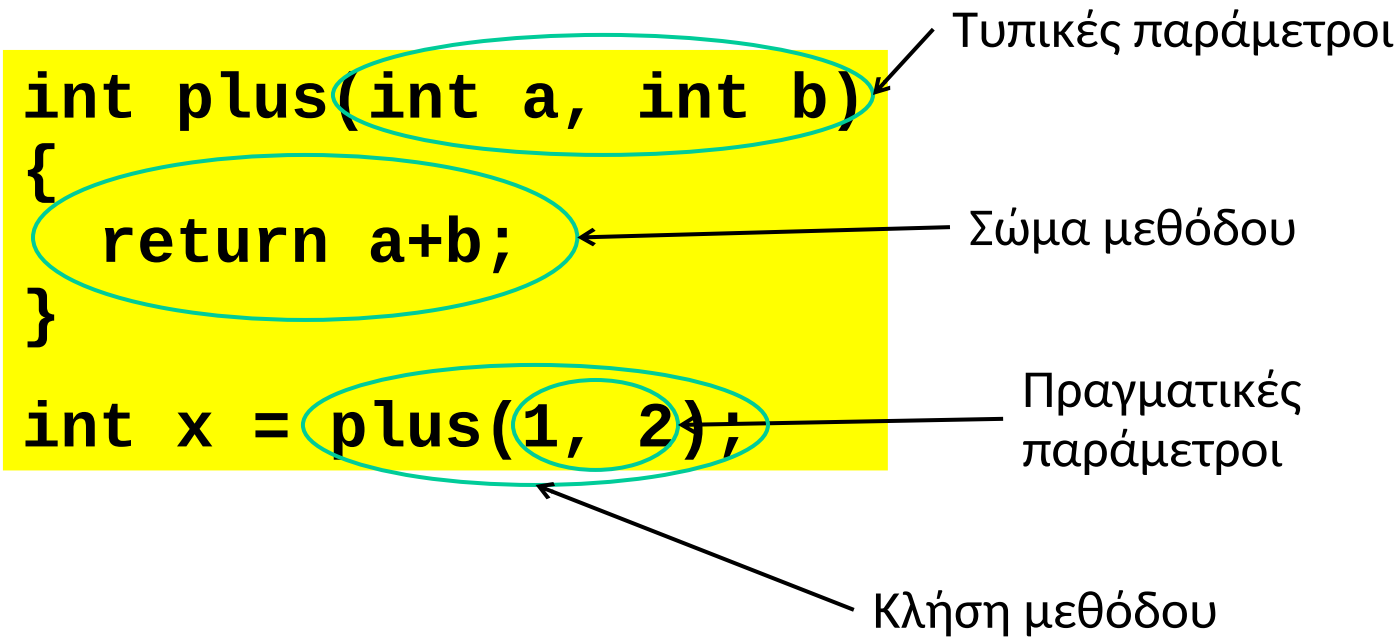
Παράμετροι

Claude Monet, *Poppies*, 1873

Κωστής Σαγώνας <kostis@cs.ntua.gr>
Νίκος Παπασπύρου <nickie@softlab.ntua.gr>

Πέρασμα παραμέτρων



- Πώς περνιούνται οι παράμετροι;
- Φαίνεται αρκετά «προφανής» διαδικασία
- Αλλά θα δούμε επτά διαφορετικές μεταξύ τους τεχνικές

Περιεχόμενα

- Αντιστοιχία παραμέτρων
- Πέρασμα παραμέτρων
  - Κλήση κατά τιμή (call by value)
  - Κλήση κατ’ αποτέλεσμα (call by result)
  - Κλήση κατά τιμή-αποτέλεσμα (call by value-result)
  - Κλήση κατ’ αναφορά (call by reference)
  - Κλήση με επέκταση μακροεντολών (call by macro expansion)
  - Κλήση κατ’ όνομα (call by name)
  - Κλήση κατ’ ανάγκη (call by need)
- Θέματα σχεδιασμού γλωσσών

Αντιστοιχία παραμέτρων

- Μια προκαταρκτική ερώτηση: με ποιο τρόπο ταιριάζει η γλώσσα τις τυπικές με τις πραγματικές παραμέτρους;

- Η πιο συνηθισμένη απάντηση στο παραπάνω ερώτημα είναι:
  - Η αντιστοιχία καθορίζεται από τις θέσεις των παραμέτρων *(positional parameters)*
  - Για παράδειγμα, η ν-οστή τυπική παράμετρος ταιριάζει με τη ν-οστή πραγματική παράμετρο

Ταίριασμα παραμέτρων μέσω ονομάτων

- Η αντιστοιχία μπορεί επίσης να καθοριστεί με χρήση ταιριάσματος ονομάτων παραμέτρων

- Για παράδειγμα στην Ada:

```
DIVIDE(DIVIDEND => i+10, DIVISOR => 3);
```

- Ταιριάζει την πραγματική παράμετρο **`i+10`** με την τυπική παράμετρο **`DIVIDEND`**, και την **`3`** με την **`DIVISOR`**
- Στην περίπτωση αυτή η σειρά εμφάνισης των παραμέτρων δεν παίζει κάποιο ρόλο

Θέσεις και ονόματα στο ταίριασμα παραμέτρων

- Οι περισσότερες γλώσσες που επιτρέπουν ταίριασμα παραμέτρων μέσω ονομάτων (π.χ. Ada, Fortran, Dylan, Python, Objective C, Smalltalk) επιτρέπουν επίσης και παραμέτρους θέσεων
- Οι πρώτες παράμετροι συνήθως καθορίζονται μέσω θέσεων, ενώ οι υπόλοιπες είναι παράμετροι που καθορίζονται μέσω ονομάτων

Προαιρετικές παράμετροι

- Προαιρετικές παράμετροι με χρήση default τιμών: η ακολουθία των τυπικών παραμέτρων περιλαμβάνει default τιμές που χρησιμοποιούνται εάν οι αντίστοιχες πραγματικές παράμετροι λείπουν
  - Μέσω προαιρετικών παραμέτρων μπορούμε να γράψουμε εύκολα κάποιους υπερφορτωμένους ορισμούς συναρτήσεων, όπως π.χ.

```
int f(int a=1, int b=2, int c=3) { body }
```

  - Η παραπάνω έκφραση στη C++ είναι ισοδύναμη με:

```
int f() { f(1,2,3); }
int f(int a) { f(a,2,3); }
int f(int a, int b) { f(a,b,3); }
int f(int a, int b, int c) { body }
```

Μη περιορισμένες ακολουθίες παραμέτρων

- Κάποιες γλώσσες επιτρέπουν ακολουθίες πραγματικών παραμέτρων μη καθορισμένου μήκους: π.χ. η C, η C++, και γλώσσες σεναρίων όπως η JavaScript, η Python και η Perl επιτρέπουν τον ορισμό

```
int printf(char *format, ...) { body }
```

- Οι επιπλέον παράμετροι προσπελάζονται μέσω ειδικών μεθόδων της βιβλιοθήκης
- Οι μη περιορισμένες ακολουθίες παραμέτρων αποτελούν «τρύπα» των στατικών συστημάτων τύπων
  - Διότι οι τύποι των επιπλέον παραμέτρων δεν είναι εύκολο να ελεγχθούν κατά το χρόνο μεταγλώττισης του προγράμματος

Πέρασμα Παραμέτρων

Κλήση κατά τιμή (call by value)

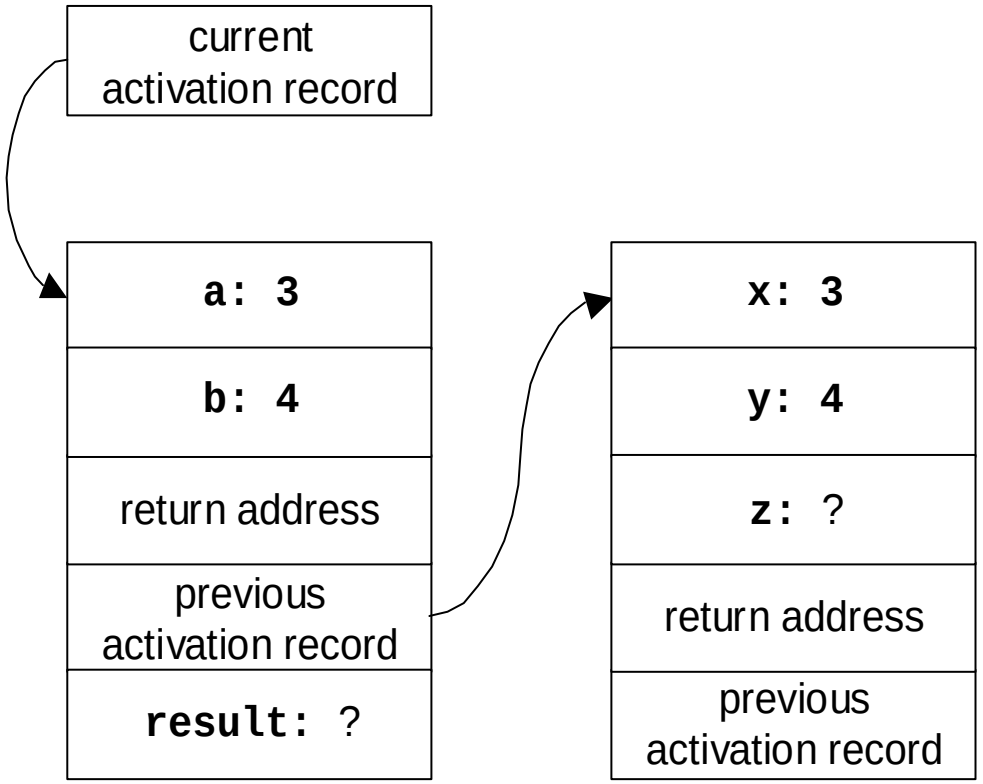
Στο πέρασμα παραμέτρων κατά τιμή, η τυπική παράμετρος συμπεριφέρεται ως μια τοπική μεταβλητή στην εγγραφή δραστηριοποίησης της κληθείσας συνάρτησης με μια σημαντική διαφορά: αρχικοποιείται με την τιμή της αντίστοιχης πραγματικής παραμέτρου, πριν η κληθείσα συνάρτηση ξεκινήσει την εκτέλεσή της.

- Η απλούστερη μέθοδος περάσματος παραμέτρων
- Χρησιμοποιείται ευρέως
- Η μόνη μέθοδος στη Java

```
int plus(int a, int b) {
  a += b;
  return a;
}
int f() {
  int x = 3;
  int y = 4;
  int z = plus(x, y);
  return 2*x*z;
}
```

Η συνάρτηση **f** επιστρέφει **42**

Όταν η **plus** αρχίζει την εκτέλεσή της

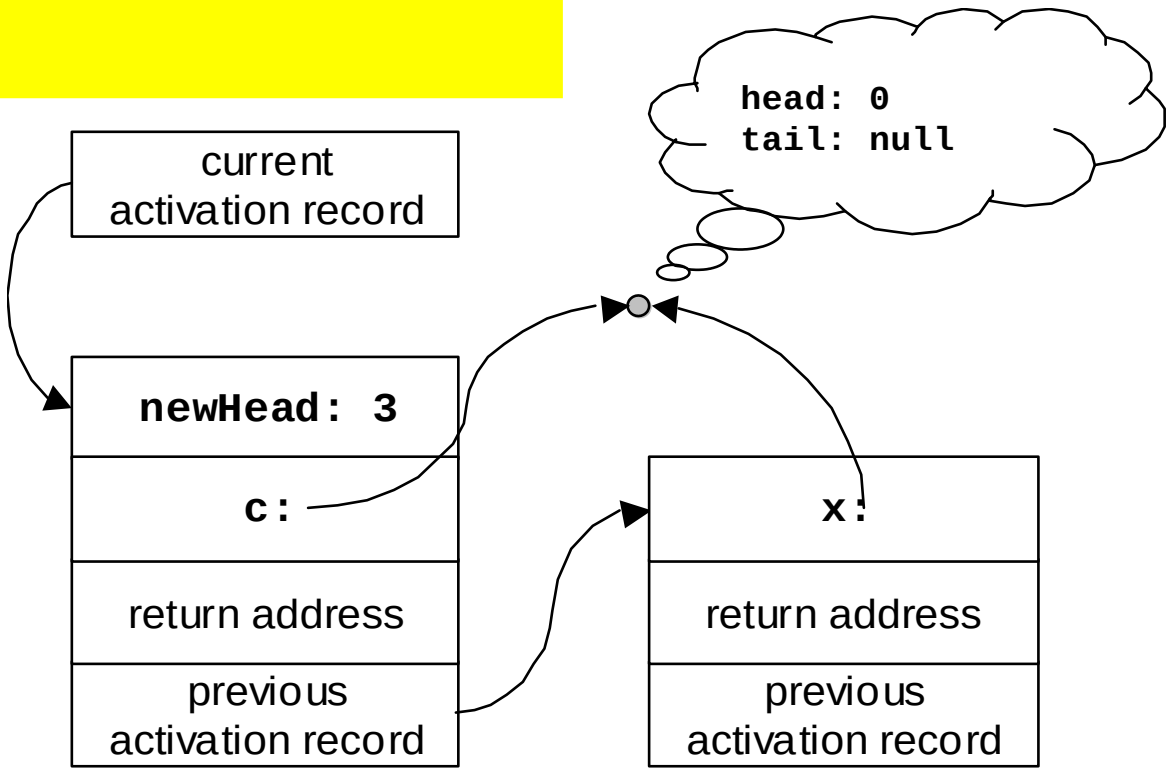


Ορατότητα αλλαγών

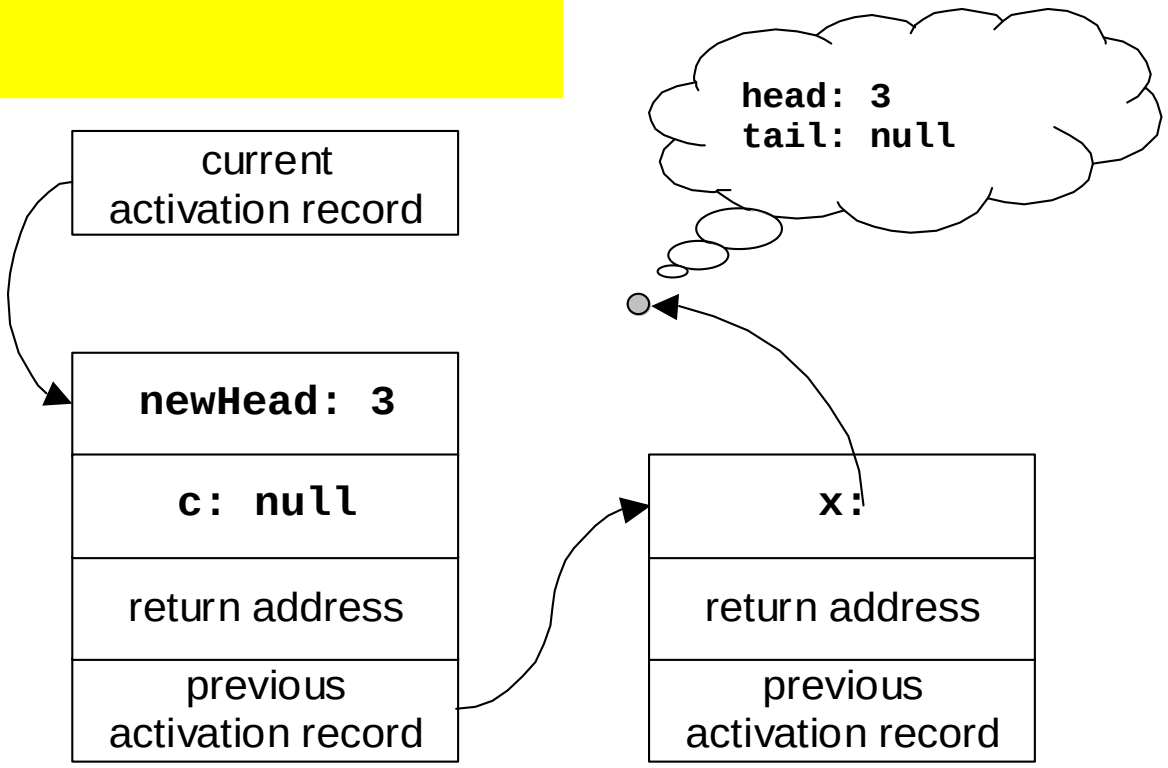
- Όταν οι παράμετροι περνιούνται κατά τιμή, οι αλλαγές σε μια τυπική παράμετρο δεν επηρεάζουν την αντίστοιχη πραγματική
- Παρόλα αυτά είναι δυνατό να γίνουν αλλαγές που να είναι ορατές στην καλούσα μέθοδο
- Για παράδειγμα, η τιμή της παραμέτρου μπορεί να είναι ένας δείκτης (στη Java, μια αναφορά)
- Στην περίπτωση αυτή η πραγματική παράμετρος δεν αλλάζει τιμή, αλλά το αντικείμενο το οποίο αναφέρεται από την παράμετρο μπορεί να τροποποιηθεί

```
void f() {
  ConsCell x = new ConsCell(0, null);
  alter(3, x);
}

void alter(int newHead, ConsCell c) {
  c.setHead(newHead);
  c = null;
}
```

Όταν η **alter** αρχίζει την εκτέλεσή της



```
void f() {
  ConsCell x = new ConsCell(0, null);
  alter(3, x);
}

void alter(int newHead, ConsCell c) {
  c.setHead(newHead);
  c = null;
}
```



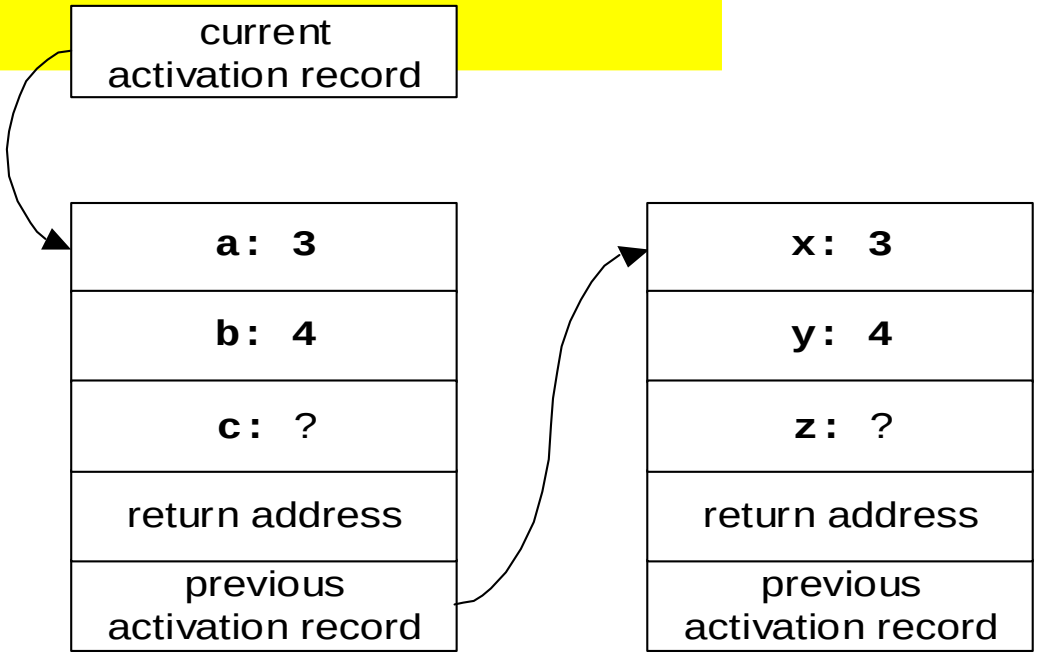
Όταν η **alter** τελειώνει την εκτέλεσή της

Κλήση κατ' αποτέλεσμα (call by result)

Στο πέρασμα παραμέτρων κατ' αποτέλεσμα, η τυπική παράμετρος συμπεριφέρεται ως μια τοπική μεταβλητή στην εγγραφή δραστηριοποίησης της κληθείσας συνάρτησης χωρίς κάποια αρχική τιμή. Όταν η κληθείσα συνάρτηση τελειώσει την εκτέλεσή της, η τελική τιμή της τυπικής παραμέτρου ανατίθεται στην αντίστοιχη πραγματική παράμετρο.

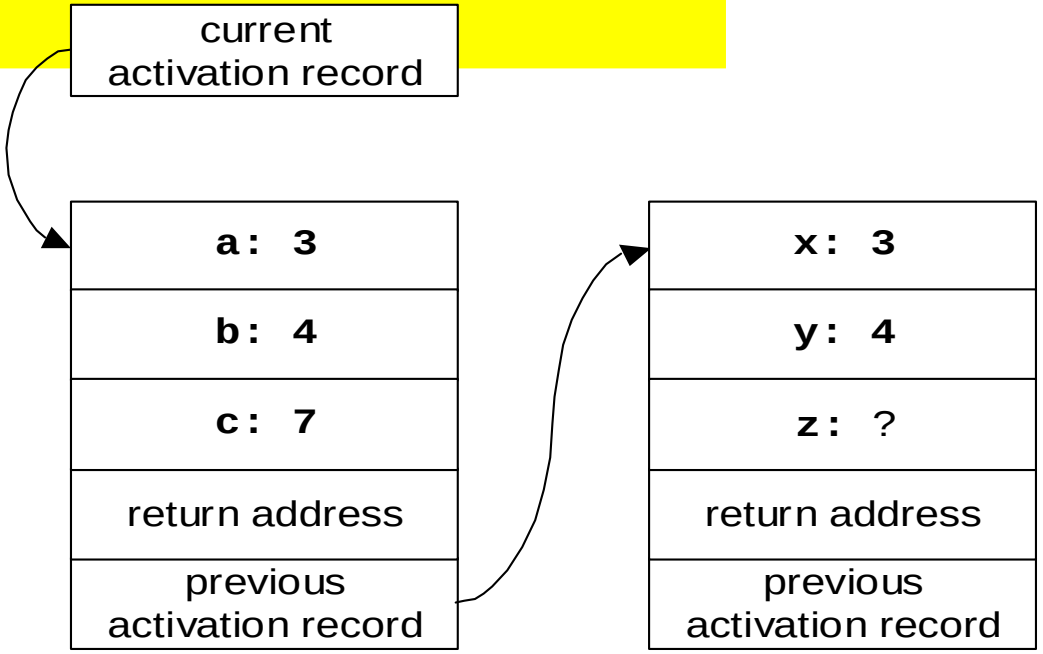
- Ονομάζεται επίσης *copy-out*
- Η πραγματική παράμετρος πρέπει να είναι μια lvalue
- Η κλήση κατ΄ αποτέλεσμα εισήχθη στην Algol 68 και χρησιμοποιήθηκε και στην Ada

```
void plus(int a, int b, by-result int c) {
  c = a+b;
}

int f() {
  int x = 3;
  int y = 4;
  int z;
  plus(x, y, z);
  return 2*x*z;
}
```



Όταν η **plus** αρχίζει την εκτέλεσή της

```
void plus(int a, int b, by-result int c) {
  c = a+b;
}

int f() {
  int x = 3;
  int y = 4;
  int z;
  plus(x, y, z);
  return 2*x*z;
}
```

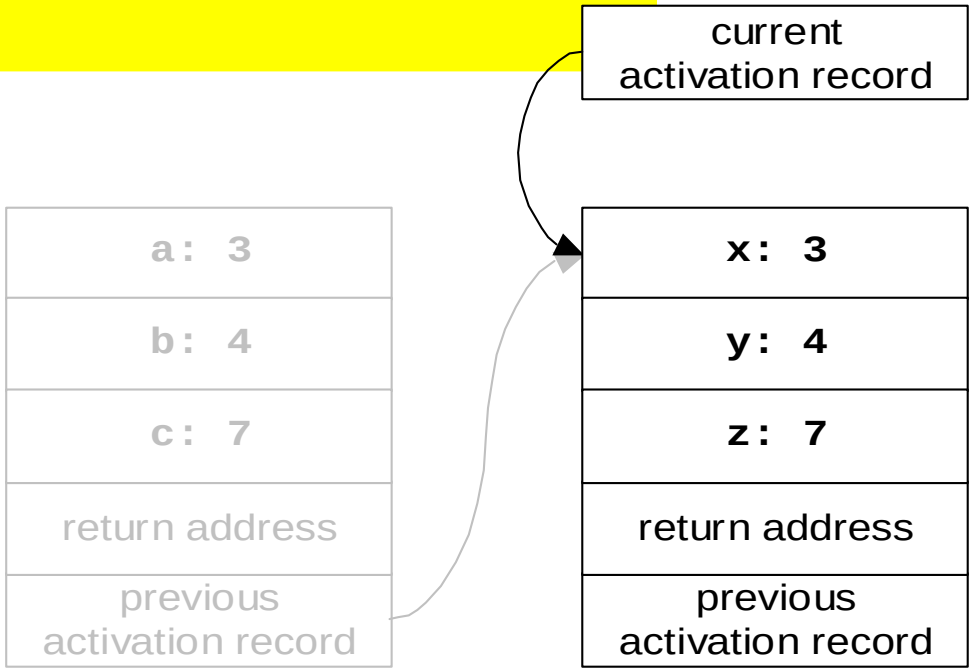


Όταν η **plus** είναι έτοιμη να επιστρέψει

```
void plus(int a, int b, by-result int c) {
  c = a+b;
}

int f() {
  int x = 3;
  int y = 4;
  int z;
  plus(x, y, z);
  return 2*x*z;
}
```

Η συνάρτηση **f** επιστρέφει **42**

Αφότου η **plus** επιστρέψει

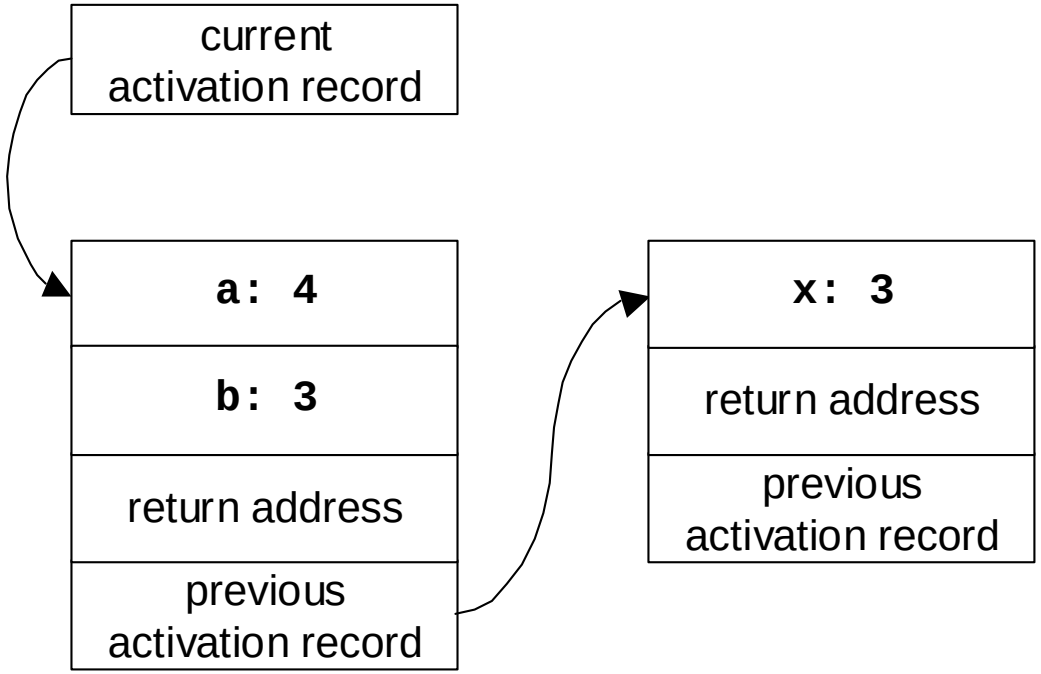


Κλήση κατά τιμή-αποτέλεσμα (call by value-result)

Στο πέρασμα παραμέτρων κατά τιμή-αποτέλεσμα, η τυπική παράμετρος συμπεριφέρεται ως μια τοπική μεταβλητή στην εγγραφή δραστηριοποίησης της κληθείσας συνάρτησης. Αρχικοποιείται με την τιμή της αντίστοιχης πραγματικής παραμέτρου, πριν η κληθείσα συνάρτηση ξεκινήσει την εκτέλεσή της. Μετά το πέρας της εκτέλεσης της κληθείσας συνάρτησης, η τελική τιμή της τυπικής παραμέτρου ανατίθεται στην πραγματική παράμετρο.

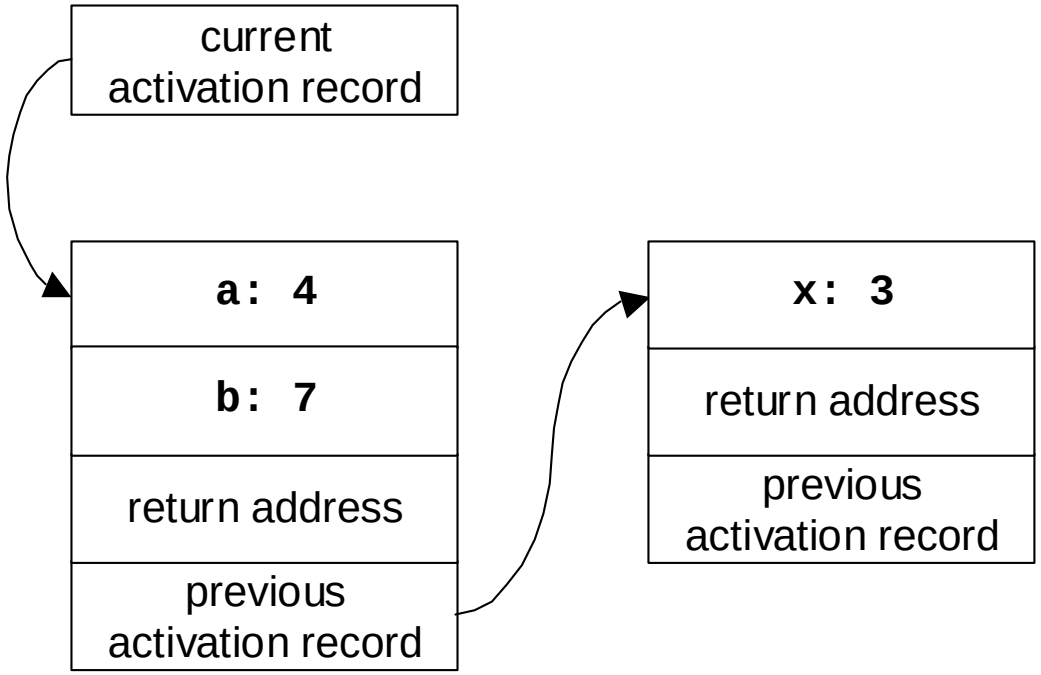
- Ονομάζεται επίσης *copy-in/copy-out*
- Η πραγματική παράμετρος πρέπει να είναι μια lvalue

```
void plus(int a, by-value-result int b) {
  b += a;
}

int f() {
  int x = 3;
  plus(4, x);
  return x*x - x;
}
```



Όταν η **plus** αρχίζει την εκτέλεσή της

```
void plus(int a, by-value-result int b) {
  b += a;
}

int f() {
  int x = 3;
  plus(4, x);
  return x*x - x;
}
```

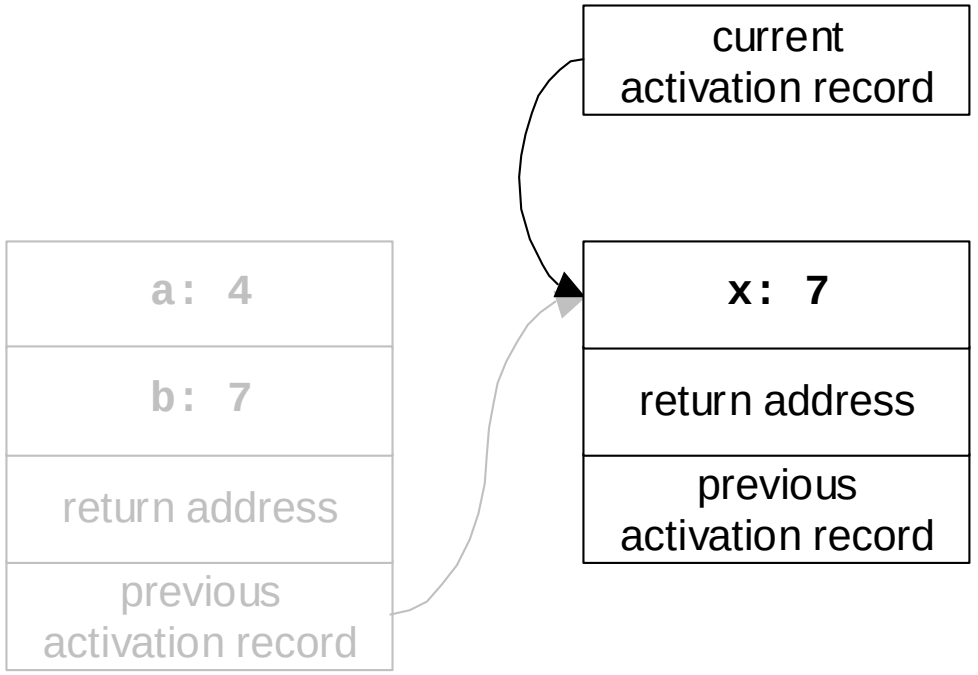


Όταν η **plus** είναι έτοιμη για επιστροφή

```
void plus(int a, by-value-result int b) {
  b += a;
}

int f() {
  int x = 3;
  plus(4, x);
  return x*x - x;
}
```

Η συνάρτηση **f** επιστρέφει **42**

Όταν η **plus** επιστρέψει



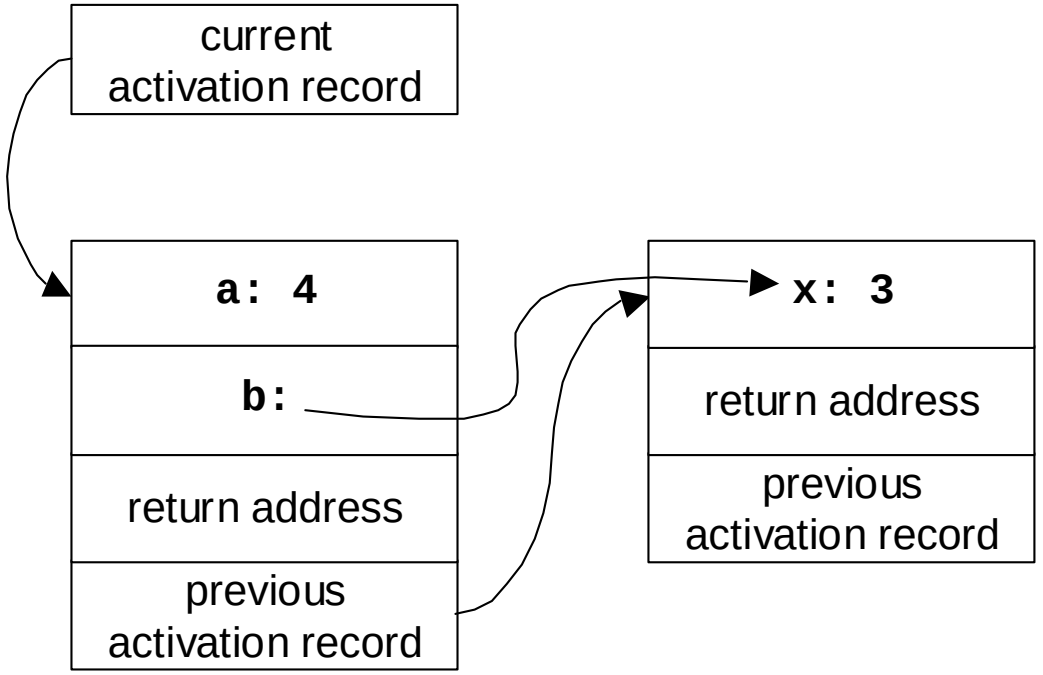
Κλήση κατ' αναφορά (call by reference)

Στο πέρασμα παραμέτρων κατ' αναφορά, η lvalue της παραμέτρου υπολογίζεται πριν αρχίσει η εκτέλεση της κληθείσας συνάρτησης. Μέσα στη συνάρτηση, η lvalue χρησιμοποιείται ως lvalue της αντίστοιχης τυπικής παραμέτρου. Κατά συνέπεια, η τυπική παράμετρος είναι ένα επιπλέον όνομα (δηλαδή ένα συνώνυμο - alias) για τη θέση μνήμης της πραγματικής παραμέτρου.

- Ένας από τους παλαιότερους τρόπους κλήσης
- Χρησιμοποιήθηκε στη Fortran
- Αποδοτικός για μεγάλα αντικείμενα
- Χρησιμοποιείται ευρέως και σήμερα

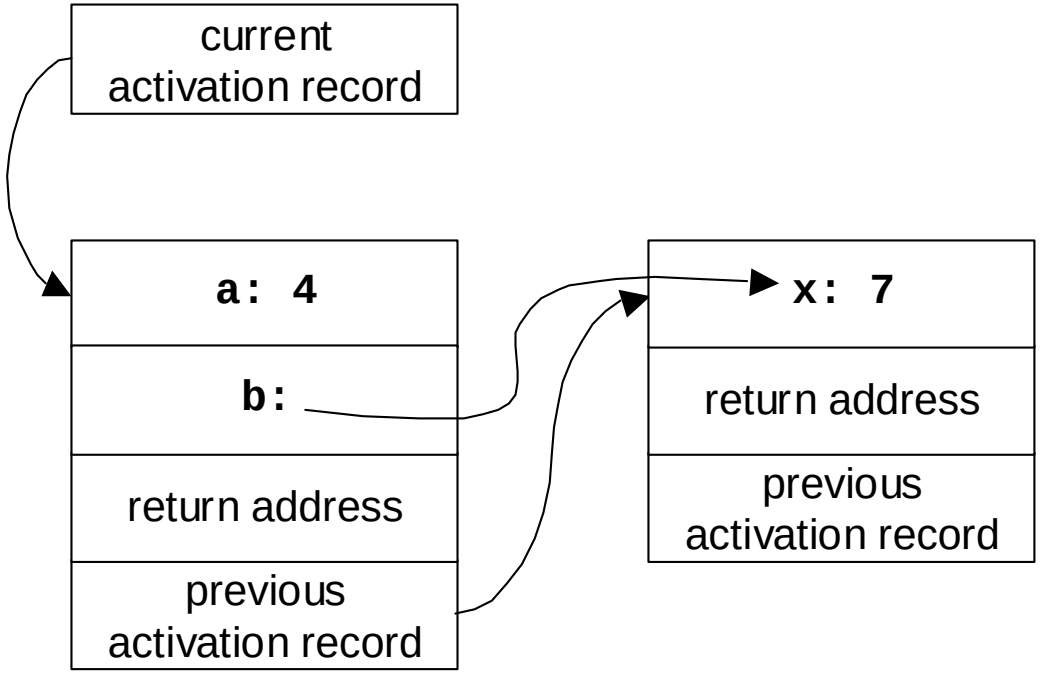
```
void plus(int a, by-reference int b) {
  b += a;
}

void f() {
  int x = 3;
  plus(4, x);
}
```

Όταν η **plus** αρχίζει την εκτέλεσή της



```
void plus(int a, by-reference int b) {
  b += a;
}

void f() {
  int x = 3;
  plus(4, x);
}
```

Όταν η **plus** εκτελέσει την ανάθεση



Υλοποίηση κλήσης κατ' αναφορά

```
void plus(int a, by-reference int b) {
  b += a;
}
void f() {
  int x = 3;
  plus(4, x);
}
```

Το προηγούμενο παράδειγμα

```
void plus(int a, int *b) {
  *b += a;
}
void f() {
  int x = 3;
  plus(4, &x);
}
```

Η υλοποίησή του σε C

κατ' αναφορά = διεύθυνση κατά τιμή

Συνωνυμία (aliasing)

- Όταν δύο εκφράσεις έχουν την ίδια lvalue, τότε λέμε ότι είναι συνώνυμα (aliases) η μία της άλλης
- Κάποιες προφανείς περιπτώσεις aliasing

```
ConsCell x = new ConsCell(0,null);
ConsCell y = x;
```

```
A[i] = A[j] + A[k];
```

- Όπως θα δούμε στην επόμενη διαφάνεια, η κλήση κατ΄ αναφορά οδηγεί σε πιο περίπλοκες περιπτώσεις aliasing...

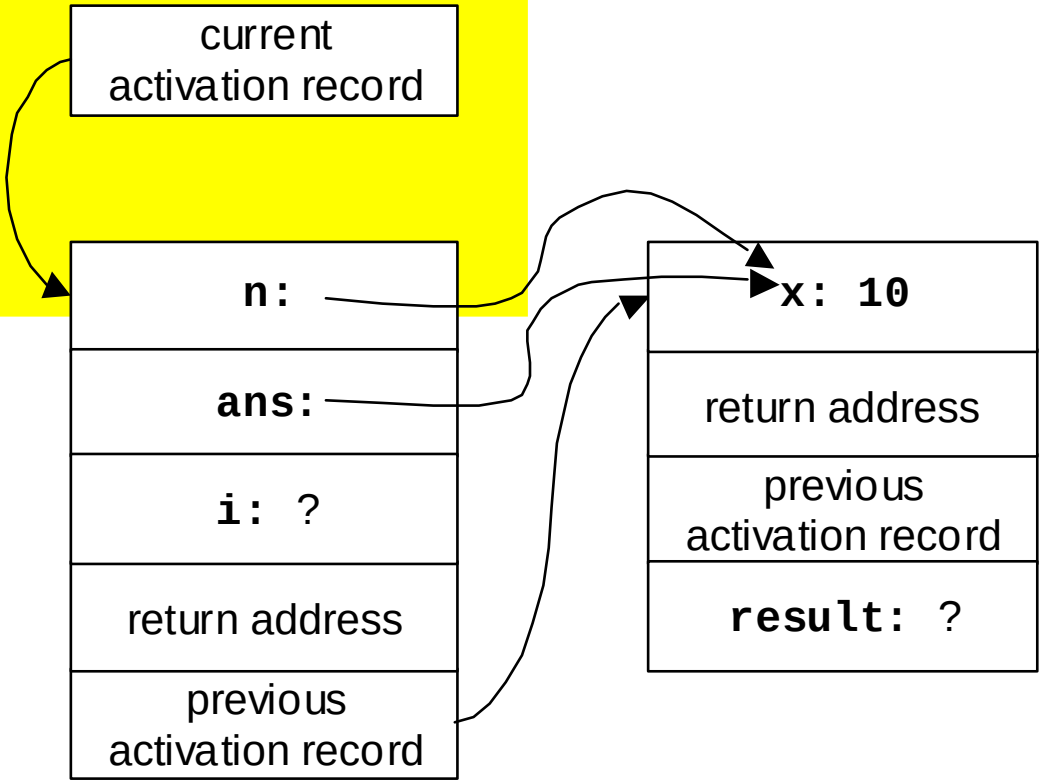
Παράδειγμα

```
void sigsum(by-reference int n,
            by-reference int ans) {
  ans = 0;
  int i = 1;
  while (i <= n) ans += i++;
}

int f() {
  int x, y;
  x = 10;
  sigsum(x,y);
  return y;
}
```

```
int g() {
  int x;
  x = 10;
  sigsum(x,x);
  return x;
}
```

```
void sigsum(by-reference int n,
            by-reference int ans) {
  ans = 0;
  int i = 1;
  while (i <= n) ans += i++;
}

int g() {
  int x = 10;
  sigsum(x,x);
  return x;
}
```

Η συνάρτηση **g** επιστρέφει **0**

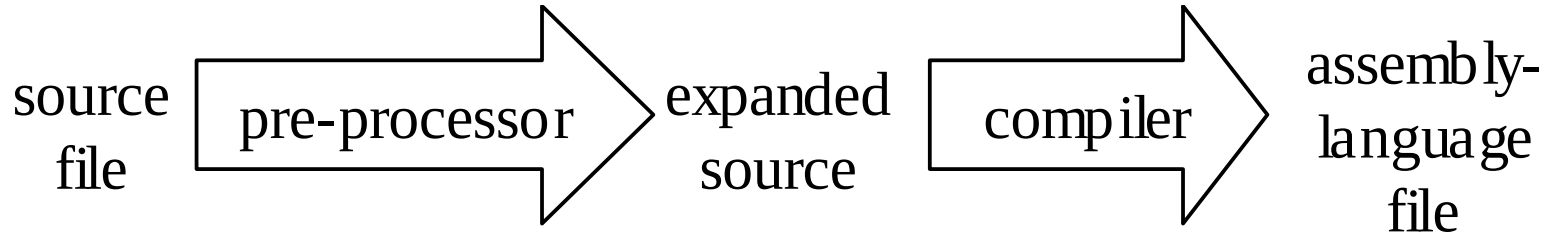


Επέκταση μακροεντολών (macro expansion)

Στην επέκταση μακροεντολών, το σώμα της μακροεντολής αποτιμάται στο περιβάλλον εκτέλεσης της καλούσας μεθόδου. Κάθε μία από τις πραγματικές παραμέτρους αποτιμάται για κάθε χρήση της αντίστοιχης τυπικής στο περιβάλλον εμφάνισης της τυπικής παραμέτρου (το οποίο με τη σειρά του βρίσκεται στο περιβάλλον της καλούσας μεθόδου).

- Με άλλα λόγια, όπως δουλεύουν οι μακροεντολές της C
- Φυσική υλοποίηση: αντικατάσταση κειμένου πριν τη μεταγλώττιση

Επέκταση μακροεντολών στη C



- Η προεπεξεργασία είναι μία επιπλέον φάση της μεταγλώττισης
- Επεκτείνει τις μακροεντολές πριν τη μετάφραση
- Παράδειγμα

```
#define MIN(X,Y) ((X)<(Y)?(X):(Y))
a = MIN(b,c);
```

```
a = ((b)<(c)?(b):(c))
```

Προεπεξεργασία (preprocessing)

- Αντικαθιστούμε κάθε χρήση της μακροεντολής με ένα αντίγραφο του σώματος της μακροεντολής, όπου οι τυπικές παράμετροι έχουν αντικατασταθεί από τις πραγματικές
- Αποτελεί μια παλιά μέθοδο: χρησιμοποιήθηκε στους assemblers πριν από την ύπαρξη των πρώτων «ανώτερων» γλωσσών προγραμματισμού
- Έχει κάποιες περίεργες παρενέργειες
  - Επανεκτίμηση
  - Πιάσιμο ονομάτων (name capture)

Επανεκτίμηση (repeated evaluation)

- Κάθε πραγματική παράμετρος επανεκτιμάται κάθε φορά που χρησιμοποιείται

πηγαίος κώδικας:

```
#define MIN(X,Y) ((X)<(Y)?(X):(Y))
a = MIN(b++,c++);
```

κώδικας μετά την επέκταση:

```
a = ((b++)<(c++)?(b++):(c++))
```

Παράδειγμα πιασίματος ονομάτων

πηγαίος κώδικας:

```
#define intswap(X,Y) {int temp=X; X=Y; Y=temp;}
int main() {
  int temp = 1, b = 2;
  intswap(temp, b);
  printf("%d, %d\n", temp, b);
}
```

κώδικας μετά την επέκταση:

```
int main() {
  int temp = 1, b = 2;
  {int temp = temp; temp = b; b = temp;};
  printf("%d, %d\n", temp, b);
}
```

Πιάσιμο ονομάτων (name capture)

- Σ’ ένα κομμάτι κώδικα, κάθε εμφάνιση μιας μεταβλητής η οποία δεν είναι δεσμευμένη στατικά ονομάζεται *ελεύθερη (free)*
- Όταν κάποιο κομμάτι κώδικα μετακινηθεί σε ένα διαφορετικό περιβάλλον, οι ελεύθερες μεταβλητές του μπορεί να δεσμευθούν (become bound)
- Το φαινόμενο αυτό ονομάζεται *capture*:
  - Οι ελεύθερες μεταβλητές των πραγματικών παραμέτρων μπορεί να «πιαστούν» (be captured) από ορισμούς μεταβλητών στο σώμα της μακροεντολής
  - Επίσης, μεταβλητές που είναι ελεύθερες στο σώμα της μακροεντολής μπορεί να «πιαστούν» από ορισμούς μεταβλητών της καλούσας μεθόδου

Κλήση κατ' όνομα (call by name)

Στο πέρασμα παραμέτρων κατ' όνομα, κάθε παράμετρος αποτιμάται στο περιβάλλον της καλούσας συνάρτησης, για κάθε χρήση της αντίστοιχης τυπικής παραμέτρου.

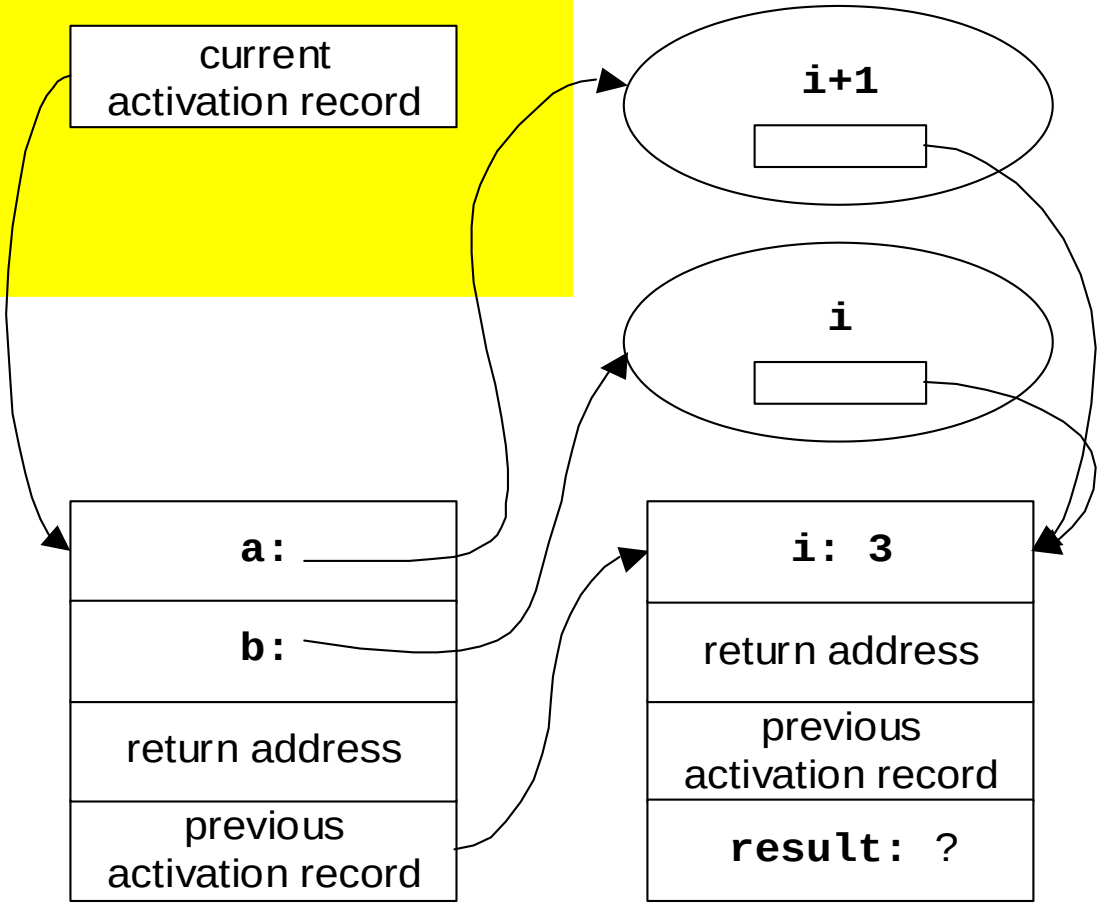
- Δουλεύει όπως η επέκταση μακροεντολών αλλά αποφεύγει το πρόβλημα του πιασίματος ονομάτων
- Χρησιμοποιήθηκε στην Algol 60 και στις επεκτάσεις της
- Σήμερα δεν είναι δημοφιλής μέθοδος περάσματος

Υλοποίηση της κλήσης κατ' όνομα

- Χειριζόμαστε την πραγματική παράμετρο ως μια μικρή ανώνυμη συνάρτηση
- Όταν η καλούσα μέθοδος χρειαστεί την τιμή της τυπικής παραμέτρου (είτε την rvalue της ή την lvalue της) καλεί τη συνάρτηση για να της την επιστρέψει
- Η συνάρτηση πρέπει να περαστεί μαζί με το σύνδεσμο φωλιάσματός της (its nesting link), έτσι ώστε να μπορεί να αποτιμηθεί στο περιβάλλον της καλούσας μεθόδου

```
void f(by-name int a, by-name int b) {
  b = 6;
  b = a;
}

int g() {
  int i = 3;
  f(i+1, i);
  return i;
}
```

Η συνάρτηση **g** επιστρέφει **7**



Όταν η **f** αρχίζει την εκτέλεσή της

Σύγκριση

- Όπως και στην επέκταση μακροεντολών, οι παράμετροι που περνιούνται κατ' όνομα επανεκτιμώνται κάθε φορά που χρησιμοποιούνται
- Η παραπάνω ιδιότητα μπορεί να φανεί αρκετά χρήσιμη σε κάποιες περιπτώσεις, όμως στις περισσότερες περιπτώσεις αποτελεί χάσιμο χρόνου
- Όμως, αντίθετα με την επέκταση μακροεντολών, η κλήση κατ΄ όνομα αποφεύγει το πρόβλημα του πιασίματος ονομάτων

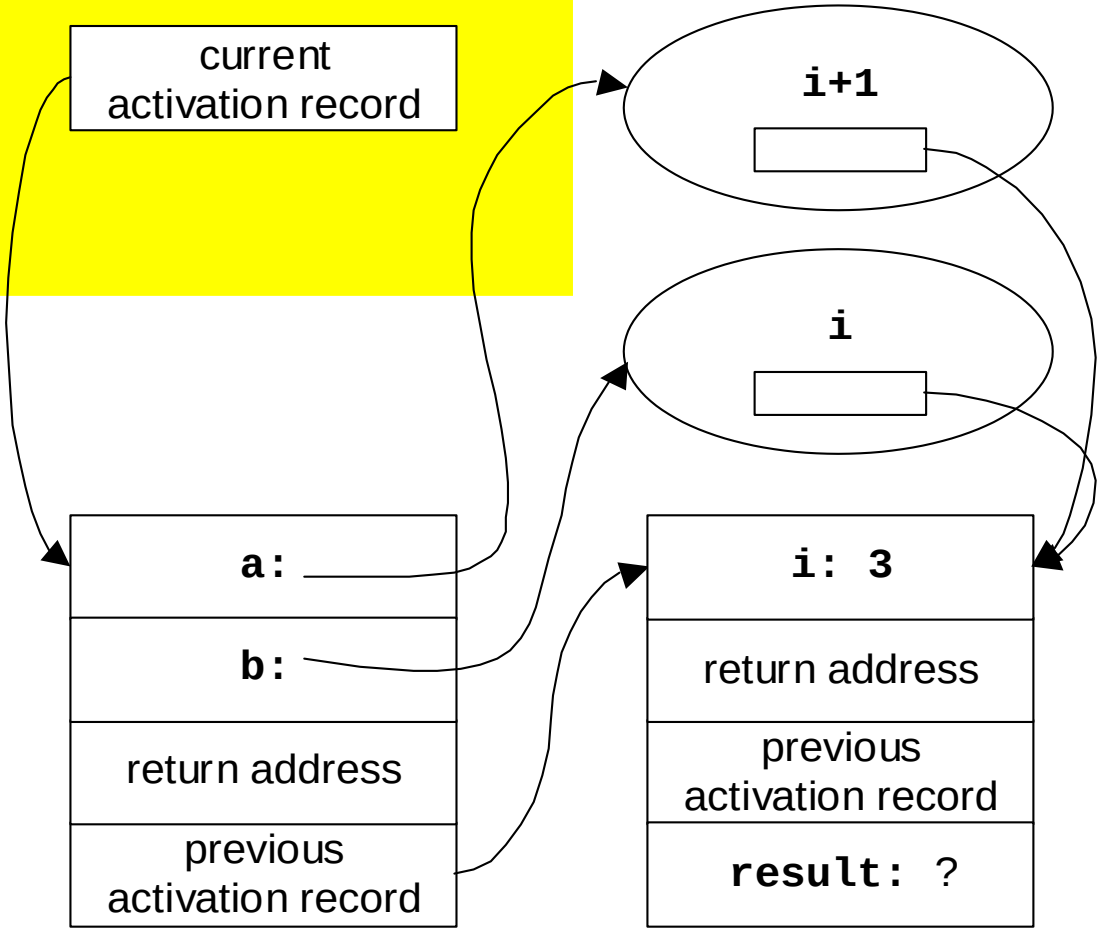
Κλήση κατ' ανάγκη (call by need)

Στο πέρασμα παραμέτρων κατ' ανάγκη, κάθε παράμετρος αποτιμάται στο περιβάλλον της καλούσας μεθόδου, τη στιγμή της πρώτης χρήσης της αντίστοιχης τυπικής παραμέτρου. Μετά την αποτίμηση, η τιμή της πραγματικής παραμέτρου αποθηκεύεται (cached), ούτως ώστε περαιτέρω χρήσεις της αντίστοιχης τυπικής παραμέτρου να μη χρειάζονται επανεκτίμηση.

- Χρησιμοποιείται στις οκνηρές (lazy) συναρτησιακές γλώσσες προγραμματισμού (Miranda, Haskell, Clean)
- Αποφεύγει το χάσιμο χρόνου λόγω της επανεκτίμησης την οποία πολλές φορές κάνει η κλήση κατ΄ όνομα

```
int f(by-need int a, by-need int b) {
  b = a;
  b = a;
  return b;
}
int g() {
  int i = 3;
  return f(i+1, i);
}
```

Η συνάρτηση **g** επιστρέφει **4**



Όταν η **f** αρχίζει την εκτέλεσή της

Οκνηρία (laziness)

```
boolean andand(by-need boolean a,
               by-need boolean b) {
  if (!a) return false;
  else return b;
}

boolean g() {
  while (true) {
  }
  return true;
}

void f() {
  andand(false, g());
}
```

Η συνάρτηση **andand** βραχυκυκλώνει όπως η **andalso** της ML και ο τελεστής **&&** της Java.

Με κλήση κατ΄ ανάγκη η μέθοδος **f** θα τερματίσει.

Το αποτέλεσμα θα ήταν το ίδιο τόσο με κλήση κατ΄ όνομα όσο και με επέκταση μακροεντολών.

Θέματα Σχεδιασμού Γλωσσών

Θέματα σχεδιασμού γλωσσών

Ερώτηση: Αυτά που εξετάσαμε σχετικά με το πέρασμα παραμέτρων είναι απλώς θέματα υλοποίησης μιας γλώσσας ή είναι και μέρος των προδιαγραφών της;

Απάντηση: Εξαρτάται από τη γλώσσα

- Σε γλώσσες χωρίς παρενέργειες, η τεχνική που χρησιμοποιείται για το πέρασμα παραμέτρων μπορεί να είναι διαφανής και μη ανιχνεύσιμη από τον προγραμματιστή
- Όμως ακόμα και γλώσσες με παρενέργειες πολλές φορές καθορίζουν μόνο μερικώς την τεχνική περάσματος παραμέτρων που χρησιμοποιείται από τη γλώσσα

Γλώσσες χωρίς παρενέργειες

Ερώτηση: πρέπει οι παράμετροι πάντα να αποτιμώνται (*eager evaluation*), ή αυτό συμβαίνει μόνο όταν υπάρχει ανάγκη χρήσης των τιμών τους (*lazy evaluation*);

- Το παραπάνω είναι μέρος του μοντέλου κόστους της γλώσσας και μπορεί επίσης να χρησιμοποιηθεί από τον προγραμματιστή:
  - Πόσο στοιχίζει η επανεκτίμηση κάποιας τυπικής παραμέτρου;
  - Έχει το πέρασμα παραμέτρων κόστος ανάλογο με το μέγεθος των αντικειμένων (που περιλαμβάνονται στις παραμέτρους);

Γλώσσες με παρενέργειες

- Στις γλώσσες αυτής της κατηγορίας, ένα πρόγραμμα μπορεί να ανιχνεύσει ποια τεχνική περάσματος παραμέτρων χρησιμοποιείται από την υλοποίηση της γλώσσας
- Όμως αυτό μπορεί να είναι μια λεπτομέρεια υλοποίησης στην οποία τα προγράμματα δεν πρέπει να μπορούν να βασιστούν — διότι δεν είναι μέρος της προδιαγραφής της γλώσσας
- Παράδειγμα τέτοιας γλώσσας είναι η Ada

Ada modes

- Τρεις τρόποι (modes) περάσματος παραμέτρων:
  - **`in`**: μπορούν να διαβαστούν από την κληθείσα συνάρτηση αλλά δε μπορεί να τους γίνει κάποια ανάθεση — με άλλα λόγια συμπεριφέρονται σαν σταθερές
  - **`out`**: μπορεί να τους γίνει κάποια ανάθεση αλλά δε μπορούν να διαβαστούν
  - **`in out`**: μπορούν τόσο να διαβαστούν όσο και να τους γίνει κάποια ανάθεση
- Η προδιαγραφή της Ada επίτηδες αφήνει κάποιο περιθώριο για ευελιξία στις υλοποιήσεις της γλώσσας

Υλοποιήσεις της Ada

- Για βαθμωτές παραμέτρους η υλοποίηση γίνεται μέσω αντιγραφής:
  - **`in`** = value, **`out`** = result, **`in out`** = value/result
- Οι συλλογές αντικειμένων (π.χ. arrays και records) *μπορεί* να περαστούν κατ΄ αναφορά αντί με αντιγραφή
- Κάθε πρόγραμμα που μπορεί να ανιχνεύσει τη διαφορά της υλοποίησης (όπως προγράμματα αντίστοιχα με αυτά της σημερινής διάλεξης) είναι μη νόμιμο πρόγραμμα Ada

Συμπερασματικά

- Σήμερα είδαμε:
  - Πώς ταιριάζουμε τυπικές και πραγματικές παραμέτρους
  - Επτά διαφορετικές τεχνικές για πέρασμα παραμέτρων
  - Ιδέες για το πού βρίσκεται η διαχωριστική γραμμή μεταξύ ορισμού γλώσσας και λεπτομέρειας υλοποίησης
- Αυτές δεν είναι οι μόνες τεχνικές που έχουν δοκιμαστεί, αλλά απλά κάποιες από τις πιο συνηθισμένες