Κεφάλαιο 6: Εμβέλεια και Δέσμευση Μνήμης

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γ. Γαροφαλάκης, Σ. Σιούτας, Π. Χατζηδούκας, Γ. Βασιλόπουλος

Ορισμοί (1)

Εμβέλεια ενός *ονόματος* (Scope), είναι το τμήμα του προγράμματος στο οποίο όλες οι *χρήσεις* του ονόματος είναι ίδιες.

Δηλαδή:
Τρόπος που ρυθμίζεται η δυνατότητα *οντοτήτων* με κάποιο όνομα (μεταβλητές, τύποι δεδομένων, συναρτήσεις, ετικέτες, ..), να επιδρούν σε ένα πρόγραμμα.

Χρησιμότητα:

- Δεν χρειάζεται ο προγραμματιστής να θυμάται όλα τα ονόματα που έχουν χρησιμοποιηθεί στο πρόγραμμα.
- Modular κατασκευή και χρήση.

Ορισμοί (2)

Χρόνος Δέσμευσης ενός *ονόματος* (Extent), είναι ο χρόνος κατά την εκτέλεση του προγράμματος, στον οποίο η μνήμη που περιέχει μια τιμή, συνδέεται με το όνομα της τιμής αυτής (binding).

ALGOL 60: Εισήγαγε την ιδέα της εμβέλειας σε συνάρτηση με την έννοια της σύνθετης εντολής (block): BEGIN … END

- Πολλές εντολές θεωρούνται ως μία.
- Τοπικές δηλώσεις: Μεταβλητές, συναρτήσεις, ετικέτες με εμβέλεια εντός της σύνθετης εντολής.

C, C++: Χρήση blocks:

```
if (m[i] < m[j])
{
    int temp;
    temp = m[i];
      …..
}
```

Ονόματα

Οντότητες ενός προγράμματος που μπορούν να έχουν όνομα:

1. **Μεταβλητές**
2. **Τυπικές παράμετροι**
3. **Υποπρογράμματα (συναρτήσεις, διαδικασίες)**
4. Τύποι Δεδομένων χρήστη
5. Σταθερές χρήστη
6. Ετικέτες (labels)

*Πιο σημαντικά τα **1, 2, 3***: Γίνονται αλλαγές στις συνδέσεις τους κατά την εκτέλεση, σε αντίθεση με τα 4, 5, 6 που οι συνδέσεις τους γίνονται κατά τη μετάφραση και δεν αλλάζουν.

Ακολουθία Εκτέλεσης Προγράμματος

1. Αρχή εκτέλεσης main: Υπάρχουν *συνδέσεις* (bindings – Κεφ. 4) των ονομάτων που ορίζονται στο main, με οντότητες (μεταβλητές, υποπρογρ/τα).
2. Με την εκτέλεση, χρησιμοποιούνται *Referencing Operations* (Ref-Ops, *αναφορές*) για πρόσβαση στις τιμές των οντοτήτων. Π.χ. στην εντολή A=B+F(C), χρειάζονται 4 Ref-Ops.
3. Όταν το main καλέσει ένα υποπρόγραμμα, δημιουργείται ένα νέο σύνολο συνδέσεων (τοπικών) για το υπ/μα, που μπορεί να κάνει ανενεργές κάποιες από τις συνδέσεις του main.
4. Με την εκτέλεση του υποπρογράμματος, χρησιμοποιούνται Ref-Ops σε συνδέσεις του υπ/τος (τοπικές) και του main (global).
5. Αν το υπ/μα καλέσει άλλο υπ/μα, δημιουργείται νέο σύνολο συνδέσεων που μπορεί να κάνει ανενεργές κάποιες από τις συνδέσεις κ.ο.κ.
6. Όταν ένα υπ/μα τελειώσει την εκτέλεσή του, οι συνδέσεις του καταστρέφονται ή γίνονται ανενεργές.
7. Όταν ο έλεγχος επιστρέφει στο main, χρησιμοποιούνται οι αρχικές συνδέσεις .

Χρειάζεται κατάλληλη διαχείριση της μνήμης…

Βασικές Έννοιες (1)

Περιβάλλον Αναφοράς (referencing environment)

Το σύνολο των συνδέσεων ονομάτων με οντότητες που είναι διαθέσιμες για Ref-Ops κατά την εκτέλεση προγράμματος ή υποπρογράμματος (**ΠΑ**).

Τοπικό Περιβάλλον Αναφοράς (ΤΠΑ) ενός υπ/τος

Το σύνολο των συνδέσεων ονομάτων με οντότητες που δημιουργούνται με την είσοδο σε ένα υποπρόγραμμα. Αντιπροσωπεύουν:

- Τοπικές μεταβλητές
- Τυπικές παραμέτρους
- Τοπικά υποπρογράμματα

Μία Ref-Op σε σύνδεση του ΤΠΑ, καθορίζεται χωρίς να καταφεύγουμε έξω από το υποπρόγραμμα.

Βασικές Έννοιες (2)

Μη-Τοπικό Περιβάλλον Αναφοράς (Μη-ΤΠΑ) υπ/τος

Το σύνολο των συνδέσεων ονομάτων με οντότητες που μπορούν να χρησιμοποιηθούν από το υπ/μα, αλλά δεν δημιουργούνται με την είσοδο σε αυτό.

Καθολικό Περιβάλλον Αναφοράς (ΚΠΑ) ενός υπ/τος

Το σύνολο των συνδέσεων ονομάτων με οντότητες που δημιουργούνται με την έναρξη του main και είναι διαθέσιμες στο υπ/μα. Είναι μέρος του Μη-ΤΠΑ.

Προκαθορισμένο Περιβάλλον Αναφοράς (ΠΠΑ)

Το σύνολο των προκαθορισμένων συνδέσεων από τη ΓΠ που είναι διαθέσιμες σε όλο το πρόγραμμα.
Π.χ. MAXINT, read, write, …

Άλλες Έννοιες

Ορατότητα (visibility)

Μία σύνδεση ονόματος είναι ορατή, αν είναι μέρος του ΠΑ του εκτελούμενου υπ/τος.

Δυναμική Εμβέλεια (dynamic scope)

μιας σύνδεσης, είναι το τμήμα εκτέλεσης του προγράμματος κατά το οποίο η σύνδεση υπάρχει ως μέρος κάποιου ΠΑ. Δηλαδή, αποτελείται από το σύνολο των *ενεργοποιήσεων υπ/των* στα οποία είναι ορατή η σύνδεση.

Ref-Op: id <Ref-Op> ΠΑ ⟹ <data object> ή <υπ/μα>

Κανόνες Εμβέλειας σε Block-Structured Προγράμματα (1)

- Οι δηλώσεις στην αρχή του block ορίζουν το ΤΠΑ του block. Κάθε αναφορά σε όνομα μέσα στο block (εκτός από τα εσωτερικά block), θεωρείται αναφορά στην τοπική δήλωση του ονόματος (αν υπάρχει).
- Αν γίνεται αναφορά σε όνομα στο block και δεν υπάρχει τοπική δήλωση (δηλαδή η σύνδεση δεν είναι μέρος του ΤΠΑ), τότε η δήλωση αναζητείται στο αμέσως πιο εξωτερικό block (δηλαδή στο Μη-ΤΠΑ) κ.ο.κ. έως το ΚΠΑ. Αν δεν βρεθεί εκεί, εξετάζεται το ΠΠΑ. Αν δεν βρεθεί και εκεί: ERROR.
- Αν ένα block περιλαμβάνει άλλο block, κάθε δήλωση στο εσωτερικό block (και τα εσωτερικά αυτού), είναι εντελώς κρυμμένη από το εξωτερικό block και δεν μπορεί να προσπελασθεί από αυτό.

Κανόνες Εμβέλειας σε Block-Structured Προγράμματα (2)

- Αν ένα block έχει όνομα (στη C μπορεί και να μην έχει), το *όνομα* αυτό είναι μέρος του ΤΠΑ του block στο οποίο περιλαμβάνεται.

  Π.χ. αν στο main ενός προγράμματος Pascal περιλαμβάνεται ο ορισμός:

  procedure **P** (**A**: real);
  ……

  τότε το όνομα **P** είναι μέρος του ΤΠΑ του main, ενώ το **A** είναι μέρος του ΤΠΑ του **P**.

- Αν το block επιστρέφει τιμή (δηλαδή είναι function, όχι void), με την έναρξη εκτέλεσης του block, δημιουργείται ***και*** *τοπική μεταβλητή* με το όνομά του (δηλαδή, μέρος του ΤΠΑ του block).

Παράδειγμα

```
Program MAIN;
var A, B, C: real;
procedure SUB1(A: real);
    var D: real;
    procedure SUB2(C: real);
        var D: real;
        begin
            C:=C+B;
        end;
    begin
        SUB2(B);
    end;
begin
    SUB1(A);
end.
```

Προσοχή: Εξ' ορισμού ισχύει πως το Τοπικό ΠΑ του MAIN είναι και Μη-τοπικό, και Καθολικό.

ΠΑ της SUB2:
ΤΠΑ: C, D
Μη-ΤΠΑ: A, SUB2, B, SUB1
ΚΠΑ: B, SUB1

ΠΑ της SUB1:
ΤΠΑ: A, D, SUB2
Μη-ΤΠΑ και ΚΠΑ: B, C, SUB1

ΠΑ του MAIN:
ΤΠΑ: A, B, C, SUB1
Μη-ΤΠΑ και ΚΠΑ: A, B, C, SUB1

Παράδειγμα (ίδιο, σε C)

```
int main() {
  float A, B, C;
  void SUB1(float A) {
    float D;
    void SUB2(float C) {
      float D;
      C=C+B;
    }
    SUB2(B);
  };

  SUB1(A);
  return 0;
}
```

ΠΑ της SUB2:
ΤΠΑ: C, D
Μη-ΤΠΑ: A, SUB2, B, SUB1
ΚΠΑ: B, SUB1

ΠΑ της SUB1:
ΤΠΑ: A, D, SUB2
Μη-ΤΠΑ και ΚΠΑ: B, C, SUB1

ΠΑ του MAIN:
ΤΠΑ: A, B, C, SUB1
Μη-ΤΠΑ και ΚΠΑ: A, B, C, SUB1

Καθορισμός της εμβέλειας μεταβλητών κ.α.

- FORTRAN: Global Variables με χρήση COMMON.
- PL/1: Δεν απαιτεί ρητές δηλώσεις όλων των μεταβλητών. Αν μια μεταβλητή δεν έχει δηλωθεί, θεωρείται ότι έχει τις ιδιότητες της διαδικασίας στην οποία χρησιμοποιείται.
- ALGOL68: Δηλώσεις και εμβέλεια μέσα σε: begin-end, if-fi, then-else, else-fi.
- Pascal: Τα begin-end χρησιμοποιούνται μόνο για την οριοθέτηση σύνθετων εντολών. Δηλώσεις και εμβέλεια μόνο στα: Main, Procedures, Functions.
- C, C++: Δηλώσεις και εμβέλεια μέσα σε { }.

Θετικά Εμβέλειας με χρήση blocks

- Ενθαρρύνεται η δήλωση μεταβλητών «κοντά» στο τμήμα προγράμματος όπου θα χρησιμοποιηθούν (**τοπικότητα** – locality).
- Λόγω τοπικότητας, το Λ.Σ. θα κάνει **λιγότερα** Ref-Ops και page faults.
- Διευκολύνει τη χρήση **βιβλιοθηκών**, με σύνδεσή τους σε εξωτερικό block, ώστε να μην συγχέονται τα ονόματα μεταβλητών κ.λ.π.
- Ενθαρρύνεται η **modular** κατασκευή του προγράμματος, στοιχείο απαραίτητο για αποδοτικό updating.

Εύρεση των δηλώσεων ονομάτων (1)

```
Program main;
   var x, y: integer;
   procedure R;
         var y: real;
         begin
            x:= x + 1;
            writeln(x);
         end;
   procedure Q;
         var x: real; i: integer;
         begin
            x:= 3.0;
            R;
         end;
   procedure P;
         var x: boolean;
         begin
            Q;
         end;
BEGIN
   x:= 0;
   P;
   writeln(x);
END.
```

Ερώτημα:
Όταν εκτελείται η R,
ποιο x τυπώνεται;

Εύρεση των δηλώσεων ονομάτων (2)

- **Στατικός Κανόνας Εμβέλειας** (static scoping rule)

  Για κάθε μεταβλητή του Μη-Τοπικού ΠΑ, ψάχνει στο αμέσως πιο εξωτερικό block **στο κείμενο του προγράμματος**. Αν δεν βρεθεί εκεί, ψάχνει στο αμέσως πιο εξωτερικό block κ.ο.κ.

  C, C++, FORTRAN, COBOL, Pascal, Ada, …

- **Δυναμικός Κανόνας Εμβέλειας** (dynamic scoping)

  Για κάθε μεταβλητή του Μη-Τοπικού ΠΑ, ψάχνει στη **διαδικασία ή block που κάλεσε** την τρέχουσα. Αν δεν βρεθεί εκεί, ψάχνει στη διαδικασία ή block που κάλεσε αυτήν κ.ο.κ. Δηλαδή, εξετάζονται οι διαδικασίες και τα blocks με την αντίθετη σειρά από τη σειρά κλήσης τους.

  LISP, APL, …

Εύρεση των δηλώσεων ονομάτων (3)

```
Program main;
   var x, y: integer;
   procedure R;
         var y: real;
         begin
            x:= x + 1;
            writeln(x);
         end;
   procedure Q;
         var x: real; i: integer;
         begin
            x:= 3.0;
            R;
         end;
      procedure P;
            var x: boolean;
            begin
               Q;
            end;
BEGIN
   x:= 0;
   P;
   writeln(x);
END.
```

Ερώτημα:
Όταν εκτελείται η R,
ποιο x τυπώνεται;

Απάντηση:
ΣΚΕ: x = 1 (από main, x=0)
ΔΚΕ: x = 4.0 (από Q, x=3.0)

Εύρεση των δηλώσεων ονομάτων (4)

Πλεονεκτήματα του Στατικού Κανόνα Εμβέλειας

- Ο *Έλεγχος Τύπων Δεδομένων* (type checking) των ονομάτων οντοτήτων ενός προγράμματος είναι σημαντικός στον έλεγχο ορθότητας.
- Με Στατικό Κανόνα Εμβέλειας, ο έλεγχος αυτός γίνεται κατά τη Μετάφραση (Static Type Checking):
  - Αν υπάρχει σύνδεση της οντότητας με ΤΔ
  - Ιδιότητες της οντότητας
  - Συμβατότητα ΤΔ
- Πιο Ευανάγνωστα Προγράμματα.
- Πιο γρήγορα και πιο αξιόπιστα Προγράμματα

Χρόνος Δέσμευσης Μνήμης (1)

Δύο Αρχές για *History Sensitivity*:

1. **Διατήρηση (retention)**

   Η σύνδεση μιας τοπικής μεταβλητής με τη μνήμη, δημιουργείται με την κλήση του υπ/τος, και μετά την έξοδο από αυτό, διατηρείται μέχρι να ξανακληθεί το υπ/μα. (COBOL, μερικές FORTRAN)

2. **Διαγραφή (deletion)**

   Η σύνδεση τοπικής μεταβλητής με τη μνήμη, δημιουργείται με την κλήση του υπ/τος, και μετά την έξοδο από αυτό, καταστρέφεται (default στη C, Pascal, Ada, LISP, …)

Μερικές γλώσσες (π.χ. C, κάποιες Pascal, ALGOL, PL/1) επιτρέπουν και τα δύο.

Χρόνος Δέσμευσης Μνήμης (2)

- ALGOL 60: Τύπος μεταβλητών OWN → retained
- FORTRAN (από I ως και IV), όλες οι μεταβλητές static → retained.
- PL/1: Ο προγραμματιστής καθορίζει τον τρόπο δέσμευσης μνήμης με ιδιότητες της μεταβλητής:
  - STATIC → retained.
  - AUTOMATIC → deleted.
  - CONTROLLED → Δεσμεύεται μνήμη με εντολή ALLOCATE και αποδεσμεύεται με εντολή FREE (για pointers).
- Pascal: Υπάρχουν υλοποιήσεις στις οποίες μπορούν να οριστούν μεταβλητές static → retained. (default: deleted)
- C: με static → retained. (default: deleted)

Ενεργοποίηση Υποπρογράμματος (1)

Παράδειγμα:

```
function FN (X: real; Y: integer): real;
        const MAX = 20;
        VAR M: array [1..MAX] of real;
                N: integer;
        begin
          N:= MAX;
          X:= 2*X+M[5];
          {FN:= X;}
        end;
```

Ενεργοποίηση Υποπρογράμματος (2)

Τα στοιχεία που χρειάζονται για την ενεργοποίηση του υπ/τος FN κατά την εκτέλεση:

```
function FN (X: real; Y: integer): real;
    const MAX = 20;
    VAR M: array [1..MAX] of real;
        N: integer;
    begin
      N:= MAX;
      X:= 2*X+M[5];
    end;
```

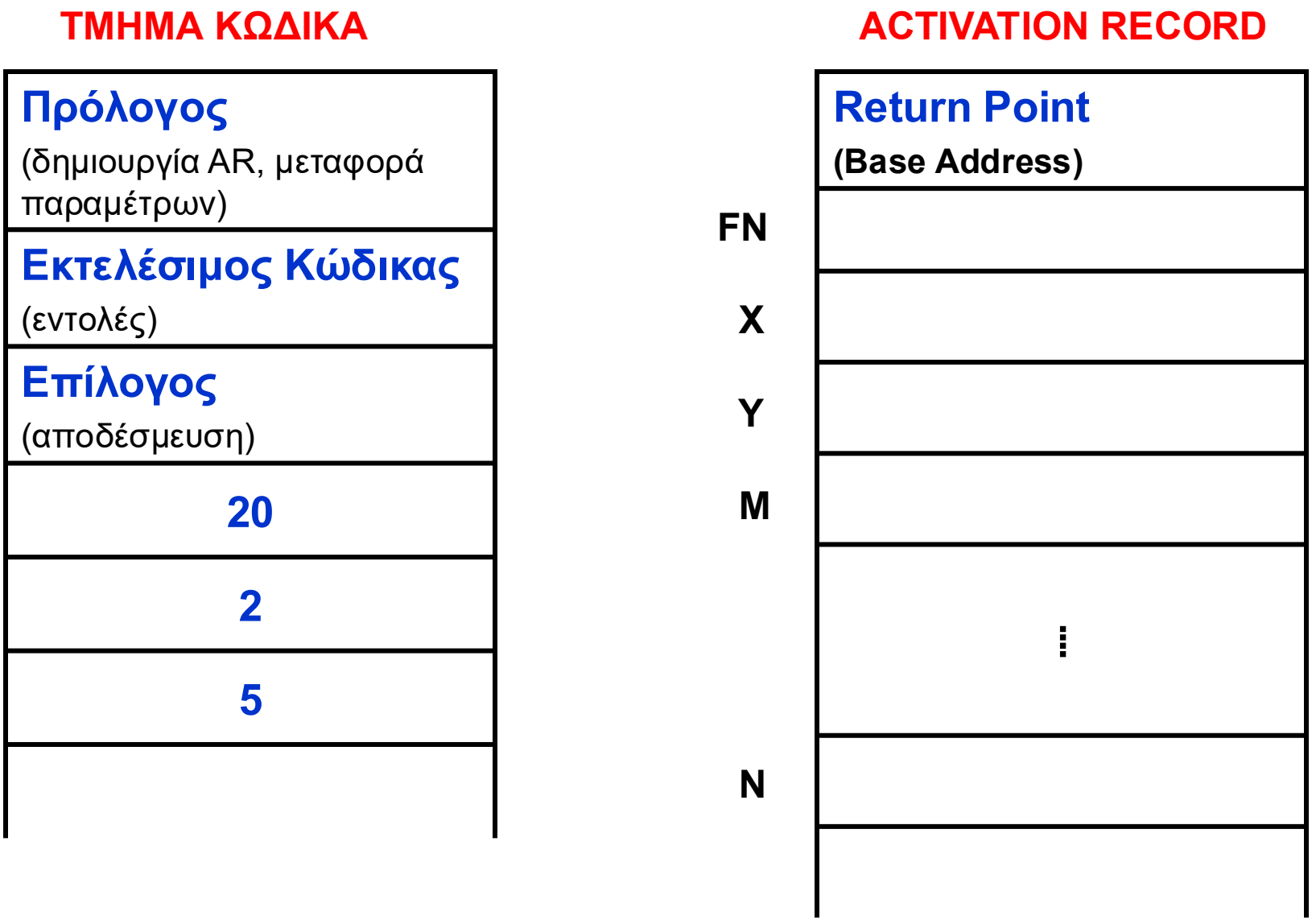
1. Αποθήκευση των παραμέτρων X, Y
2. Αποθήκευση του αποτελέσματος FN, τύπου real
3. Αποθήκευση των τοπικών μεταβλητών M, N
4. Αποθήκευση των σταθερών 20, 2, 5
5. Αποθήκευση του εκτελέσιμου κώδικα της FN

Η μνήμη της ενεργοποίησης μπορεί να χωριστεί σε 2 τμήματα:

A. Ένα *Στατικό Τμήμα* – **ΤΜΗΜΑ ΚΩΔΙΚΑ** – που περιλαμβάνει τα **4, 5**. Κάθε ενεργοποίηση χρησιμοποιεί το ίδιο αντίγραφο.

B. Ένα *Δυναμικό Τμήμα* – **ACTIVATION RECORD (AR)** – για **1, 2, 3**, και άλλα (προσωρινή μνήμη, return points, συνδέσεις κ.α.). Ίδια δομή σε κάθε ενεργοποίηση, αλλά διαφορετικό αντίγραφο, σε διαφορετικές θέσεις μνήμης, με άλλες τιμές.

Ενεργοποίηση Υποπρογράμματος (3)



Ενεργοποίηση Υποπρογράμματος (4)

Activation Record - AR

- Το μέγεθος και η δομή του καθορίζονται κατά τη μετάφραση.
- Ο μεταφραστής και το run-time περιβάλλον, μπορούν να καθορίσουν το μέγεθος και τη θέση κάθε αντικειμένου (δηλαδή την πρόσβαση σε αυτό), με χρήση του: **Base Address + Offset**
- Δεν χρειάζεται να αποθηκεύονται τα ονόματα των αντικειμένων.
- Οι retained μεταβλητές μπορούν να παίρνουν μνήμη από το *Τμήμα Κώδικα*.
- Οι deleted μεταβλητές παίρνουν μνήμη από το *Activation Record*.

Υλοποίηση Μηχανισμού Δέσμευσης (1)

```
Program main;
  var x, y: integer;
  procedure R;
      var y: real;
      begin
        x:= x + 1;
        writeln(x);
      end;
  procedure Q;
      var x: real; i: integer;
      begin
        x:= 3.0;
        R;
      end;
```

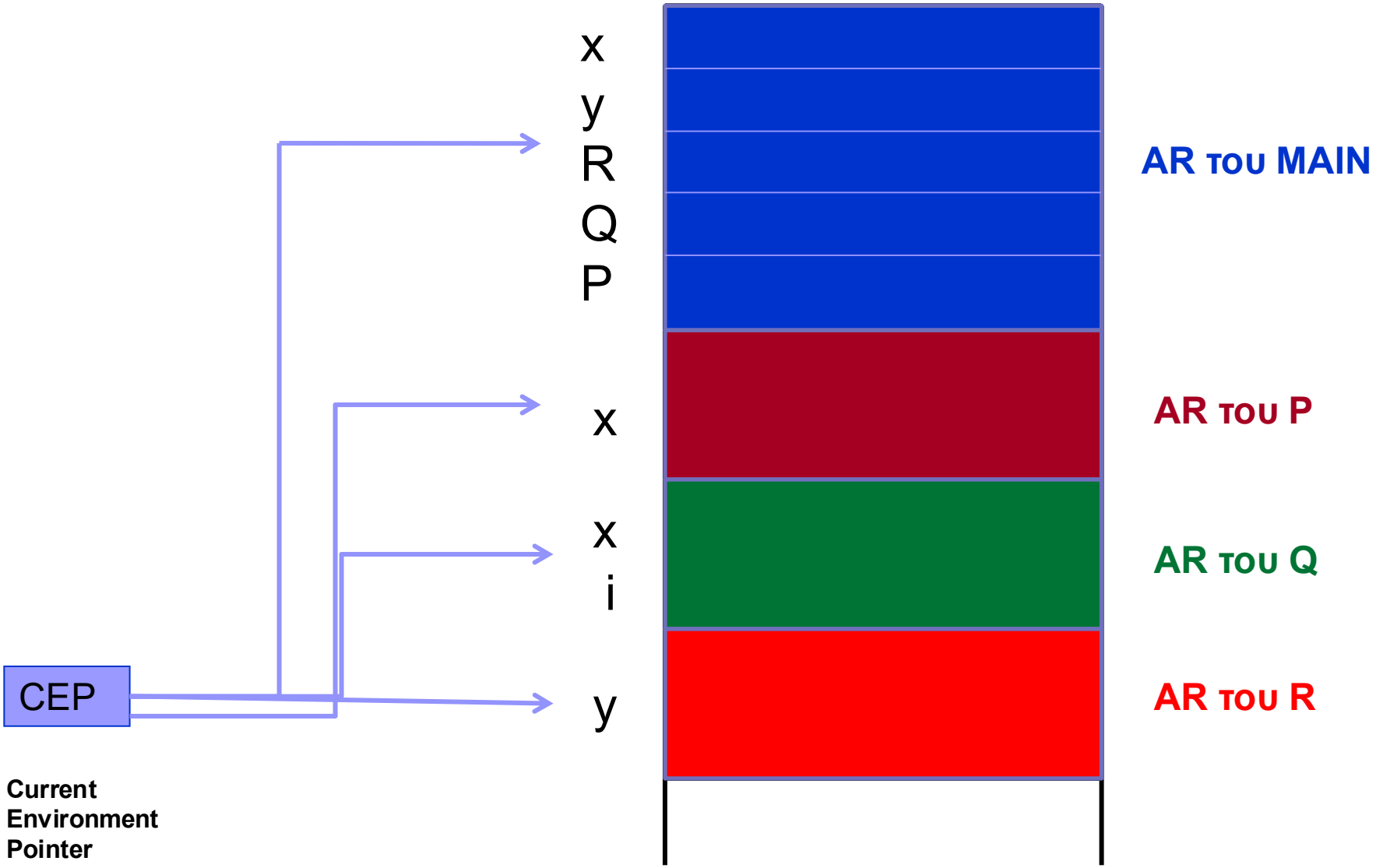
```
procedure P;
    var x: boolean;
    begin
      Q;
    end;
BEGIN
  x:= 0;
  P;
  writeln(x);
END.
```

Το Παράδειγμα

Ερώτημα:
Όταν εκτελείται η R,
ποιο x τυπώνεται;

Απάντηση:
ΣΚΕ: x = 1 (από main, x=0)
ΔΚΕ: x = 4.0 (από Q, x=3.0)

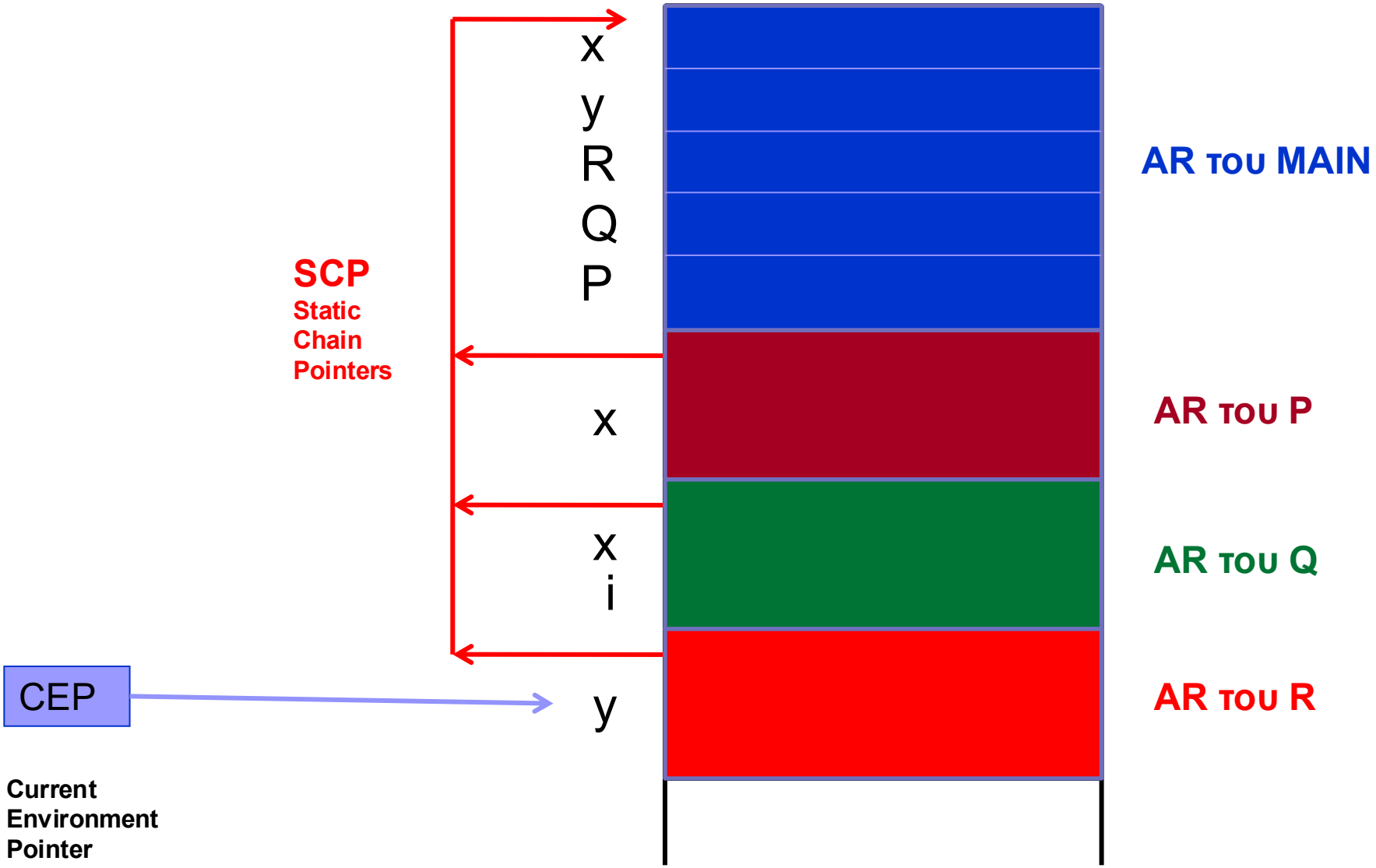
Κεντρική run-time stack



Υλοποίηση Μηχανισμού Δέσμευσης (2)

- Κάθε AR **δημιουργείται** με την έναρξη εκτέλεσης του υποπρογράμματος, και **διαγράφεται** με τη λήξη της εκτέλεσής του.
- Για την εύρεση των δηλώσεων των μεταβλητών που χρησιμοποιεί το τρέχον υπ/μα, γίνεται έλεγχος πρώτα στο AR που δείχνει ο **CEP** (Current Environment Pointer).
- Στη συνέχεια, αναζητούνται οι δηλώσεις στα «από πάνω» AR. Αυτό, όμως, υλοποιεί το Δυναμικό Κανόνα Εμβέλειας.
- Για την υλοποίηση του Στατικού Κανόνα Εμβέλειας, απαιτείται η χρήση ειδικών δεικτών, των Static Chain Pointers (Δείκτες Στατικής Αλυσίδας) που δείχνουν στην *πιο πρόσφατη ενεργοποίηση* του εξωτερικού μπλοκ με βάση το κείμενο του προγράμματος.

Κεντρική run-time stack



ΑΣΚΗΣΗ 1: Αναφέρετε το περιεχόμενο των ΠΑ (ΤΠΑ, Μη-ΤΠΑ και ΚΠΑ) του προγράμματος MAIN και των υποπρογραμμάτων που καλεί.

```
Program MAIN;
var X, Y, Z: int;
procedure PROC1(X: int);
    var W: int;
    procedure PROC2(Z: int);
        var W: int
        begin
            Z:=Z^Y;
        end;
    begin
        PROC2(Y);
    end;
begin
    PROC1(X);
end.
```

Προσοχή: Εξ' ορισμού ισχύει πως το Τοπικό ΠΑ του MAIN είναι και Μη-τοπικό, και Καθολικό.

ΠΑ της PROC2:
ΤΠΑ: Z, W
Μη-ΤΠΑ: X, PROC2, Y, PROC1
ΚΠΑ: Y, PROC1

ΠΑ της PROC1:
ΤΠΑ: X, W, PROC2
Μη-ΤΠΑ και ΚΠΑ: Y, Z, PROC1

ΠΑ του MAIN:
ΤΠΑ: X, Y, Z, PROC1
Μη-ΤΠΑ,ΚΠΑ: X, Y, Z, PROC1

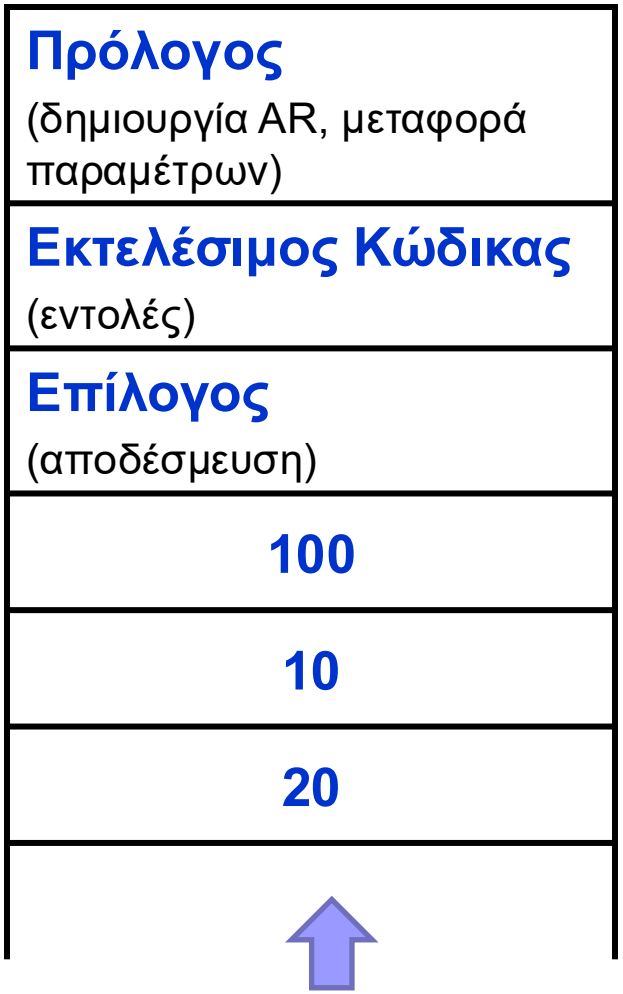
ΑΣΚΗΣΗ 2: Δίνεται το παρακάτω υποπρόγραμμα – συνάρτηση.

```
function FX (A: real; B: integer): real;
        const MAX = 100;
        VAR C: array [1..MAX] of real;
                X: integer;
        begin
          {C[20] := 33}
          X:= MAX;
          A:= 10*C[20];
          {FX:= A;}
        end;
```

Ποια είναι τα περιεχόμενα του στατικού και δυναμικού τμήματος της μνήμης κατά την εκτέλεση της FX?

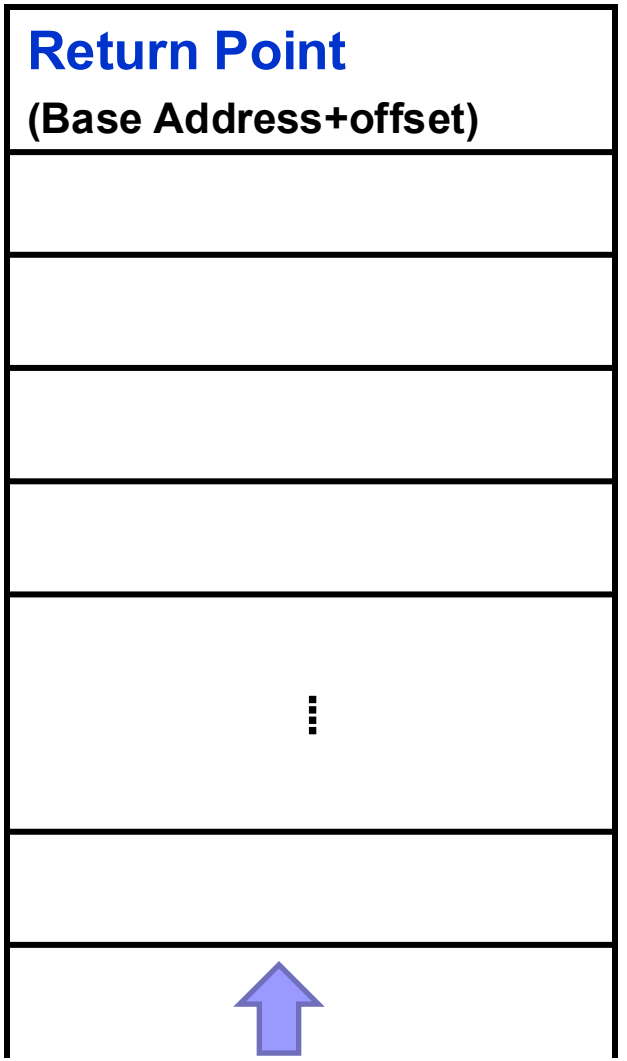
```
function FX (A: real; B: integer): real;
    const MAX = 100;
    VAR C: array [1..MAX] of real;
        X: integer;
    begin
      X:= MAX;
      A:= 10*C[20];
    end;
```

ΤΜΗΜΑ ΚΩΔΙΚΑ



RETAINED ΜΕΤΑΒΛΗΤΕΣ

ACTIVATION RECORD



DELETED ΜΕΤΑΒΛΗΤΕΣ

ΑΣΚΗΣΗ 2 (ΣΥΝ.)

Activation Record - AR

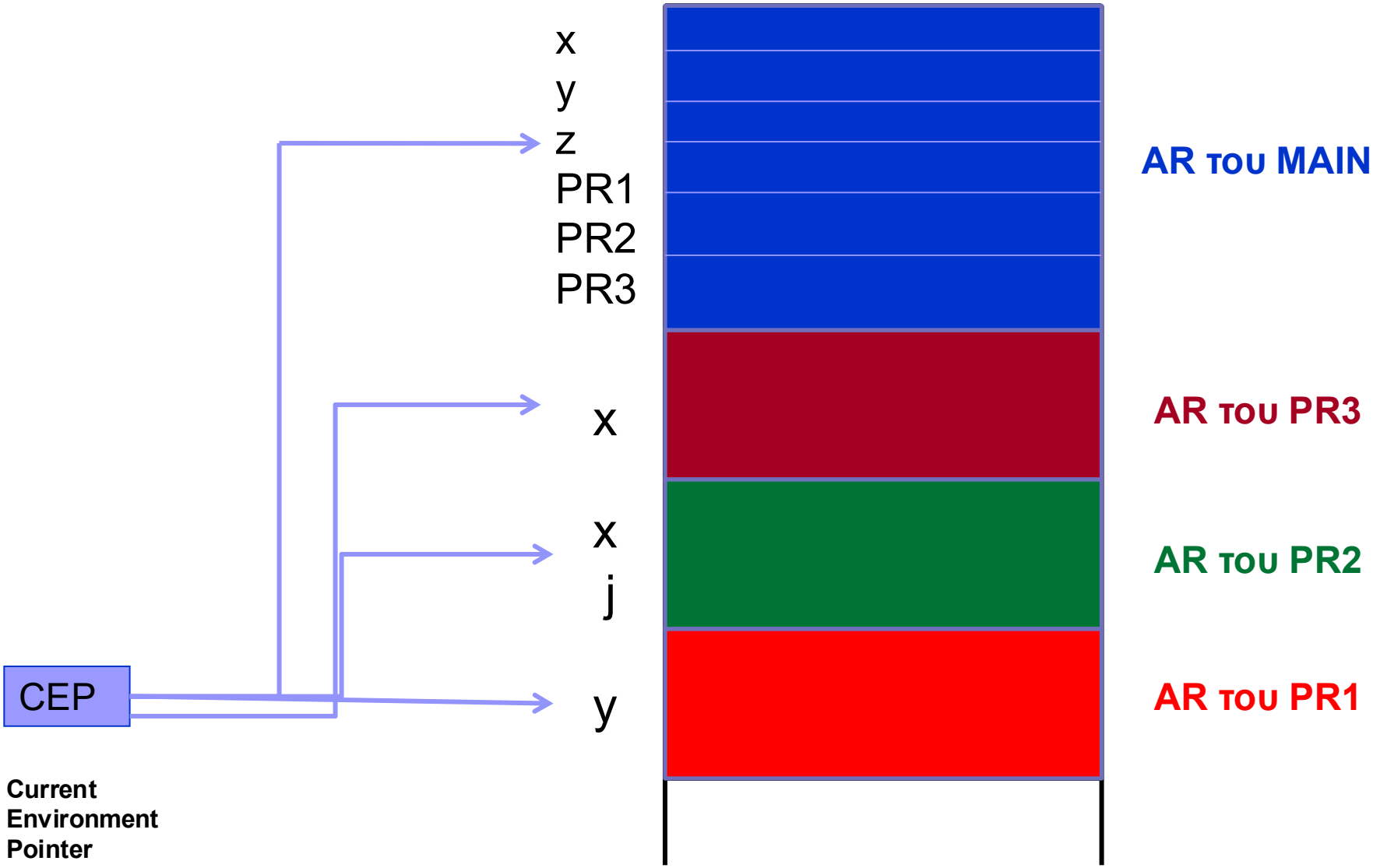
- Το μέγεθος και η δομή του καθορίζονται κατά τη μετάφραση.
- Ο μεταφραστής και το run-time περιβάλλον, μπορούν να καθορίσουν το μέγεθος και τη θέση κάθε αντικειμένου (δηλαδή την πρόσβαση σε αυτό), με χρήση του: **Base Address + Offset**
- Δεν χρειάζεται να αποθηκεύονται τα ονόματα των αντικειμένων.
- Οι retained μεταβλητές μπορούν να παίρνουν μνήμη από το *Τμήμα Κώδικα*.
- Οι deleted μεταβλητές παίρνουν μνήμη από το *Activation Record*.

ΑΣΚΗΣΗ 3: Τί θα τυπώσει η main ρουτίνα, αν εφαρμόσουμε στατικό κανόνα εμβέλειας και τι αν εφαρμόσουμε δυναμικό κανόνα εμβέλειας? Σχεδιάστε το περιεχόμενο της run-time stack και του δείκτη CEP σε κάθε περίπτωση.

```
Program main;
  var x,y,z: integer;
  procedure PR1;
      var y: real;
      begin
        x:= x + 1;
        writeln(x);
      end;
  procedure PR2;
      var x: real; j: integer;
      begin
        x:= 0.0;
        PR1;
      end;
  procedure PR3;
      var x: boolean;
      begin
        PR2;
      end;
BEGIN
  x:= 100;
  PR3;
  writeln(x);
END.
```

Ερώτημα:	**Απάντηση**:
Όταν εκτελείται η PR1, ποιο x τυπώνεται;	ΣΚΕ: x = 101 (από main, x=100) ΔΚΕ: x = 1.0 (από PR2, x=0.0)

Κεντρική run-time stack



ΑΣΚ3 (ΣΥΝ…)

- Κάθε AR **δημιουργείται** με την έναρξη εκτέλεσης του υποπρογράμματος, και **διαγράφεται** με τη λήξη της εκτέλεσής του.
- Για την εύρεση των δηλώσεων των μεταβλητών που χρησιμοποιεί το τρέχον υπ/μα, γίνεται έλεγχος πρώτα στο AR που δείχνει ο **CEP** (Current Environment Pointer).
- Στη συνέχεια, αναζητούνται οι δηλώσεις στα «από πάνω» AR. Αυτό, όμως, υλοποιεί το Δυναμικό Κανόνα Εμβέλειας.
- Για την υλοποίηση του Στατικού Κανόνα Εμβέλειας, απαιτείται η χρήση ειδικών δεικτών, των Static Chain Pointers (Δείκτες Στατικής Αλυσίδας) που δείχνουν στην *πιο πρόσφατη ενεργοποίηση* του εξωτερικού μπλοκ με βάση το κείμενο του προγράμματος.

Κεντρική run-time stack