Προγραμματισμός Ι (ΗΥ120)

Διάλεξη 9:
Συναρτήσεις

Χρήστος Δ. Αντωνόπουλος
2/11/2011

Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων
Πανεπιστήμιο Θεσσαλίας

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).

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.
  - Και ο μεταγλωττιστής εμφανίζει μια προειδοποίηση.
- 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 αποτίμησης** 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.


```
/* 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 in1, in2, max;

  printf("enter 2 ints: ");
  scanf("%d %d", &in1, &in2);
  max = max2(in1, in2);
  printf("max = %d\n", max);
  return(0);
}
```


```
/* 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 in1, in2, max;

  printf("enter 2 int: ");
  scanf("%d %d", &in1, &in2);
  max = max2(in1, in2);
  printf(“max = %d\n", max);
  return(0);
}
```


```
/* 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 in1 ,in2, max;

  printf("enter 2 int: ");
  scanf("%d %d", &in1, &in2);
  max = max2(in1, in2);
  printf(“max = %d\n", max);
}
```

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 T μιας συνάρτησης μπορεί κατά την κλήση να δοθεί σαν πραγματική παράμετρος μια **οποιαδήποτε** τιμή τύπου T.
- 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 T:
    - κυριολεκτικό τύπου T
    - μεταβλητή τύπου T
    - έκφραση που αποτιμάται σε τιμή T
  - Σαν παράμετρος κλήσης μιας συνάρτησης μπορεί να δοθεί μια κλήση συνάρτησης τύπου T.

```
/* 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 in1, in2, max;

  printf("enter 2 int: ");
  scanf("%d %d", &in1, &in2);
  max = max2(in1, in2);
  printf("max = %d\n", max);
}
```

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

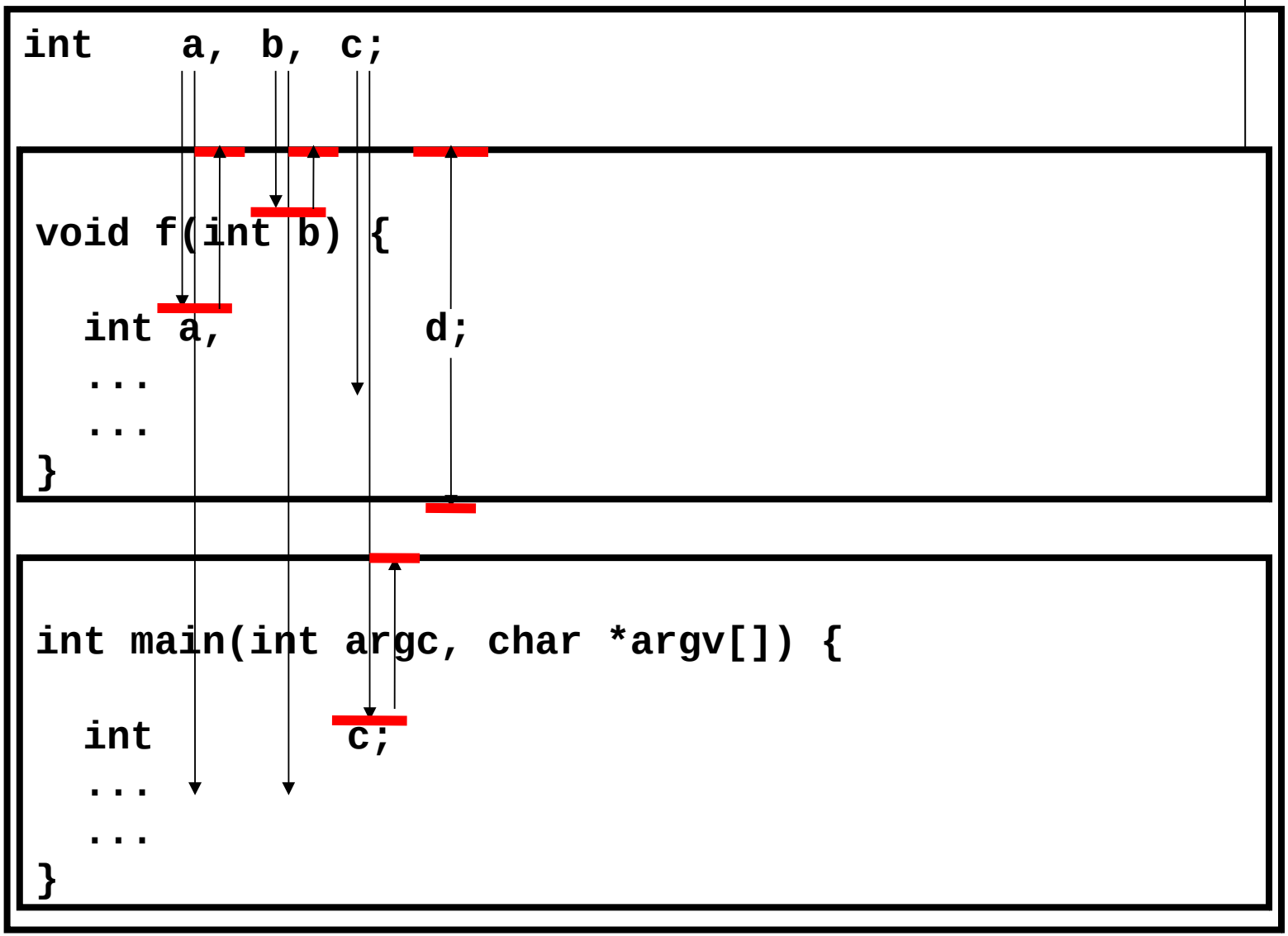
```
#include <stdio.h>

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

int main(int argc, char* argv[]) {
  int in1, in2, in3;

  printf("enter 3 int: ");
  scanf("%d %d %d", &in1, &in2, &in3);

  printf("%d\n", max2(in1, in2));

  printf("%d\n", max2(in1 + in2, 25);

  printf("%d\n", max2(in1, max2(in2, in3)) );
}
```

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.
  - 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 διαχωρισμού ανάμεσα στην τυπική παράμετρο και την τοπική μεταβλητή με το ίδιο όνομα 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.

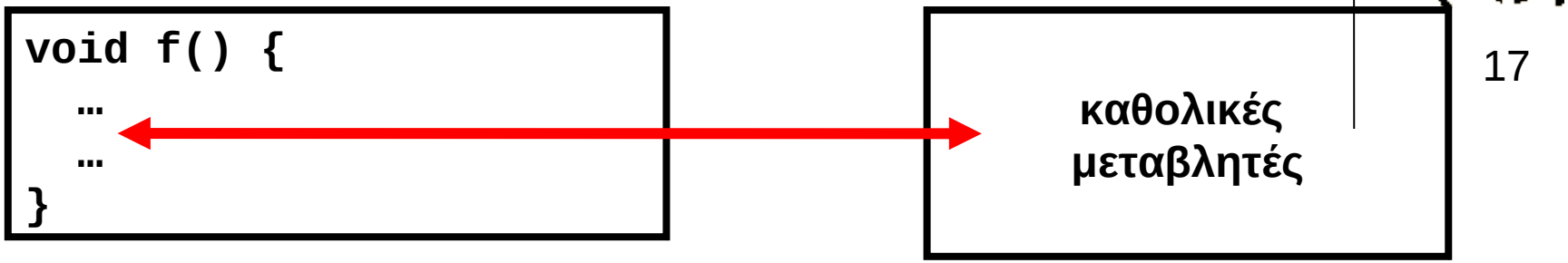

int a, b, c;
void f(int b) {
int a, d;
...
...
}
int main(int argc, char *argv[]) {
int c;
...
...
}


```
#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);
}
```

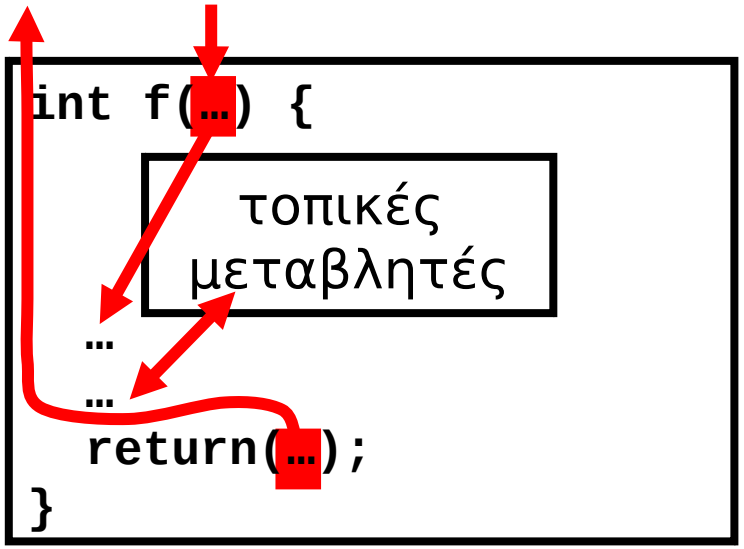
Συναρτήσεις και καθολικές μεταβλητές

- 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



προγραμματισμός χωρίς παρενέργειες



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

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

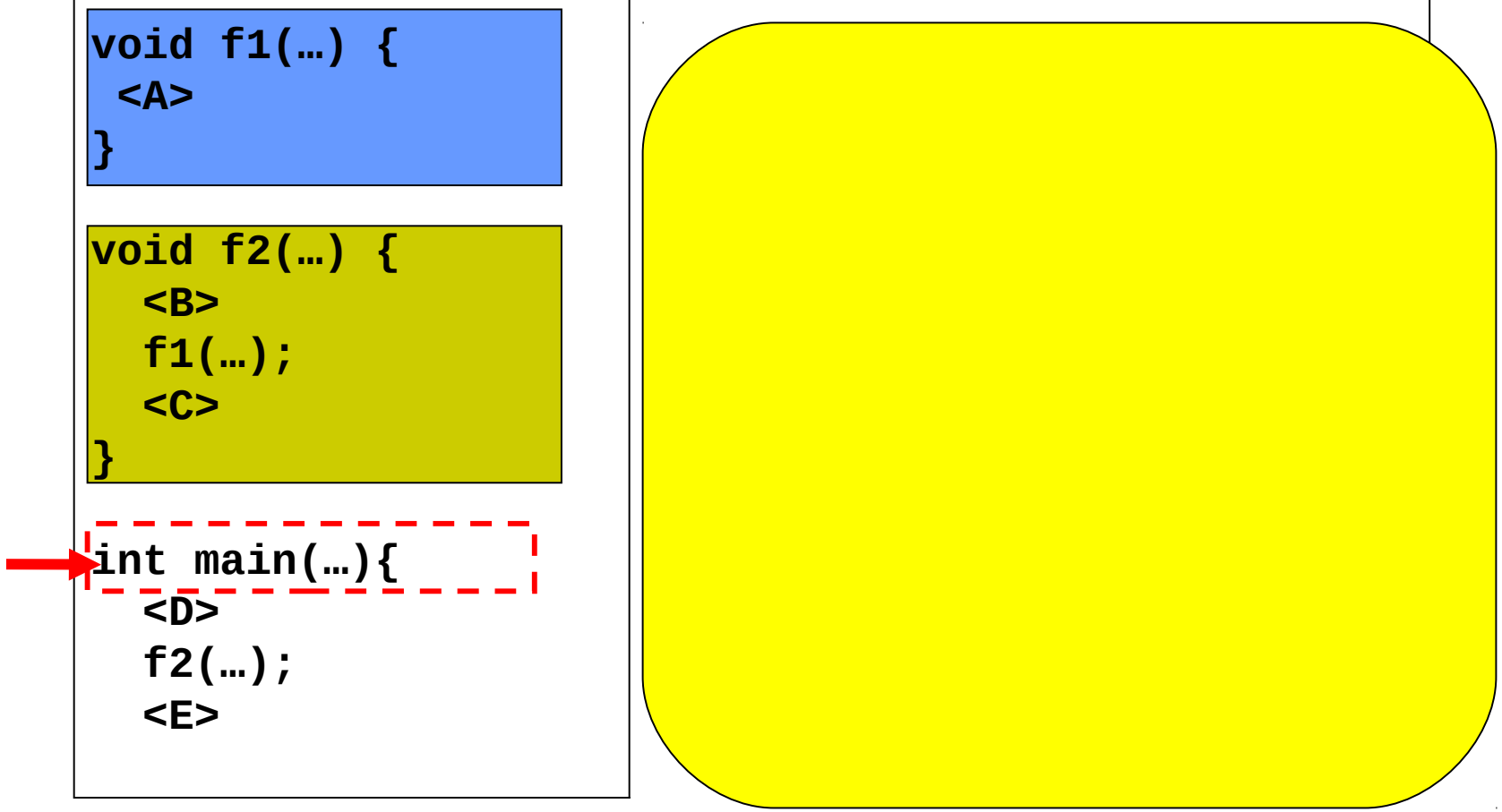
  1. Η εκτέλεση της συνάρτησης Α σταματά στο σημείο όπου γίνεται η κλήση της συνάρτησης Β.
  2. Αρχικοποιούνται οι παράμετροι και τοπικές μεταβλητές της συνάρτησης Β.
  3. Αρχίζει η εκτέλεση του κώδικα της συνάρτησης Β (που μπορεί να καλέσει και άλλες συναρτήσεις).
  4. Όταν τερματίζεται η εκτέλεση της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

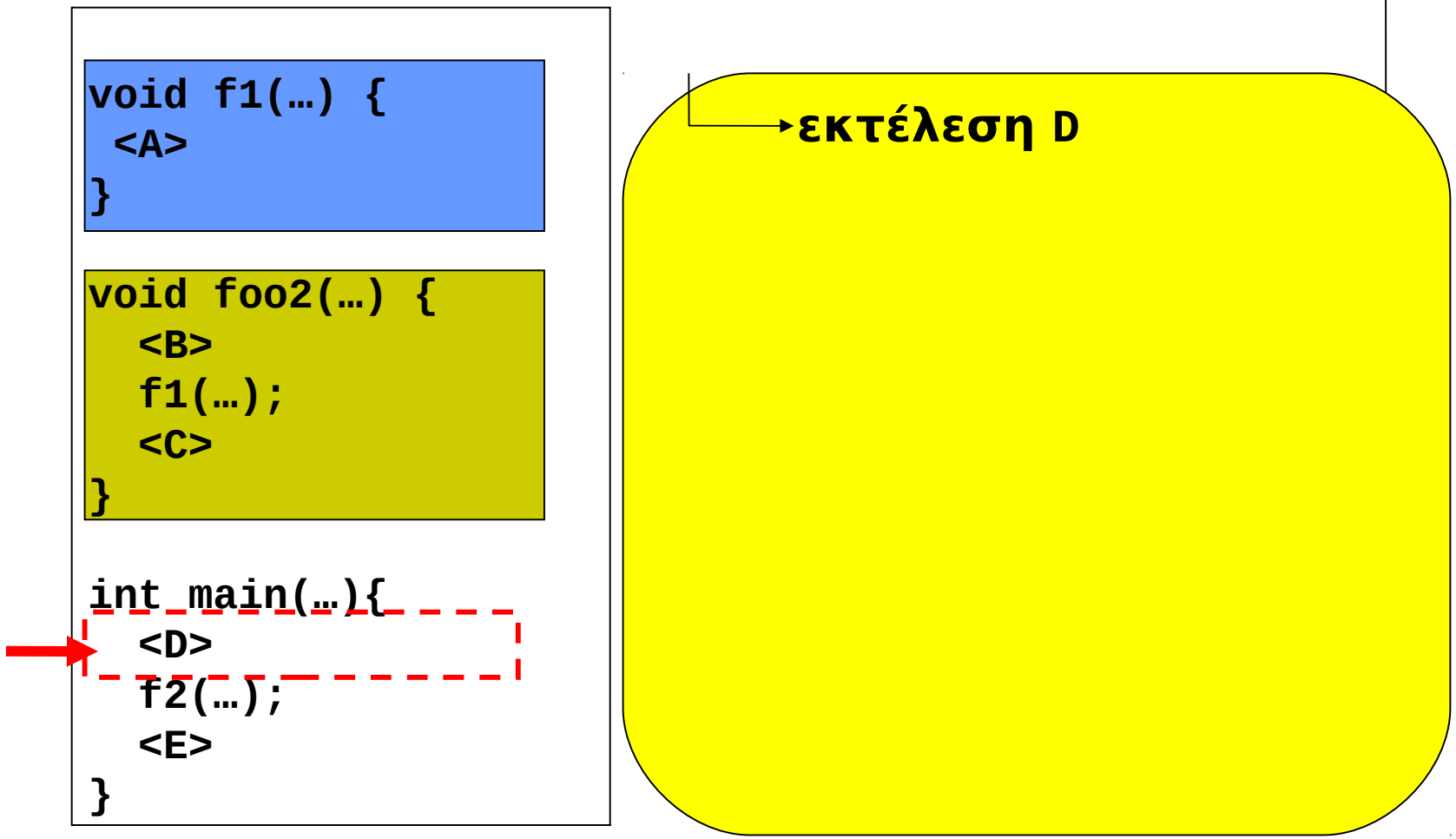
Πλαίσιο εκτέλεσης συνάρτησης

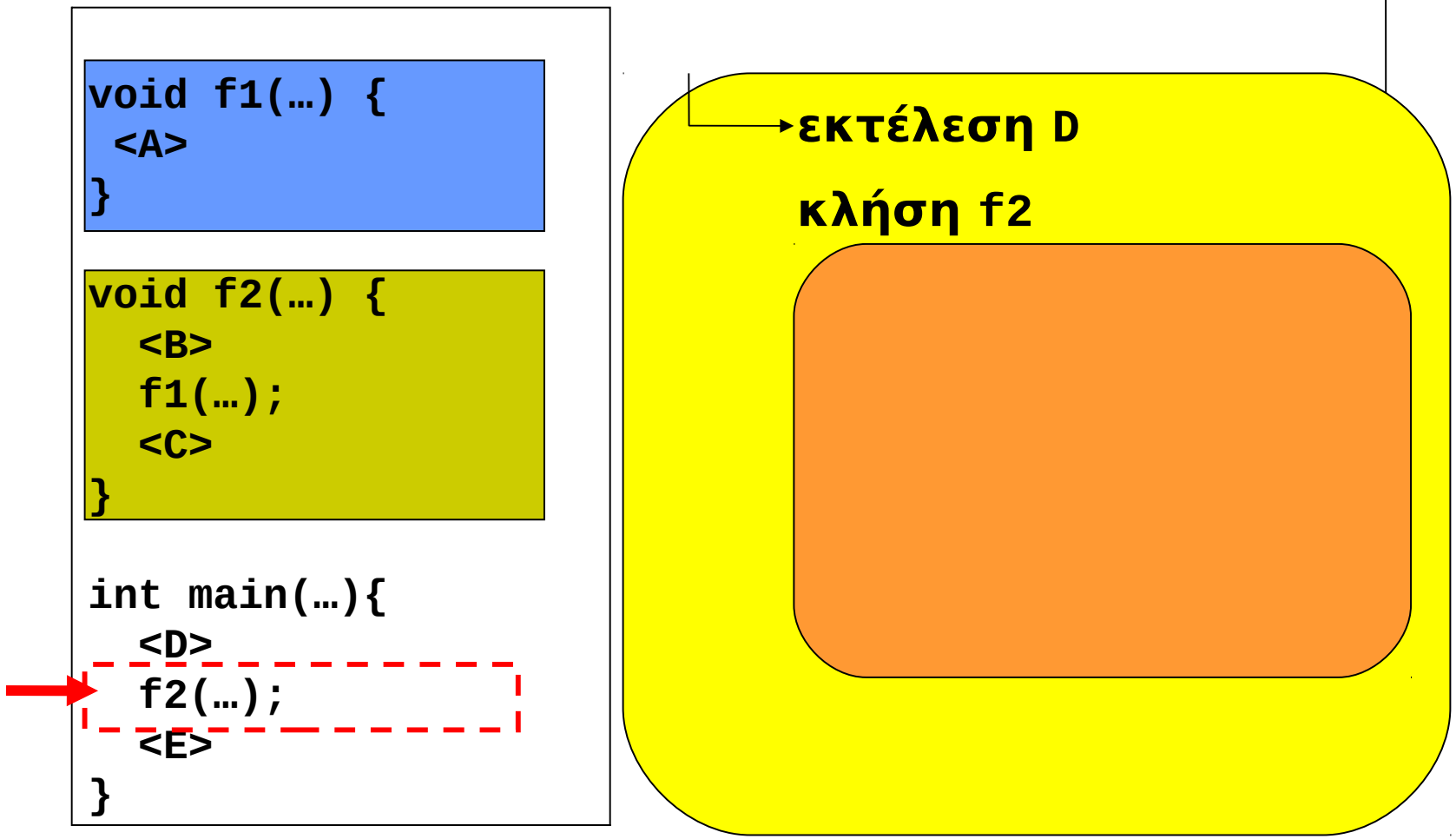
- Ο **ίδιος** κώδικας της συνάρτησης εκτελείται κάθε φορά σε ένα **διαφορετικό** πλαίσιο εκτέλεσης (σε έναν ξεχωριστό «μικρόκοσμο»).
  - Το πλαίσιο εκτέλεσης **δημιουργείται** (εκ νέου) πριν αρχίσει η εκτέλεση του κώδικα της συνάρτησης και **καταστρέφεται** όταν ολοκληρωθεί η εκτέλεση της.
  - Για κάθε κλήση, δημιουργείται ένα **καινούργιο** και **ξεχωριστό** πλαίσιο εκτέλεσης, που **δεν** έχει σχέση με προηγούμενα πλαίσια εκτέλεσης.

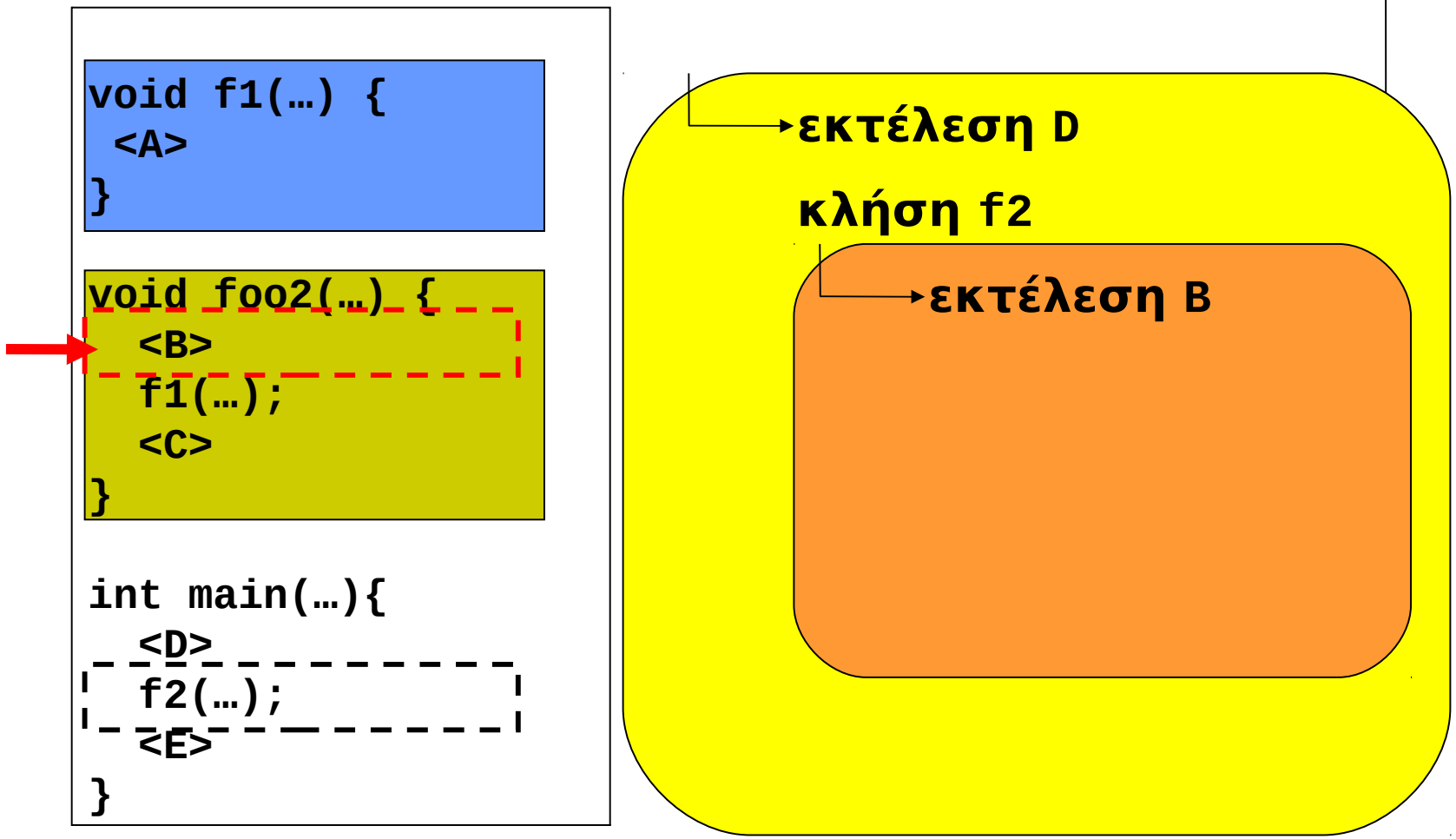
- Το πλαίσιο εκτέλεσης χρησιμεύει για την αποθήκευση των τοπικών μεταβλητών και των πραγματικών παραμέτρων για την **συγκεκριμένη** κλήση.

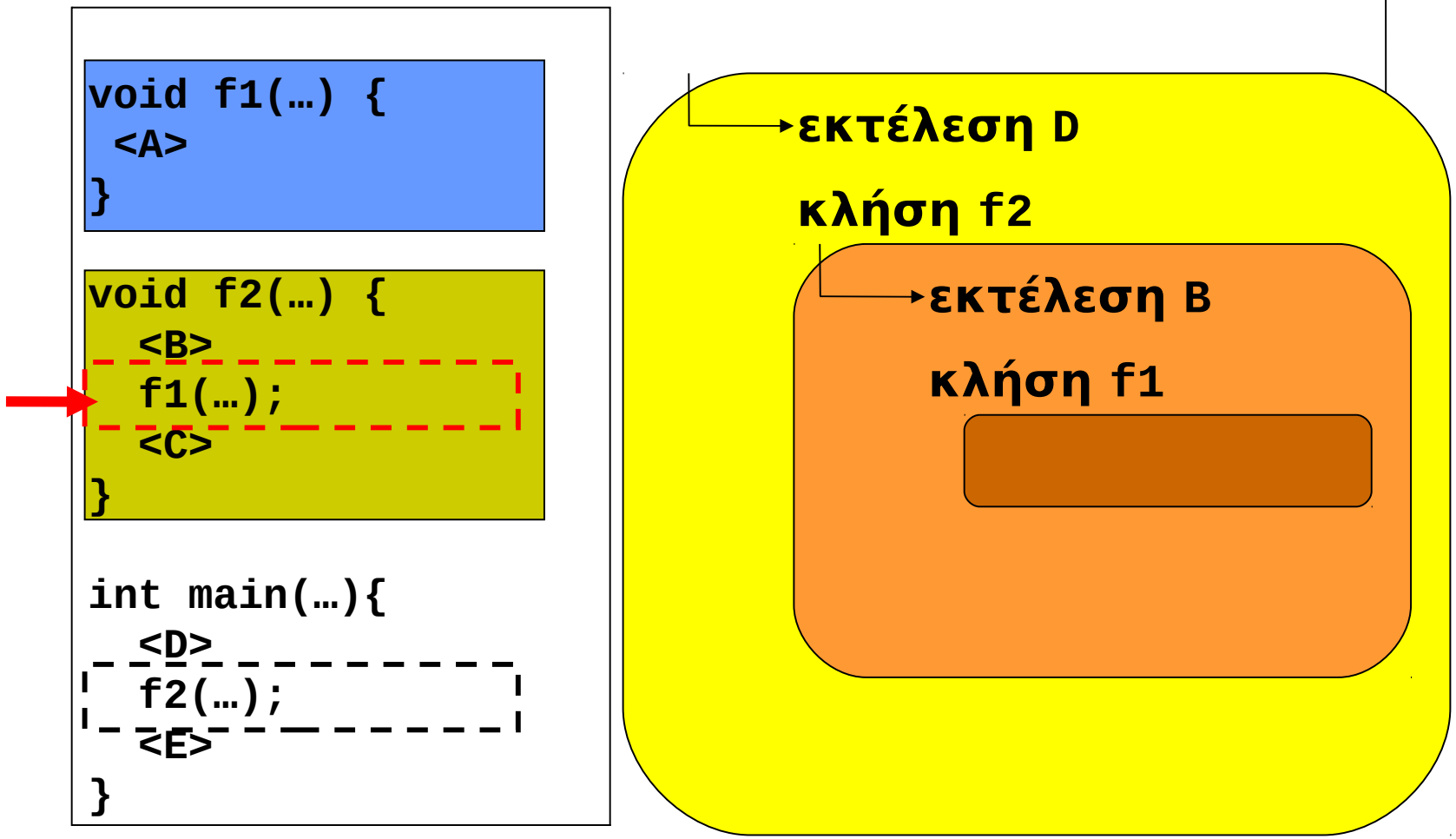
```
void f1(…) {
 <A>
}

void f2(…) {
  <B>
  f1(…);
  <C>
}

int main(…){
  <D>
  f2(…);
  <E>
```

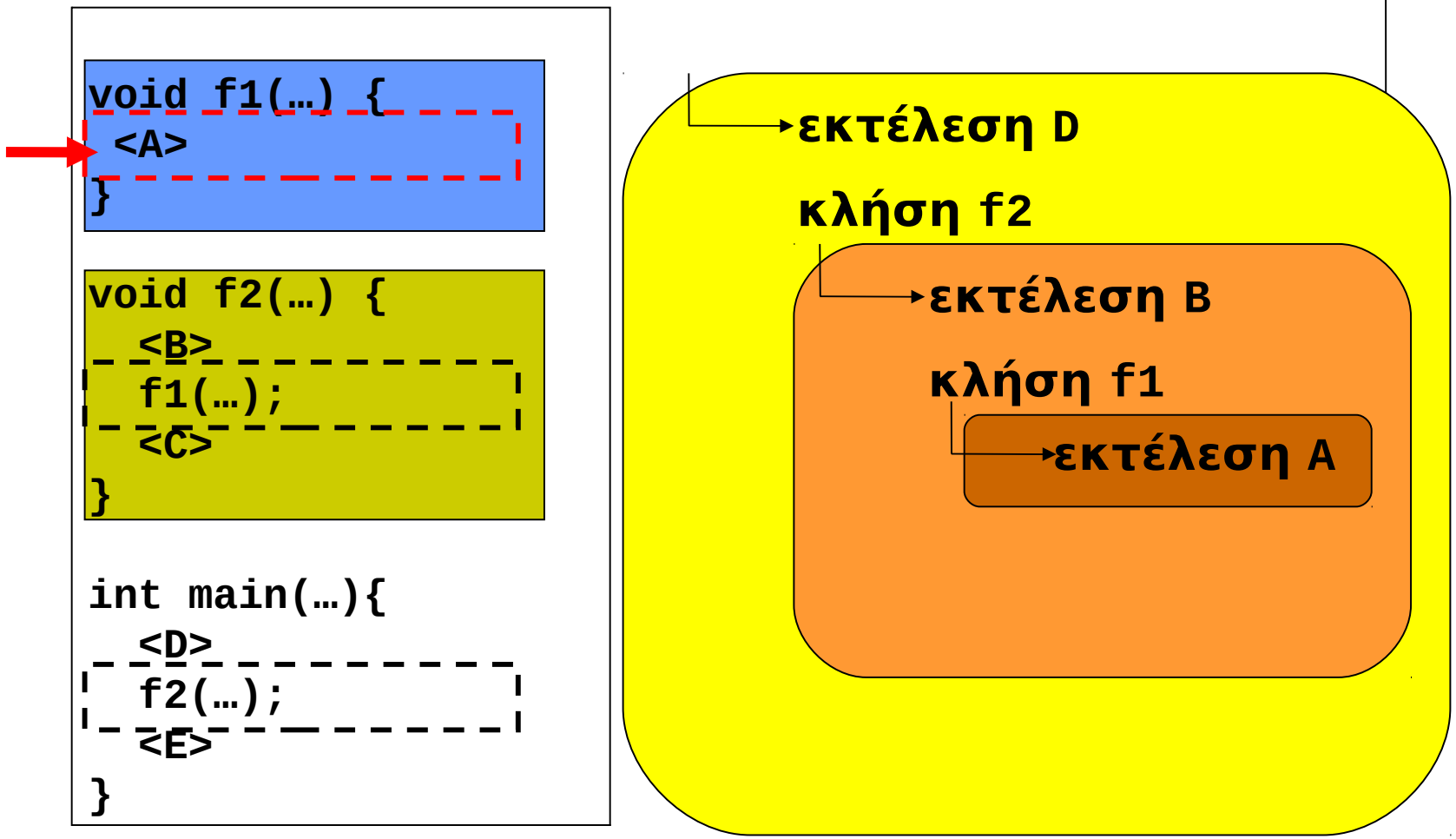


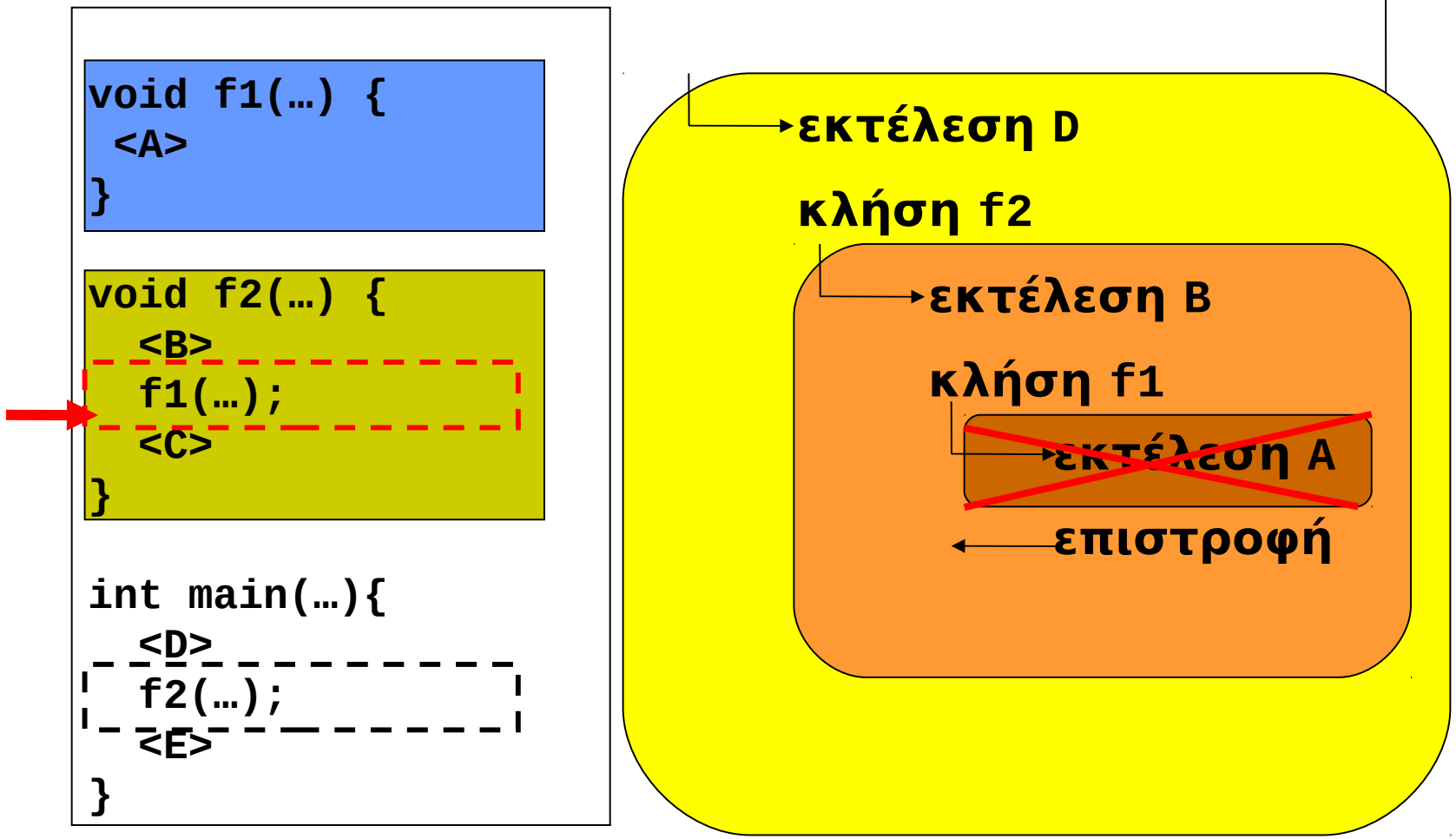

void f1(…) {
<A>
}
void foo2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D

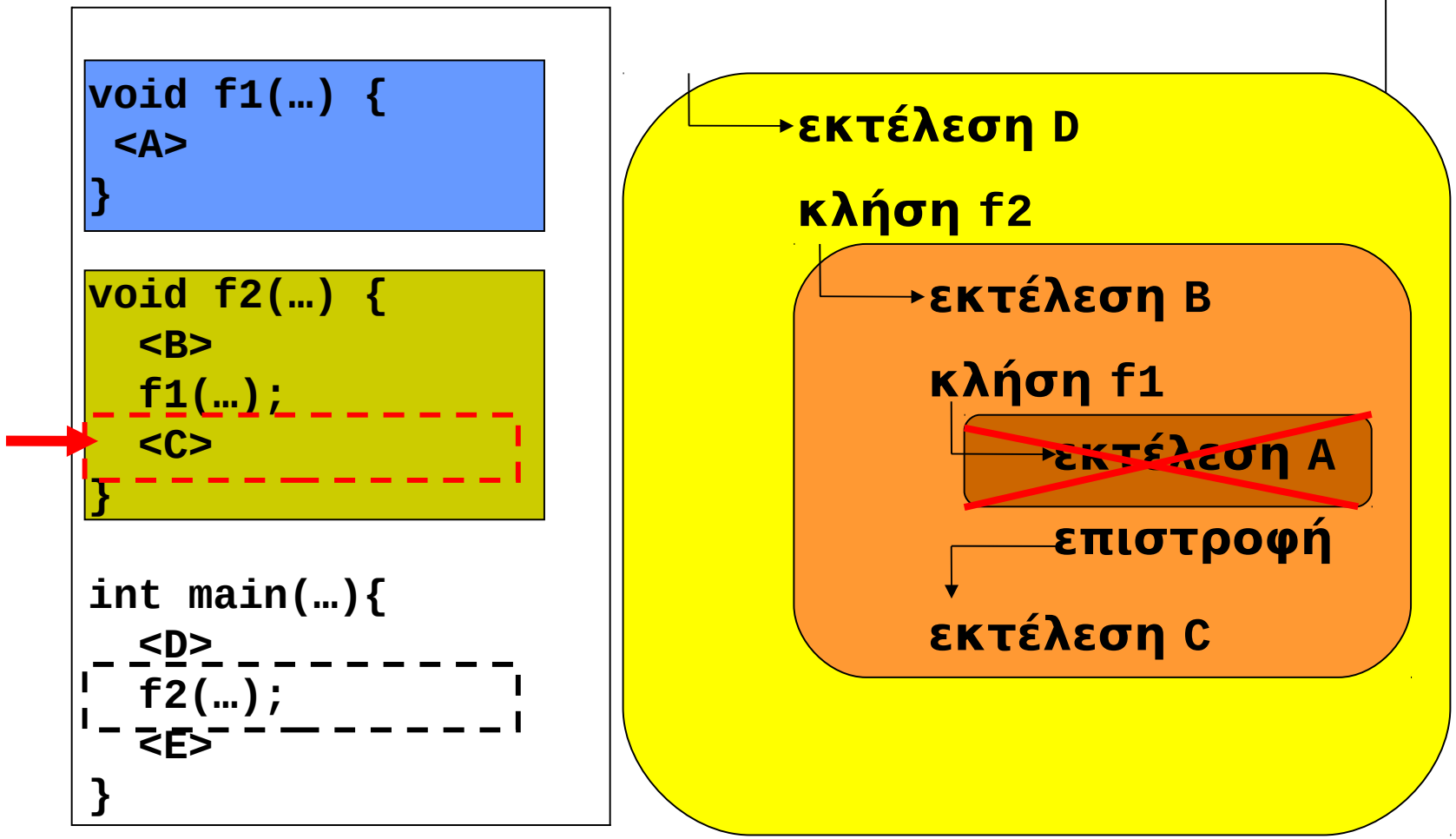

void f1(…) {
<A>
}
void f2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D
κλήση f2

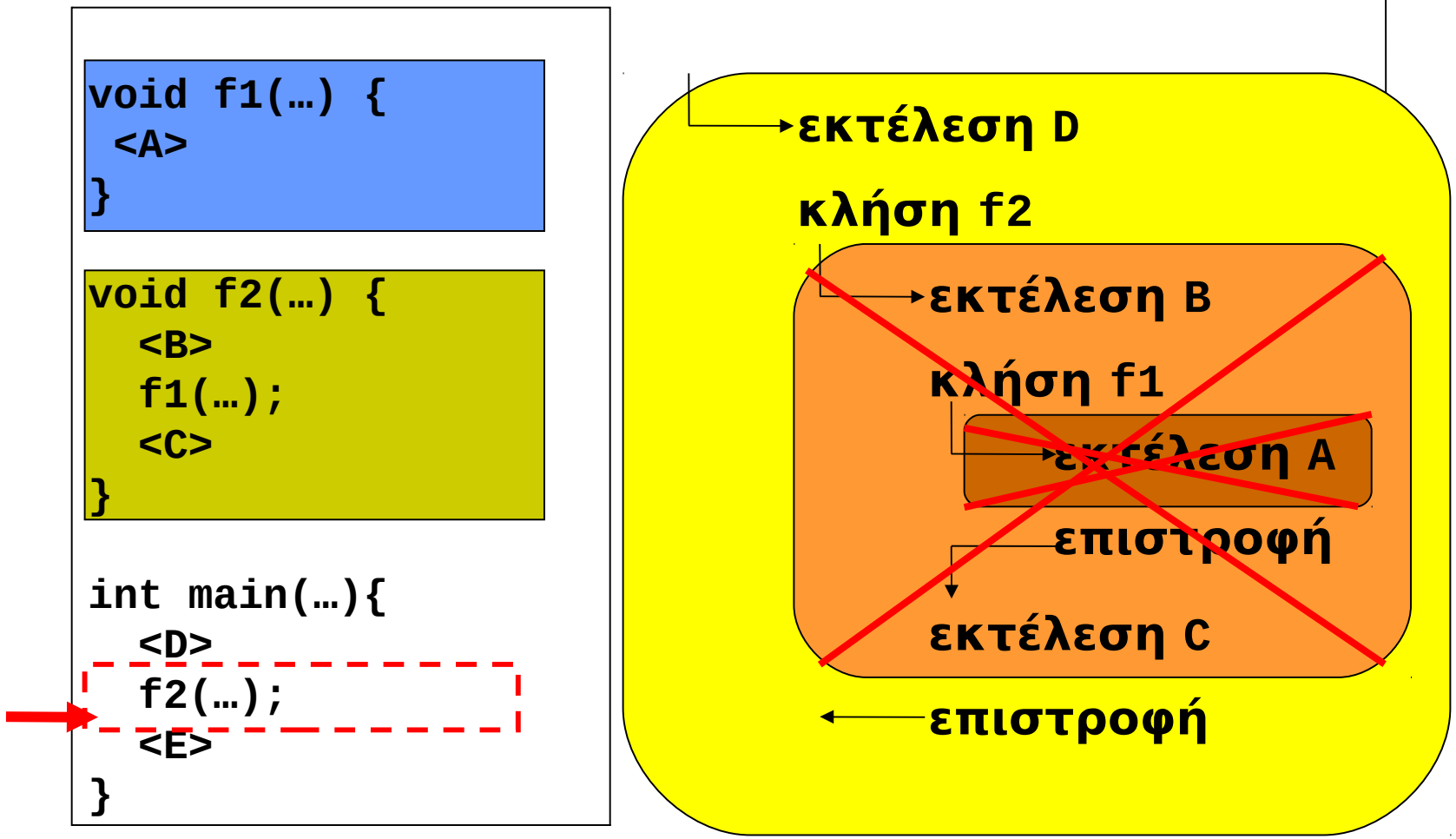

void f1(…) {
<A>
}
void foo2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D
κλήση f2
εκτέλεση B


void f1(…) {
<A>
}
void f2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D
κλήση f2
εκτέλεση B
κλήση f1

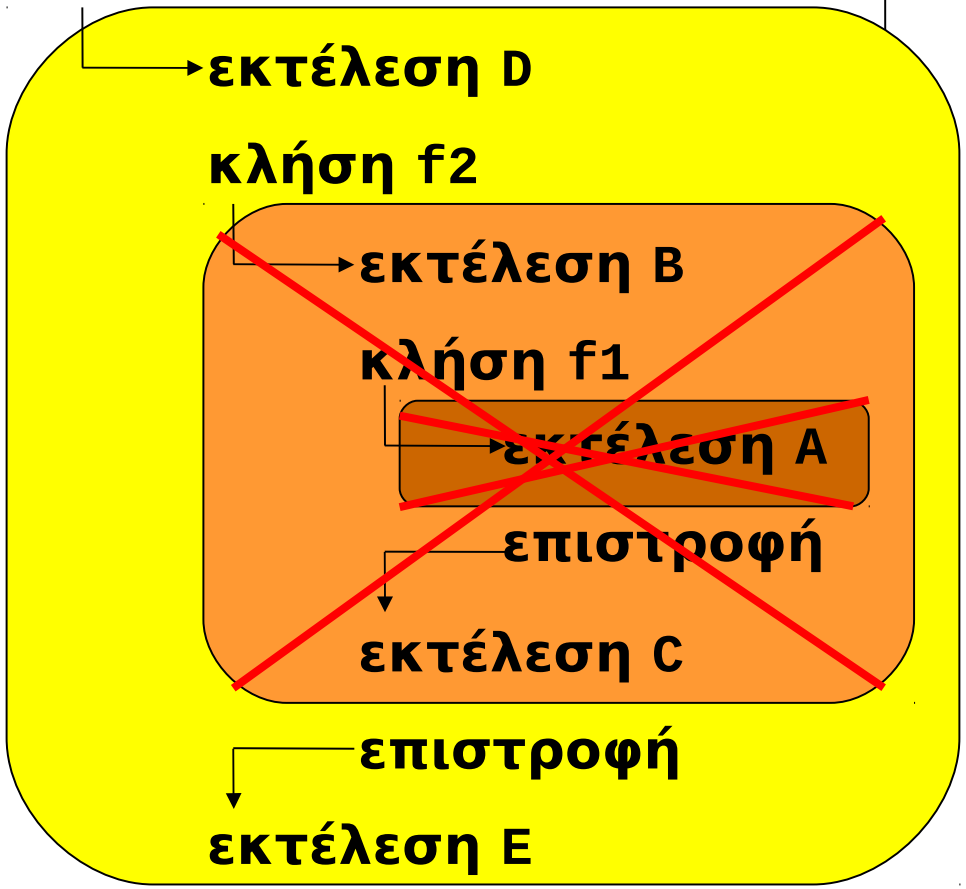

void f1(…) {
<A>
}
void f2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D
κλήση f2
εκτέλεση B
κλήση f1
εκτέλεση A


void f1(…) {
<A>
}
void f2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D
κλήση f2
εκτέλεση B
κλήση f1
εκτέλεση A
επιστροφή

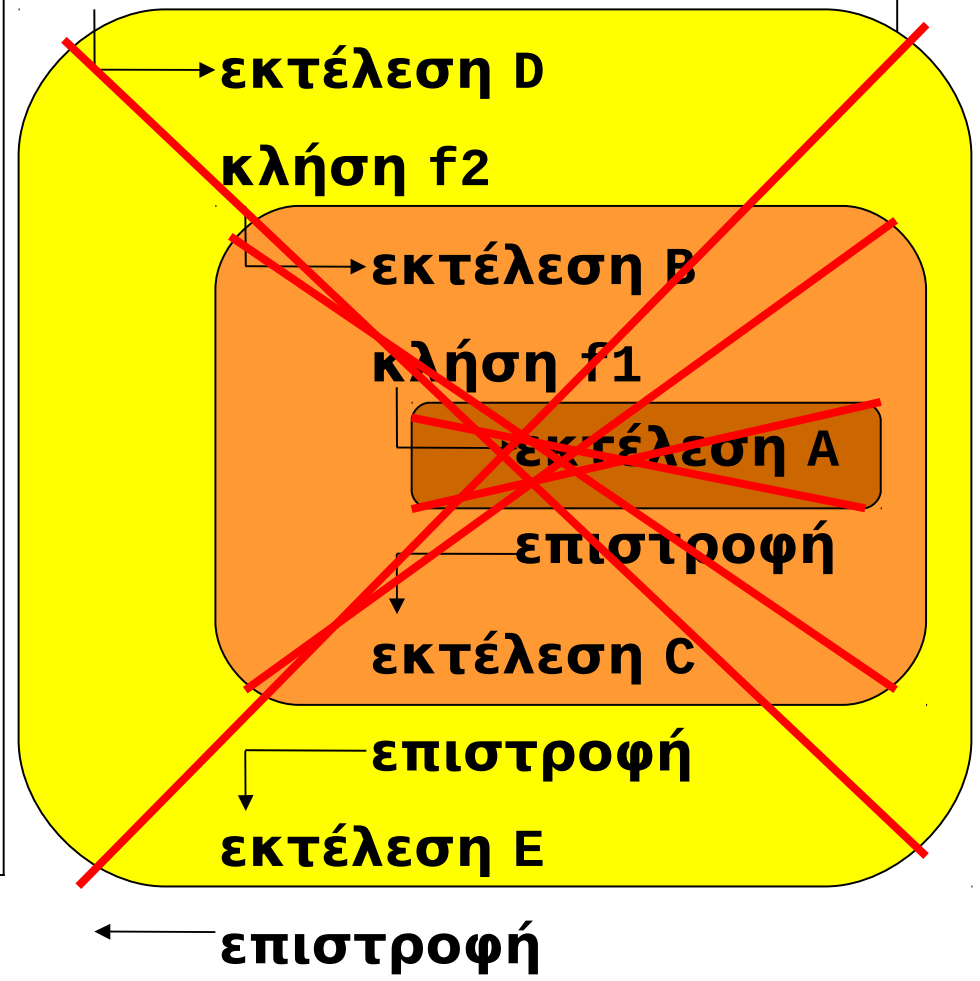

void f1(…) {
<A>
}
void f2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
εκτέλεση D
κλήση f2
εκτέλεση B
κλήση f1
εκτέλεση A
επιστροφή
εκτέλεση C


void f1(…) {
<A>
}
void f2(…) {
<B>
f1(…);
<C>
}
int main(…){
<D>
f2(…);
<E>
}
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>
}
```



```
void f1(…) {
 <A>
}

void f2(…) {
  <B>
  f1(…);
  <C>
}

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



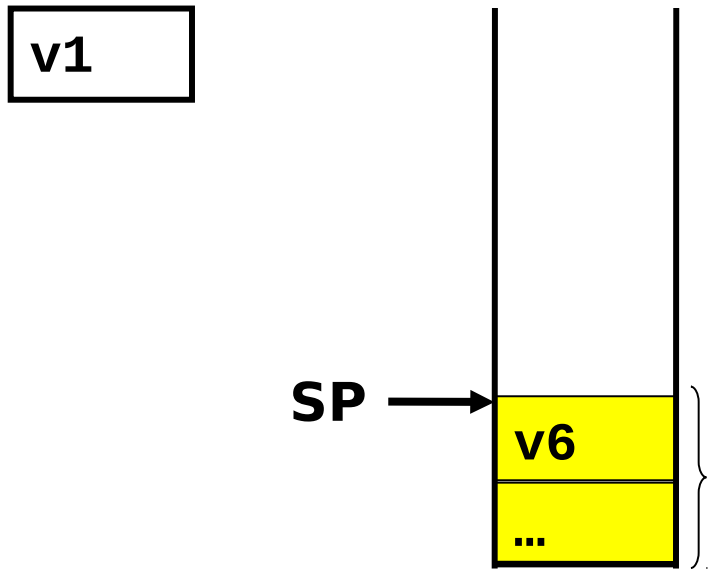
Δέσμευση μνήμης μεταβλητών

- 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**
  - **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.

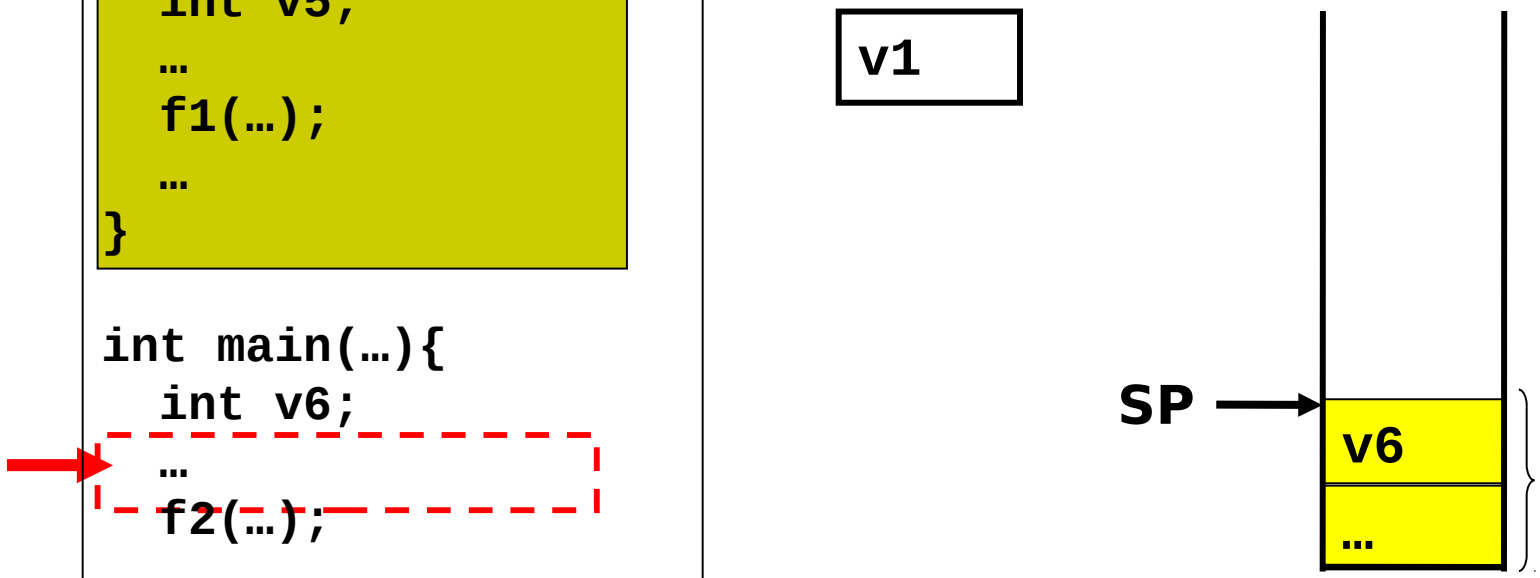
Στοίβα

- Δεσμεύεται ένα (μεγάλο) συνεχόμενο τμήμα μνήμης, που χρησιμοποιείται σύμφωνα με την λογική της **στοίβας** (Last In First Out - LIFO queue) για την αποθήκευση των τιμών των παραμέτρων και τοπικών μεταβλητών των συναρτήσεων.
  - Και των καταχωρητών της CPU μεταξύ κλήσεων διαφορετικών 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.

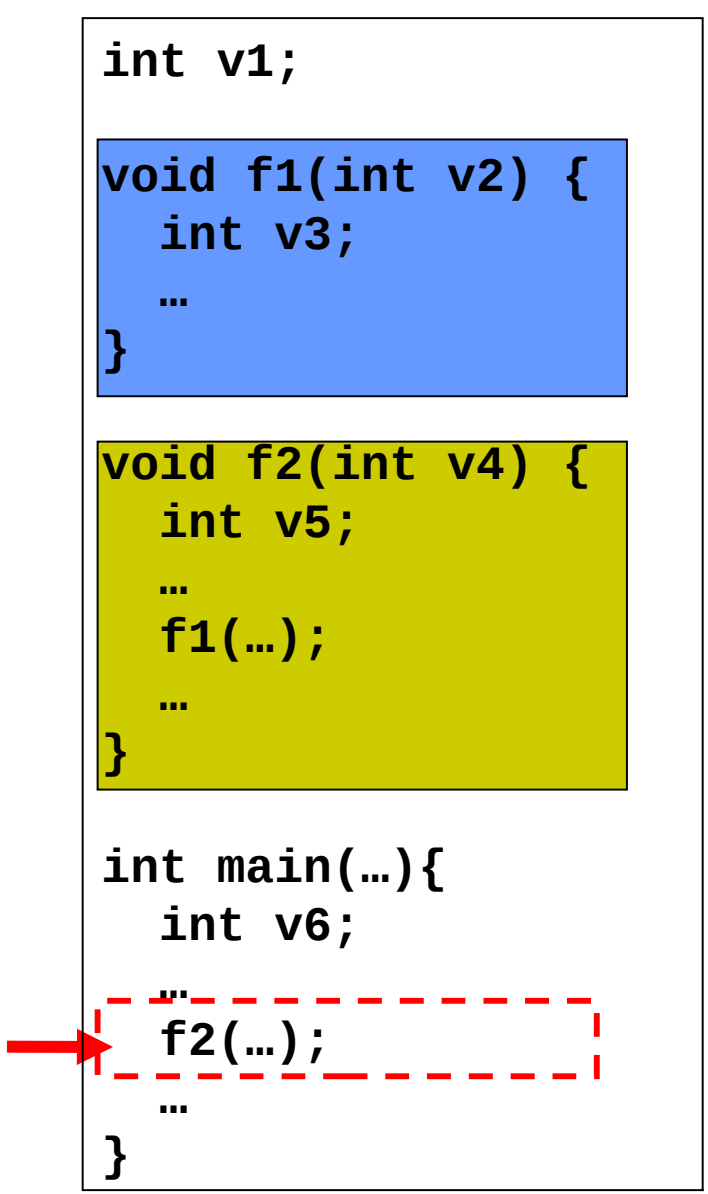
```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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

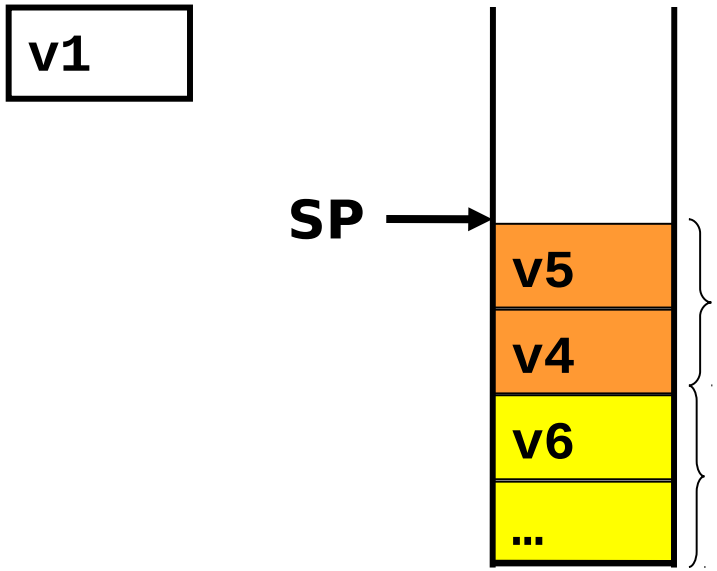


```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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

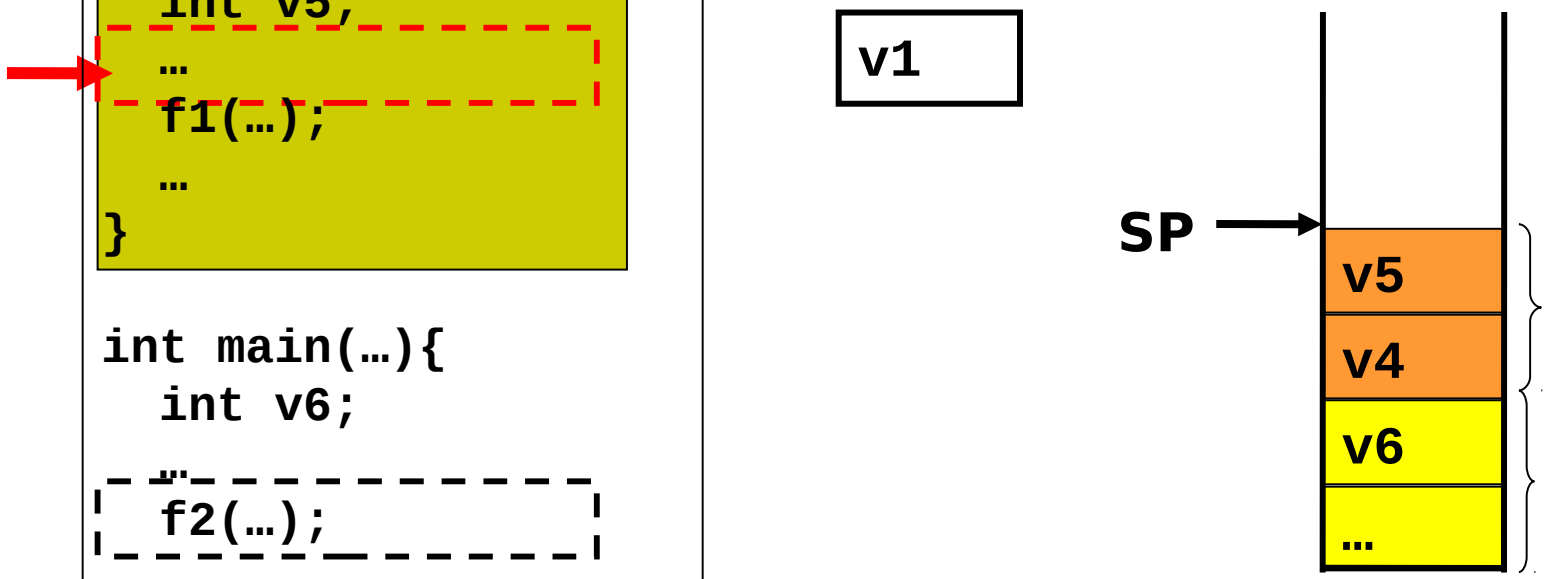


```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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

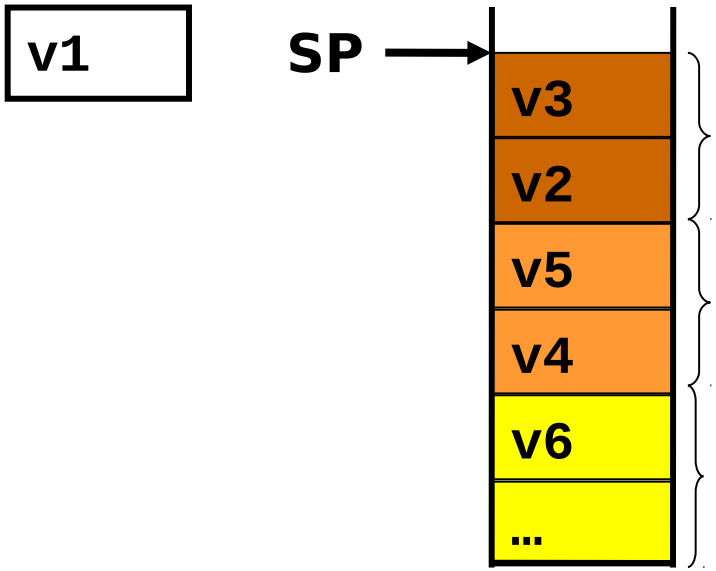




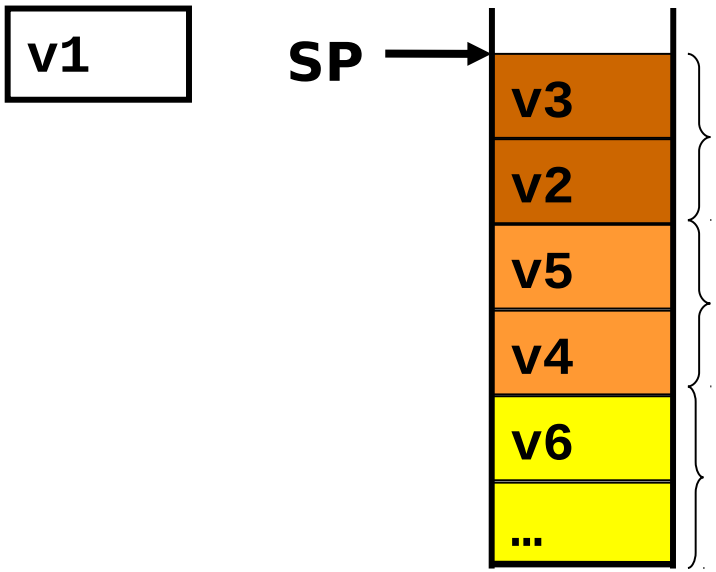
```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



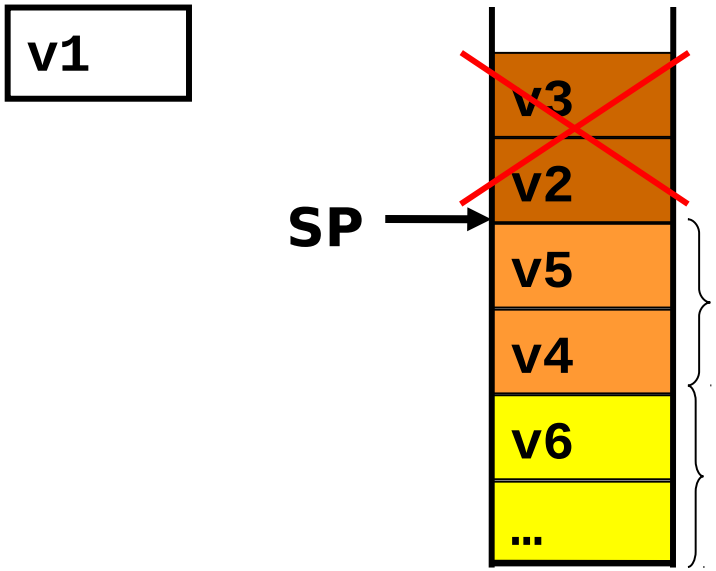
```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



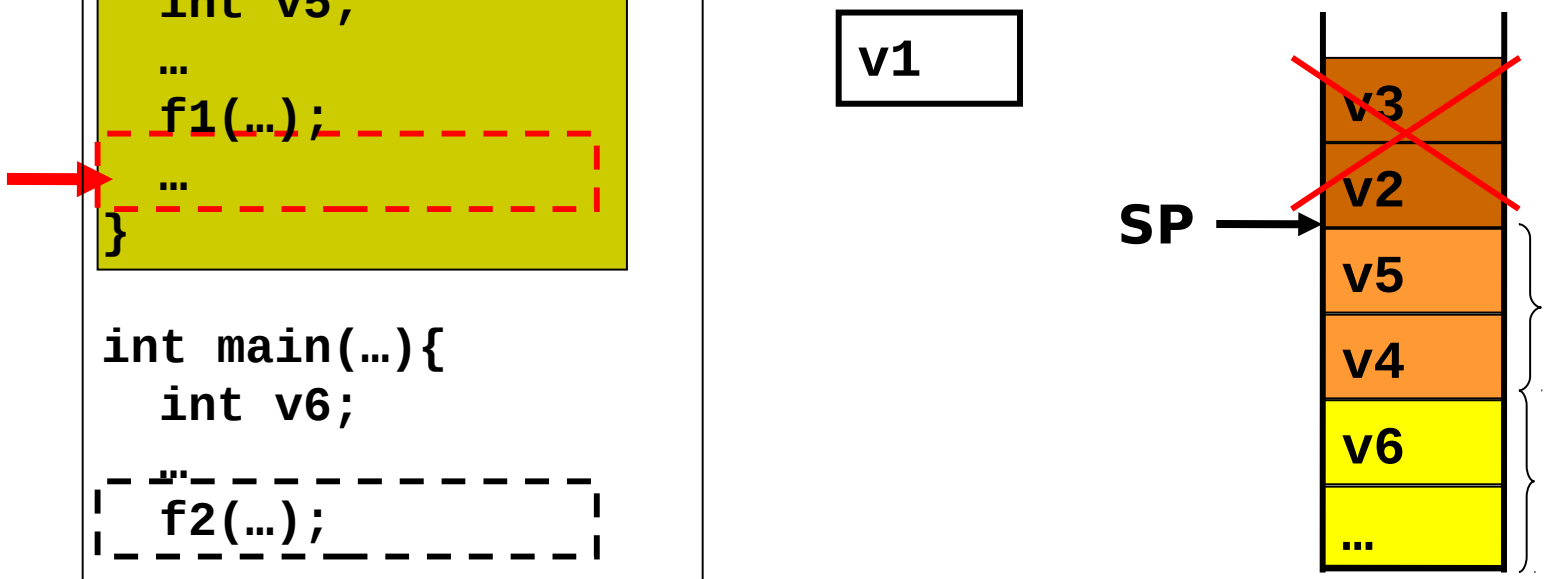
```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



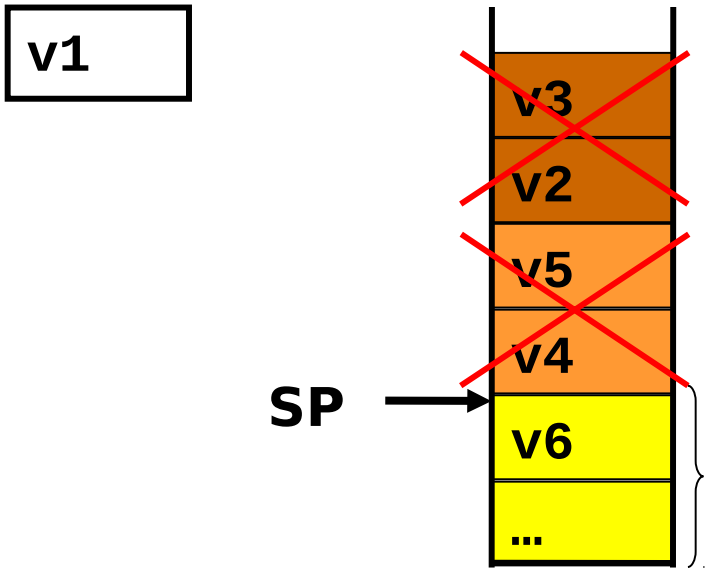
```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



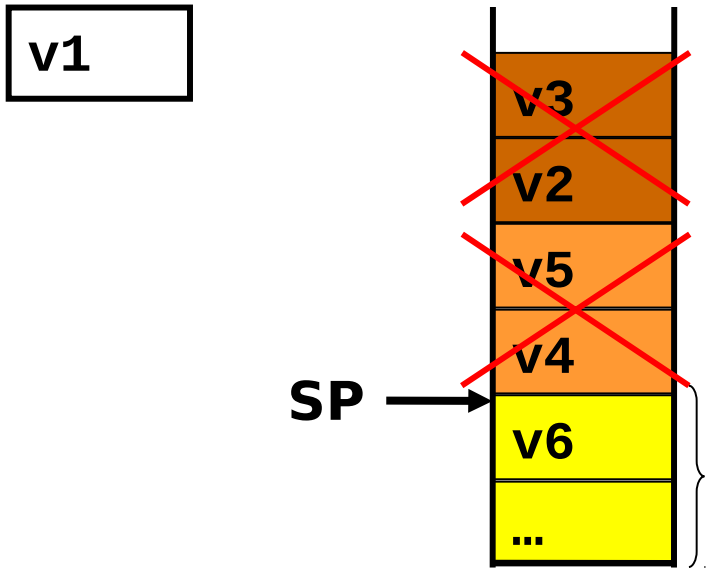
```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



```
int v1;

void f1(int v2) {
  int v3;
  …
}

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

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



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.