



Προγραμματισμός Ι (ECE115)

#8

συναρτήσεις (functions)

Συναρτήσεις

- Συχνά, θέλουμε να εκτελούνται (σχεδόν) οι ίδιες εντολές, σε **διαφορετικά** σημεία του προγράμματος
 - δεν μας καλύπτουν οι δομές επανάληψης
- Δεν είναι καλή πρακτική να γράφουμε τις ίδιες εντολές πολλές φορές μέσα στο πρόγραμμα
 - ο κώδικας μεγαλώνει, χειροτερεύει η αναγνωσιμότητα
 - η διόρθωση λαθών γίνεται ακόμα πιο επίπονη
- Χρησιμοποιούμε **συναρτήσεις!**
 - περιέχουν τον κώδικα που επαναλαμβάνεται
 - σε **παραμετροποιημένη** μορφή (αν/όπου χρειάζεται)

Δομημένος προγραμματισμός

- Οι συναρτήσεις είναι το πιο βασικό στοιχείο του λεγόμενου **δομημένου** προγραμματισμού
- Μπορεί να χρησιμοποιηθούν «απλά» για να υπάρχει καλύτερη δόμηση του κώδικα ενός προγράμματος
 - ακόμα και για κώδικα που δεν επαναλαμβάνεται
- Το σπάσιμο του κώδικα σε μικρότερα κομμάτια **βελτιώνει** από μόνο του την αναγνωσιμότητα καθώς και την διαχείριση του κώδικα
- Ιδανικά, η λειτουργία κάθε συνάρτησης πρέπει να είναι κατανοητή από μόνη της
 - χωρίς γνώση του υπόλοιπου κώδικα του προγράμματος
- Οι βιβλιοθήκες είναι «συλλογές» συναρτήσεων
 - π.χ. `stdio`, `string`, `math`, κλπ

Παραμετροποίηση

- Η κύρια χρησιμότητα των συναρτήσεων βρίσκεται στην **παραμετροποιημένη** εκτέλεση κώδικα
 - όπως ισχύει και για το ίδιο το πρόγραμμα (συνάρτηση `main`)
- Ο κώδικας που γράφουμε παραμένει «σταθερός»
- Το αποτέλεσμα που παράγει **εξαρτάται** από τις παραμέτρους (είσοδο) που δέχεται κάθε φορά
- Εκτελέσεις με **διαφορετικές** παραμέτρους μπορεί να οδηγήσουν σε **διαφορετικά** αποτελέσματα
 - **χωρίς** να πρέπει να γραφτεί νέος κώδικας για αυτό

Αναλογία με τον πραγματικό κόσμο

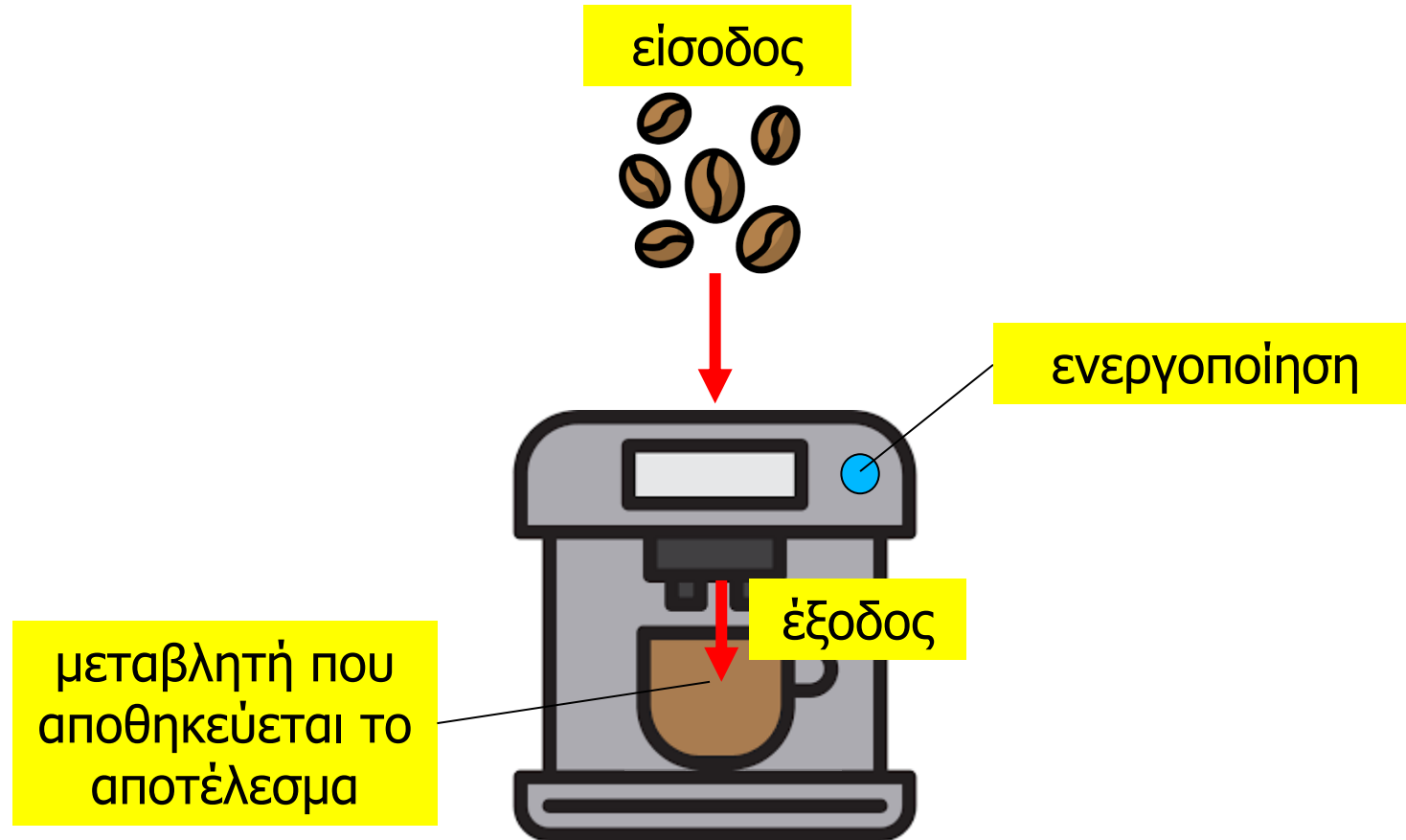
- Συνάρτηση = ένα κουτί που περιέχει ένα μηχανισμό
- Παράμετροι = είσοδος / παράμετροι λειτουργίας
- Τιμή επιστροφής = έξοδος / αποτέλεσμα
- Απαιτείται **σχεδιασμός της συνάρτησης!**
 1. Τι είσοδο θα παίρνει και τι έξοδο θα παράγει;
 2. Πως θα λειτουργεί εσωτερικά;
- Για να χρησιμοποιήσει κάποιος ένα τέτοιο κουτί αρκεί να ξέρει το (1)
- Χωρίς απαραίτητα να γνωρίζει ή να κατανοεί σε βάθος το (2) – the black box principle

Το κουτί

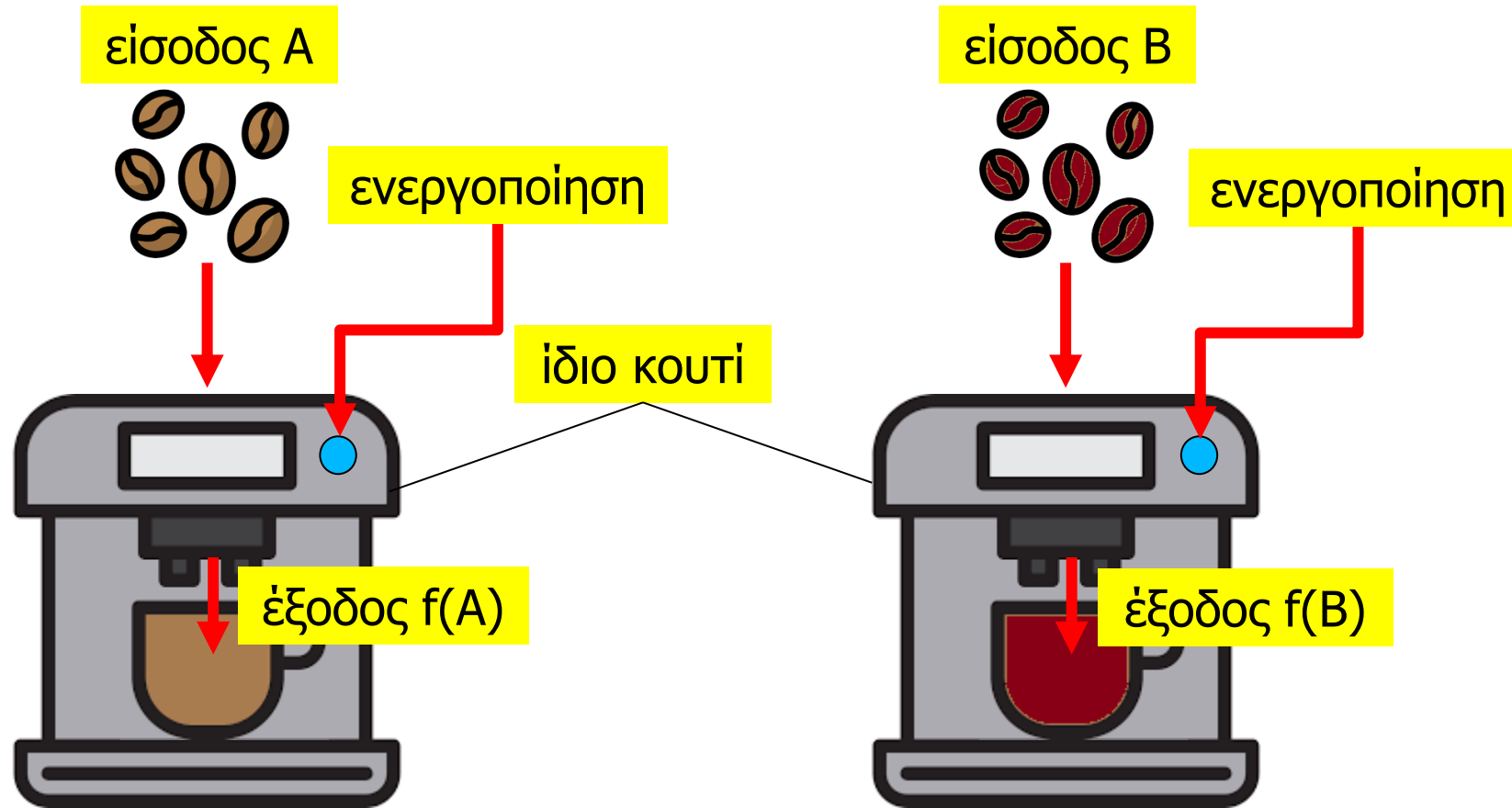
δεν γνωρίζουμε
ακριβώς τι έχει μέσα
και το πως λειτουργεί



Είσοδος-ενεργοποίηση-έξοδος



Διαφορετική είσοδος -> διαφορετική έξοδος



Δομή συνάρτησης

- Μια συνάρτηση ορίζεται δίνοντας
 1. το όνομα της
 2. τις «τυπικές» παραμέτρους της
 3. τον τύπο του αποτελέσματος που επιστρέφει
 4. το σώμα της
- Τα (1), (3), (3) αποτελούν την **επικεφαλίδα**
- Το (4) τον **κώδικα / υλοποίηση** της συνάρτησης

- Μια συνάρτηση μπορεί να **δηλωθεί** (ξεχωριστά)
 - μέσω της επικεφαλίδας της (χωρίς το σώμα)
- Η υλοποίηση μπορεί να δίνεται σε **παρακάτω** σημείο του κώδικα (ή ακόμα και σε **διαφορετικό** αρχείο)

επικεφαλίδα

<τύπος> <όνομα> (<παράμετρος> , ... , <παράμετρος>) {

<τοπικές μεταβλητές>

<εντολές επεξεργασίας / ελέγχου>

}

σώμα
υλοποίηση

Επιστροφή αποτελέσματος

- Μια συνάρτηση μπορεί να **επιστρέφει** μια συγκεκριμένη τιμή
- Αυτό γίνεται με την **ειδική εντολή** `return`
 - αν δεν υπάρχει `return` ή χρησιμοποιηθεί χωρίς κάποια τιμή, τότε η συνάρτηση επιστρέφει μια **τυχαία** τιμή
 - ο μεταγλωττιστής εκτυπώνει μήνυμα προειδοποίησης
- Η `return` **τερματίζει** την εκτέλεση της συνάρτησης
 - μπορεί να υπάρχει σε **πολλά σημεία** του σώματος της συνάρτησης – προσοχή έτσι ώστε να επιστέφεται το επιθυμητό αποτέλεσμα σε κάθε περίπτωση
- Μια συνάρτηση μπορεί να μην επιστρέφει κάποιο αποτέλεσμα: επιστρέφει `void`
 - η εντολή `return` δίνεται προαιρετικά και χωρίς κάποια τιμή

```
int add(int a, int b);
```

```
int add(int a, int b) {  
    return (a+b);  
}
```

```
int add(int a, int b);
```

```
int add(int a, int b) {  
    int c;  
  
    c = a + b;  
  
    return(c);  
}
```

```
int sum(int n);
```

```
int sum(int n) {  
    int i, s;  
  
    s = 0;  
    for (i=1; i<=n; i++) {  
        s = s + i;  
    }  
  
    return(s);  
}
```

```
void printRange(int low, int high);
```

```
void printRange(int low, int high) {  
    int i;  
  
    for (i=low; i<=high; i++) {  
        printf("%d ", i);  
    }  
    printf("\n");  
  
    return;  
}
```

προαιρετικό

Κλήση συνάρτησης

- Για να **καλέσουμε** μια συνάρτηση μέσα στον κώδικα μας, γράφουμε
 - το όνομα της συνάρτησης
 - τιμές για τις παραμέτρους της
 - σε `()` χρησιμοποιώντας το `,` ως διαχωριστικό
- Αν μια συνάρτηση επιστρέφει αποτέλεσμα τύπου $\mathbb{T}$, τότε η κλήση της συνάρτησης αποτελεί **έκφραση αποτίμησης** που δίνει τιμή τύπου $\mathbb{T}$
- Το αποτέλεσμα της κλήσης μπορεί
 - Να ανατεθεί σε μια μεταβλητή τύπου $\mathbb{T}$
 - Να χρησιμοποιηθεί σε μια σύνθετη έκφραση αποτίμησης αντί μιας κυριολεκτικής τιμής ή μεταβλητής τύπου $\mathbb{T}$
 - Να χρησιμοποιηθεί ως πραγματική τιμή μιας τυπικής παραμέτρου τύπου $\mathbb{T}$ μιας συνάρτησης


```
/* υπολογισμός μέγιστης τιμής 2 ακεραίων */  
  
#include <stdio.h>  
  
int max2(int num1, int num2) {  
    if (num1 > num2) {  
        return(num1);  
    }  
    else {  
        return(num2);  
    }  
}  
  
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 num1, int num2) {  
    if (num1 > num2) {  
        return(num1);  
    }  
    else {  
        return(num2);  
    }  
}  
  
int main(int argc, char *argv[]) {  
    int in1, in2  
  
    printf("enter 2 ints: ");  
    scanf("%d %d", &in1, &in2);  
  
    printf("max = %d\n", max2(in1, in2));  
  
    return(0);  
}
```

Τυπικές και πραγματικές παράμετροι

- Οι παράμετροι που ορίζονται κατά την υλοποίηση μιας συνάρτησης ονομάζονται **τυπικές παράμετροι**
 - τα ονόματά τους είναι **συμβολικά**
 - έτσι ώστε ο κώδικας της συνάρτησης να μπορεί να αναφερθεί στις τιμές που **θα περαστούν** κατά την κλήση
 - ο τύπος τους 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 ονομάζονται **πραγματικές παράμετροι**
 - για κάθε κλήση, η συνάρτηση μπορεί να δέχεται **διαφορετικές πραγματικές** παραμέτρους για τις **ίδιες τυπικές** παραμέτρους

τυπικές παράμετροι

```
int max2(int num1, int num2) {  
    if (num1 > num2)  
        return(num1);  
    else  
        return(num2);  
}
```

αναφορές σε τυπικές παραμέτρους

πραγματικές
παράμετροι

```
...  
res = max2(5, 10);  
...
```

10

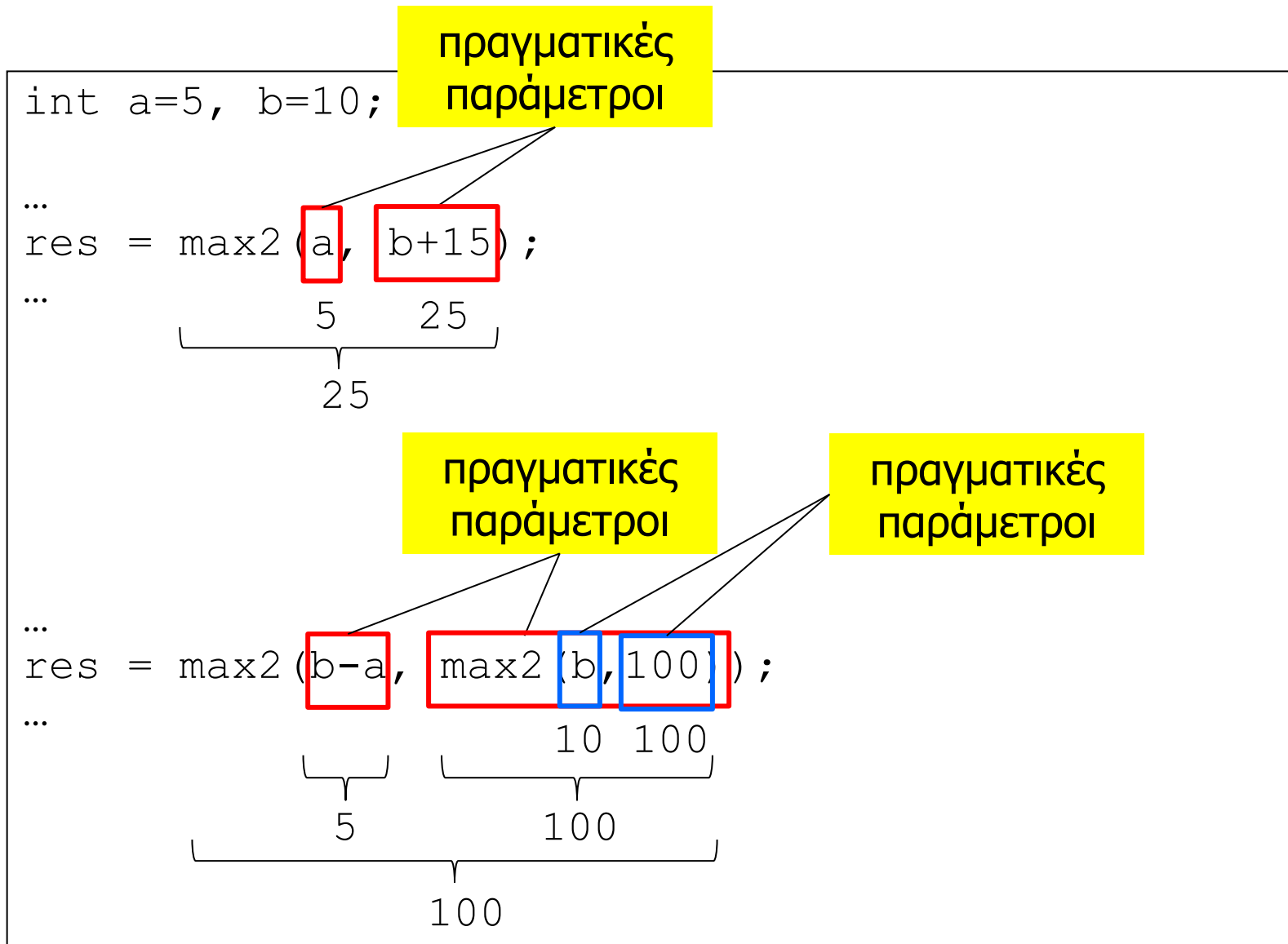
πραγματικές
παράμετροι

```
...  
res = max2(25, 10);  
...
```

25

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
- Παράμετρος κλήσης για τυπική παράμετρο T μπορεί να είναι οποιαδήποτε έκφραση αποτιμάται σε T
 - κυριολεκτικό, μεταβλητή, έκφραση, κλήση συνάρτησης
- Η έκφραση που δίνεται ως παράμετρος κλήσης **αποτιμάται πριν** πραγματοποιηθεί η κλήση
 - σαν πραγματική παράμετρος της κλήσης συνάρτησης θα περαστεί το **αποτέλεσμα** της αποτίμησης



```
#include <stdio.h>

int max2(int num1, int num2) {
    if (num1 > num2) {
        return(num1);
    }
    else {
        return(num2);
    }
}

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

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

    printf("max is %d\n", max2(in1, max2(in2, in3)));

    return(0);
}
```



```

#include <stdio.h>

int max2(int num1, int num2) {
    if (num1 > num2) {
        return(num1);
    }
    else {
        return(num2);
    }
}

int max3(int num1, int num2, int num3) {
    return( max2(num1, max2(num2, num3)) );
}

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

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

    printf("max is %d\n", max3(in1, in2, in3));

    return(0);
}

```

Εκτέλεση συνάρτησης

Έστω ότι μέσα από την συνάρτηση f_1 καλείται μια άλλη συνάρτηση f_2

1. Η εκτέλεση της f_1 **σταματάει** (χωρίς να τερματίζει) στο σημείο όπου γίνεται η κλήση της f_2
2. Γίνεται **υπολογισμός** των πραγματικών παραμέτρων και **αρχικοποίηση** των τυπικών παραμέτρων της f_2 .
3. Γίνεται **δέσμευση** των τοπικών μεταβλητών της f_2 .
4. **Αρχίζει** η εκτέλεση του κώδικα της f_2 (που με την σειρά της μπορεί να καλέσει μια άλλη συνάρτηση)
5. Ολοκληρώνεται η εκτέλεση της f_2
6. Η εκτέλεση **συνεχίζεται** στην f_1 στο σημείο που έγινε η κλήση, όπου **επιστρέφεται το αποτέλεσμα** της κλήσης της f_2

Πλαίσιο εκτέλεσης συνάρτησης

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

Η συνάρτηση `main`

- Η `main` είναι η συνάρτηση με την κλήση της οποίας **αρχίζει** η εκτέλεση του προγράμματος
- Η επιστροφή (`return`) μέσα από την `main` οδηγεί στον **τερματισμό** του προγράμματος
- Ποιος καλεί την `main` και ποιος χρησιμοποιεί την τιμή που επιστρέφεται μέσα από την `main`;
 - το «περιβάλλον εκτέλεσης» – βλέπε επόμενο εξάμηνο
- Τι είναι / συμβολίζουν οι παράμετροι της `main`;
 - βλέπε αργότερα – λίγη υπομονή ακόμα ...
- Υπάρχει κώδικας που δεν έχει `main`;
 - ναι, για παράδειγμα οι βιβλιοθήκες – βλέπε επόμενο εξάμηνο

```
... f2 (...) {  
    <A>  
}
```

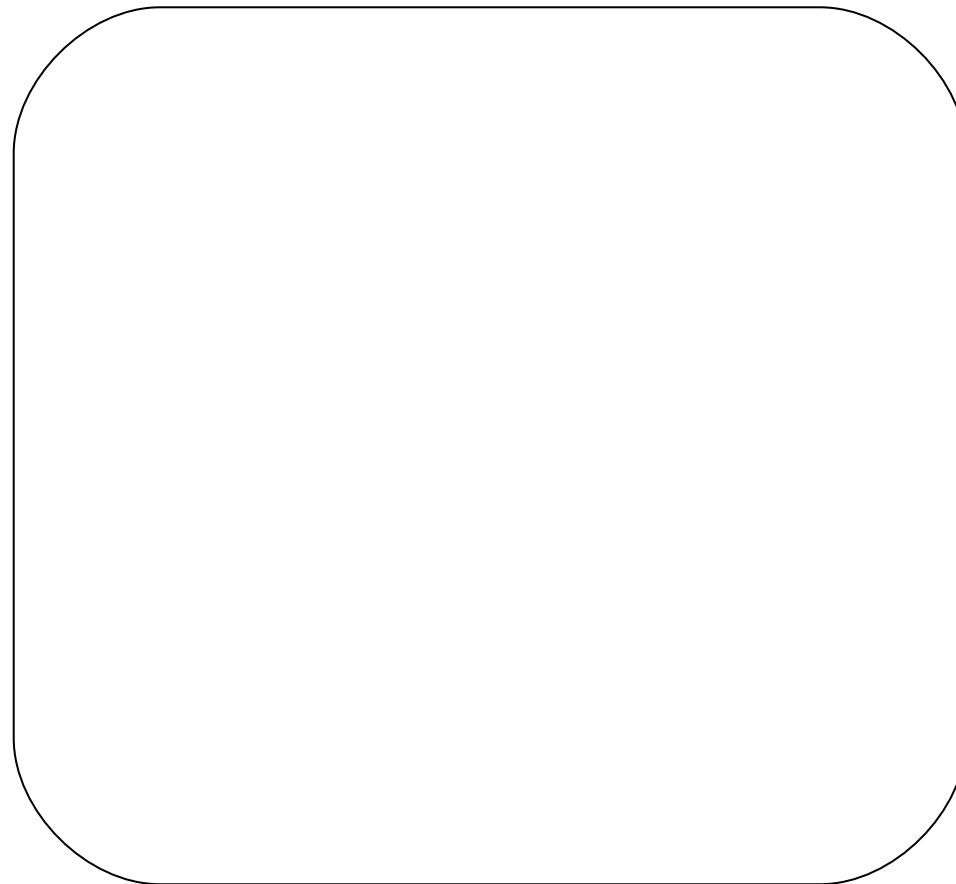
```
... f1 (...) {  
    <B>  
    f2 (...);  
    <C>  
}
```

```
... main (...) {  
    <D>  
    f1 (...);  
    <E>  
}
```

```
... f2 (...) {  
  <A>  
}
```

```
... f1 (...) {  
  <B>  
  f2 (...);  
  <C>  
}
```

```
→ ... main (...) {  
  <D>  
  f1 (...);  
  <E>  
}
```



```
... f2 (...) {  
  <A>  
}
```

```
... f1 (...) {  
  <B>  
  f2 (...);  
  <C>  
}
```

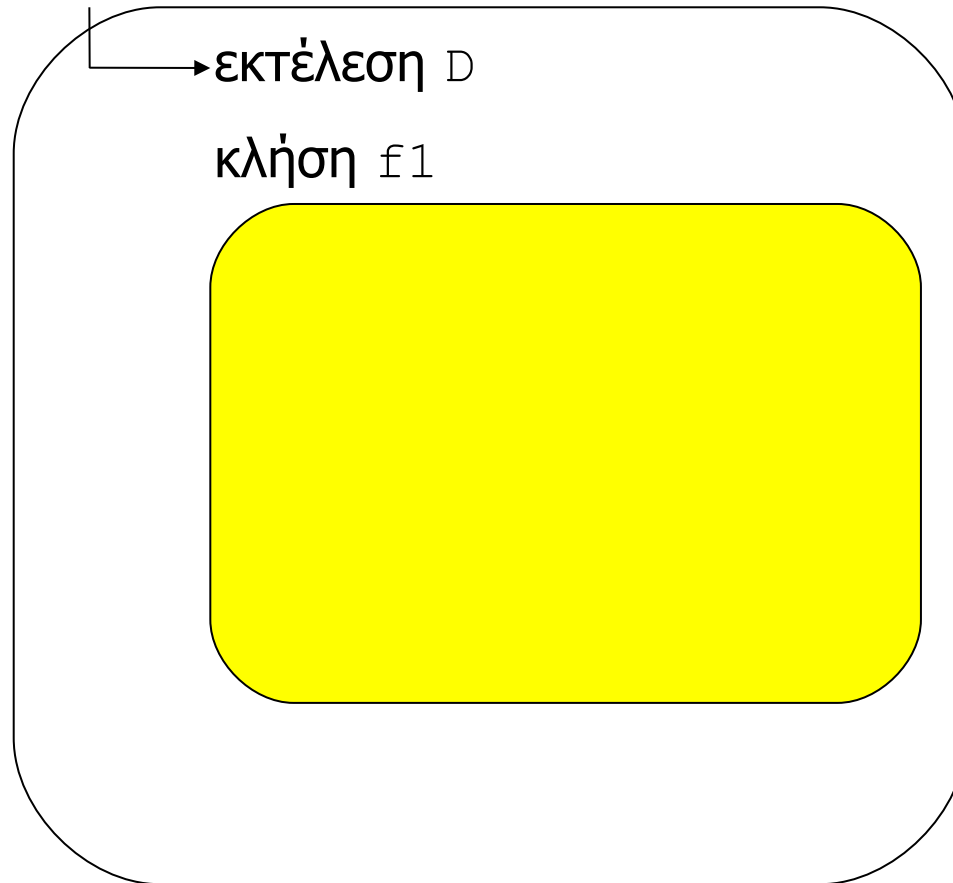
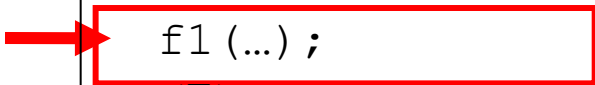
```
... main (...) {  
  <D>  
  f1 (...);  
  <E>  
}
```

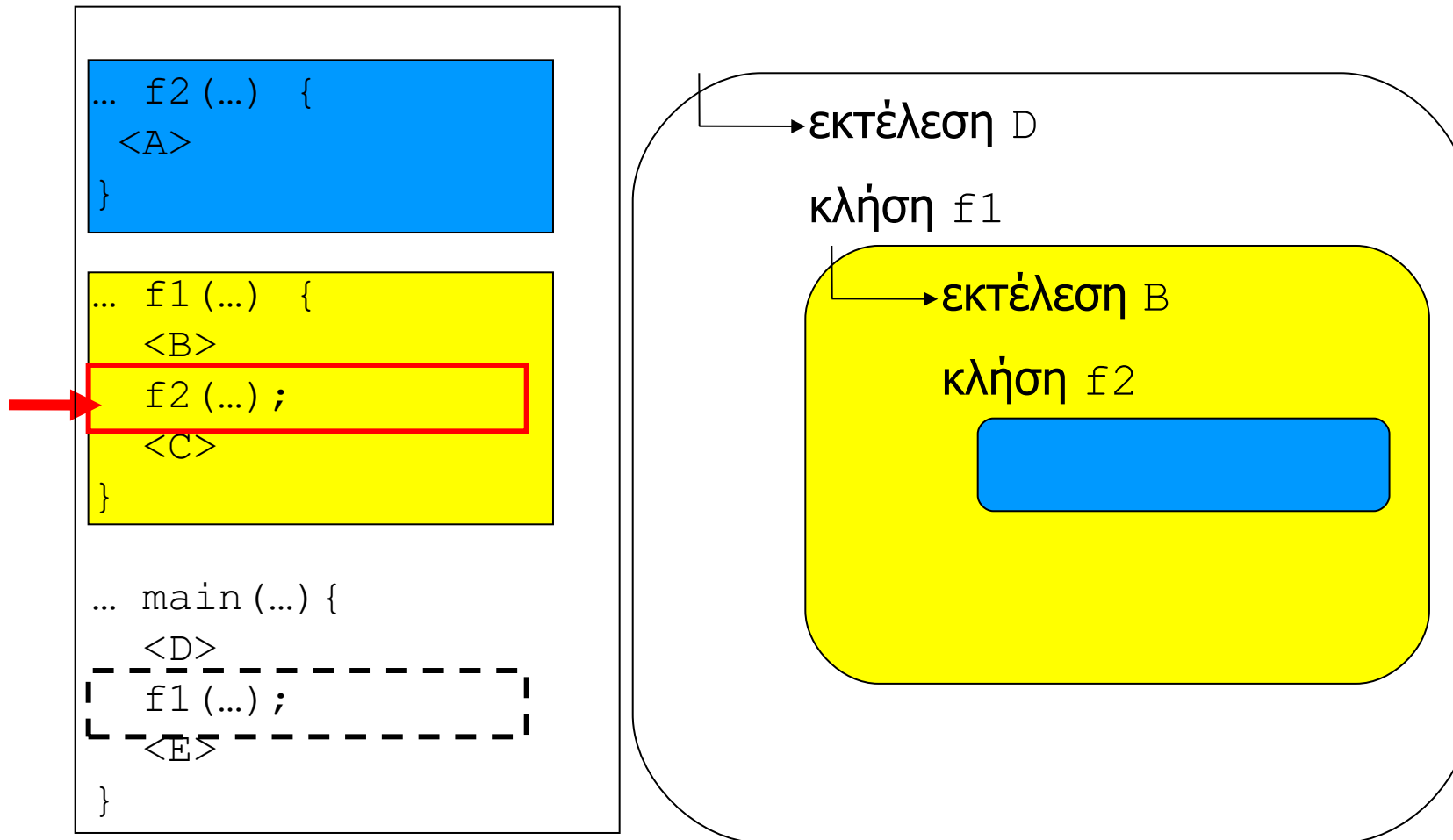
→ εκτέλεση D

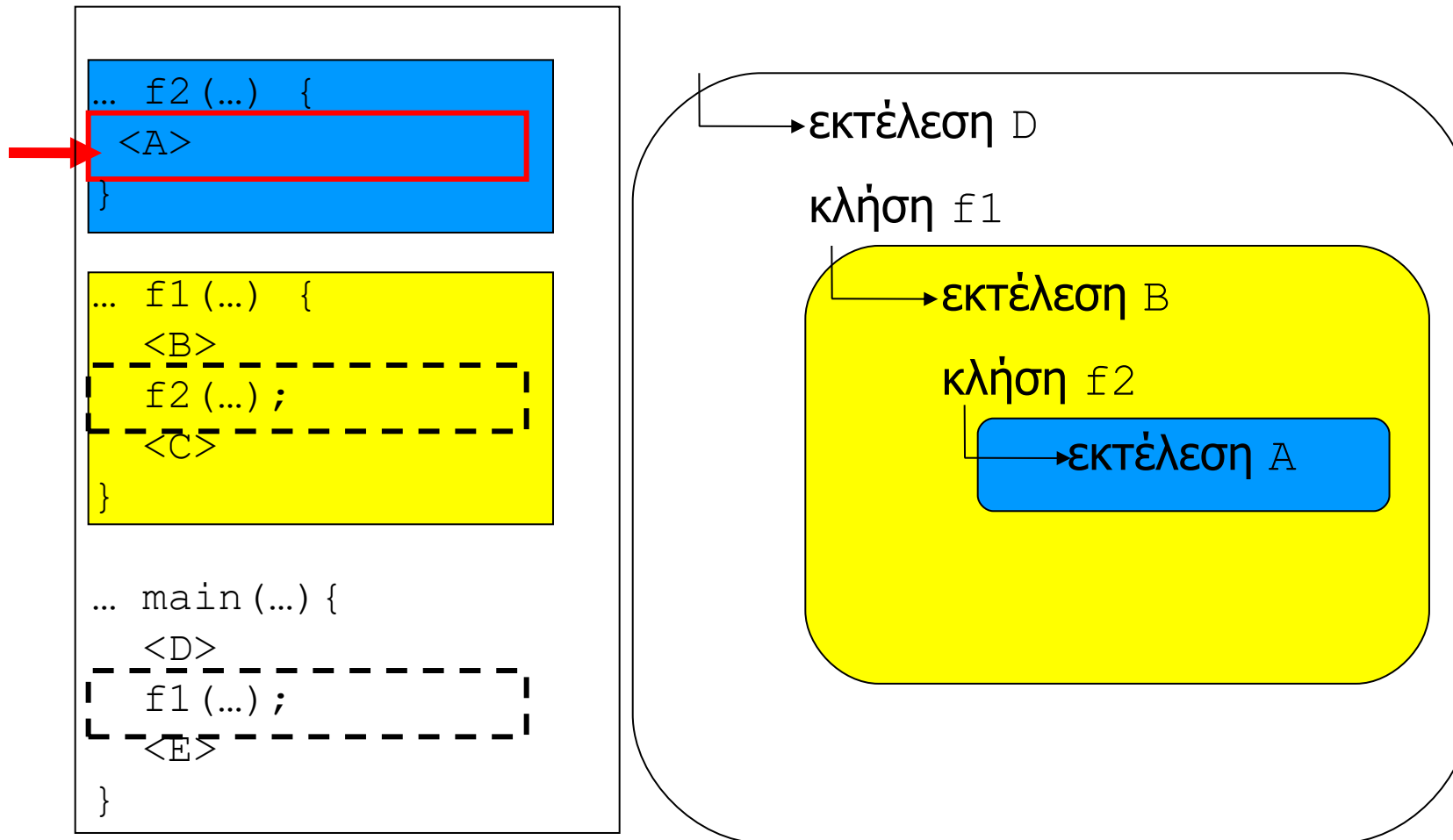
```
... f2 (...) {  
  <A>  
}
```

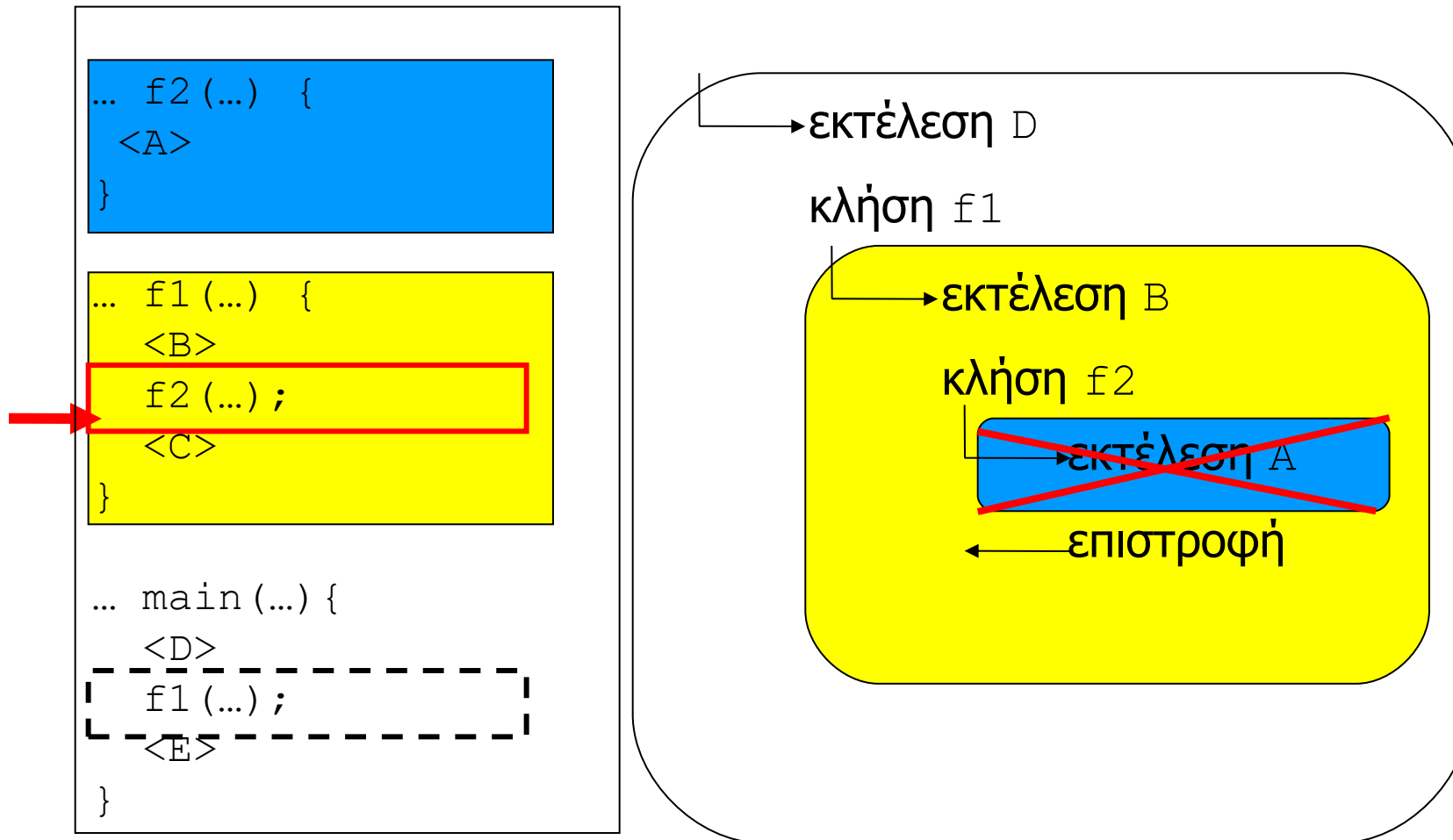
```
... f1 (...) {  
  <B>  
  f2 (...);  
  <C>  
}
```

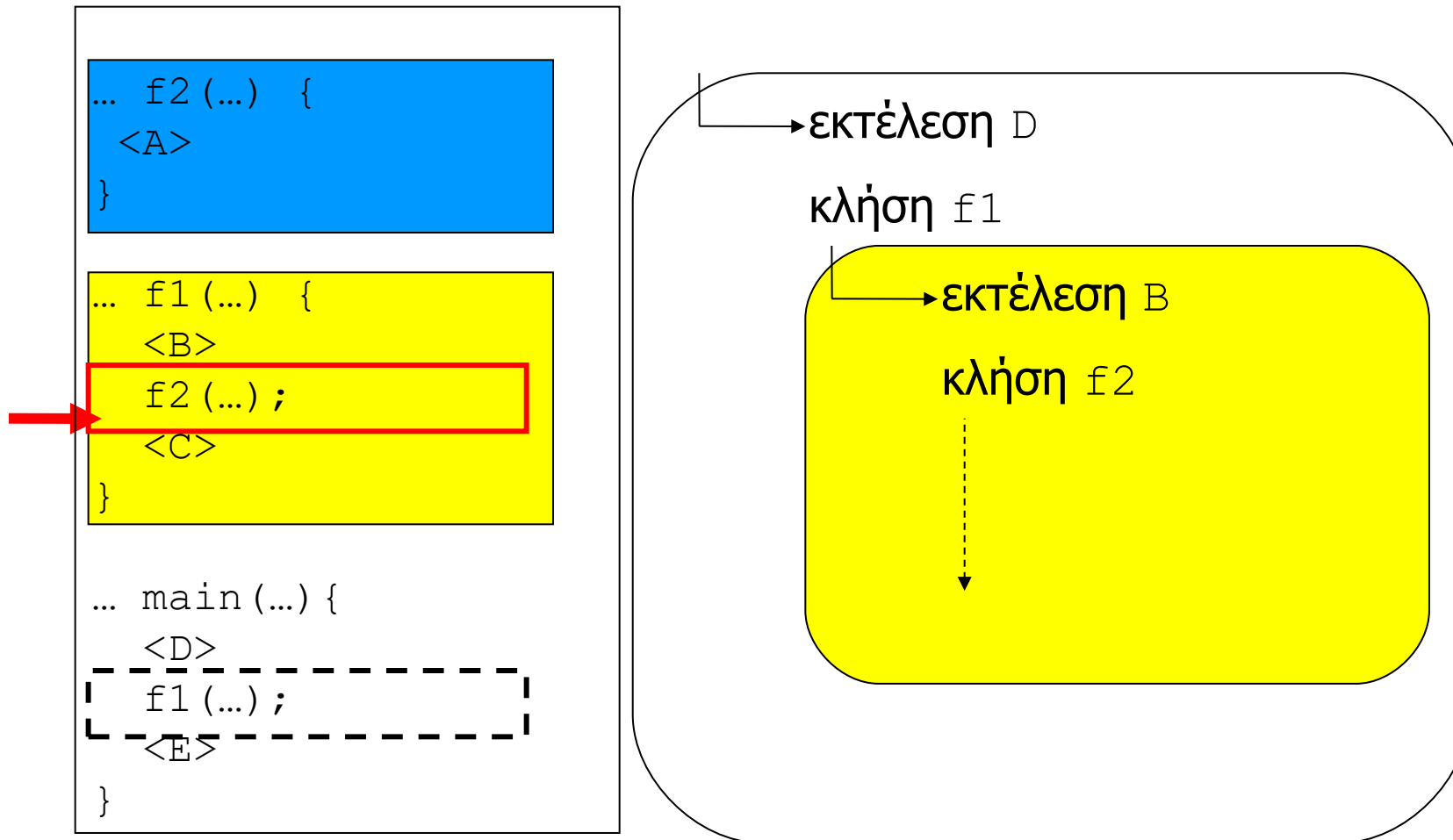
```
... main (...) {  
  <D>  
  f1 (...);  
  <E>  
}
```

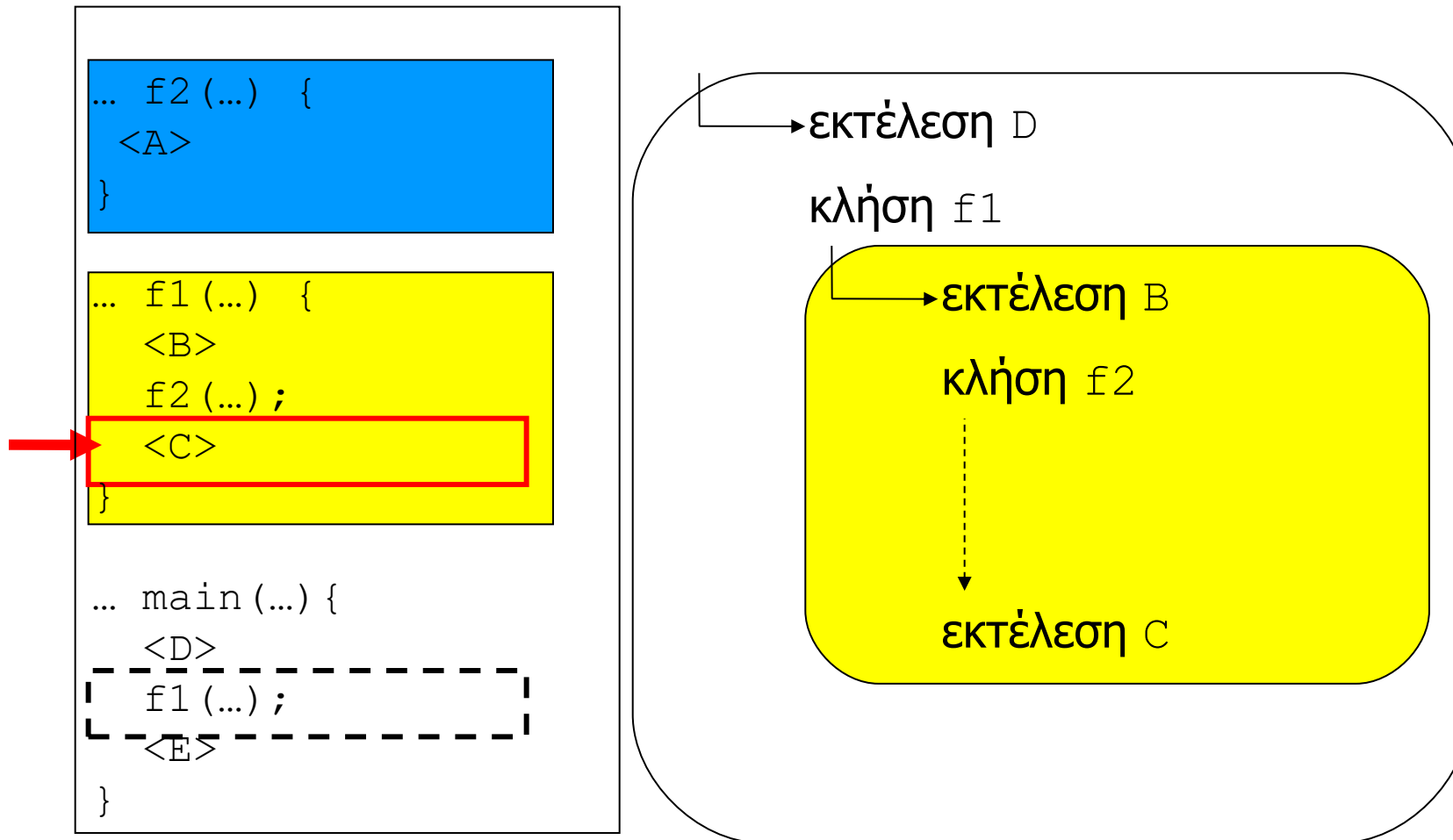








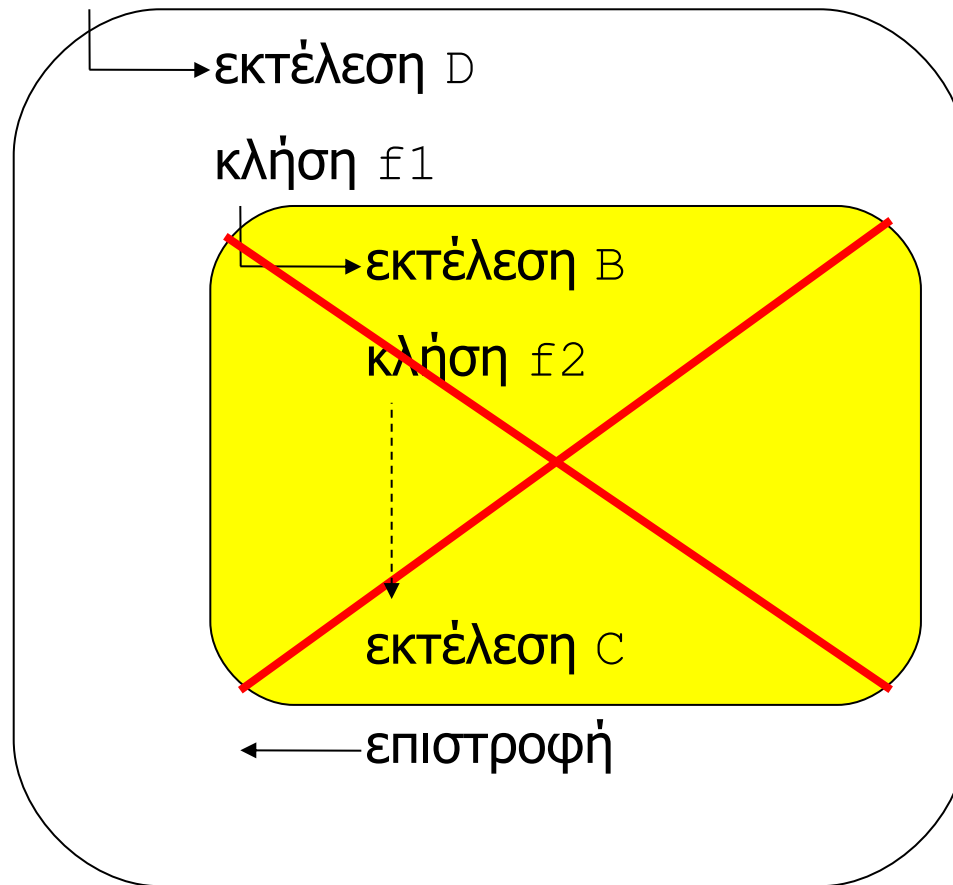
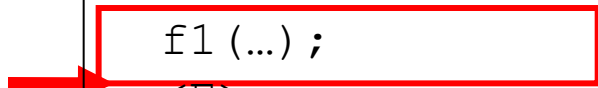




```
... f2 (...) {  
  <A>  
}
```

```
... f1 (...) {  
  <B>  
  f2 (...);  
  <C>  
}
```

```
... main (...) {  
  <D>  
  f1 (...);  
  <E>  
}
```



```
... f2 (...) {  
  <A>  
}
```

```
... f1 (...) {  
  <B>  
  f2 (...);  
  <C>  
}
```

```
... main (...) {  
  <D>  
  f1 (...);  
  <E>  
}
```

εκτέλεση D

κλήση f1


```
... f2 (...) {  
  <A>  
}
```

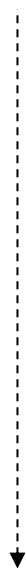
```
... f1 (...) {  
  <B>  
  f2 (...);  
  <C>  
}
```

```
... main (...) {  
  <D>  
  f1 (...);  
  <E>  
}
```

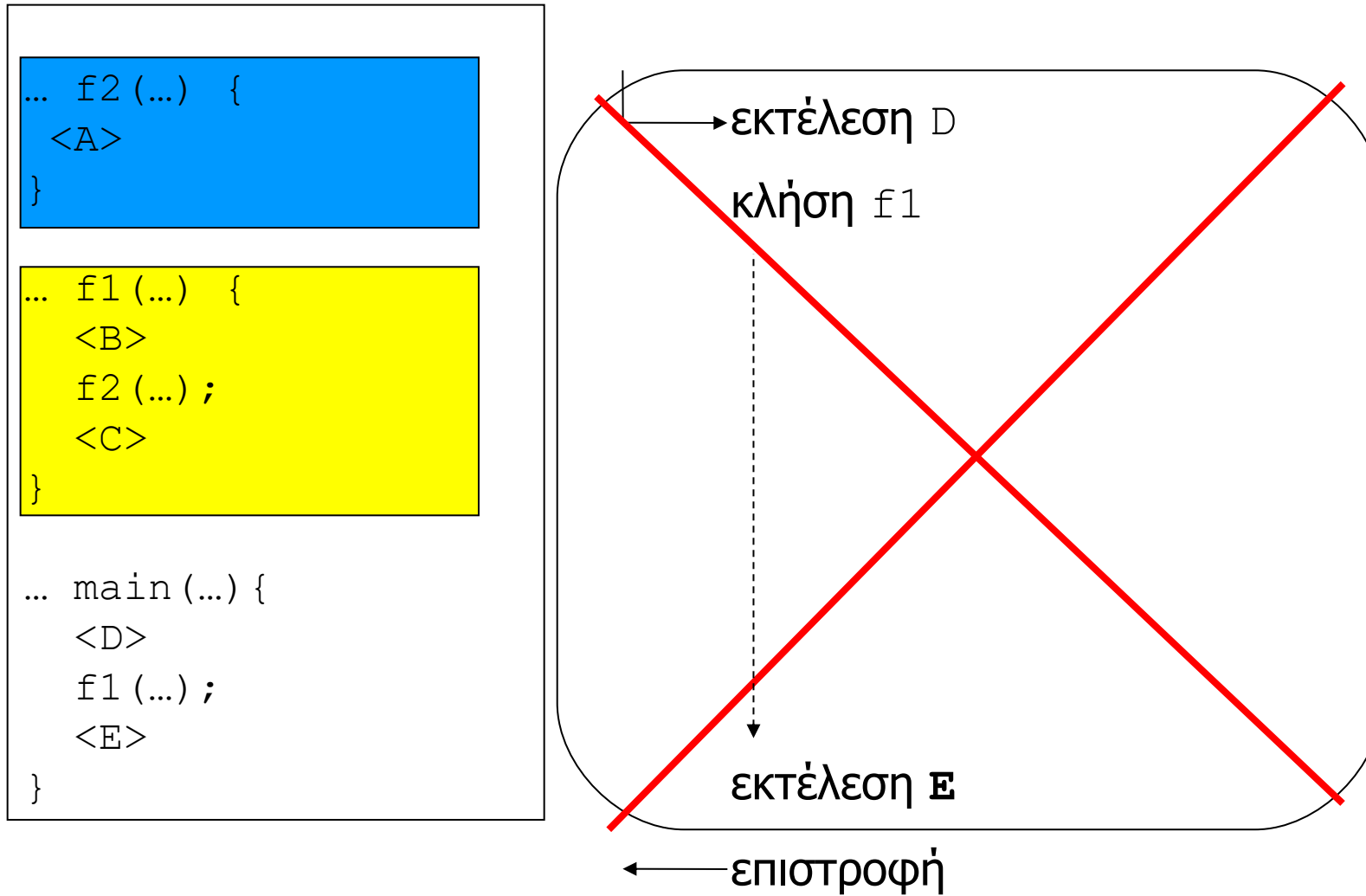


εκτέλεση D

κλήση f1



εκτέλεση E



```
... f2 (...) {  
    <A>  
}
```

```
... f1 (...) {  
    <B>  
    f2 (...);  
    <C>  
}
```

```
... main (...) {  
    <D>  
    f1 (...);  
    <E>  
}
```

Τοπικές και καθολικές μεταβλητές

Τοπικές μεταβλητές

- Κάθε συνάρτηση μπορεί να δηλώνει **νέες** «δικές της» (**τοπικές**) μεταβλητές
 - που χρησιμοποιεί για τους δικούς της σκοπούς / επεξεργασία
- Οι τυπικές παράμετροι μπορεί να θεωρηθούν ως **ειδικές** τοπικές μεταβλητές που χρησιμοποιούνται για την αποθήκευση (και πρόσβαση) των πραγματικών παραμέτρων 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
 - για να μπορεί να γίνει διαχωρισμός ανάμεσα τους

Δέσμευση μνήμης τοπικών μεταβλητών

- Η μνήμη των προσωρινών (τοπικών) μεταβλητών μιας συνάρτησης είναι **δυναμική**
 - δεσμεύεται/αποδεσμεύεται με το αντίστοιχο πλαίσιο εκτέλεσης
 - αντίθετα με τις καθολικές μεταβλητές που είναι στατικές
- Το ίδιο ισχύει για την μνήμη που χρησιμοποιείται για την αποθήκευση των πραγματικών παραμέτρων της συνάρτησης
- Η διαχείριση της τοπικής μνήμης των συναρτήσεων γίνεται μέσω ενός ειδικού μηχανισμού: της **στοίβας**
 - εναλλάξ ή/και αλυσιδωτή εκτέλεση συναρτήσεων
 - με τους λιγότερους δυνατούς πόρους
 - και την μεγαλύτερη δυνατή ταχύτητα εκτέλεσης

Στοιίβα

- Δεσμεύεται ένα (μεγάλο) **συνεχόμενο** τμήμα μνήμης
- Χρησιμοποιείται ως ουρά Last In First Out (LIFO queue) για την αποθήκευση των τιμών των παραμέτρων και των τοπικών μεταβλητών για κάθε κλήση συνάρτησης
 - και των καταχωρητών της CPU μεταξύ κλήσεων διαφορετικών συναρτήσεων (σώσιμο κατάστασης)
- Το όριο της μνήμης της στοίβας που χρησιμοποιείται ανά πάσα στιγμή υποδεικνύεται από τον **stack pointer**
- Η διαχείριση του γίνεται από το περιβάλλον εκτέλεσης
 - με **διαφανή** τρόπο, χωρίς εμπλοκή του προγραμματιστή
 - κάθε φορά που γίνεται μια νέα κλήση και κάθε φορά που τερματίζεται μια κλήση, η τιμή του stack pointer αλλάζει ώστε να δείχνει στο τρέχον πλαίσιο εκτέλεσης

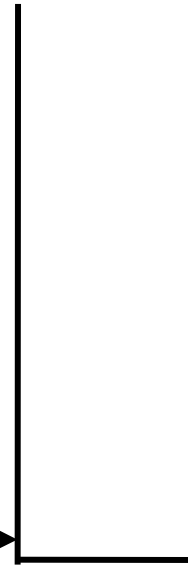
```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```

στοίβα

stack pointer SP →

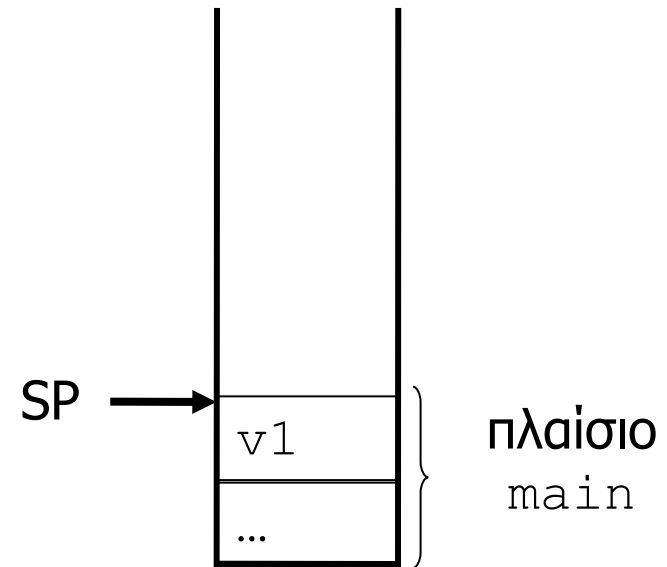



```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
→ ... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```

στοίβα



```
... f2 (...) {  
    int v5;  
    <A>  
}
```

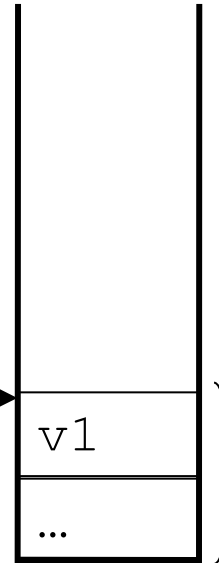
```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```



στοίβα

SP



πλαίσιο
main

```
... f2 (...) {  
    int v5;  
    <A>  
}
```

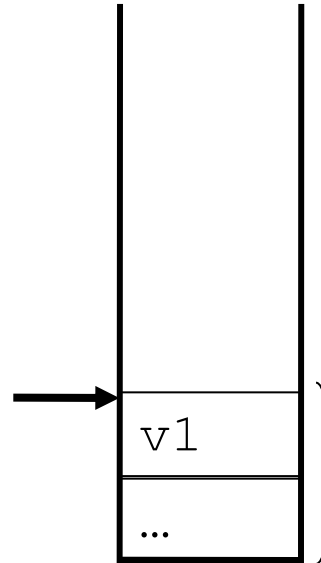
```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```



στοίβα

SP



πλαίσιο
main

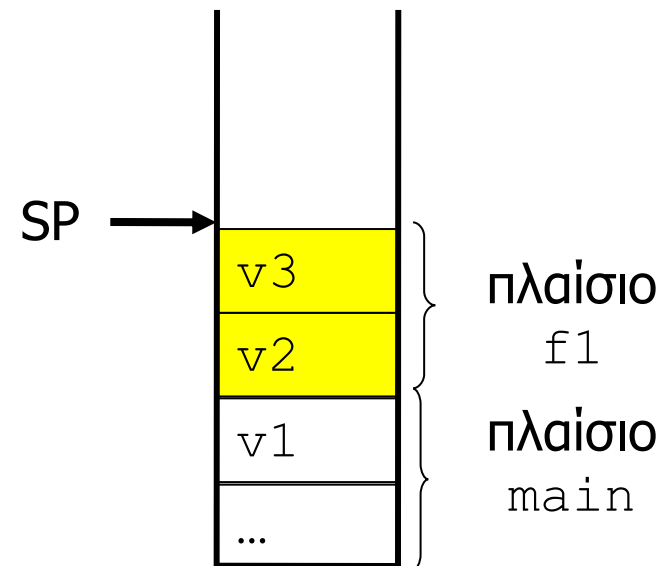
```
... f2 (...) {  
    int v5;  
    <A>  
}
```

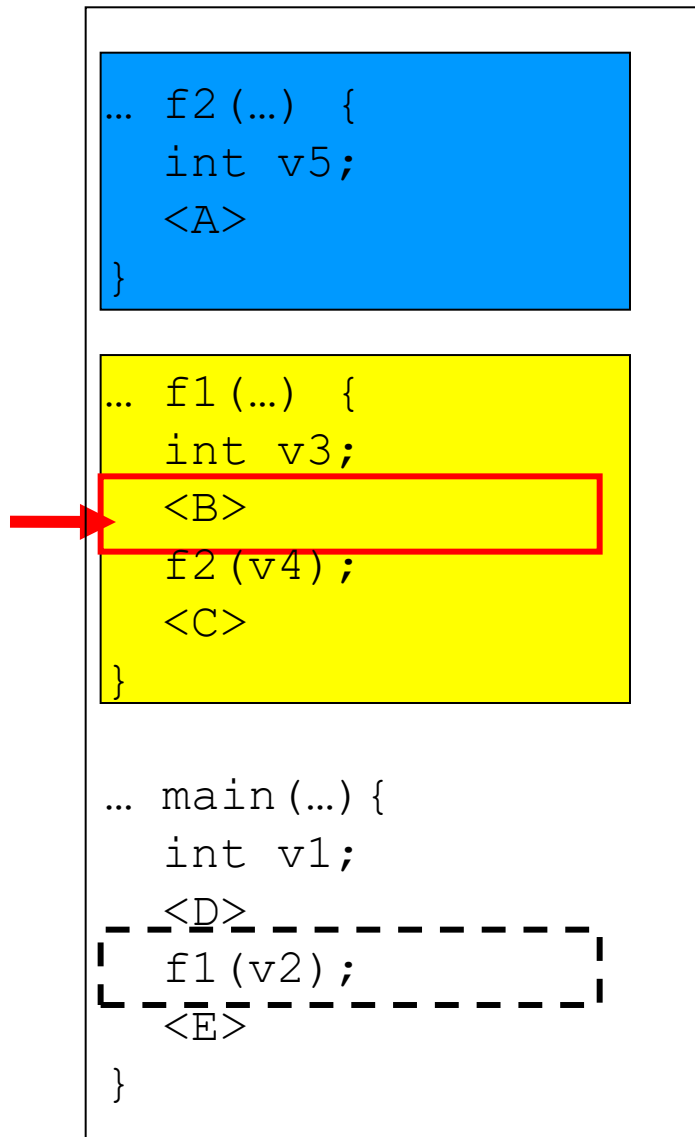


```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4);  
    <C>  
}
```

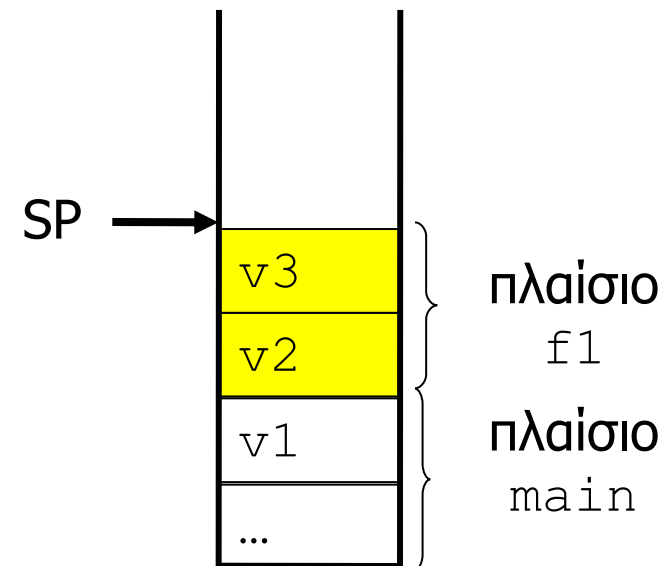
```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2);  
    <E>  
}
```

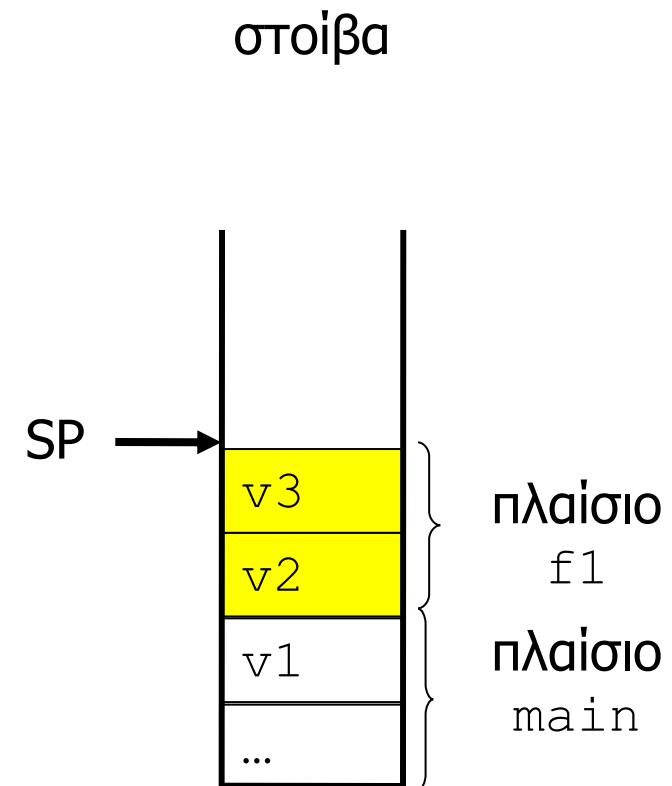
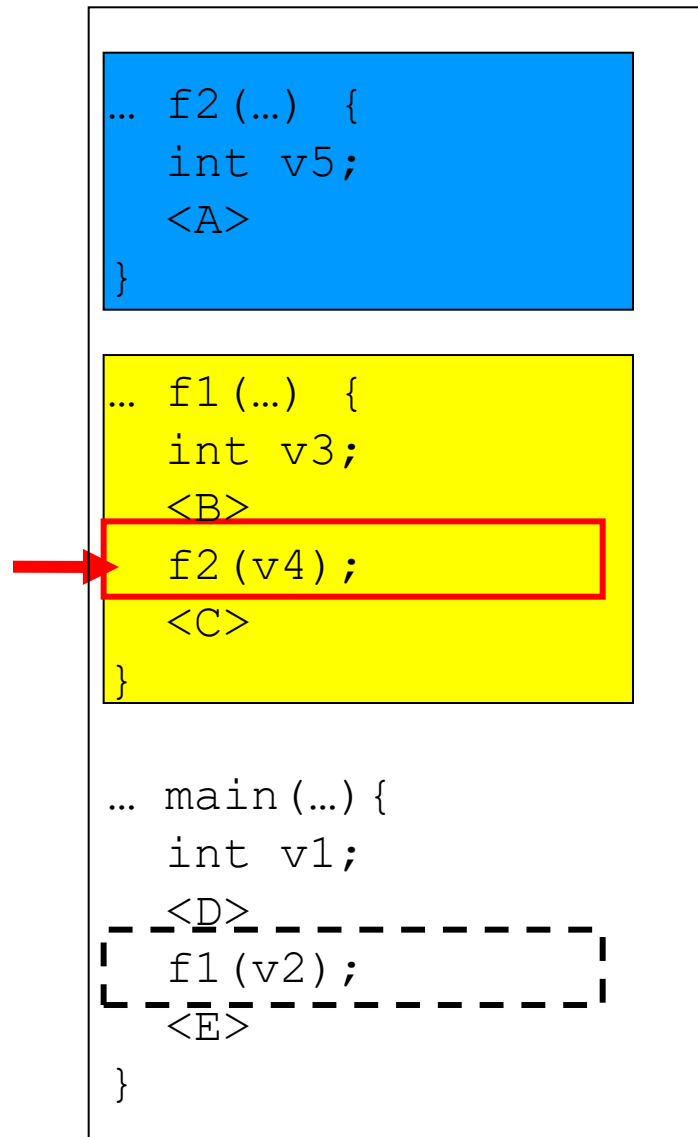
στοίβα

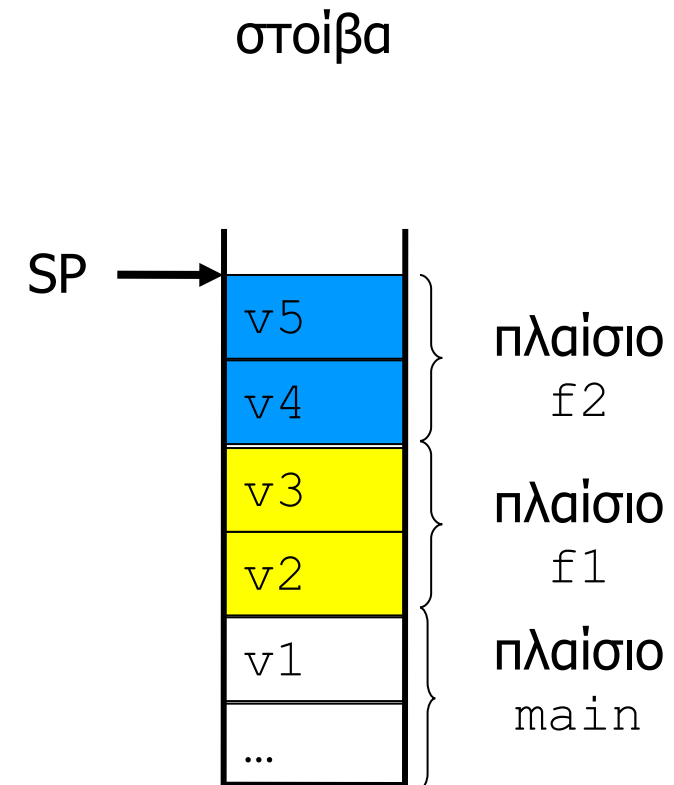
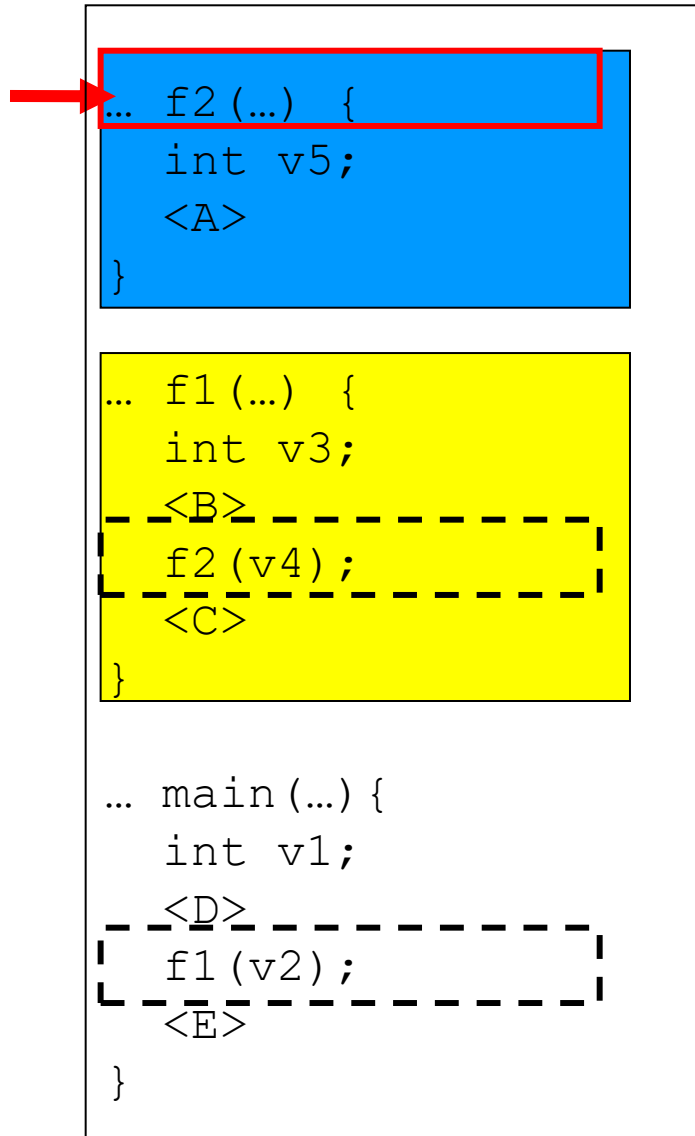


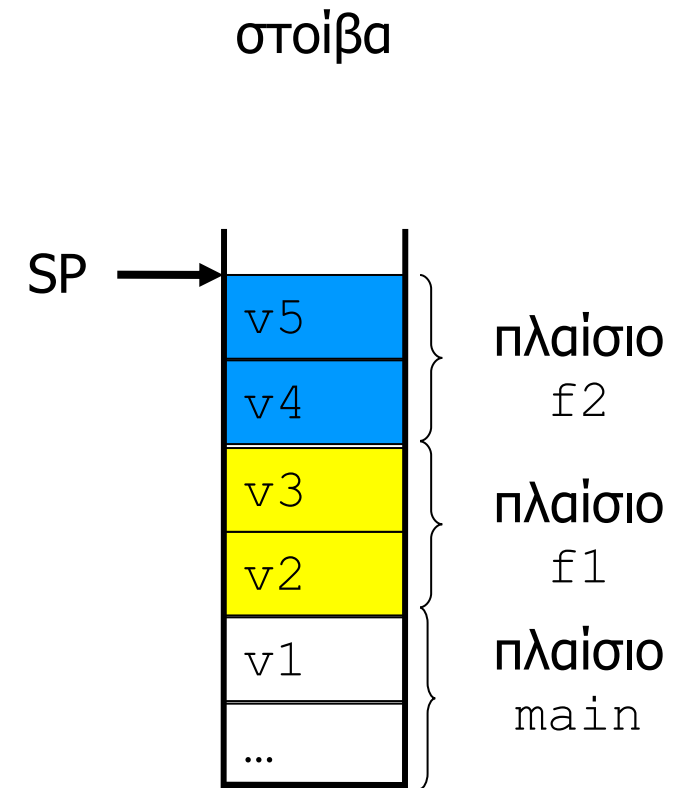
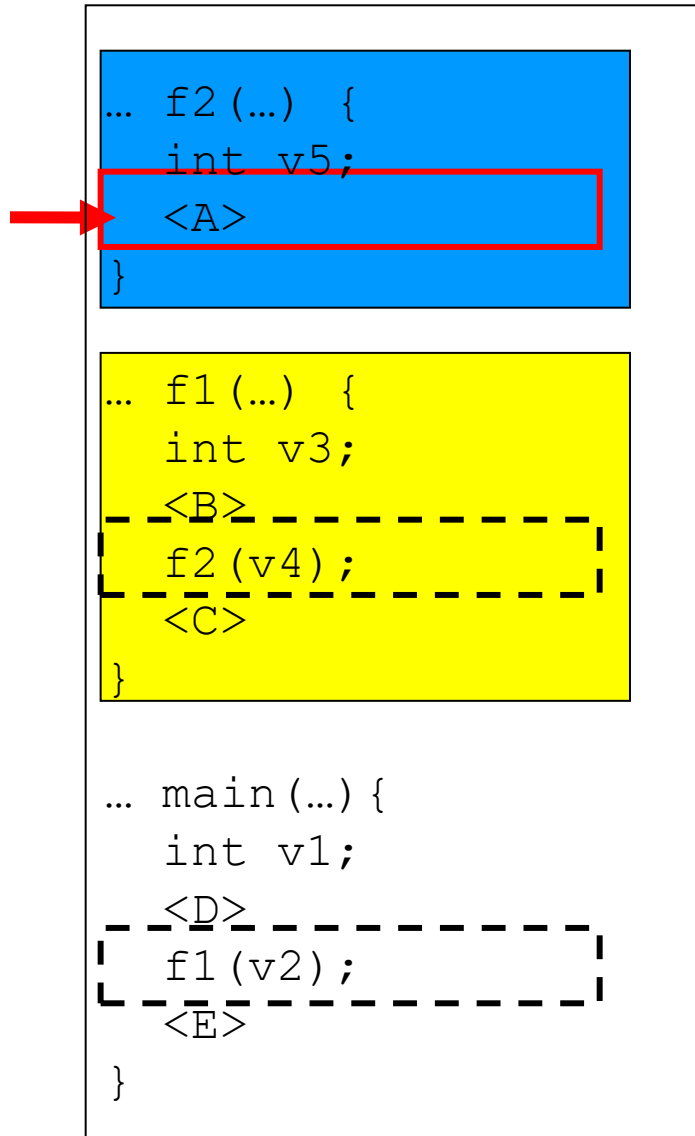


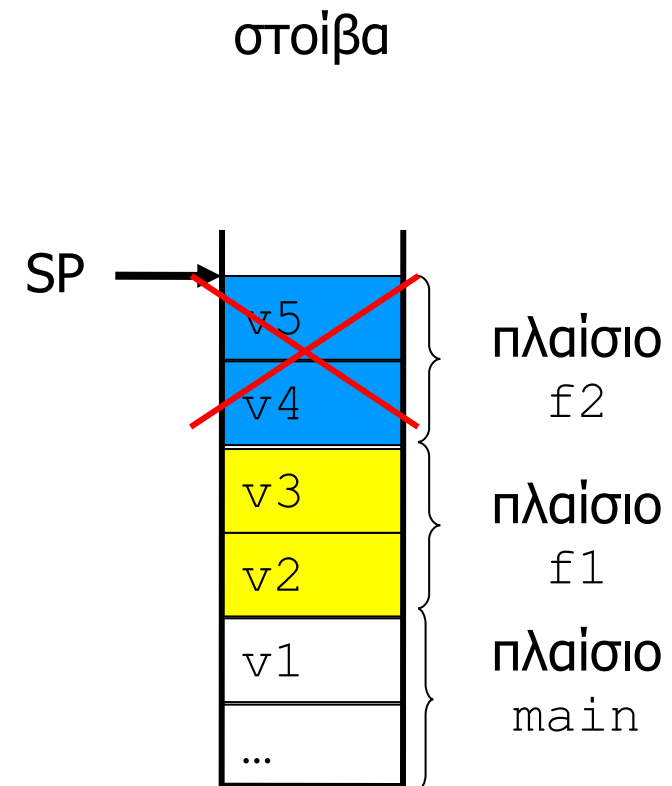
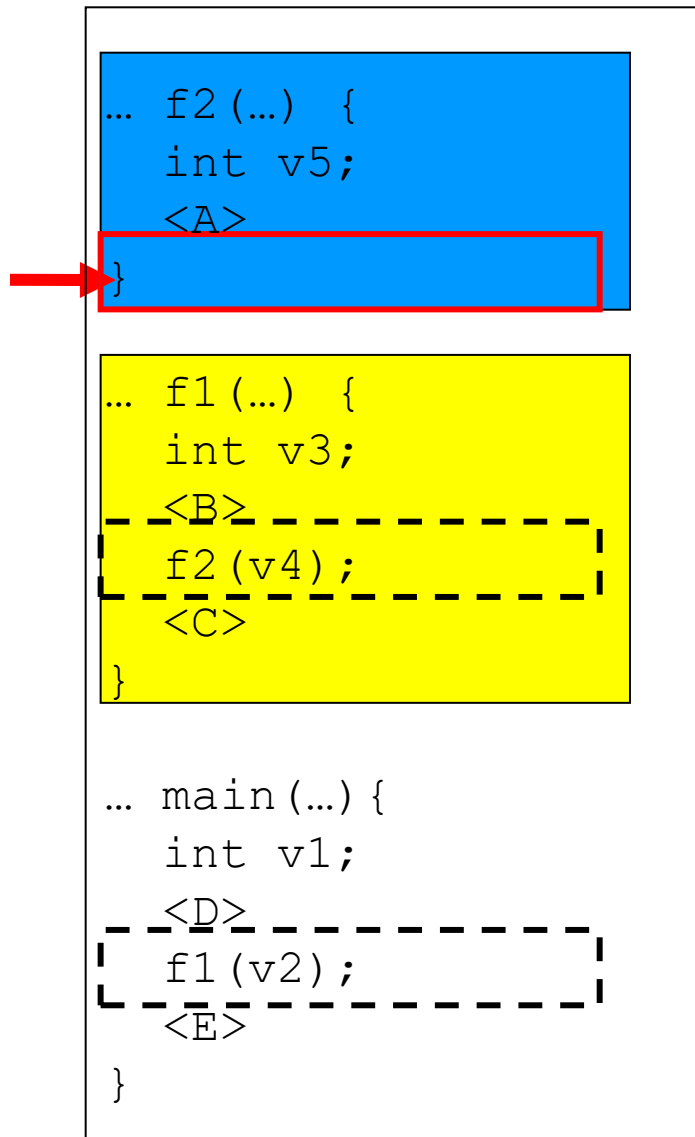
στοίβα









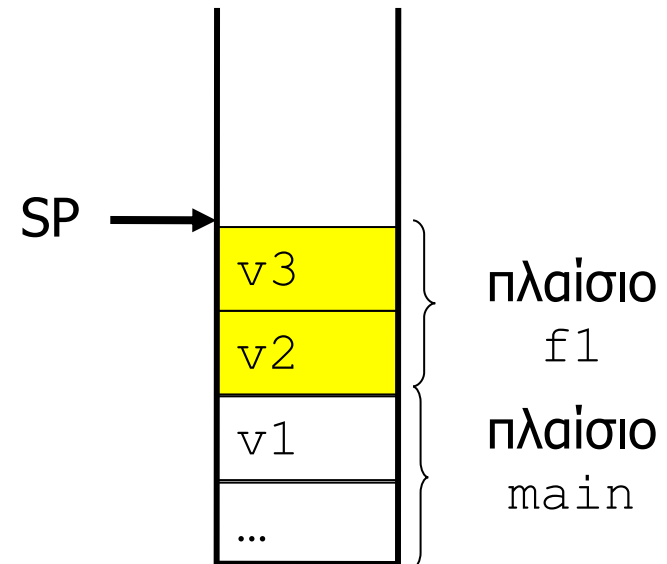


```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```

στοίβα

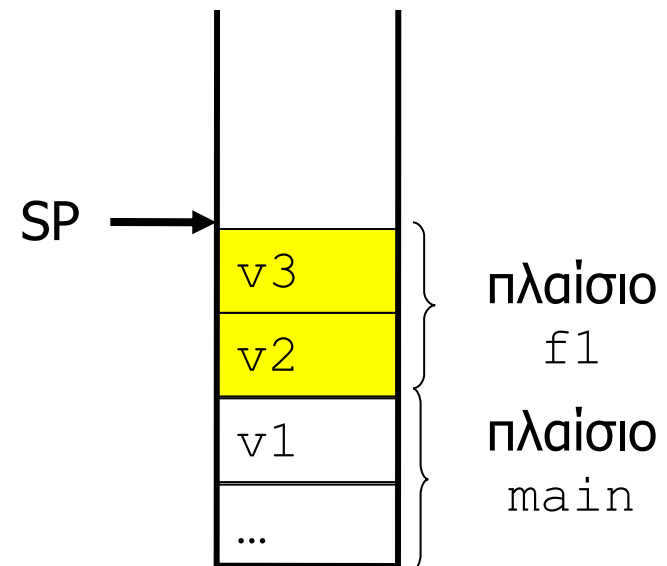


```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4);  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2);  
    <E>  
}
```

στοίβα



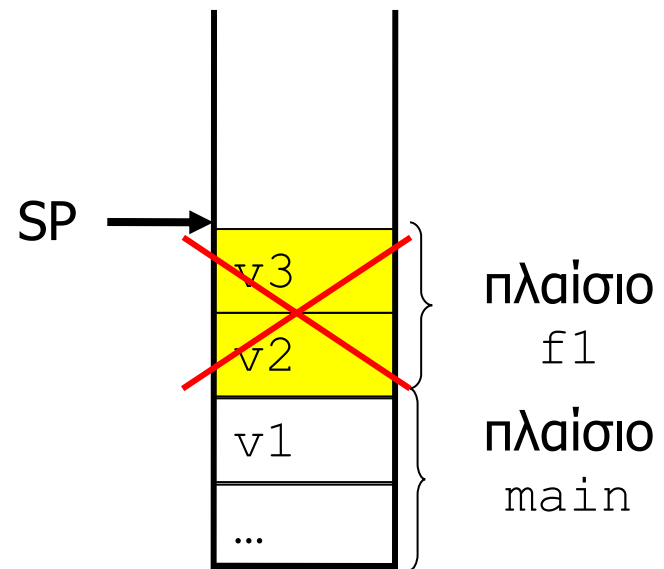
```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4);  
    <C>
```

```
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2);  
    <E>  
}
```

στοίβα



```
... f2 (...) {  
    int v5;  
    <A>  
}
```

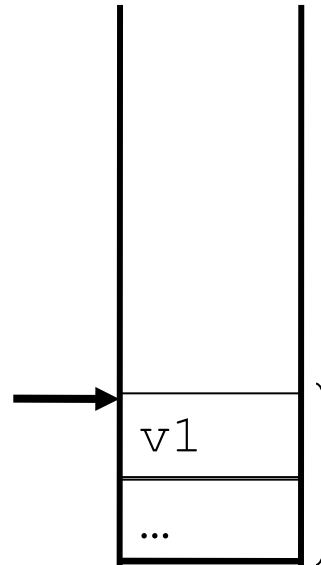
```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```



στοίβα

SP



πλαίσιο
main

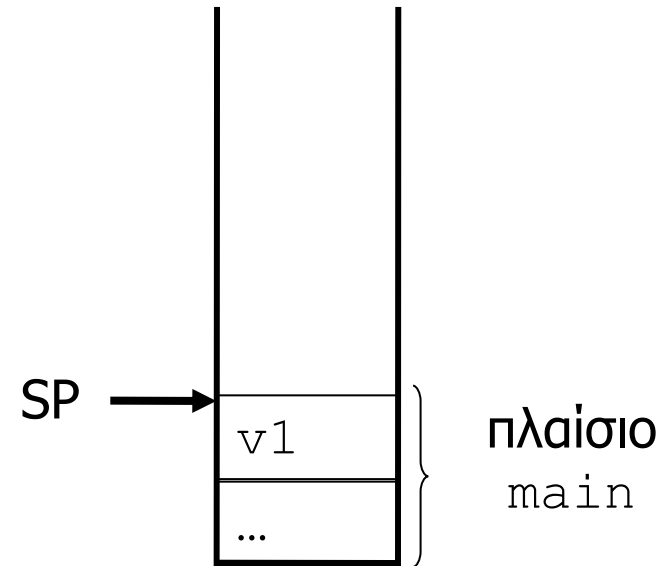
```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4);  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2);  
    <E>  
}
```



στοίβα



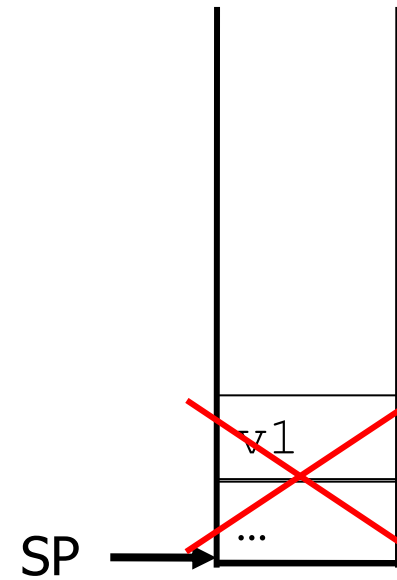
```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```



στοίβα

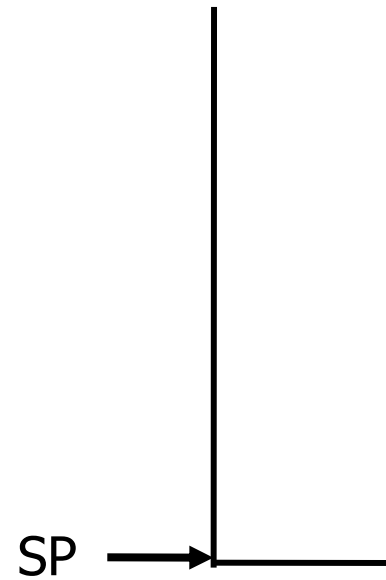


```
... f2 (...) {  
    int v5;  
    <A>  
}
```

```
... f1 (...) {  
    int v3;  
    <B>  
    f2 (v4) ;  
    <C>  
}
```

```
... main (...) {  
    int v1;  
    <D>  
    f1 (v2) ;  
    <E>  
}
```

στοίβα




```
#include <stdio.h>

void fool (int in1) {
    int var1;
    printf("Addr of in1 is %p and of var1 is %p\n", &in1, &var1);
}

void foo2 (int in2) {
    int var2;
    printf("Addr of in2 is %p and of var2 is \n", &in2, &var2);
    fool(in2);
}

void foo3 (int in3) {
    int var3;
    printf("Addr of in3 is %p and of var3 is %p\n", &in3, &var3);
    foo2(in3);
}

int main (int argc, char *argv[]) {
    int var0;
    printf("Addr of var0 is %p\n", &var0);
    foo3(var0);
    return 0;
}
```

Υπερχείλιση στοίβας

- Το μέγεθος της στοίβας είναι συνήθως **περιορισμένο**
- Το πρόγραμμα μπορεί να εξαντλήσει την μνήμη της στοίβας, και αυτή να υπερχειλίσει (**stack overflow**)
 - γίνονται πολλές αλυσιδωτές κλήσεις συνάρτησης
 - το συνολικό μέγεθος των τοπικών μεταβλητών είναι υπερβολικά μεγάλο για να χωρέσει στην στοίβα
- Τότε η εκτέλεση του προγράμματος **τερματίζεται**
 - με μήνυμα λάθους (αναλόγως με το λειτουργικό σύστημα)

Καθολικές (global) μεταβλητές

- Ορίζονται στην αρχή του κειμένου του προγράμματος
- **Έξω** από όλες τις συναρτήσεις (και την `main`)
- Αντίθετα από τις τοπικές μεταβλητές που ορίζονται **μέσα** στο πλαίσιο μιας συνάρτησης

Διάρκεια ζωής μεταβλητών

- Οι καθολικές μεταβλητές είναι **μόνιμες**
 - υφίστανται **καθ' όλη την διάρκεια** της εκτέλεσης
 - η μνήμη τους δεσμεύεται **στατικά** προτού αρχίσει η εκτέλεση της `main` (ανεξάρτητα από την στοίβα)
- Οι τοπικές μεταβλητές (και οι πραγματικές παράμετροι μιας συνάρτησης) είναι **προσωρινές**
 - υφίστανται **μόνο** όσο κρατά η εκτέλεση της συνάρτησης
 - η μνήμη τους δεσμεύεται **δυναμικά** (στην στοίβα) και **εκ νέου** κάθε φορά που καλείται η συνάρτηση
- **Εξαιρέση:** `static` τοπικές μεταβλητές
 - είναι **μόνιμες**: υφίστανται και κρατούν την τιμή τους **ανάμεσα** στις διάφορες εκτελέσεις της συνάρτησης
 - αρχικοποιούνται κατά τη δήλωση – η εντολή αρχικοποίησης εκτελείται **μια φορά**, στην πρώτη κλήση της συνάρτησης

```
#include <stdio.h>

int foo(int val) {
    static int hiddenval = 0;

    hiddenval++;
    return(val+hiddenval);
}

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

    res = foo(0);
    printf("res = %d\n", res);

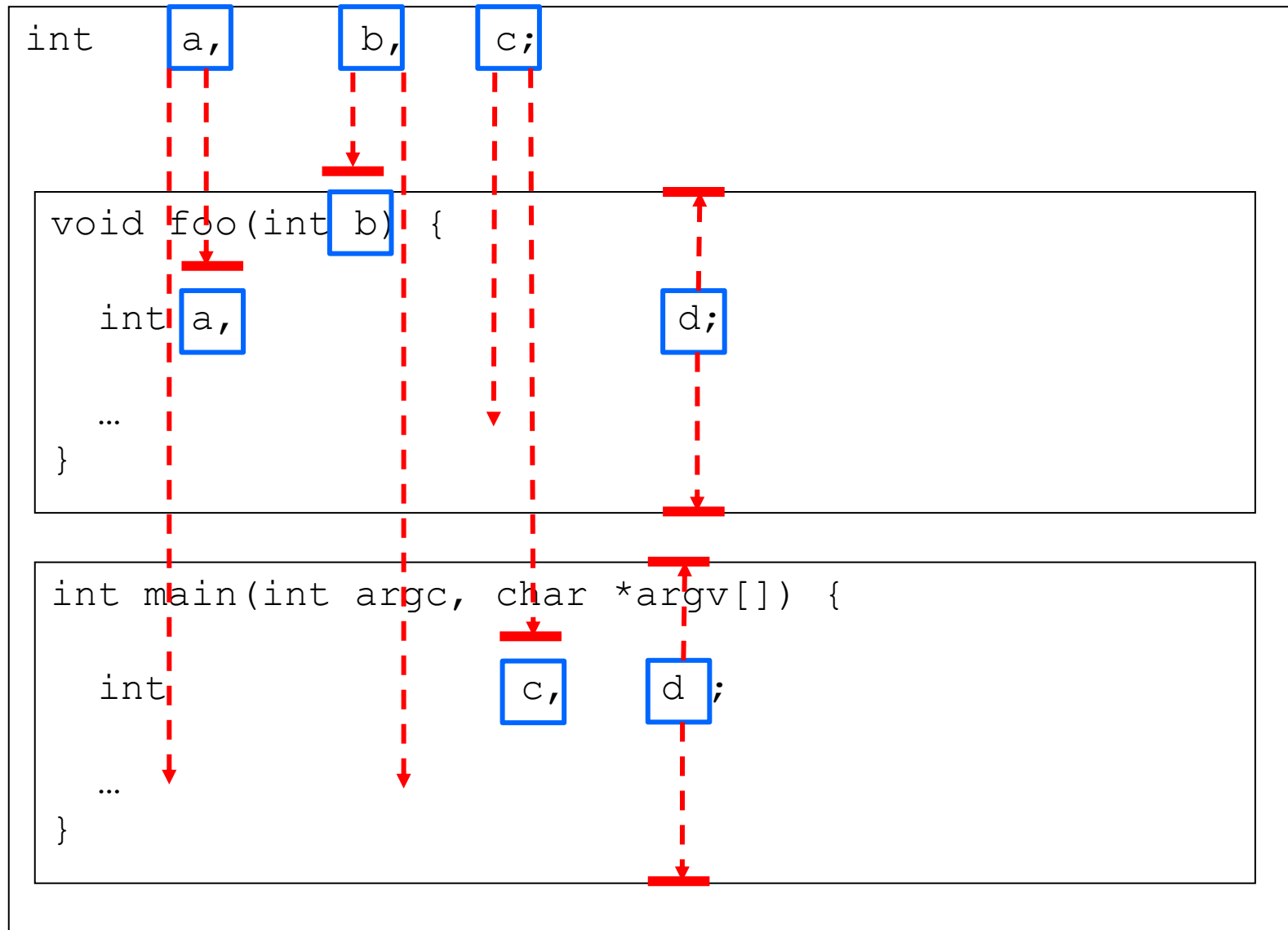
    res = foo(0);
    printf("res = %d\n", res);

    res = foo(0);
    printf("res = %d\n", res);

    return(0);
}
```

Εμβέλεια/ορατότητα μεταβλητών

- **Καθολικές μεταβλητές:** προσπελάσιμες (ορατές) από παντού / μέσα από **κάθε** συνάρτηση
- **Τοπικές μεταβλητές και τυπικές παράμετροι:** προσπελάσιμες (ορατές) **μόνο** από τον κώδικα της αντίστοιχης συνάρτησης
- Αν μια τοπική μεταβλητή (ή τυπική παράμετρος) μιας συνάρτησης έχει το ίδιο όνομα με μια καθολική μεταβλητή, **αποκρύπτει** την καθολική μεταβλητή
 - την καθιστά μη προσπελάσιμη μέσα από την συνάρτηση
- Ανά επίπεδο εμβέλειας, τα ονόματα μεταβλητών (και παραμέτρων) πρέπει να είναι μοναδικά
 - διαφορετικά υπάρχει πρόβλημα κατά την μετάφραση



```
#include <stdio.h>

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

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

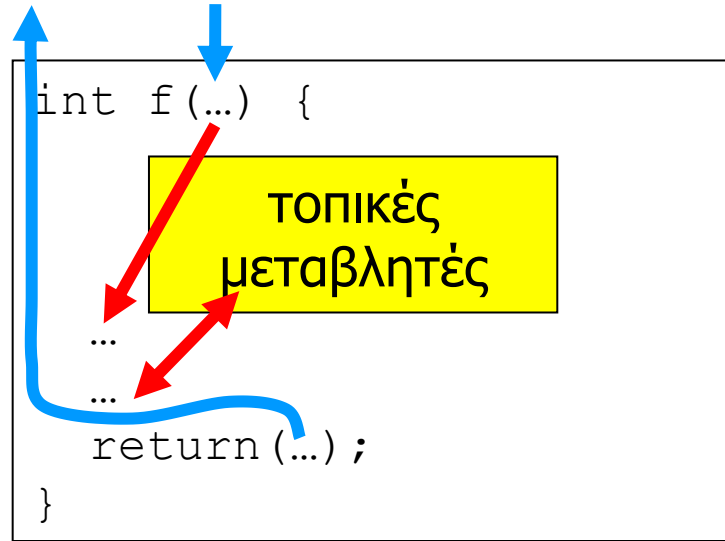
int main(int argc, char *argv[]) {
    int c = 1, d = 1;
    c = a + b;
    b = b + 1;
    printf("main: a=%d, b=%d, c=%d, d=%d\n", a, b, c, d);
    foo(c);
    printf("main: a=%d, b=%d, c=%d, d=%d\n", a, b, c, d);
    c = a + b;
    b = b + 1;
    printf("main: a=%d, b=%d, c=%d, d=%d\n", a, b, c, d);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

    return(0);
}
```

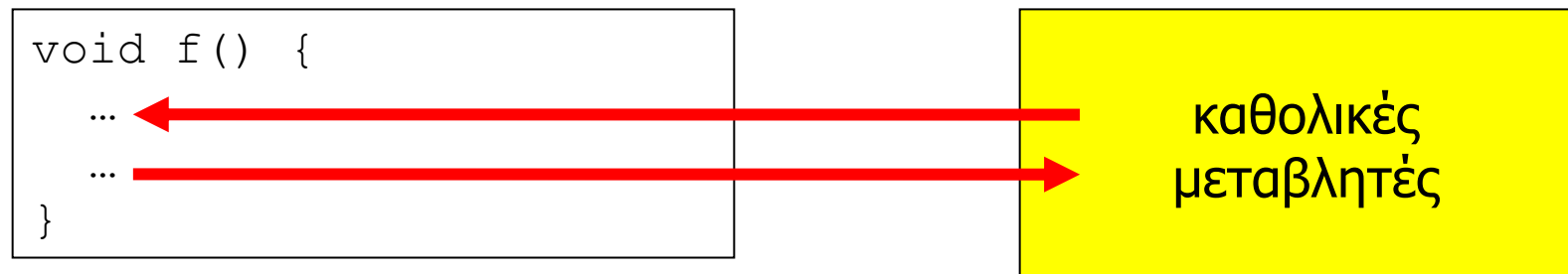

Αλλαγή καθολικών μεταβλητών

- Η αλλαγή μιας καθολικής μεταβλητής μέσα από μια συνάρτηση συνιστά μια λεγόμενη **παρενέργεια**
 - η αλλαγή δεν μπορεί να εντοπιστεί εύκολα, χωρίς να διαβάσουμε τον κώδικα της συγκεκριμένης συνάρτησης
 - οι παρενέργειες θεωρούνται κακό στυλ προγραμματισμού
- Βεβαίως, οι καθολικές μεταβλητές υπάρχουν ακριβώς για να επιτρέπουν το «πέρασμα» τιμών/δεδομένων (επικοινωνία) ανάμεσα σε διαφορετικές συναρτήσεις
- Αυτή η λύση πρέπει να επιλέγεται με **σύνεση**
 - όταν το επιθυμητό αποτέλεσμα **δεν** μπορεί να επιτευχθεί (με πιο απλό/καθαρό τρόπο) με πέρασμα κατάλληλων παραμέτρων και επιστροφή αποτελεσμάτων
 - και να τεκμηριώνεται κατάλληλα (π.χ. σύντομο σχόλιο)

προγραμματισμός χωρίς παρενέργειες



προγραμματισμός με παρενέργειες



```

int sum(int n) {
    int i, s;

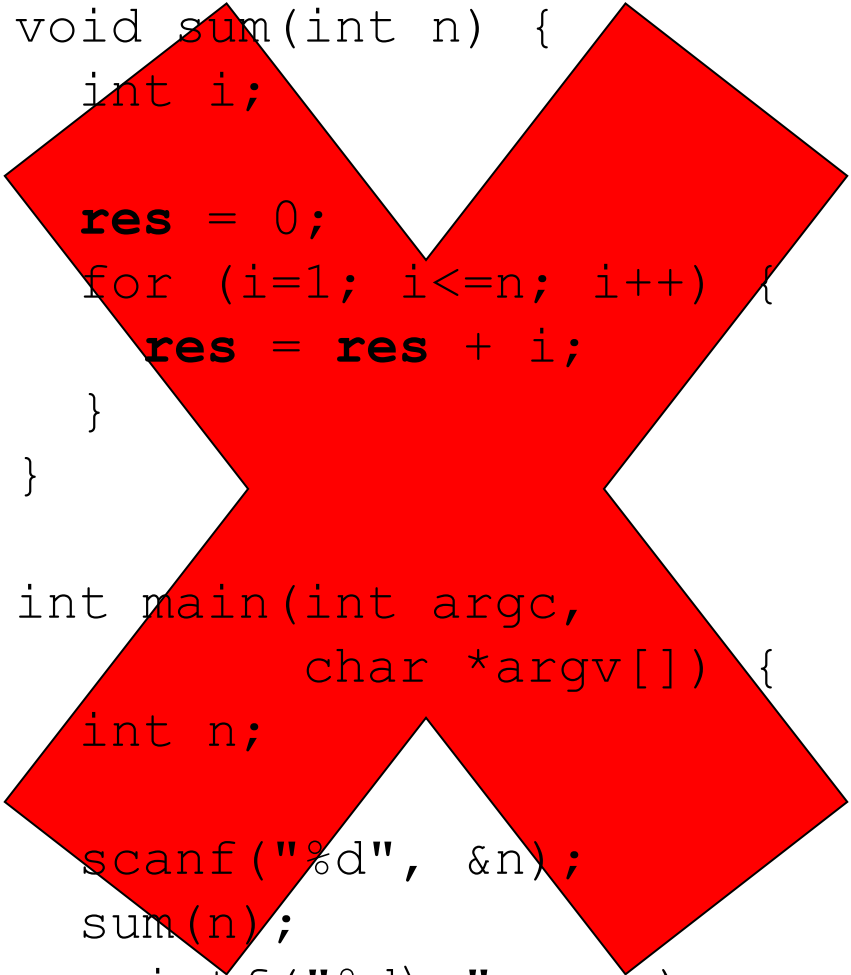
    s = 0;
    for (i=1; i<=n; i++) {
        s = s + i;
    }

    return(s);
}

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

    scanf("%d", &n);
    res = sum(n);
    printf("%d\n", res);
}

```



```

int res;

void sum(int n) {
    int i;

    res = 0;
    for (i=1; i<=n; i++) {
        res = res + i;
    }
}

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

    scanf("%d", &n);
    sum(n);
    printf("%d\n", res);
}

```