

## Ονόματα και Εμβέλεια



Wassily Kandinsky, *Black lines*, 1913

Κωστής Σαγώνας <kostis@cs.ntua.gr>

## Ανακύκλωση ονομάτων

- Η κατανόηση της εμβέλειας είναι άμεση εάν το κάθε τι έχει το δικό του όνομα

```
fun square a = a * a;  
fun double b = b + b;
```

- Αλλά στις μοντέρνες γλώσσες προγραμματισμού, συχνά χρησιμοποιούμε το ίδιο όνομα ξανά και ξανά

```
fun square x = x * x;  
fun double x = x + x;
```

- Τι ακριβώς συμβαίνει;

Ονόματα και Εμβέλεια

2

## Περιεχόμενα

- Ορισμοί και εμβέλεια
- Εμβέλεια μέσω μπλοκ
- Εμβέλεια με προσδιορισμένους τόπους ονομάτων (labeled namespaces)
- Εμβέλεια με πρωτόγονους τόπους ονομάτων (primitive namespaces)
- Δυναμική εμβέλεια
- Ξεχωριστή μεταγλώττιση

Ονόματα και Εμβέλεια

3

## Ορισμοί

- Όταν υπάρχουν διαφορετικές μεταβλητές με το ίδιο όνομα, υπάρχουν και διαφορετικά πιθανά δεσίματα για κάθε συγκεκριμένο όνομα
- Το παραπάνω δεν αναφέρεται μόνο σε ονόματα μεταβλητών, αλλά σε ονόματα τύπων, ονόματα σταθερών, ονόματα συναρτήσεων, κ.λπ.
- Ένας ορισμός είναι οτιδήποτε εγκαθιστά ένα πιθανό δεσίμο για ένα όνομα

Ονόματα και Εμβέλεια

4

## Παραδείγματα

```
fun square n = n * n;  
fun square square = square * square;
```

```
const  
  Low = 1;  
  High = 10;  
type  
  Ints = array [Low..High] of Integer;  
var  
  X: Ints;
```

## Εμβέλεια

- Μπορεί να υπάρχουν περισσότεροι από ένας ορισμοί για κάποιο όνομα
- Κάθε εμφάνιση του ονόματος (εκτός των ορισμών) πρέπει να είναι δεμένη με κάποιον από τους ορισμούς του ονόματος
- Λέμε ότι η εμφάνιση ενός ονόματος **είναι στην εμβέλεια** κάποιου ορισμού του ονόματος όποτε ο ορισμός κυβερνά το δέσιμο της συγκεκριμένης εμφάνισης

## Παραδείγματα

```
- fun square square = square * square;  
val square = fn : int -> int  
- square 3;  
val it = 9 : int
```

- Κάθε εμφάνιση πρέπει να είναι δεμένη με κάποιον από τους ορισμούς
- Αλλά με ποιόν από όλους;
- Υπάρχουν πολλοί τρόποι να λύσουμε αυτό το πρόβλημα εμβέλειας

## Μπλοκ (blocks)

- Ένα μπλοκ είναι ένα οικοδόμημα των γλωσσών που περιέχει ορισμούς, και περικλείει μια περιοχή του προγράμματος στην οποία έχουν εφαρμογή οι ορισμοί

```
let  
  val x = 1;  
  val y = 2;  
in  
  x + y  
end
```

## Είδη μπλοκ στην ML

- Ένα `let` απλώς ορίζει ένα μπλοκ: δεν έχει άλλο σκοπό
- Ένας `fun` ορισμός περιλαμβάνει ένα μπλοκ:

```
fun cube x = x * x * x;
```

- Οι διαφορετικές εναλλακτικές προτάσεις των συναρτήσεων ορίζουν διαφορετικά μπλοκ:

```
fun f (a::b::_) = a+b  
  | f [a] = a  
  | f [] = 0;
```

- Κάθε κανόνας σε ένα ταίριασμα αποτελεί ένα μπλοκ:

```
case x of (a,0) => a | (_,b) => b
```

## Μπλοκ στη Java

- Στη Java και σε άλλες γλώσσες σαν τη C, μπορούμε να συνδυάσουμε εντολές σε μια **σύνθετη εντολή (compound statement)** περικλείοντάς τες σε { και }
- Μια σύνθετη εντολή χρησιμεύει επίσης σαν ένα μπλοκ:

```
while (i < 0) {  
  int c = i*i*i;  
  p += c;  
  q += c;  
  i -= step;  
}
```

## Φωλιασμένα μπλοκ (nested blocks)

- Τι συμβαίνει όταν ένα μπλοκ περιλαμβάνει ένα άλλο μπλοκ, και τα δύο μπλοκ εμπεριέχουν ορισμούς για το ίδιο όνομα;
- Παράδειγμα σε ML: ποια είναι η τιμή της διπλανής έκφρασης;

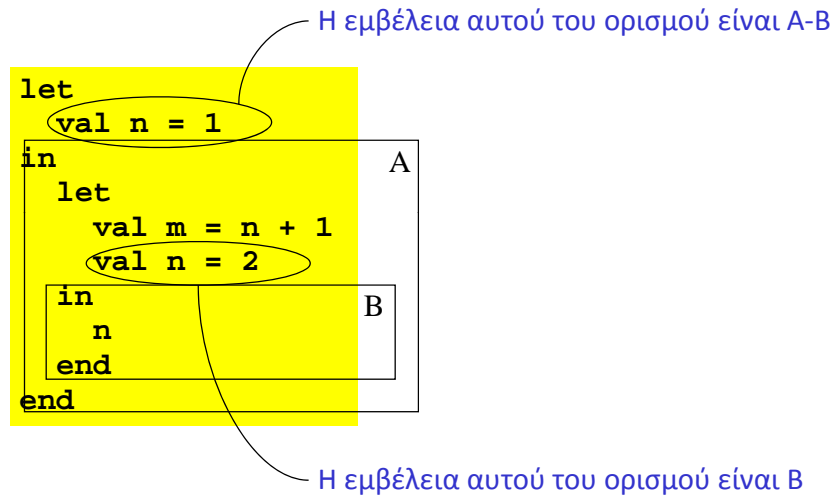
```
let  
  val n = 1  
in  
  let  
    val m = n + 1  
    val n = 2  
  in  
    n * m  
  end  
end
```



## Κλασικός κανόνας εμβέλειας σε μπλοκ

- Η εμβέλεια ενός ορισμού αποτελείται από το μπλοκ που περιέχει τον ορισμό, από το σημείο του ορισμού μέχρι το τέλος του μπλοκ, εκτός από την εμβέλεια των πιθανών ορισμών του ίδιου ονόματος σε εσωτερικά μπλοκ
- Αυτός είναι ο κανόνας εμβέλειας που χρησιμοποιείται στην ML
- Οι περισσότερες γλώσσες με δομή μπλοκ και στατική εμβέλεια χρησιμοποιούν αυτόν τον κανόνα ή κάποια μικρή παραλλαγή του

## Παράδειγμα



## Προσδιορισμένοι τόποι ονομάτων

- Ένας **προσδιορισμένος τόπος ονομάτων** είναι οτιδήποτε οικοδόμημα της γλώσσας περιέχει ορισμούς και μια περιοχή του προγράμματος στην οποία ισχύουν οι ορισμοί
- Ο προσδιορισμένος τόπος έχει ένα όνομα που μπορεί να χρησιμοποιηθεί για προσπέλαση των ορισμών του από έξω από το συγκεκριμένο οικοδόμημα
- Η ML έχει ένα τέτοιο οικοδόμημα – τη **δομή (structure)**

## Δομές στην ML (ML structures)

```
structure Fred = struct
  val a = 1;
  fun f x = x + a;
end;
```

- Οι δομές είναι περίπου σα μπλοκ: η τιμή **a** μπορεί να χρησιμοποιηθεί παντού από τον ορισμό της μέχρι το τέλος της δομής
- Αλλά οι ορισμοί μέσα στη δομή είναι προσπελάσιμοι και από εκτός της δομής, με χρήση του ονόματός της: **Fred.a** και **Fred.f**

## Άλλοι προσδιορισμένοι τόποι ονομάτων

- Τόποι ονομάτων που είναι απλώς τόποι ονομάτων:
  - C++ `namespace`
  - Modula-3 `module`
  - Ada `package`
  - Java `package`
- Τόποι ονομάτων που έχουν και άλλους σκοπούς:
  - Ορισμοί κλάσεων σε αντικειμενοστρεφείς γλώσσες προγραμματισμού που είναι βασισμένες σε κλάσεις

## Παράδειγμα σε Java

```
public class Month {
    public static int min = 1;
    public static int max = 12;
    ...
}
```

- Οι μεταβλητές `min` και `max` είναι ορατές σε όλη την υπόλοιπη κλάση
- Αλλά είναι προσπελάσιμες και εκτός της κλάσης, ως `Month.min` και `Month.max`
- Φυσικά οι κλάσεις έχουν και άλλους σκοπούς...

## Πλεονεκτήματα τύπων ονομάτων

- Δύο αλληλοσυγκρουόμενοι στόχοι:
  - Χρησιμοποίηση ενός απλού μνημονικού ονόματος όπως π.χ. `max`
  - Για ότι θέλουμε να είναι απροσπέλαστο από τον υπόλοιπο κόσμο, χρησιμοποιούμε ασυνήθη ονόματα όπως `maxSupplierBid`, δηλαδή ονόματα που δε θα συγκρούονται με ονόματα σε άλλα μέρη του προγράμματος
- Με χρήση τύπων ονομάτων, μπορούμε να επιτύχουμε και τους δύο στόχους:
  - Εντός του τύπου ονόματος, μπορούμε να χρησιμοποιήσουμε το όνομα `max`
  - Από έξω, το όνομα `SupplierBid.max`

## Εκλέπτυνση τύπων ονομάτων

- Οι περισσότεροι τύποι ονομάτων έχουν κάποιο τρόπο να επιτρέπουν μέρος του τύπου ονομάτων να παραμένει ιδιωτικής (δηλ. μη κοινής) χρήσης
- Με τον τρόπο αυτό κρύβουμε πληροφορία από κάποια μέρη
- Τα προγράμματα είναι εύκολα συντηρούμενα όταν οι χρησιμοποιούν ονόματα με σχετικά μικρή εμβέλεια
- Για παράδειγμα, οι **αφηρημένοι τύποι δεδομένων (abstract data types)** αποκαλύπτουν μια αυστηρή διαπροσωπεία και ταυτόχρονα κρύβουν τις λεπτομέρειες της υλοποίησης...

## Παράδειγμα αφηρημένου τύπου δεδομένων

```
namespace dictionary contains
    a constant definition for initialSize
    a type definition for hashTable
    a function definition for hash
    a function definition for reallocate
    a function definition for create
    a function definition for insert
    a function definition for search
    a function definition for delete
end namespace
```

Οι λεπτομέρειες της υλοποίησης πρέπει να μείνουν κρυφές

Οι ορισμοί της διαπροσωπείας όμως πρέπει να είναι ορατοί

## Δύο προσεγγίσεις

- Σε κάποιες γλώσσες, όπως η C++, ο τύπος ονομάτων καθορίζει λεπτομερώς την ορατότητα των συστατικών στοιχείων του
- Σε άλλες γλώσσες, όπως η ML, κάποιο ξεχωριστό οικοδόμημα ορίζει τη διαπροσωπεία του τύπου ονομάτων (π.χ. ένα **signature** στην ML)
- Και κάποιες γλώσσες, όπως η Ada και η Java, συνδυάζουν και τις δύο παραπάνω προσεγγίσεις

## Ο τύπος ονομάτων καθορίζει την ορατότητα

```
namespace dictionary contains
  private:
    a constant definition for initialSize
    a type definition for hashTable
    a function definition for hash
    a function definition for reallocate
  public:
    a function definition for create
    a function definition for insert
    a function definition for search
    a function definition for delete
end namespace
```

## Ξεχωριστή διαπροσωπεία

```
interface dictionary contains
  a function type definition for create
  a function type definition for insert
  a function type definition for search
  a function type definition for delete
end interface
```

```
namespace myDictionary implements dictionary contains
  a constant definition for initialSize
  a type definition for hashTable
  a function definition for hash
  a function definition for reallocate
  a function definition for create
  a function definition for insert
  a function definition for search
  a function definition for delete
end namespace
```

## Μη το δοκιμάσετε στο σπίτι!

```
- val int = 3;
val int = 3 : int
```

- Είναι επιτρεπτό να ορίσουμε μια μεταβλητή με όνομα `int`
- Η ML δε μπερδεύεται
- Ακόμα και το επόμενο επιτρέπεται (η ML καταλαβαίνει ότι η έκφραση `int * int` δεν αναφέρεται σε κάποιο τύπο):

```
- fun f int = int * int;
val f = fn : int -> int
- f 3;
val it = 9 : int
```

## Πρωτόγονοι τόποι ονομάτων

- Η σύνταξη της ML κρατάει ξεχωριστούς τους τύπους και τις εκφράσεις
- Η ML πάντα ξέρει κατά πόσο κάτι είναι ένας τύπος ή κάτι άλλο, διότι υπάρχει ξεχωριστός τόπος ονομάτων για τους τύπους

```
fun f((int:int) = (int:int)*(int:int));
```

Τόπος ονομάτων  
των μεταβλητών

Τόπος ονομάτων  
των τύπων

## Πρωτόγονοι τόποι ονομάτων

- Είναι τόποι ονομάτων που δε μπορούν να δημιουργηθούν με χρήση της γλώσσας (όπως οι πρωτόγονοι τύποι)
- Είναι μέρος του ορισμού της γλώσσας
- Κάποιες γλώσσες έχουν αρκετούς διαφορετικούς μεταξύ τους πρωτόγονους τόπους ονομάτων
- Π.χ. η Java: τα πακέτα (packages), οι τύποι, οι μέθοδοι, τα πεδία, και οι ετικέτες των εντολών βρίσκονται σε διαφορετικούς τόπους ονομάτων

## Δυναμική εμβέλεια

## Πότε επιλύεται η εμβέλεια;

- Όλες οι μέθοδοι εμβέλειας που είδαμε μέχρι στιγμής είναι στατικές
- Απαντούν κατά το χρόνο μεταγλώττισης στην ερώτηση του κατά πόσο μια εμφάνιση κάποιου ονόματος είναι στην εμβέλεια κάποιου ορισμού
- Κάποιες γλώσσες καθυστερούν την παραπάνω απόφαση μέχρι το χρόνο εκτέλεσης: λέμε ότι οι γλώσσες αυτές υποστηρίζουν **δυναμική εμβέλεια**

## Δυναμική εμβέλεια

- Κάθε συνάρτηση έχει ένα περιβάλλον από ορισμούς
- Εάν ο ορισμός ενός ονόματος που βρίσκεται σε μια συνάρτηση  $f$  δεν βρίσκεται στο περιβάλλον της συνάρτησης, τότε ψάχνουμε στο περιβάλλον της συνάρτησης που κάλεσε την  $f$
- Εάν ο ορισμός δε βρεθεί ούτε εκεί, τότε η αναζήτηση συνεχίζεται ακολουθώντας την αλυσίδα των κλήσεων των συναρτήσεων
- Στη δυναμική εμβέλεια ο κανόνας είναι κάπως περίεργος

## Κλασικός κανόνας δυναμικής εμβέλειας

- Η εμβέλεια ενός ορισμού είναι η συνάρτηση που περιέχει τον ορισμό, από το σημείο του ορισμού μέχρι το τέλος της συνάρτησης, μαζί με όλες τις συναρτήσεις που καλούνται (άμεσα ή έμμεσα) από την εμβέλεια—εκτός της εμβέλειας οποιωνδήποτε νέων ορισμών των ίδιων ονομάτων στις καλούμενες συναρτήσεις

## Στατική εμβέλεια έναντι δυναμικής

- Οι κανόνες εμβέλειας είναι παρόμοιοι
- Και οι δύο μιλάνε για **τρύπες εμβέλειας**—μέρη στα οποία η εμβέλεια διακόπτεται εξαιτίας νέων ορισμών
- Αλλά ο κανόνας της στατικής εμβέλειας αναφέρεται μόνο σε κομμάτια του κειμένου του προγράμματος, και κατά συνέπεια μπορεί να εφαρμοστεί κατά το χρόνο μεταγλώττισης του προγράμματος
- Ο κανόνας της δυναμικής εμβέλειας αναφέρεται σε γεγονότα χρόνου εκτέλεσης, συγκεκριμένα σε “ακολουθίες κλήσεων συναρτήσεων”

## Παράδειγμα

```
fun g x =  
  let  
    val inc = 1;  
    fun f y = y+inc;  
    fun h z =  
      let  
        val inc = 2;  
      in  
        f z  
      end;  
  in  
    h x  
  end;
```

Ποια θα ήταν η τιμή της κλήσης  $g\ 5$  αν η γλώσσα χρησιμοποιούσε  
α) στατική εμβέλεια;  
β) δυναμική εμβέλεια;



## Στατική εμβέλεια (εμβέλεια μπλοκ)

```
fun g x =  
  let  
    val inc = 1;  
    fun f y = y+inc;  
    fun h z =  
      let  
        val inc = 2;  
      in  
        f z  
      end;  
  in  
    h x  
  end;
```

Στη στατική εμβέλεια, η μεταβλητή `inc` δένεται με τον προηγούμενο ορισμό στο ίδιο μπλοκ. Ο ορισμός στο περιβάλλον κλήσης της `f` είναι μη προσπελάσιμος.

`g 5 = 6`  
σε αυτήν την περίπτωση (όπως και στην ML)

## Δυναμική εμβέλεια

```
fun g x =  
  let  
    val inc = 1;  
    fun f y = y+inc;  
    fun h z =  
      let  
        val inc = 2;  
      in  
        f z  
      end;  
  in  
    h x  
  end;
```

Στη δυναμική εμβέλεια, η μεταβλητή `inc` δένεται με τον ορισμό στο περιβάλλον κλήσης της `f`.

`g 5 = 7`  
σε αυτήν την περίπτωση

## Που χρησιμοποιείται η δυναμική εμβέλεια

- Σε λίγες μόνο γλώσσες: σε κάποιες διαλέκτους της Lisp και της APL
- Υπάρχει διαθέσιμη ως επιλογή στην Common Lisp
- Μειονεκτήματα:
  - Δυσκολία αποδοτικής υλοποίησης
  - Δημιουργεί μεγάλες και περίπλοκες εμβέλειες (οι εμβέλειες επεκτείνονται στις καλούμενες συναρτήσεις)
  - Η επιλογή του ονόματος κάποιας μεταβλητής στη συνάρτηση κλήσης μπορεί να επηρεάσει τη συμπεριφορά της καλούμενης συνάρτησης
- Πλεονέκτημα:
  - Δυνατότητα αλλαγής/επιλογής του περιβάλλοντος εκτέλεσης

## Χρησιμότητα της δυναμικής εμβέλειας

```
program messages;  
var message : string;  
  
procedure complain;  
begin  
  writeln(message);  
end;  
  
procedure out_of_mem;  
var message : string;  
begin  
  message := "Out of memory"; complain;  
end;  
  
procedure timeout;  
var message : string;  
begin  
  message := "Timeout"; complain;  
end;
```

## Δηλώσεις forward

- Γλώσσες όπως η C, C++ και η Pascal χρειάζονται **δηλώσεις forward** για αμοιβαία αναδρομικούς ορισμούς

```
int foo();  
int bar() { ... foo(); ... }  
int foo() { ... bar(); ... }
```



- Εν μέρει το παραπάνω είναι παρενέργεια της διαδικασίας που ακολουθείται από το μεταγλωττιστή: επιτρέπει τη μετάφραση με ένα μόνο πέρασμα

## Ξεχωριστή μεταγλώττιση

- Μέρη του προγράμματος μεταγλωττίζονται ξεχωριστά, και στη συνέχεια συνδέονται (linked) μαζί
- Τα θέματα εμβέλειας επεκτείνονται από το μεταγλωττιστή (compiler) στο συνδέτη (linker) το οποίο χρειάζεται να συνδέσει αναφορές σε ορισμούς που έχουν προκύψει από ξεχωριστή μεταγλώττιση
- Οι περισσότερες γλώσσες έχουν κάποια ειδική υποστήριξη για το πως γίνεται η σύνδεση των κομματιών του προγράμματος που έχουν μεταγλωττιστεί ξεχωριστά

## Η προσέγγιση της C: η μεριά του μεταγλωττιστή

- Δύο διαφορετικά είδη ορισμών ονομάτων:
  - Πλήρης **ορισμός (definition)**
  - Όνομα και τύπος μόνο: μια **δήλωση (declaration)** κατά τη C
- Εάν πολλές διαφορετικές μεταγλωττίσεις/αρχεία θέλουν να χρησιμοποιήσουμε την ίδια ακέραια μεταβλητή **x**:
  - Μόνο ένα αρχείο θα έχει τον ορισμό:  
`int x = 3;`
  - Όλα τα άλλα αρχεία θα έχουν τη δήλωση:  
`extern int x;`

## Η προσέγγιση της C: η μεριά του συνδέτη

- Όταν τρέχει ο συνδέτης, θεωρεί κάθε δήλωση ως μια αναφορά σε ένα όνομα που ορίζεται σε κάποιο άλλο αρχείο
- Περιμένει να βρει ακριβώς έναν πλήρη ορισμό του κάθε ονόματος (σε κάποιο αρχείο)
- Προσέξτε ότι οι δηλώσεις δεν αναφέρουν σε ποιο σημείο βρίσκεται ο πλήρης ορισμός και πως αυτός θα βρεθεί— απλώς αξιώνουν από το συνδέτη να ψάξει να τον βρει κάπου

## Τάσεις ξεχωριστής μεταγλώττισης

---

- Στις σύγχρονες γλώσσες, η ξεχωριστή μεταγλώττιση είναι λιγότερο ξεχωριστή από ότι ήταν πριν από κάποια χρόνια
  - Οι κλάσεις της Java μπορεί να εξαρτώνται κυκλικά μεταξύ τους, και ο μεταγλωττιστής της Java πρέπει να είναι σε θέση να μεταφράσει πολλές ξεχωριστές κλάσεις ταυτόχρονα
  - Η ML δεν προσφέρεται καθόλου για ξεχωριστή μεταγλώττιση, αλλά υπάρχει ο CM (Compilation Manager, ένα ξεχωριστό εργαλείο του συστήματος SML/NJ system) μπορεί να το κάνει για τα περισσότερα προγράμματα σε ML

## Συμπερασματικά

---

- Τέσσερις διαφορετικές προσεγγίσεις για εμβλήματα
- Υπάρχουν πολλές διαφοροποιήσεις στις προσεγγίσεις
- Οι περισσότερες γλώσσες χρησιμοποιούν τουλάχιστον κάποιες από αυτές

