά που τερματίζεται μια κλήση, η τιμή του stack pointer αλλάζει ώστε να δείχνει στο τρέχων πλαίσιο εκτέλεσης.

```
int v1;

void f1(int v2) {
   int v3;
   …
}

void f2(int v4) {
   int v5;
   …
   f1(…);
   …
}

int main(…){
   int v6;
   …
   f2(…);
   …
}
```

κλήση `main`

στατική μνήμη

στοίβα



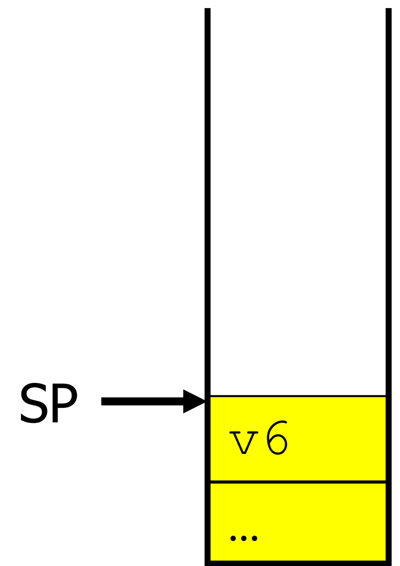
```
int v1;

void f1(int v2) {
  int v3;
  …
}

void f2(int v4) {
  int v5;
  …
  f1(…);
  …
}

int main(…){
  int v6;
  …
  f2(…);
  …
}
```

εκτέλεση `main`

στατική μνήμη

στοίβα

SP



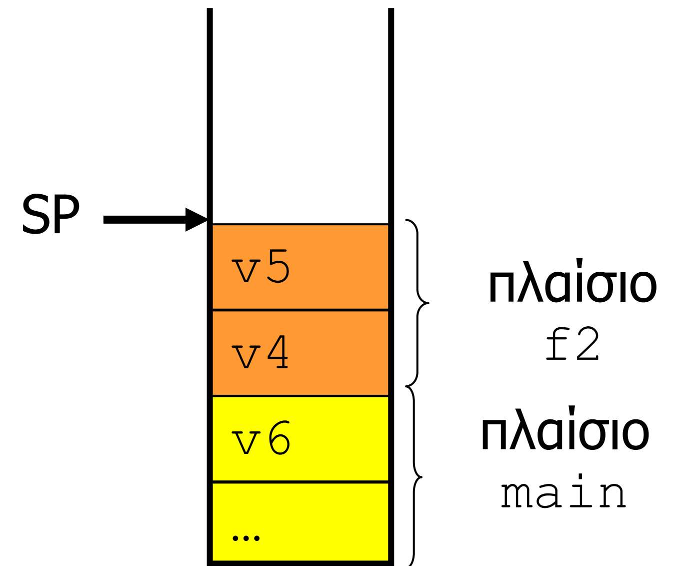
πλαίσιο `main`

```
int v1;

void f1(int v2) {
   int v3;
   …
}

void f2(int v4) {
   int v5;
   …
   f1(…);
   …
}

int main(…){
   int v6;
   …
   f2(…);
   …
}
```

κλήση `f2`

στατική μνήμη

στοίβα

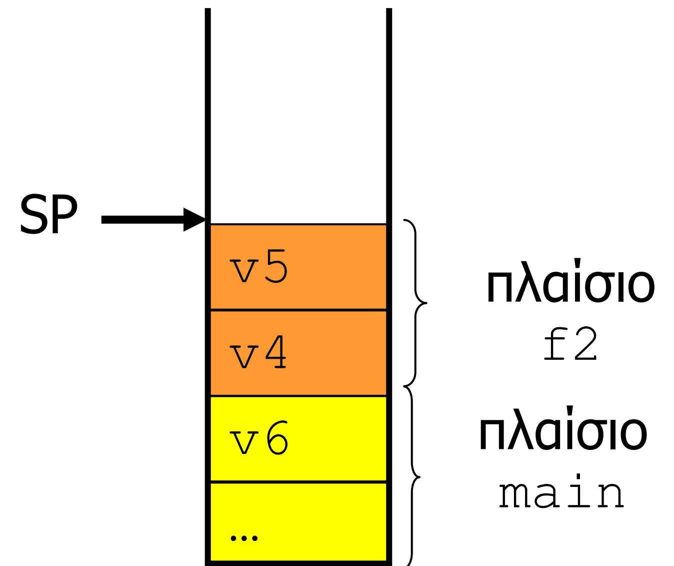


```
int v1;

void f1(int v2) {
   int v3;
   …
}

void f2(int v4) {
   int v5;
   …
   f1(…);
   …
}

int main(…){
   int v6;
   …
   f2(…);
   …
}
```

εκτέλεση `f2`

στατική μνήμη

στοίβα

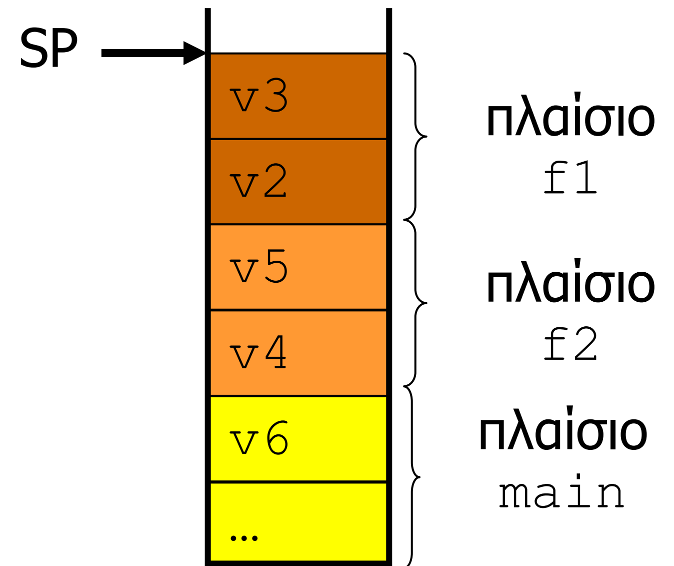


```
int v1;

void f1(int v2) {
   int v3;
   …
}

void f2(int v4) {
   int v5;
   …
   f1(…);
   …
}

int main(…){
   int v6;
   …
   f2(…);
   …
}
```

κλήση `f1`

στατική μνήμη

στοίβα

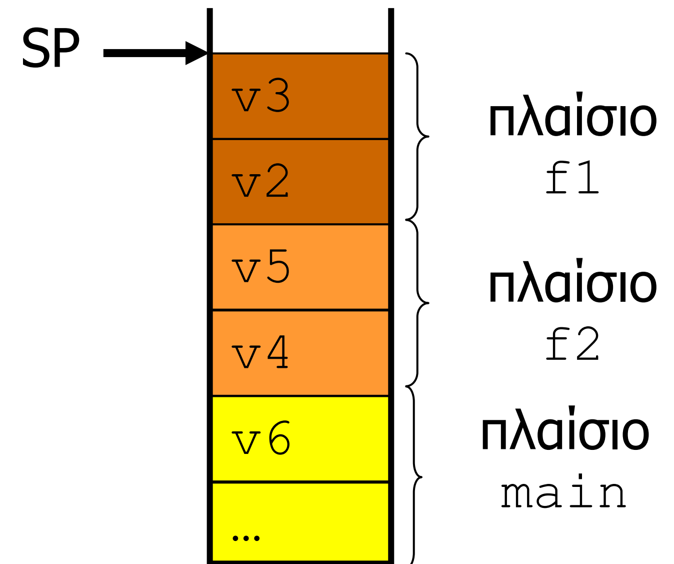


```
int v1;

void f1(int v2) {
   int v3;
   …
}

void f2(int v4) {
   int v5;
   …
   f1(…);
   …
}

int main(…){
   int v6;
   …
   f2(…);
   …
}
```

εκτέλεση f1

στατική μνήμη

στοίβα

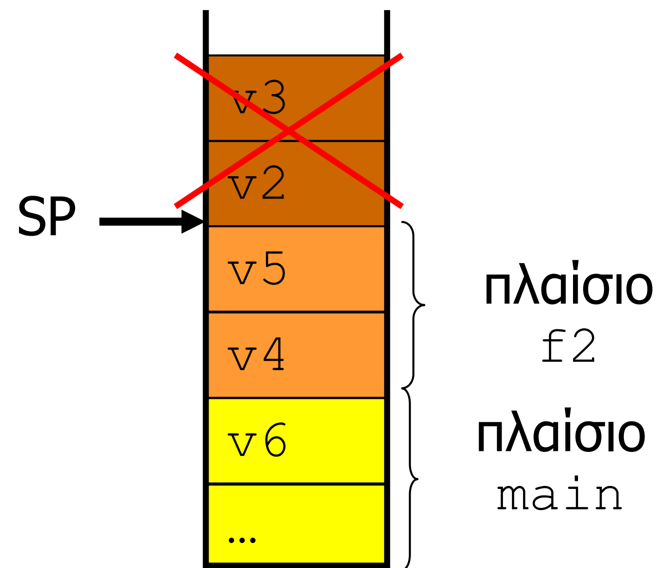


```
int v1;

void f1(int v2) {
  int v3;
  …
}

void f2(int v4) {
  int v5;
  …
  f1(…);
  …
}

int main(…){
  int v6;
  …
  f2(…);
  …
}
```

επιστροφή f1

στατική μνήμη

στοίβα

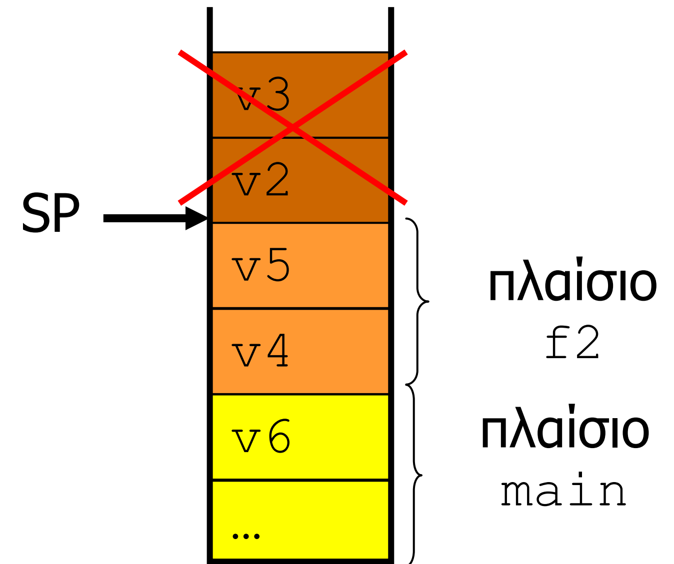


```
int v1;

void f1(int v2) {
  int v3;
  …
}

void f2(int v4) {
  int v5;
  …
  f1(…);
  …
}

int main(…){
  int v6;
  …
  f2(…);
  …
}
```

εκτέλεση `f2`

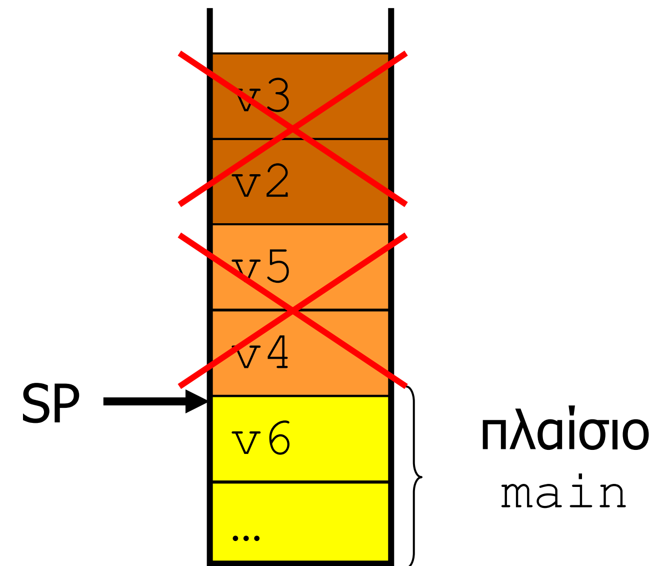
στατική μνήμη

στοίβα



```
int v1;

void f1(int v2) {
   int v3;
   …
}

void f2(int v4) {
   int v5;
   …
   f1(…);
   …
}

int main(…){
   int v6;
   …
   f2(…);
   …
}
```

επιστροφή `f2`

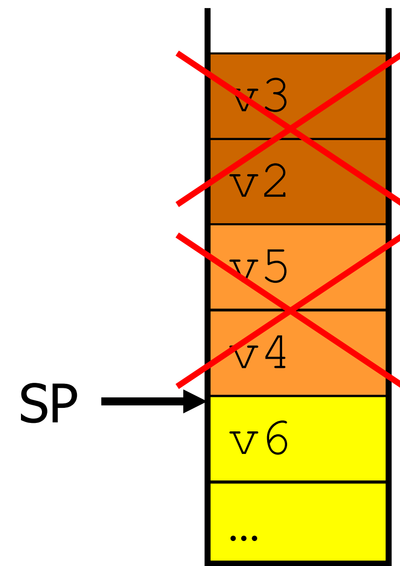


```
int v1;

void f1(int v2) {
  int v3;
  …
}

void f2(int v4) {
  int v5;
  …
  f1(…);
  …
}

int main(…){
  int v6;
  …
  f2(…);
  …
}
```

εκτέλεση `main`

στατική μνήμη

στοίβα



πλαίσιο `main`

Υπερχείλιση στοίβας

- Το μέγεθος της στοίβας ενός προγράμματος είναι (συνήθως) περιορισμένο – γιατί;
- Υπάρχει περίπτωση ένα πρόγραμμα να εξαντλήσει την μνήμη της στοίβας του, με αποτέλεσμα αυτή να υπερχειλίσει (stack overflow):

1. Γίνονται πολλές αλυσιδωτές κλήσεις συνάρτησης.
2. Το μέγεθος των τοπικών μεταβλητών μιας συνάρτησης είναι μεγάλο, και δεν χωρά στην στοίβα.

- Τότε το πρόγραμμα τερματίζεται με μήνυμα λάθους (παρόμοιο με αυτό στην περίπτωση της πρόσβασης σε μη επιτρεπτή θέση μνήμη, π.χ. μέσω δείκτη).

Συναρτήσεις και δείκτες

Αλλαγή του «εξωτερικού» περιβάλλοντος

- Αν σαν παράμετρος μιας συνάρτησης δοθεί μια μεταβλητή, σαν **πραγματική** παράμετρος θα περαστεί **η τιμή** της.
- Το πέρασμα παραμέτρων είναι **καθ' αποτίμηση**.
- Η πραγματικές παράμετροι (τιμές) αποθηκεύονται σε τοπική μνήμη – προσωρινές μεταβλητές με τα ονόματα των αντίστοιχων τυπικών παραμέτρων.
- Αν ο κώδικας της συνάρτησης αλλάξει την τιμή μιας τυπικής παραμέτρου, στην πραγματικότητα αλλάζει την τιμή της αντίστοιχης τοπικής μεταβλητής, **όχι** της μεταβλητής που «περάστηκε» σαν παράμετρος.

```c
#include <stdio.h>

void inc0(int a) {
  a=a+1;
}

int main(int argc, char *argv[]) {
  int a;

  a=5;

  inc0(a);
  printf("a=%d\n", a);

  inc0(a);
  printf("a=%d\n", a);
}
```

οποιαδήποτε αλλαγή στην a δεν επηρεάζει το «εξωτερικό» περιβάλλον της κλήσης

σαν **πραγματική** παράμετρος (για την τυπική παράμετρο a) περνιέται η **τιμή** 5

Δείκτες ως παράμετροι συναρτήσεων

- Αν σαν παράμετρος μιας συνάρτησης δοθεί μια **διεύθυνση**, τότε η συνάρτηση **μπορεί** (προφανώς) να αλλάξει τα περιεχόμενα σε αυτή την θέση μνήμης.
- Η αντίστοιχη τυπική παράμετρος πρέπει να δηλωθεί ως **δείκτης-σε-T**, και ο κώδικας της συνάρτησης πρέπει να χρησιμοποιεί αυτή την τιμή αντίστοιχα, όπως ακριβώς απαιτείται σε μια μεταβλητή δείκτη.
- Αντίστοιχα, όταν καλείται η συνάρτηση, σαν παράμετρος πρέπει να δίνεται μια **διεύθυνση** που αντιστοιχεί σε ένα αντικείμενο (μεταβλητή) τύπου T.
- Σημείωση: ένα κλασικό λάθος είναι το πέρασμα της τιμής αντί της διεύθυνσης της μεταβλητής.

```
#include <stdio.h>

void inc0(int a) {
  a=a+1;
}

void inc1(int *a) {
  *a=*a+1;
}

int main(int argc, char *argv[]) {
  int a1,a2;

  printf("enter int value: ");
  scanf("%d", &a1);
  a2=a1;
  inc0(a1); inc1(&a2);
  printf("a1=%d, a2=%d\n", a1, a2);
  inc0(a1); inc1(&a2);
  printf("a1=%d, a2=%d\n", a1, a2);
}
```

δείκτης!

διεύθυνση!

```
#include <stdio.h>

void swap(int *a, int *b) {
  int tmp;

  tmp=*a; *a=*b; *b=tmp;

}

int main(int argc, char *argv[]) {
  int i,j;

  printf("enter 2 int values: ");
  scanf("%d %d", &i, &j);
  printf("i=%d, j=%d\n", i, j);

  swap(&i,&j);

  printf("i=%d, j=%d\n", i, j);

}
```

Πίνακες ως παράμετροι συναρτήσεων

- Η τυπική παράμετρος δηλώνεται ως πίνακας.
- Δεν είναι υποχρεωτικό να προσδιοριστεί το μέγεθος του πίνακα – μπορεί να δοθεί και ως παράμετρος.
- Κατά την κλήση, σαν πραγματική παράμετρος περνιέται (στην στοίβα) η **διεύθυνση** (του πρώτου στοιχείου) του πίνακα, **όχι τα περιεχόμενα του**.
- Ο κώδικας της συνάρτησης μπορεί να αλλάξει τις τιμές των στοιχείων του πίνακα, και αυτές οι αλλαγές θα **«παραμείνουν»** αφού επιστρέψει η συνάρτηση.
- **Σημείωση:** μια τυπική παράμετρος τύπου δείκτη-σε-Τ μπορεί να ερμηνευτεί ως η αρχή ενός πίνακα από Τ.

```
/* αλλαγή σε κεφαλαία */
#include <stdio.h>
#define N 16

void smallToCapitals(char s[]) {
  int i;

  for (i=0; s[i] != '\0'; i++) {
    if ( (s[i] >= 'a') && (s[i] <= 'z') ) {
      s[i] = 'A' + s[i] - 'a';
    }
  }
}

int main(int argc, char *argv[]) {
  char str[N];

  scanf("%15s",str);
  printf("%s\n",str);
  smallToCapitals(str);
  printf("%s\n",str);
}
```

```
/* αλλαγή σε κεφαλαία */
#include <stdio.h>
#define N 16

void smallToCapitals(char *s) {
  int i;

  for (i=0; s[i] != '\0'; i++) {
    if ( (s[i] >= 'a') && (s[i] <= 'z') ) {
      s[i] = 'A' + s[i] - 'a';
    }
  }
}

int main(int argc, char *argv[]) {
  char str[N];

  scanf("%15s",str);
  printf("%s\n",str);
  smallToCapitals(str);
  printf("%s\n",str);
}
```

```c
/* αλλαγή σε κεφαλαία */
#include <stdio.h>
#define N 16

void smallToCapitals(char s[]) {
  int i;

  for (i=0; s[i] != '\0'; i++) {
    if ( (s[i] >= 'a') && (s[i] <= 'z') ) {
      s[i] = 'A' + s[i] - 'a';
    }
  }
}

int main(int argc, char *argv[]) {
  char str[N];

  scanf("%7s",&str[8]);
  printf("%s\n",str);
  smallToCapitals(&str[8]);
  printf("%s\n",&str[8]);
}
```

```c
/* ανάγνωση και ταξινόμηση ακεραίων */
#include <stdio.h>
#define N 10

void swap(int *a, int *b) { … }

void sort(int t[], int len) {
  int i,j;
  for (i=0; i<len; i++) {
    for (j=i; j<len; j++) {
      if (t[i]>t[j]) { swap(&t[i],&t[j]); }
    }
  }
}

int main(int argc, char *argv[]) {
  int buf[N],i;
  for (i=0; i<N; i++) { scanf("%d", &buf[i]); }
  sort(buf,N);
  for (i=0; i<N; i++) { printf("%d ", buf[i]); }
  printf("\n");
}
```

```
/* ανάγνωση και ταξινόμηση ακεραίων */
#include <stdio.h>
#define N 10

void swap(int *a, int *b) { … }

void sort(int *t, int len) {
  int i,j;
  for (i=0; i<len; i++) {
    for (j=i; j<len; j++) {
      if (t[i]>t[j]) { swap(&t[i],&t[j]); }
    }
  }
}

int main(int argc, char *argv[]) {
  int buf[N],i;
  for (i=0; i<N; i++) { scanf("%d", &buf[i]); }
  sort(buf,N);
  for (i=0; i<N; i++) { printf("%d ", buf[i]); }
  printf("\n");
}
```

```
/* ανάγνωση και ταξινόμηση ακεραίων */
#include <stdio.h>
#define N 10

void swap(int *a, int *b) { … }

void sort(int t[], int len) {
  int i,j;
  for (i=0; i<len; i++) {
    for (j=i; j<len; j++) {
      if (t[i]>t[j]) { swap(&t[i],&t[j]); }
    }
  }
}

int main(int argc, char *argv[]) {
  int buf[N],i;
  for (i=0; i<N; i++) { scanf("%d", &buf[i]); }
  sort(buf,N/2); sort(&buf[N/2],N/2);
  for (i=0; i<N; i++) { printf("%d ", buf[i]); }
  printf("\n");
}
```

Σχόλιο

- Η αλλαγή της τιμής μιας εξωτερικής μεταβλητής μέσα από συνάρτηση, μέσω δείκτη, μπορεί να θεωρηθεί ως μια μορφή παρενέργειας.
- Η διαφορά σε σχέση με την αλλαγή καθολικών μεταβλητών είναι ότι για να γίνει αυτό πρέπει να υπάρχει **τυπική παράμετρος** που να έχει **δηλωθεί** σαν **δείκτης**, και όταν γίνεται η κλήση να δίνεται σαν παράμετρος η **διεύθυνση** της μεταβλητής.
- Αυτό είναι ορατό κατά την ανάγνωση του κώδικα, συνεπώς δεν αποτελεί «πραγματική» παρενέργεια.
- Το πρόθεμα `const` σε μια τυπική παράμετρο δείκτη, δηλώνει ότι η συνάρτηση **δεν αλλάζει** τα περιεχόμενα που βρίσκονται σε αυτή τη διεύθυνση.

Δείκτες ως αποτέλεσμα συναρτήσεων

- Μια συνάρτηση μπορεί να δηλωθεί έτσι ώστε να επιστρέφει σαν αποτέλεσμα ένα δείκτη-σε-T.
- Χρειάζεται προσοχή ώστε η τιμή που επιστρέφεται να αντιστοιχεί σε μεταβλητή (μνήμη) που είναι **μόνιμη**.
- Η επιστροφή της διεύθυνσης μιας (συμβατικής) προσωρινής τοπικής μεταβλητής μιας συνάρτησης είναι **προγραμματιστικό λάθος**, καθώς αυτή μπορεί να **μην** υφίσταται μετά την κλήση της συνάρτησης (καταστρέφεται το πλαίσιο εκτέλεσης).
- Αυτό δεν εντοπίζεται από το μεταφραστή αλλά οδηγεί (αν είμαστε τυχεροί, και όχι πάντα) σε τερματισμό της εκτέλεσης του προγράμματος.

```
#include <stdio.h>

int *add(int a, int b) {
  int c;
  c=a+b;
  return(&c); /* αυτό είναι λάθος ! */
}

int f(int a, int b) {
  int c = 0;
}

int main(int argc, char *argv[]) {
  int a,b,*c;

  printf("enter 2 int values: ");
  scanf("%d %d", &a, &b);
  c=add(a,b);
  f(a,b);
  printf("the result is %d\n", *c);

}
```

Χρήση συναρτήσεων

Χρησιμοποιούμε συναρτήσεις:

- Όταν το πρόγραμμα αποτελείται από μια ομάδα εντολών που επαναλαμβάνεται πολλές φορές, και η οποία μπορεί να **παραμετροποιηθεί** έτσι ώστε να γράψουμε τον κώδικα μια μοναδική φορά.
- Όταν επιθυμούμε, για λόγους καλύτερης δόμησης, να σπάσουμε ένα μεγάλο τμήμα κώδικα σε περισσότερα, νοηματικά ανεξάρτητα, κομμάτια.
- Όταν επιθυμούμε να έχουμε ανεξάρτητα τμήματα κώδικα σε διαφορετικά αρχεία ή/και με δυνατότητα ξεχωριστής μετάφρασης (π.χ. βιβλιοθήκες).

Χρήση μεταβλητών

- Κάθε μεταβλητή εξυπηρετεί ένα συγκεκριμένο σκοπό και ονομάζεται αντίστοιχα (χωρίς υπερβολές).
- Μεταβλητές με «ειδικό» ρόλο σχολιάζονται ώστε να διευκολύνουν την ανάγνωση του κώδικα.
- Ιδανικά, η λειτουργία κάθε συνάρτησης πρέπει να είναι κατανοητή «από μόνη της», χωρίς να γνωρίζουμε το τι (ακριβώς) κάνει ο υπόλοιπος κώδικας του προγράμματος.
- Οι καθολικές μεταβλητές πρέπει να χρησιμοποιούνται με σύνεση, μόνο όταν απλουστεύουν τον κώδικα.

Παρένθεση
(συναρτήσεις με άγνωστο αριθμό παραμέτρων)

Συναρτήσεις με <u>άγνωστο</u> αριθμό παραμέτρων

- Μπορεί να υλοποιηθούν συναρτήσεις με άγνωστο (μεταβλητό) αριθμό παραμέτρων, ορίζοντας ως τελευταία (αλλά όχι πρώτη) παράμετρο το "…".
- Αυτό είναι βολικό σε περιπτώσεις που ο αριθμός των παραμέτρων δεν μπορεί να προσδιοριστεί εκ των προτέρων ή είναι επιθυμητό η συνάρτηση να μπορεί να καλείται με μεταβλητό αριθμό παραμέτρων.
- Ο συνολικός αριθμός των παραμέτρων πρέπει να μπορεί να υπολογίζεται με βάση τις τιμές των (γνωστών) παραμέτρων της συνάρτησης.
- Κλασικό παράδειγμα: `printf` και `scanf`.

Βασικές λειτουργίες βιβλιοθήκης

- Για την πρόσβαση στις παραμέτρους που δόθηκαν κατά την κλήση, χρησιμοποιούνται οι μάκρο-εντολές (από τη βιβλιοθήκη `stdarg`):
  - `va_list`: ο τύπος της λίστας των παραμέτρων
  - `void va_start(va_list ap, last)`: αρχικοποιεί την μεταβλητή `ap` ώστε να δείχνει στην πρώτη άγνωστη παράμετρο μετά την τελευταία γνωστή παράμετρο της συνάρτησης `last`
  - `type va_arg(va_list ap, type)`: επιστρέφει την τιμή της παραμέτρου στην οποία δείχνει η `ap` ερμηνεύοντας την σύμφωνα με τον τύπο `type` και μεταθέτει το `ap` στην επόμενη θέση
  - `void va_end(va_list ap)`: καλείται πριν τον τερματισμό της συνάρτησης

```
#include<stdio.h>
#include<stdarg.h>

void sum(int nof_args, ...) {
  int i,s; va_list ap;

  va_start(ap,num_args);
  for(i=0,s=0; i<nof_args; i++) {
    s=s+va_arg(ap,int);
  }
  va_end(ap);
}

int main(int argc, char *argv[]) {
  printf("sum of 1..7 is %d\n",sum(7,1,2,3,4,5,6,7));
  printf("sum of 1..5 is %d\n",sum(5,5,7,9,11,13));
}
```

Παρένθεση
(συναρτήσεις με ανοιχτό αριθμό παραμέτρων)

Δείκτες σε συναρτήσεις και συναρτήσεις ως τύποι

Η συνάρτηση ως τύπος

- Η έννοια του «δείκτη-σε-» μπορεί να εφαρμοστεί **και σε συναρτήσεις**.
- Ένας τύπος **δείκτης-σε-συνάρτηση** ορίζει τις παραμέτρους και το αποτέλεσμα των συναρτήσεων.
- Η υλοποίηση μιας συνάρτησης στην ουσία δηλώνει ένα **σταθερό δείκτη** (στην διεύθυνση του κώδικα), ο τύπος του οποίου καθορίζεται **έμμεσα** μέσω των παραμέτρων και του αποτελέσματος της συνάρτησης.
- Υποστηρίζονται **μεταβλητές** τύπου συνάρτησης.
- Υποστηρίζονται συναρτήσεις που δέχονται σαν **παράμετρο** ένα τύπο συνάρτησης –ονομάζονται και μετα-συναρτήσεις (meta-functions) ή συναρτήσεις υψηλότερου βαθμού (higher order functions).

```
int a;                      /* μεταβλητή ακεραίου */
int *b;                     /* μεταβλητή δείκτης σε ακέραιο */
typedef int *c;             /* τύπος «δείκτης σε ακέραιο» */
c d;                        /* μεταβλητή δείκτης σε ακέραιο */

int f1(int x);              /* δήλωση συνάρτησης */

int (*f2)(int);             /* μεταβλητή δείκτης σε
                               συνάρτηση με παράμετρο int
                               που επιστρέφει int */

typedef int (*f3)(int);    /* τύπος «δείκτης σε
                              συνάρτηση με παράμετρο int
                              που επιστρέφει int» */

int f4(int (*f5)(int));    /* δήλωση συνάρτησης με παράμετρο
                                 δείκτη σε
                                 συνάρτηση με παράμετρο int
                                 που επιστρέφει int
                              που επιστρέφει int */
```

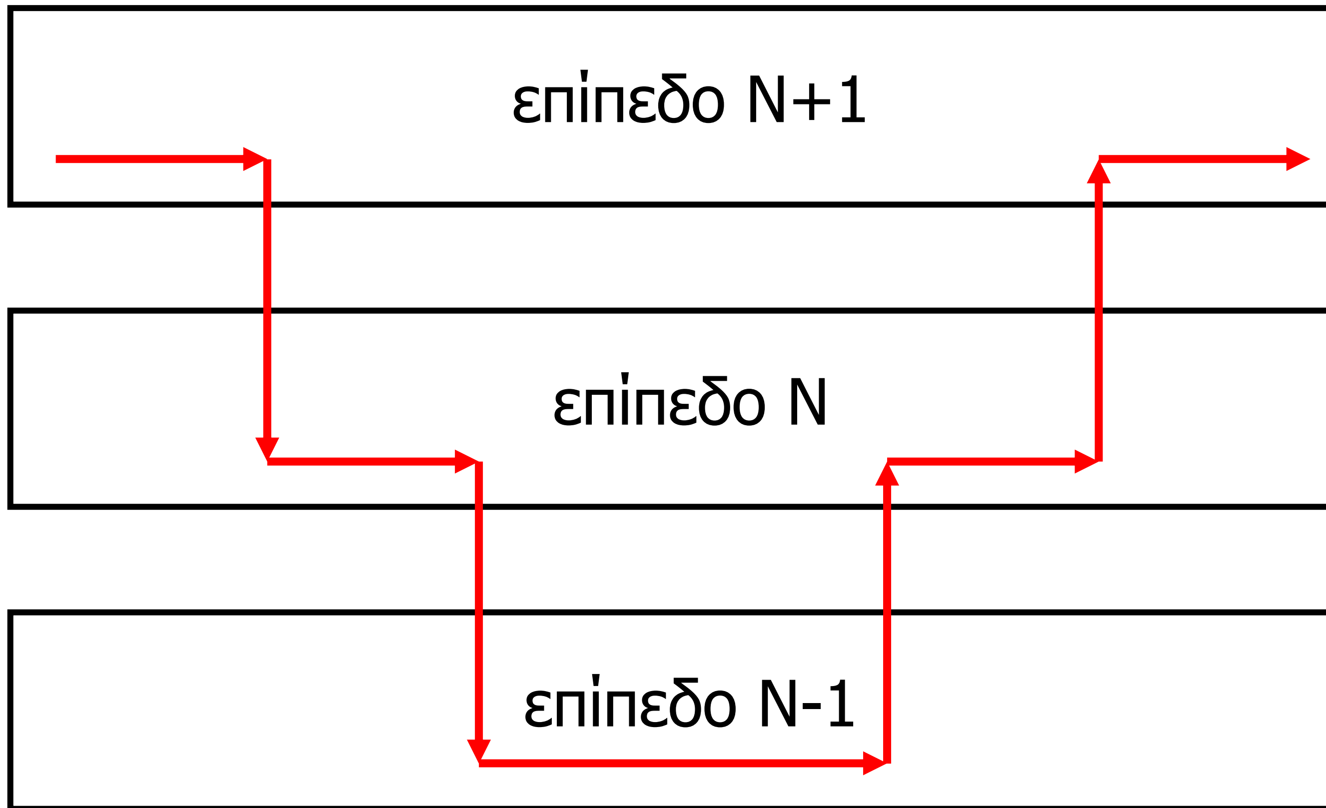
Χρησιμότητα

- Σχήματα αφηρημένης λειτουργικότητας: υλοποίηση λειτουργικότητας με **παραμετροποιημένο** τρόπο, όπου ένα μέρος **του κώδικα** προσδιορίζεται «δυναμικά» και **εκ των υστέρων**, μέσω μιας συνάρτησης που περνιέται σαν παράμετρος.
- Αντίστροφες κλήσεις: κλήση κώδικα που γράφεται την χρονική στιγμή T μέσα από κώδικα που έχει **ήδη γραφτεί** (μεταφραστεί) την χρονική στιγμή T’<T.
- Αντικειμενοστραφής προγραμματισμός: δομές δεδομένων που διαθέτουν **δικό τους** κώδικα (δείκτες στον κώδικα) επεξεργασίας των περιεχομένων τους.

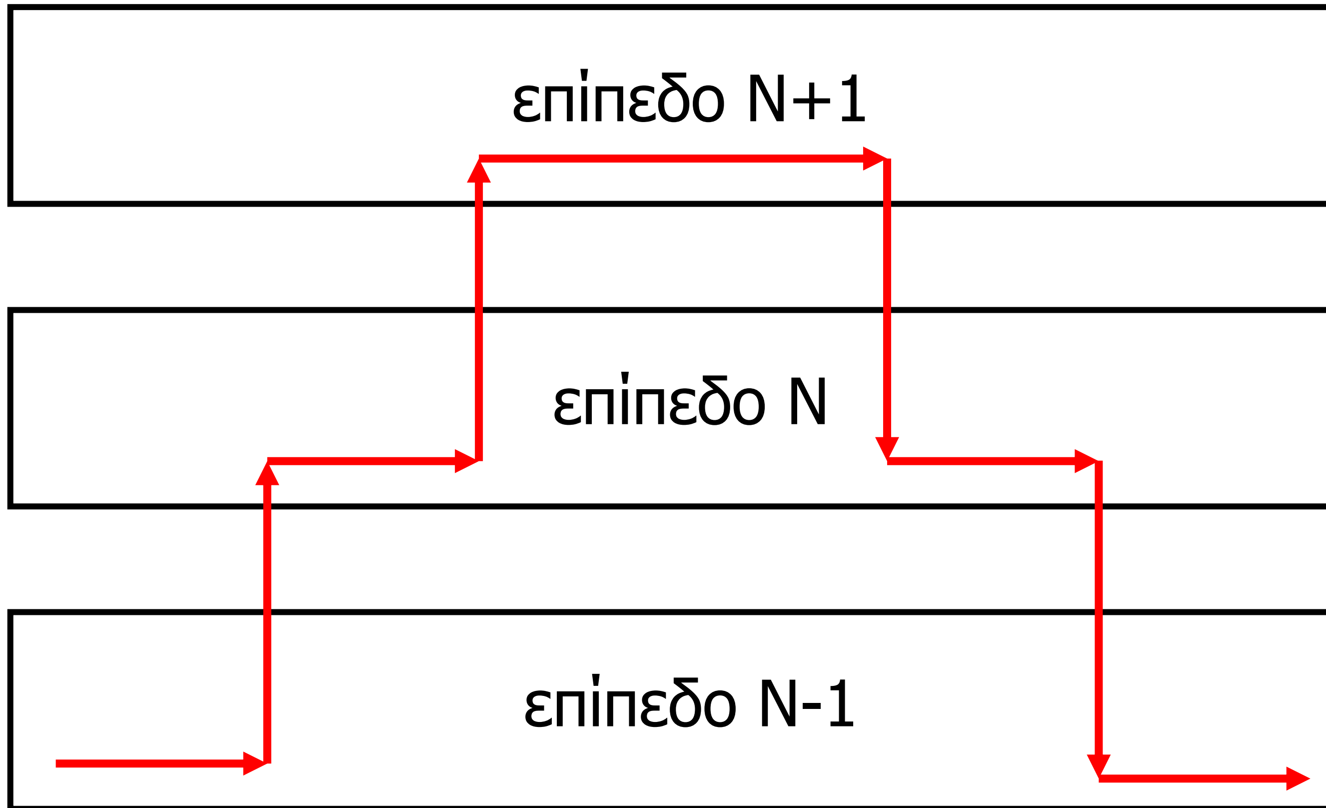
Αντίστροφες κλήσεις (προς τα πάνω)

- Ας θεωρήσουμε ο κώδικας τοποθετείται σε **επίπεδα** ανάλογα με την **χρονική στιγμή** υλοποίησης του (σε αναλογία με τα γεωλογικά στρώματα της γης).
- Ο συμβατικός τρόπος προγραμματισμού είναι να βασιζόμαστε σε λειτουργίες που ήδη υπάρχουν (κλήση συναρτήσεων βιβλιοθήκης όπως η printf).
- Σε αυτή τη περίπτωση, η κλήση κώδικα επιπέδου N γίνεται μέσα από κώδικα επιπέδου N+1, δηλαδή κλήση **από πάνω προς τα κάτω** (downcall).
- Αν η κλήση κώδικα επιπέδου N γίνεται μέσα από κώδικα επιπέδου N-1, δηλαδή **από κάτω προς τα πάνω** (upcall), τότε έχουμε αντίστροφη κλήση.

αλυσιδωτές κλήσεις **προς τα κάτω** (downcall chain)



αλυσιδωτές κλήσεις **προς τα πάνω** (upcall chain)



```
/* κώδικας επιπέδου N+1 */

…

int main(int argc, char *argv[]) {
   …
   res=f1(…);
   …
}
```

downcall

επίπεδο N+1

```
/* κώδικας επιπέδου N */

int f1(…) {
   …
}
```

επίπεδο N

```
/* κώδικας επιπέδου N+1 */

int f2(int i) {
   …
}

int main(int argc, char *argv[]) {
   …
   res=f1(f2,…);
   …
}
```

downcall

upcall

επίπεδο N+1

```
/* κώδικας επιπέδου N */

int f1(int (*f)(int), …) {
   …
   res=(*f)(var);
   …
}
```

επίπεδο N

επίπεδο N+1

```
int f1(int x) {…}

int f2(int x) {…}

int main(int argc, char *argv[]) {
  int beg,end,m;
  printf("enter range: ");
  scanf("%d %d", &beg, &end);
  m = findMax(f1,beg,end);
  printf("max for f1 is %d at %d\n", f1(m), m);
  m = findMax(f2,beg,end);
  printf("max for f2 is %d at %d\n", f2(m), m);
}
```

επίπεδο N

```
/* αφηρημένος υπολογισμός μέγιστης τιμής */

int findMax(int(*f)(int), int beg, int end) {
  int x,v,maxv,maxx;
  maxv = (*f)(beg); maxx=beg;
  for (x=beg+1; x<end; x++) {
    v = (*f)(x);
    if (v>maxv) { maxv = v; maxx = x; }
  }
  return(maxx);
}
```

Σχόλιο

- Η έννοια του «επίπεδου» μπορεί να είναι περισσότερο συμβολική παρά ουσιαστική, π.χ. μπορούμε να γράψουμε κώδικα που χρησιμοποιεί δείκτες σε συναρτήσεις που βρίσκονται μέσα στο ίδιο αρχείο.
- Ο λόγος που χρησιμοποιούμε δείκτες σε συναρτήσεις (που γνωρίζουμε ακριβώς ποιές είναι και τι κάνουν αφού τις υλοποιούμε οι ίδιοι μέσα στο ίδιο αρχείο) παραμένει ο ίδιος: προγραμματισμός μιας **γενικής** λειτουργικότητας «μια για πάντα».