



Προγραμματισμός Ι (ECE115)

#1

μνήμη & μεταβλητές
πρόγραμμα & εκτέλεση

Ψηφιακά δεδομένα, μνήμη, μεταβλητές

Δυαδικός κόσμος

- Οι υπολογιστές είναι δυαδικές μηχανές
- Όλη η πληροφορία (δεδομένα και κώδικας) κωδικοποιείται ως μια ακολουθία από 0 ή 1
- Η ελάχιστη μονάδα πληροφορίας ονομάζεται **bit**
- 1 bit μπορεί να πάρει τις τιμές 0 / 1
- Η βασική μονάδα πληροφορίας είναι το **byte**
- 1 byte αποτελείται από 8 bit

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---



1	1	0	0	1	0	1	0
---	---	---	---	---	---	---	---

Μνήμη

- **Αφαίρεση**: μια (μακριά) σειρά από bytes
- Πώς γίνεται αναφορά σε ένα συγκεκριμένο byte;
- **Διεύθυνση**
 - μας λέει το **πού** βρίσκεται ένα byte στην μνήμη
 - συγκεκριμένα: ο **αριθμός σειράς** του byte στη μνήμη
- Οι διευθύνσεις αρχίζουν (θεωρητικά) από την τιμή 0

διευθύνσεις

περιεχόμενα

0

1

2

3

4

5

N-2

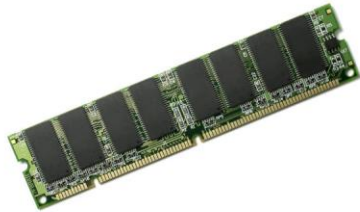
N-1

0	0	1	1	0	1	0	1
1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0
3	1	1	0	0	1	0	1
4	0	0	1	1	0	1	1
5	1	1	1	1	1	1	1
...							
N-2	0	0	0	0	0	0	0
N-1	1	1	0	0	1	0	1

Μνήμη / αποθηκευτικός χώρος Η/Υ

Κύρια μνήμη (RAM)

- γρήγορη πρόσβαση
- μερικά GB



Σκληρός δίσκος (hard disk)

- αργή πρόσβαση
- λίγα TB

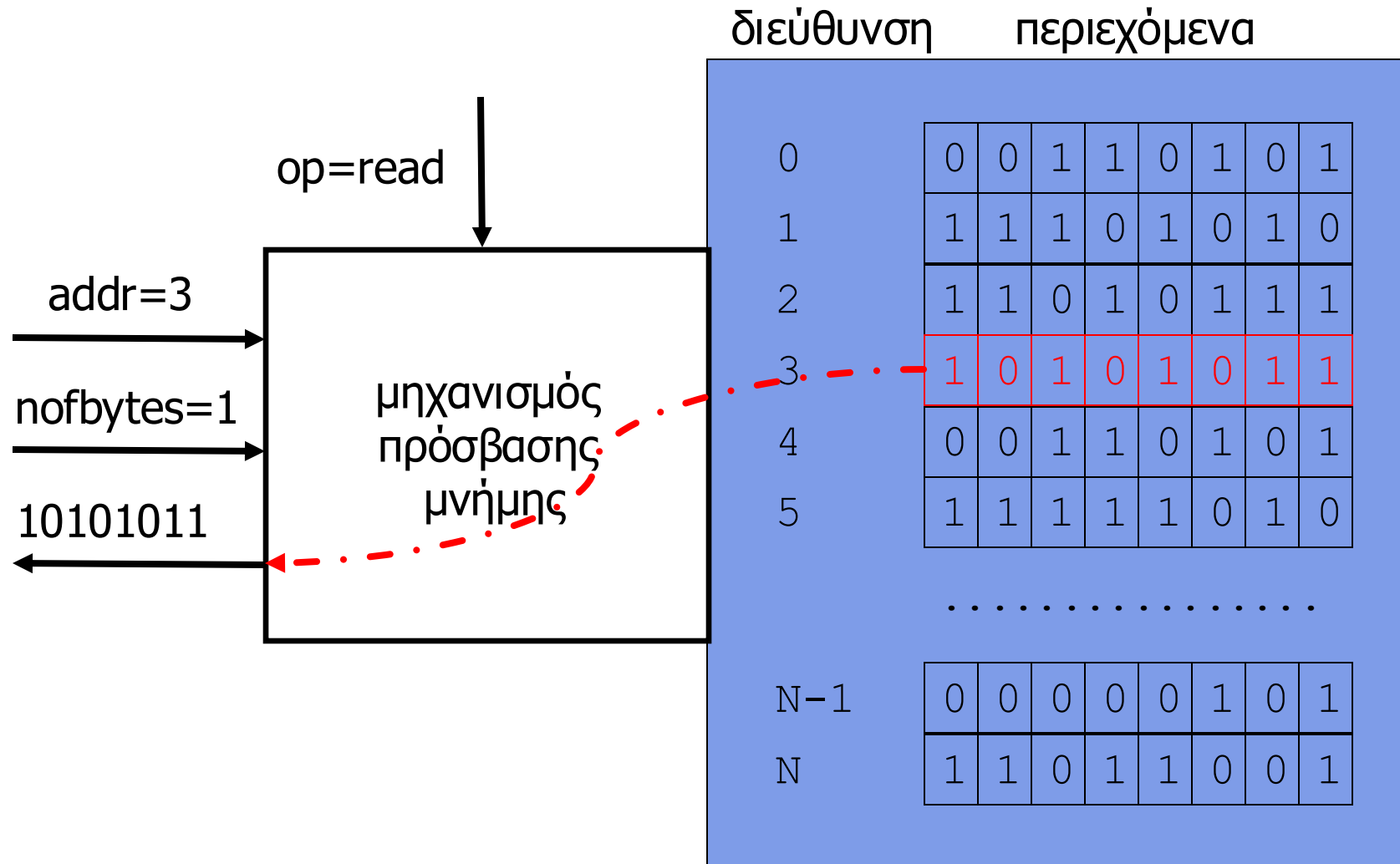


- 1 KB = 1024 B = 2^{10} B
- 1 MB = 1024 KB = 2^{20} B
- 1 GB = 1024 MB = 2^{30} B
- 1 TB = 1024 GB = 2^{40} B

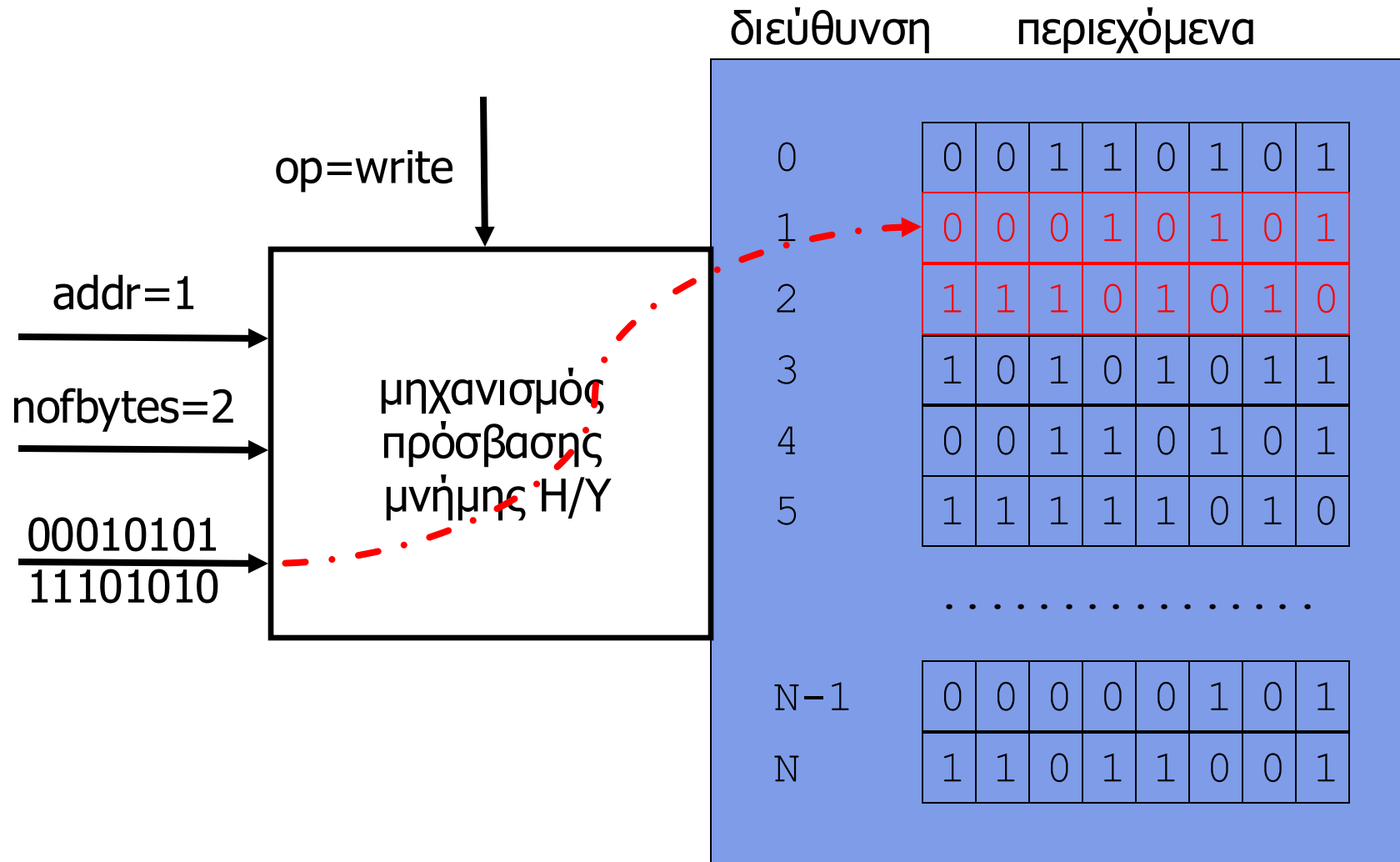
Πρόσβαση στην μνήμη

- Λειτουργίες: ανάγνωση / εγγραφή
- Σημείο αναφοράς: διεύθυνση
- Ποσότητα δεδομένων: αριθμός bytes
- **Ανάγνωση:** επιστρέφει τις τιμές των bytes που έχουν αποθηκευτεί στις αντίστοιχες θέσεις
- **Εγγραφή:** προσδιορίζει τις τιμές των bytes που θα αποθηκευτούν στις αντίστοιχες θέσεις

Ανάγνωση



Εγγραφή



Κωδικοποίηση δεδομένων

- Κωδικοποίηση: η μετατροπή δεδομένων από ένα σύστημα αναπαράστασης σε ένα άλλο
 - δεκαδικό $\rightarrow$ δυαδικό
 - χαρακτήρες $\rightarrow$ δυαδικό
- Πόσα bits χρειαζόμαστε για κάθε είδος δεδομένων;
- Περισσότερα bits/bytes
 - μεγαλύτερο πεδίο τιμών
 - μεγαλύτερη ακρίβεια
- Με N bits κωδικοποιούμε 2^N διαφορετικές τιμές
 - π.χ. πεδίο φυσικών $[0 \dots 2^N - 1]$
 - π.χ. πεδίο ακεραίων $[-2^{N-1} \dots 2^{N-1} - 1]$

Δεδομένα και προγραμματισμός

- Για κάθε δεδομένο πρέπει να καθοριστεί (θεωρητικά, **από τον προγραμματιστή**)
 - η θέση μνήμης για την αποθήκευση του
 - ο αριθμός των bytes που χρειάζονται για την αποθήκευση των τιμών που μπορεί να λάβει
 - η κωδικοποίηση που χρησιμοποιείται για κάθε τιμή στο δυαδικό σύστημα
- Από άνθρωπο!
- Χρειάζεται κατάλληλη υποστήριξη από τις γλώσσες προγραμματισμού ...

Τύποι δεδομένων

- Κάθε γλώσσα προγραμματισμού ορίζει ένα σύνολο από βασικούς **τύπους δεδομένων**
- Κάθε βασικός τύπος έχει
 - συγκεκριμένο **μέγεθος** (σε bytes)
 - συγκεκριμένη δυαδική **κωδικοποίηση** και αντίστοιχο πεδίο τιμών
- Ο προγραμματιστής ορίζει κάθε δεδομένο του προγράμματος ως «αντικείμενο» ενός τύπου
 - έτσι καθορίζεται **έμμεσα** το μέγεθος και η κωδικοποίηση (η **σημασία** των bits που αντιστοιχούν στο αντικείμενο)
- Η μνήμη χρησιμεύει για την αποθήκευση των bits/bytes **χωρίς** να γίνεται ερμηνεία τους

Μεταβλητή

- **Αντικείμενο δεδομένων** του προγράμματος
 - με συγκεκριμένο **όνομα** και **τύπο**
- «**Αναθέτουμε**» μια τιμή σε μεταβλητή
 - η (κωδικοποιημένη) τιμή (bytes) γράφεται στην αντίστοιχη θέση της μνήμης
- «**Διαβάζουμε**» μια μεταβλητή
 - η (κωδικοποιημένη) τιμή (bytes) διαβάζεται από την αντίστοιχη θέση της μνήμης
- «**Δέσμευση**» μεταβλητής
 - κράτηση αντίστοιχου χώρου στην μνήμη
- «**Ζωή**» μεταβλητής
 - διάστημα κατά το οποίο η κράτηση παραμένει σε ισχύ

```
int var;
```

θεση/διεύθυνση: 1
τύπος: int

τύπος int

μέγεθος: 2 bytes (ενδεικτικό)
ερμηνεία: ακέραιος
κωδικοποίηση: 2's complement

τι τιμή έχει η μεταβλητή var;

διάβασε από τη διεύθυνση της var
όσα bytes ορίζει ο τύπος της, και
ερμήνευσέ τα σύμφωνα με τη
σύμβαση κωδικοποίησης του

αποτέλεσμα = 255
(για little-endian)

διεύθυνση περιεχόμενα

0	0	0	1	1	0	1	0	1
1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	1	1	1	1	1	0	1	0
.....								
N-1	0	0	0	0	0	1	0	1
N	1	1	0	1	1	0	0	1

Διαφάνεια αποθήκευσης/κωδικοποίησης

- Ο προγραμματιστής **δεν** χρειάζεται να γνωρίζει το πώς ακριβώς αποθηκεύεται/κωδικοποιείται η πληροφορία στην μνήμη του υπολογιστή
- Η προσπάθεια των περιεχομένων των μεταβλητών του προγράμματος γίνεται με **διαφανή** τρόπο
 - ένα από τα πλεονεκτήματα του να χρησιμοποιεί κανείς μια γλώσσα προγραμματισμού υψηλού επιπέδου
- Αυτό δεν ισχύει όταν
 - χρησιμοποιείται γλώσσα μηχανής
 - όταν παρακάμπτεται το «σύστημα τύπων» της γλώσσας
 - βλέπε αργότερα ...

Πρόγραμμα & εκτέλεση

Πρόγραμμα / Γλώσσα προγραμματισμού

- **Πρόγραμμα:** εντολές προς τον υπολογιστή
 - έτσι όπως τις «καταλαβαίνει» ο υπολογιστής
 - με λίγη βοήθεια ... (ο υπολογιστής ξέρει μόνο από 0 και 1)
- **Γλώσσα προγραμματισμού:** γλώσσα σχεδιασμένη για να γράφουμε εντολές προς τον υπολογιστή
- Περίπου σαν τις κανονικές γλώσσες
- Αλλά πιο αυστηρή / τυπική
 - περιορισμένο **λεξιλόγιο**
 - σαφής **σημασιολογία**
 - ξεκάθαρο **συντακτικό**

Σύνταξη & σημασία

- **Σύνταξη:** κανόνες για τον σχηματισμό προτάσεων
- **Σημασία:** το νόημα που έχουν οι συντακτικά επιτρεπτές προτάσεις
- **Υπάρχουν συντακτικά επιτρεπτές προτάσεις που **δεν** έχουν επακριβώς ορισμένη (μια μοναδική) σημασία;**
- Στις ανθρώπινες γλώσσες: ναι!
 - μαζί μιλάμε και χώρια καταλαβαινόμαστε
- Σε γλώσσες προγραμματισμού: (κατά κανόνα) όχι

Για παράδειγμα

- Συντακτικά ορθές προτάσεις

$$-2+13$$

$$7-15/5$$

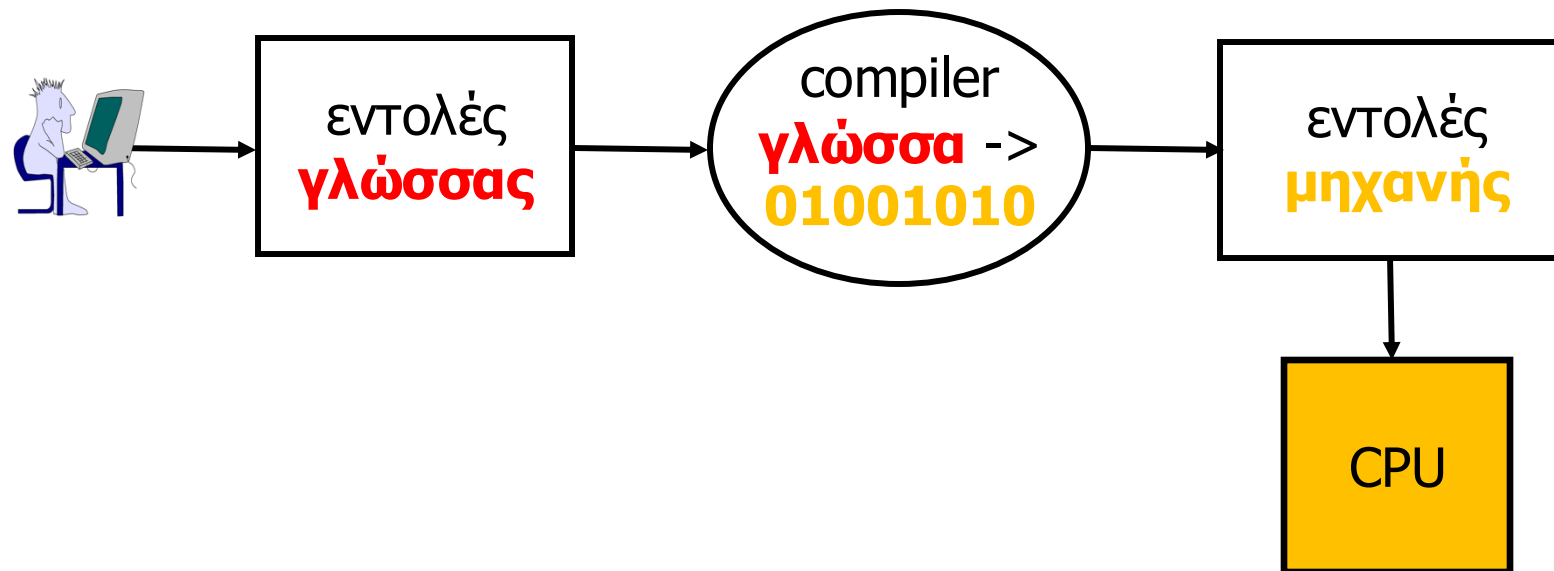
- Όμως, ποια είναι η σημασία των παραπάνω;

$$-2+13 : \quad (-2) + (13) \quad \text{ή} \quad -(2+13) \quad ;$$

$$7-15/5 : \quad (7-15) / 5 \quad \text{ή} \quad 7 - (15/5) \quad ;$$

Μεταγλώττιση (compilation)

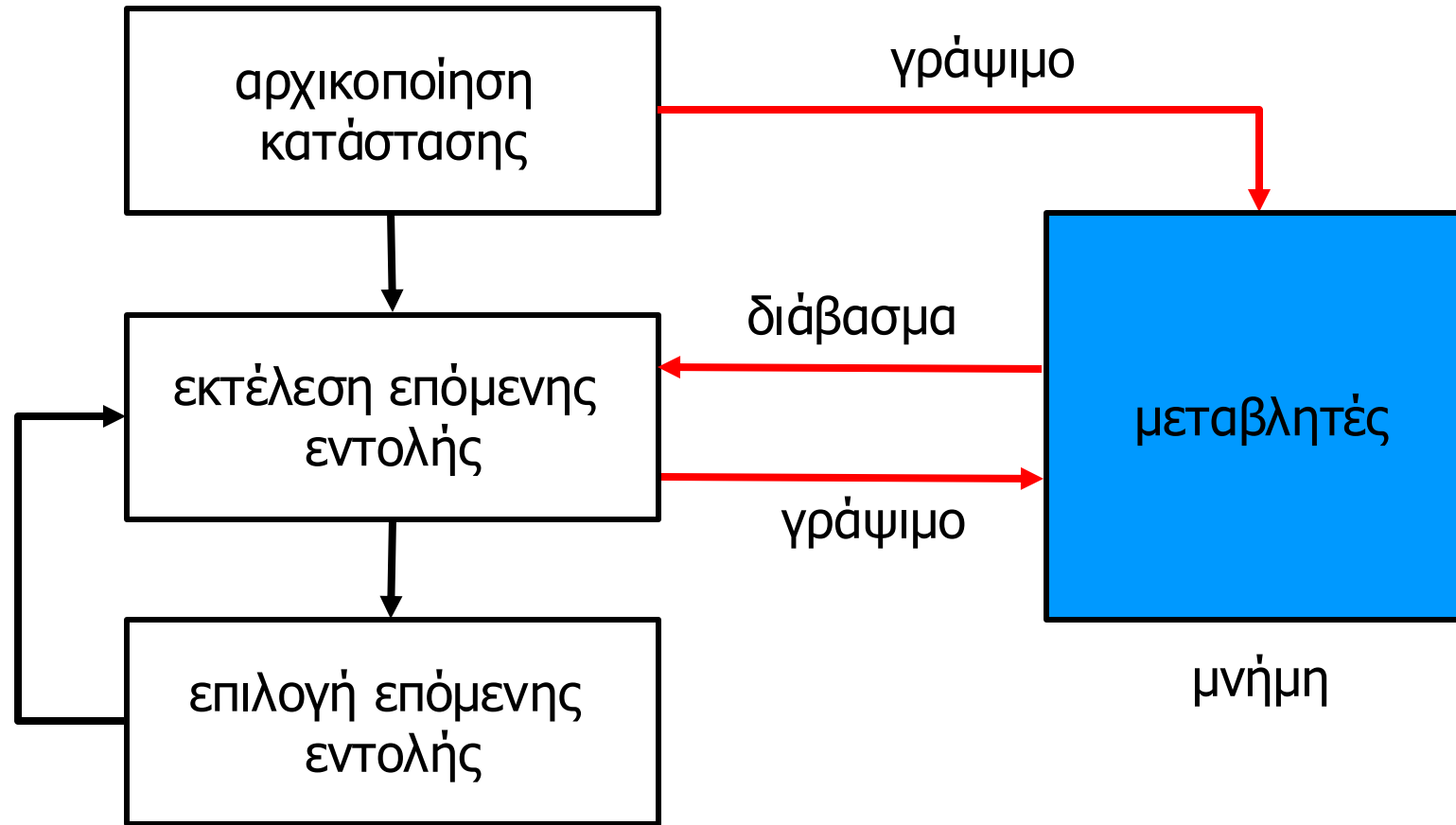
- **Κώδικας:** μια σειρά εντολών προς τον υπολογιστή, γραμμένες με βάση μια γλώσσα προγραμματισμού
- **Μεταγλώττιση:** διαδικασία μετατροπής κώδικα από μια γλώσσα σε μια άλλη
 - συνήθως από εντολές υψηλού επιπέδου σε εντολές μηχανής
 - μια εντολή γλώσσας υψηλού επιπέδου μπορεί να αντιστοιχεί σε μια σειρά εντολών γλώσσας μηχανής
- **Γίνεται από ειδικά προγράμματα, τους λεγόμενους μεταγλωττιστές (compilers)**
- Η μεταγλώττιση γίνεται (συνήθως) ως ξεχωριστή διαδικασία, πολύ πριν αρχίσει η εκτέλεση του κώδικα
 - υπάρχουν περιπτώσεις όπου η μετάφραση γίνεται ακριβώς πριν αρχίσει η εκτέλεση (just in time compilation)



Εκτέλεση κώδικα

- Κάθε γλώσσα έχει συγκεκριμένες **συμβάσεις** για την αποτίμηση και εκτέλεση των εκφράσεων/προτάσεων
- Συνήθως **«από αριστερά προς τα δεξιά»** και **«από πάνω προς τα κάτω»**
 - όπως γράφουμε/διαβάζουμε κείμενα (στον δυτικό κόσμο)
- Μπορεί να γίνουν **άλματα** μέσα στον κώδικα
 - είτε «προς τα εμπρός» είτε «προς τα πίσω»
 - παράκαμψη ή επανάληψη εκτέλεσης κομματιών κώδικα
- Εντολές **ελέγχου ροής εκτέλεσης** προγράμματος
 - διαβάζουν την κατάσταση του προγράμματος και ανάλογα μεταφέρουν την εκτέλεση σε ένα προκαθορισμένο σημείο

Εκτέλεση προγράμματος



μεταβλητή

παράμετροι

πρόγραμμα P (x, y)

```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

μεταβλητή

παράμετροι

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(0, 5)$

x ? y ? s ?
program counter ?

→ πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(0, 5)$

x	0	y	5	s	?
program counter				?	

εκτέλεση $P(0, 5)$

x 0 y 5 s ?
program counter 1

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(0, 5)$

x 0 y 5 s ?

program counter 1

s ← 0

πρόγραμμα $P(x, y)$

→ 1: $s \leftarrow 0$
2: if (x is 0) goto 6
3: $s \leftarrow s + y$
4: $x \leftarrow x - 1$
5: goto 2
6: print s

εκτέλεση $P(0, 5)$

x 0 y 5 s 0

program counter 1

$s \leftarrow 0$

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(0, 5)$

x 0 y 5 s 0
program counter 2

s <- 0

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(0, 5)$

x 0 y 5 s 0
program counter 2

```
s <- 0
if (x is 0) goto 6
```


πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(0, 5)$

x 0 y 5 s 0
program counter 2

```
s <- 0
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```



εκτέλεση $P(0, 5)$

x 0 y 5 s 0
program counter 6

```
s ← 0
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```



εκτέλεση $P(0, 5)$

x 0 y 5 s 0
program counter 6

```
s ← 0
if (x is 0) goto 6
print s
```

0

εκτέλεση $P(2, 5)$

x ? y ? s ?
program counter ?

πρόγραμμα $P(x, y)$

```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s ?
program counter ?

→ πρόγραμμα $P(x, y)$

```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s ?
program counter 1

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s ?

program counter 1

s <- 0

πρόγραμμα $P(x, y)$

→ 1: $s \leftarrow 0$
2: if (x is 0) goto 6
3: $s \leftarrow s + y$
4: $x \leftarrow x - 1$
5: goto 2
6: print s

εκτέλεση $P(2, 5)$

x 2 y 5 s 0

program counter 1

s $\leftarrow 0$

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 0
program counter 2

s ← 0

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 0

program counter 2

```
s <- 0
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 0

program counter 2

```
s <- 0
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 0
program counter 3

```
s ← 0
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 0
program counter 3

```
s ← 0
if (x is 0) goto 6
s ← s + y
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 0
program counter 3

```
s ← 0
if (x is 0) goto 6
s ← s + y
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 5
program counter 3

```
s ← 0
if (x is 0) goto 6
s ← s + y
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 5
program counter 4

```
s ← 0
if (x is 0) goto 6
s ← s + y
```


πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 5
program counter 4

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 2 y 5 s 5
program counter 4

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 4

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
```

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```



εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 5

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
```

πρόγραμμα $P(x, y)$

```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```



εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 5

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 2

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 2

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 2

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
```


πρόγραμμα P (x, y)



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση P (2, 5)

x 1 y 5 s 5
program counter 3

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 5
program counter 3

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
s ← s + y
```

πρόγραμμα P(x, y)



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση P(2, 5)

x 1 y 5 s 5
program counter 3

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 10
program counter 3

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
s ← s + y
```

πρόγραμμα $P(x, y)$



```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 1 y 5 s 10
program counter 4

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
s ← s + y
```

πρόγραμμα P(x, y)



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση P(2, 5)

x 1 y 5 s 10
program counter 4

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
```

πρόγραμμα P(x, y)



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση P(2, 5)

x 1 y 5 s 10
program counter 4

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 4

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
```


πρόγραμμα $P(x, y)$

```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```



εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 5

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
```

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```



εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 5

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 2

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 2

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$



```
1: s <- 0
2: if (x is 0) goto 6
3: s <- s + y
4: x <- x - 1
5: goto 2
6: print s
```

εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 2

```
s <- 0
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
s <- s + y
x <- x - 1
goto 2
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```



εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 6

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
```

πρόγραμμα $P(x, y)$

```
1: s ← 0
2: if (x is 0) goto 6
3: s ← s + y
4: x ← x - 1
5: goto 2
6: print s
```



εκτέλεση $P(2, 5)$

x 0 y 5 s 10
program counter 6

```
s ← 0
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
s ← s + y
x ← x - 1
goto 2
if (x is 0) goto 6
print s
```

10

Υποπρογράμματα

- Ομάδα εντολών (και δεδομένων) που είναι **συντακτικά** ή/και **εκτελεστικά** ανεξάρτητη (αυτόνομη) από τον υπόλοιπο κώδικα
 - ένα πρόγραμμα μπορεί να αποτελείται από (ή να χρησιμοποιεί) πολλά διαφορετικά υποπρογράμματα
- Όταν καλείται ένα υποπρόγραμμα, η εκτέλεση **μεταφέρεται** στο υποπρόγραμμα
- Όταν τερματιστεί η εκτέλεση του, **επιστρέφει** πίσω στο πρόγραμμα που πραγματοποίησε την κλήση
- Η χρήση υποπρογραμμάτων αποτελεί μια από τις βασικές αρχές **δομημένου προγραμματισμού**
 - πολύ σημαντικό στην πράξη

πρόγραμμα P1

```
code A
call P2
code B
```

πρόγραμμα P2

```
call P3
code C
return
```

πρόγραμμα P3

```
code D
return
```

εκτέλεση P1

```
code A
code D
code C
code B
```

Μεθοδολογία επίλυσης προβλημάτων

- **Ανάλυση** του προβλήματος
 - τι δεδομένα έχουμε στην διάθεση μας;
 - τι θέλουμε να βρούμε;
 - πώς σπάει σε μικρότερα / απλούστερα προβλήματα;
 - ποια στρατηγική πρέπει να ακολουθήσουμε;
- Σχεδιασμός **αλγορίθμου/μεθόδου**
 - ακριβής ακολουθία πεπερασμένων βημάτων που πρέπει να ακολουθηθούν για να λυθεί το πρόβλημα
- Απαιτήσεις
 - ορθότητα, καλή επίδοση, εξοικονόμηση πόρων
- Για το ίδιο πρόβλημα μπορεί να υπάρχουν περισσότεροι (κατάλληλοι) αλγόριθμοι

Παράδειγμα: γραμμική αναζήτηση

Βρες τη θέση που υπάρχει (αν υπάρχει) η τιμή 50

1ο βήμα

1	13	17	20	21	34	45	46	47	50	55	59	61	63	70
---	----	----	----	----	----	----	----	----	----	----	----	----	----	----



2ο βήμα

1	13	17	20	21	34	45	46	47	50	55	59	61	63	70
---	----	----	----	----	----	----	----	----	----	----	----	----	----	----



3ο βήμα

1	13	17	20	21	34	45	46	47	50	55	59	61	63	70
---	----	----	----	----	----	----	----	----	----	----	----	----	----	----



...

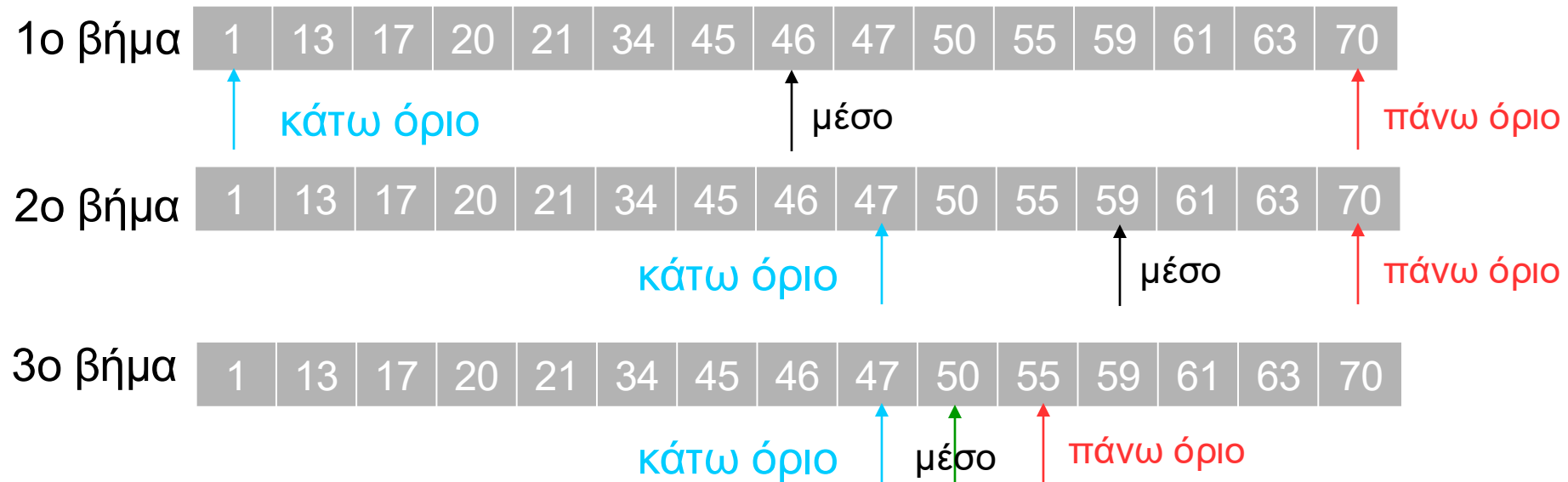
10ο βήμα

1	13	17	20	21	34	45	46	47	50	55	59	61	63	70
---	----	----	----	----	----	----	----	----	----	----	----	----	----	----



Παράδειγμα: δυαδική αναζήτηση

Βρες τη θέση που υπάρχει (αν υπάρχει) η τιμή 50



Ορθότητα (correctness)

- Ο κώδικας υλοποιεί την επιθυμητή (προδιαγεγραμμένη) λειτουργικότητα
 - για όλες τις περιπτώσεις
- Επιτυχής μετάφραση **δεν** συνεπάγεται ορθότητα
- Εκτέλεση χωρίς σφάλματα **δεν** συνεπάγεται ορθότητα
- Ακόμα και όταν το πρόγραμμα «φαίνεται» να τρέχει κανονικά, μπορεί να μην λειτουργεί (εντελώς) σωστά
 - ίσως δεν κάνει αυτό που θέλουμε, σε όλες τις περιπτώσεις
 - ίσως υπάρχουν σημασιολογικά ή/και λογικά λάθη ...
 - ... που είναι δύσκολο να εντοπιστούν

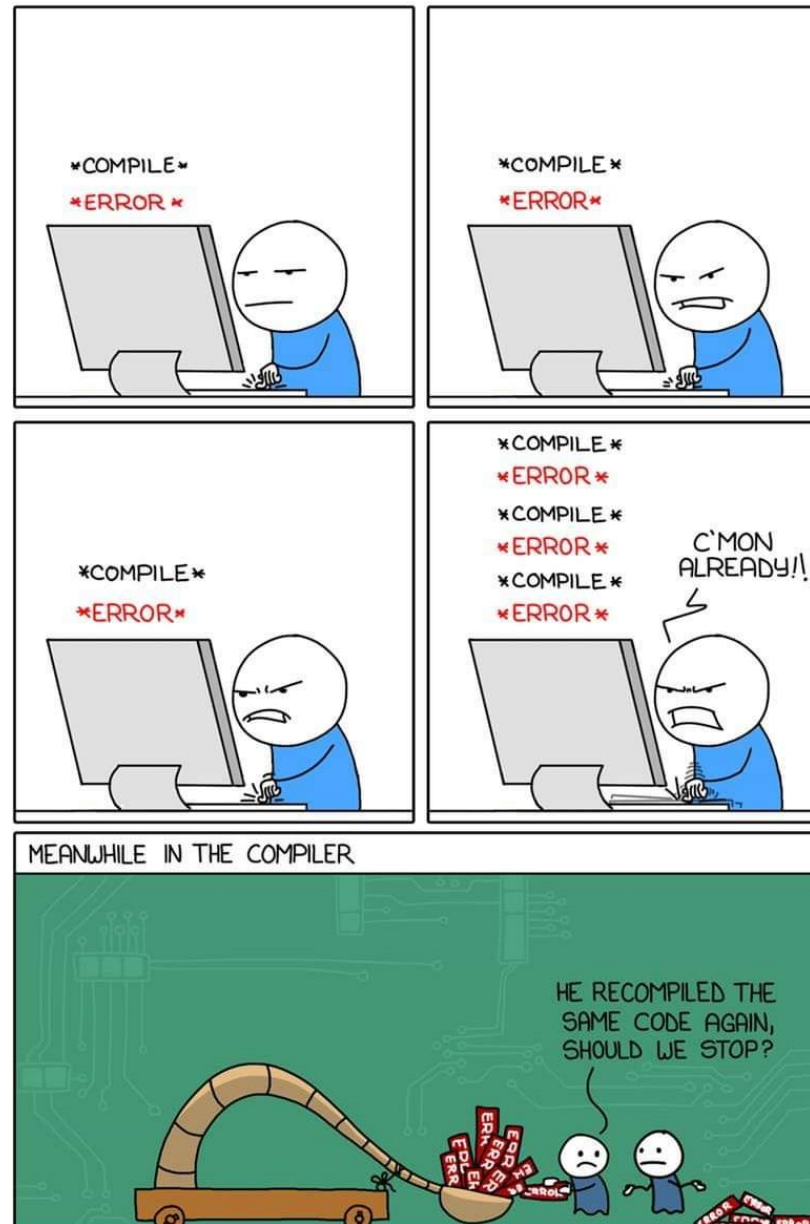
Αποσφαλμάτωση (debugging)

- **Συντακτικό** επίπεδο (εύκολο)
 - ο κώδικας δεν αντιστοιχεί σε συντακτικά επιτρεπτή πρόταση
 - ευκολάκι, να' ναι καλά αυτός που έγραψε τον compiler
- **Σημασιολογικό** επίπεδο (δύσκολο)
 - ο κώδικας είναι συντακτικά σωστός και εκτελείται (τρέχει) αλλά εμείς δεν χρησιμοποιούμε σωστά κάποια εντολή
 - άλλο θέλουμε να κάνουμε και άλλο ζητάμε από τον υπολογιστή να κάνει: παράγεται λάθος αποτέλεσμα
 - επίπονο, αλλά η κατάσταση βελτιώνεται με την εμπειρία
- **Λογικό** επίπεδο (πιο δύσκολο)
 - ο κώδικας είναι συντακτικά και σημασιολογικά σωστός
 - υπάρχει λάθος σε επίπεδο αλγορίθμου/μεθόδου (**σκέψης**)
 - το πιο δύσκολο, ακόμα και για πολύ έμπειρους

Εμφάνιση λαθών/σφαλμάτων

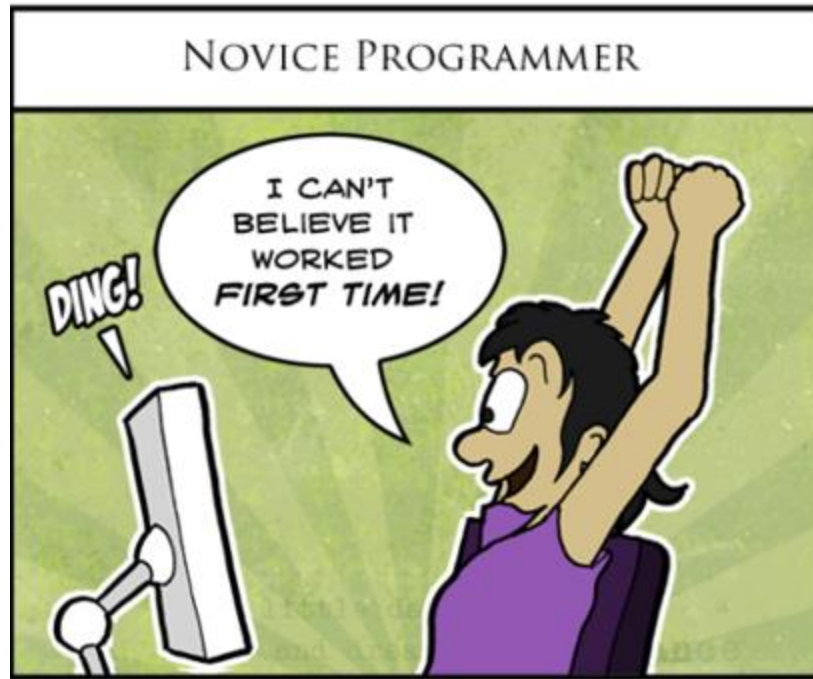
- **Χρόνος μετάφρασης** (compile time)
 - ο εντοπισμός γίνεται από τον μεταγλωττιστή, αναλόγως με την γλώσσα προγραμματισμού
 - π.χ. συντακτικά λάθη, λάθη ανάθεσης τιμών, ...
- **Χρόνος εκτέλεσης** (run time / execution time)
 - ο εντοπισμός γίνεται από το περιβάλλον εκτέλεσης (λειτουργικό ή ακόμα και τον ίδιο τον επεξεργαστή)
 - π.χ. προσπέλαση άκυρων διευθύνσεων, αριθμητικά λάθη, ...
 - συνήθως προκαλείται τερματισμός της εκτέλεσης
- **Μετά την εκτέλεση**
 - ο εντοπισμός γίνεται από τον χρήστη ή ένα άλλο πρόγραμμα
 - με βάση τα αποτελέσματα που παράγει το πρόγραμμα, χωρίς να έχει παρουσιάσει κάποιο «ορατό» σφάλμα

Για όλα φταίει αυτός ...



... ο compiler

Ο νέος (ψάρι, γατάκι)



Ο παλιός

