ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΙΚΑ ΜΑΘΗΜΑΤΑ

# ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ(Θ)

## Ενότητα 1: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ


ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ



Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

# Χρηματοδότηση


- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.


# Ενότητα 1

## ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

**ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ**

# Περιεχόμενα ενότητας

# Σκοποί ενότητας

# Ιστορικό των γλωσσών C και C++

## Ιστορικό της C

- Εξέλιξη των γλωσσών BCPL και B
- Dennis Ritchie (Εργαστήρια Bell)
- Γλώσσα ανάπτυξης του Λ.Σ. UNIX
- Ανεξάρτητη του λειτουρ-γικού στο οποίο εκτελού-νται τα προγράμματά της
  - Φορητά προγράμματα
- 1989: Πρότυπο ANSI
- 1990: Πρότυπα ANSI και ISO
  - ✓ANSI/ISO 9899: 1990
  - ✓Τελευταία πρότυπα ISO: C1999 και C2011

| Sep 2014 | Sep 2013 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | C | 16.721% | -0.25% |
| 2 | 2 | | Java | 14.140% | -2.01% |
| 3 | 4 | ⌃ | Objective-C | 9.935% | +1.37% |
| 4 | 3 | ⌄ | C++ | 4.674% | -3.99% |
| 5 | 6 | ⌃ | C# | 4.352% | -1.21% |
| 6 | 7 | ⌃ | Basic | 3.547% | -1.29% |
| 7 | 5 | ⌄ | PHP | 3.121% | -3.31% |
| 8 | 8 | | Python | 2.782% | -0.39% |
| 9 | 9 | | JavaScript | 2.448% | +0.43% |
| 10 | 10 | | Transact-SQL | 1.675% | -0.32% |
| 11 | 11 | | Visual Basic .NET | 1.532% | -0.31% |
| 12 | 12 | | Perl | 1.369% | -0.32% |
| 13 | 13 | | Ruby | 1.281% | -0.10% |
| 14 | - | ⌃⌃ | Visual Basic | 1.272% | +1.27% |
| 15 | 14 | ⌄ | Delphi/Object Pascal | 1.157% | +0.26% |
| 16 | 26 | ⌃⌃ | F# | 0.990% | +0.49% |
| 17 | 15 | ⌄ | Pascal | 0.893% | +0.01% |
| 18 | - | ⌃⌃ | Swift | 0.852% | +0.85% |
| 19 | 19 | | MATLAB | 0.818% | +0.18% |
| 20 | 17 | ⌄ | PL/SQL | 0.809% | +0.13% |

# Ιστορικό των γλωσσών C και C++

Ιστορικό της C++

- Επέκταση της C
- 1983: Bjarne Stroustrup (Εργαστήρια Bell)
- "Περιποιημένη" C
- Παρέχει τη δυνατότητες αντικειμενοστραφούς προγραμματισμού
  - Αντικείμενα: επαναχρησιμοποιήσιμα συστατικά λογισμικού
  - Αντικειμενοστραφή προγράμματα
    - Εύκολα στην κατανόηση, διόρθωση και τροποποίηση
- Υβριδική γλώσσα
  - Περιέχει στοιχεία και φορμαλισμό από τη C
  - Έχει την τεχνοτροπία του αντικειμενοστραφούς προγραμματισμού

# Ιστορικό των γλωσσών C και C++

•Η C++ προέρχεται από τη γλώσσα C. Για την ακρίβεια είναι ένα υπερσύνολο της C. Κάθε σωστή πρόταση της C, είναι και πρόταση της C++. Τα επιπλέον στοιχεία που προστέθηκαν στην C για να προκύψει η C++, είναι οι κλάσεις και τα αντικείμενα και γενικά ο αντικειμενοστραφής προγραμματισμός. Επιπρόσθετα, η C++ έχει πολλά νέα χαρακτηριστικά που περιλαμβάνουν, κυρίως, μία βελτιωμένη προσέγγιση της εισόδου/εξόδου δεδομένων.

•Πρότυπα C++:

ANSI/ISO: 1998

ISO: 2011

# C++ Πρότυπη βιβλιοθήκη

Ο αντικειμενοστραφής προγραμματισμός παριστάνει ένα σύστημα ως μία συλλογή από αντικείμενα τα οποία αλληλεπιδρούν και αλλάζουν με το χρόνο.

Ένα αντικειμενοστραφές πρόγραμμα αποτελείται από κλάσεις και αντικείμενα.

Τα αντικείμενα έχουν ιδιότητες (χαρακτηριστικά) και ενέργειες (συμπεριφορά) που συνδέονται μ'αυτά.

Η πρότυπη βιβλιοθήκη περιέχει μία ευρύτατη συλλογή από κλάσεις και συναρτήσεις.

# Java

- 1991: Sun Microsystems
  «Πράσινο πρόγραμμα» - Green project (Patrick Naughton, James Gosling, Mike Sheridan).
  Στόχος ο προγραμματισμός μικροσυσκευών, χωρίς την πολυπλοκότητα της C++.

- 1995: Sun Microsystems
  Επίσημη ονοματοδοσία της νέας γλώσσας ως Java

- Ιστοσελίδες με δυναμικό και διαδραστικό περιεχόμενο

- Ανάπτυξη εφαρμογών μεγάλης κλίμακας

- Διευρυμένη λειτουργικότητα web servers

- Εφαρμογές για καταναλωτικές συσκευές
  Κινητά τηλέφωνα, βομβητές, PDA, κ.λ.π.

# Λόγοι εκμάθησης C++

- Γλώσσα αντικειμενοστραφούς προγραμματισμού υψηλού επιπέδου, με στοιχεία από διαδικαστικές γλώσσες (C, Pascal κ.λ.π.)
- Απαιτούμενο προσόν για πολλές θέσεις εργασίας.
- Παρέχεται πλήρης έλεγχος στον προγραμματιστή:
  - Υποκαθιστά σε πολλές περιπτώσεις τη χρήση γλώσσας μηχανής (όπως και η γλώσσα C).
  - Επιτυγχάνεται ακριβής έλεγχος παντού, κυρίως δε στη διαχείριση της μνήμης.
- Υψηλή ταχύτητα εκτέλεσης.
- Μειονεκτήματα:
  - Περίπλοκη γλώσσα
  - Απαιτεί προσοχή και σχολαστικότητα στη συγγραφή κώδικα, καθώς και ιδιαίτερη εξάσκηση

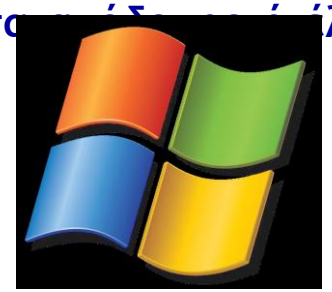
# C++ vs Java

- Στο πρόγραμμα σπουδών προβλέπεται και η εκμάθηση της Java (στο μάθημα Μεθοδολογία Προγραμματισμού του 6ου εξαμήνου).

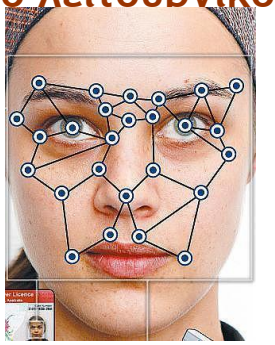
- Η κάθε γλώσσα προορίζεται για άλλες εφαρμογές και διαφορετικές απαιτήσεις. Παρόλο που η Java μοιάζει με τη C++, πρόκειται για διαφορετικές γλώσσες. Π.χ:
  - Η C++ επιτρέπει συναρτήσεις που δεν είναι μέθοδοι τάξεων.
  - Η C++ επιτρέπει τη χρήση δεικτών και δεν παρέχει αυτόματη αποκομιδή απορριμμάτων (garbage collection). Βέβαια το πρότυπο C++11 καθορίζει μηχανισμούς αποκομιδής απορριμάτων, αλλά η υποστήριξή τους από τους μεταγλωττιστές είναι προαιρετική.
  - Ο κάθε μεταγλωττιστής της C++ παράγει εκτελέσιμο κώδικα για συγκεκριμένο επεξεργαστή.

# C++ vs Java

**Σε πολλές εφαρμογές που απαιτείται ταχύτητα και μεγάλη προβλεψιμότητα μέσω αυστηρού έλεγχος, η Java δε μπορεί να ανταπεξέλθει:**



- **Προγραμματισμός σε επίπεδο λειτουργικού συστήματος**



- **Επιστημονικοί υπολογισμοί**


- **Μαζικά δεδομένα (π.χ. συστήματα διαχείρισης δεδομένων)**

- **Αλληλεπιδραστικά συστήματα (π.χ. γραφικά Η/Υ, GUIs)**

# C++ vs Java

**Προτιμάται η C++ όταν η ταχύτητα του προγράμματος είναι κρίσιμος παράγοντας:**

- **Παράγεται κώδικας μηχανής από την αρχή, όχι byte code και Just in-time (JIT) παραγωγή κώδικα**
- **Ολική βελτιστοποίηση του κώδικα από τον compiler, κάτι που δεν μπορεί να γίνει στη Java.**
- **Δε μεσολαβεί κάποια Virtual Machine (π.χ. JVM) που σπαταλά πόρους για τη δική της λειτουργία.**
- **Μπορούν να χρησιμοποιηθούν βελτιστοποιημένες υλοποιήσεις και ειδικά CPU Instruction Sets (π.χ. SSE4).**
- **Ο προγραμματιστής μπορεί να παρέμβει σε πολύ χαμηλό επίπεδο στη βελτιστοποίηση της απόδοσης.**

# C++ vs Java

**Προτιμάται η C++ όταν η κατανάλωση μνήμης του προγράμματος είναι κρίσιμος παράγοντας:**

- **Δεν παρέχεται άυτόματη αποκομιδή απορριμάτων. Ο προγραμματιστής αποφασίζει πώς και πότε θα αποδεσμεύσει μνήμη.**
- **Γίνεται ακριβής έλεγχος των δεδομένων που αποθηκεύονται.**
- **Παράγεται ιδιαίτερα συμπαγής κώδικας.**
- **Μπορούν να υλοποιηθούν δικοί μας μηχανισμοί διαχείρισης μνήμης (caches, memory pooling, lazy allocators).**

**Επίσης προτιμάται η C++ όταν απαιτείται χαμηλή κατανάλωση ενέργειας:**

- **Δε μεσολαβεί Virtual Machine. Η VM είναι και η ίδια ένα εκτελέσιμο πρόγραμμα, οπότε όταν εκτελείται ο κώδικά, χρησιμοποιούνται παράλληλα επιπρόσθετοι πόροι του συστήματος (κυρίως κύκλους CPU) που καταναλώνουν ενέργεια.**
- **Είναι σημαντικός παράγοντας σε αρχιτεκτονικές κινητών συσκευών (κινητά τηλάφωνα, tablets, ενσωματωμένα συστήματα κ.λ.π.).**

# Δομημένος Προγραμματισμός

**Δομημένος Προγραμματισμός (δεκαετία του 1960)**

- **Συστηματική προσέγγιση στη συγγραφή προγραμμάτων**
- **Προγράμματα εύκολα στην κατανόηση, τον έλεγχο, την αποσφαλμάτωση και την τροποποίηση**

**Pascal**

- **1971: Niklaus Wirth**

**Ada**

- **Δεκαετία του 1970s – αρχές της δεκαετίας του 1980s: ΥΠ.ΕΘ.Α. Η.Π.Α. (Department of Defense - DoD)**
- **Πολυεπεξεργασία (Multitasking):**
  - **Καθορισμός πολλαπλών δραστηριοτήτων που εκτελούνται παράλληλα**

# Τι είναι ο αντικειμενοστραφής προγραμματισμός;

•Οι διαδικαστικές γλώσσες δίνουν έμφαση στις ενέργειες που πρέπει να εκτελέσει ένα πρόγραμμα.

•Ο αντικειμενοστραφής προγραμματισμός αναλύει πολύπλοκα συστήματα εξετάζοντας τα χαρακτηριστικά των στοιχείων τους και τον τρόπο που αυτά αλληλεπιδρούν.

Ο αντικειμενοστραφής προγραμματισμός γεννήθηκε και άρχισε να αναπτύσσεται όταν πλέον ήταν φανερό ότι:

•Οι παραδοσιακές προσεγγίσεις στον προγραμματισμό δεν μπορούσαν να ανταποκριθούν στις νέες απαιτήσεις ανάπτυξης προγραμμάτων.

•Καθώς τα προγράμματα μεγάλωναν, γίνονταν υπερβολικά πολύπλοκα και η χρήση διαδικαστικών γλωσσών προγραμματισμού (PASCAL, C, κλπ.) παρουσίαζε αδυναμίες.

# Γιατί είναι διαφορετικός;

Ο προγραμματισμός παραδοσιακά ήταν προσανατολισμένος στο έργο που θα έπρεπε να παραχθεί.

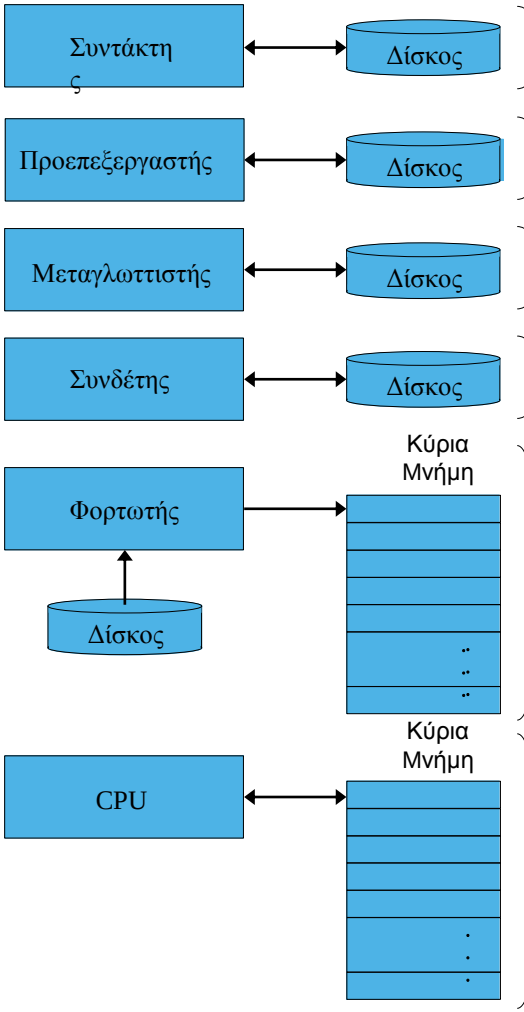
Ένα έργο διαιρείτο σε υποέργα και αυτά διαιρούνταν σε μικρότερα υποέργα, έως ότου οι ορισμοί τους να είναι τόσο απλοί ώστε να μπορούν να μετατραπούν σε κώδικα.

Η αντικειμενοστραφής ανάλυση είναι πολύ πιο κοντά στο πώς εμείς βλέπουμε τον κόσμο γύρω μας.

# Τυπικό περιβάλλον ανάπτυξης της C++

## Στάδια των προγραμμάτων C++:

1. Σύνταξη
2. Προεπεξεργασία
3. Μεταγλώττιση
4. Σύνδεση
5. Φόρτωση
6. Εκτέλεση




Το πρόγραμμα συντάσσεται στο συντάκτη και αποθηκεύεται στο δίσκο.

Ο προεπεξεργαστής επεξεργάζεται τον κώδικα.

Ο μεταγλωττιστής δημιουργεί τον αντικείμενο κώδικα και τον αποθηκεύει στο δίσκο.

Ο συνδέτης διασυνδέει τον αντικείμενο κώδικα με τις βιβλιοθήκες, δημιουργεί το εκτελέσιμο αρχείο και το αποθηκεύει στο δίσκο.

Ο φορτωτής φορτώνει το πρόγραμμα στη μνήμη.

Η CPU παίρνει κάθε οδηγία και την εκτελεί, αποθηκεύοντας πιθανώς νέες τιμές δεδομένων κατά την εκτέλεση του προγράμματος.

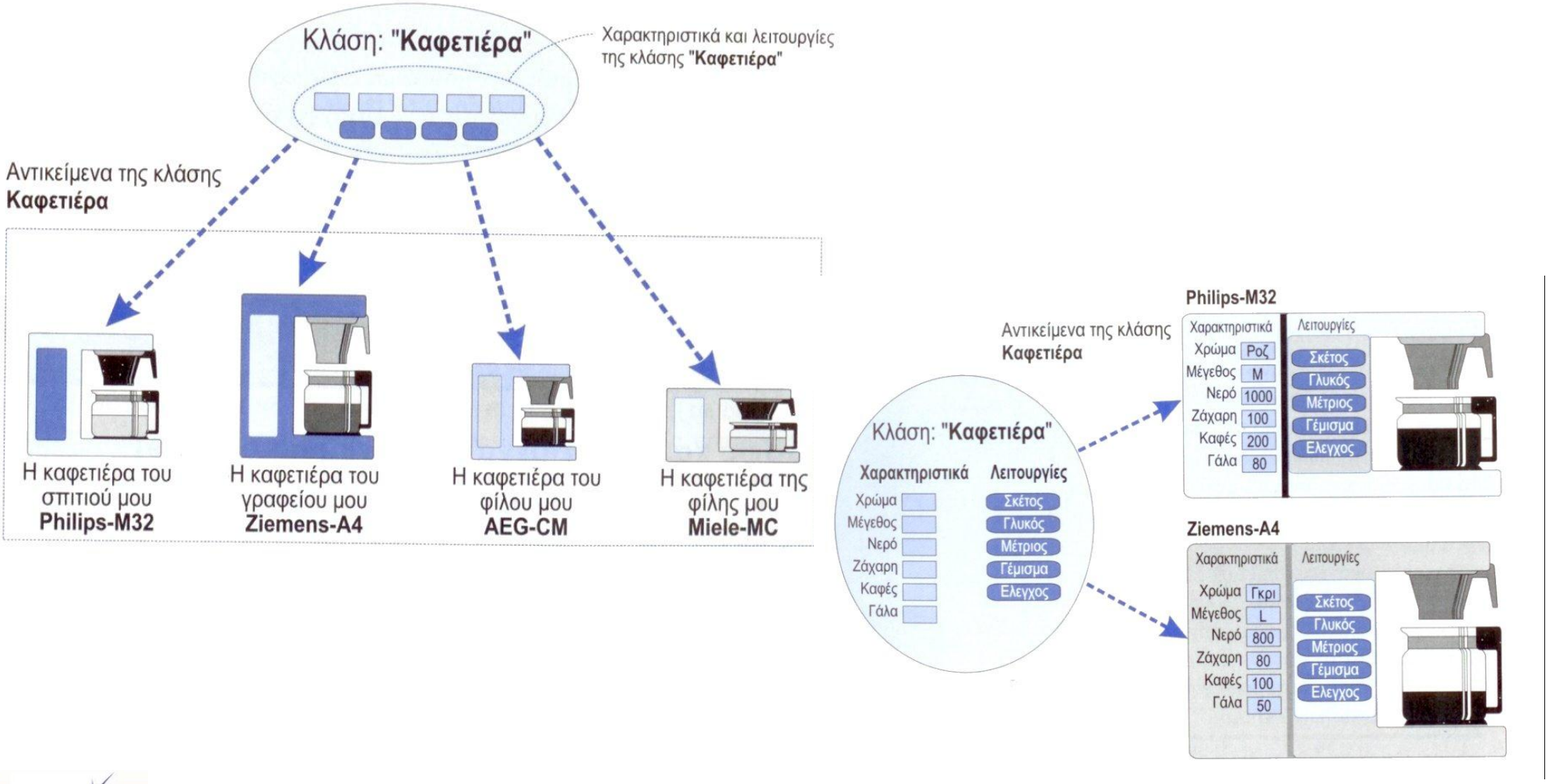
# Ορολογία

## Κλάση (Class)

Μία κλάση είναι μία συλλογή από αντικείμενα, που μοιράζονται τα ίδια χαρακτηριστικά και επιδρούν με το σύστημα με τον ίδιο τρόπο. Τα χαρακτηριστικά και η επίδραση ορίζονται για τις κλάσεις.

## Αντικείμενο (Object)

Ένα αντικείμενο είναι ένα μέλος μίας κλάσης. Αν και ο ορισμός γίνεται σε επίπεδο κλάσης, η πραγματική επίδραση συμβαίνει με ανεξάρτητα αντικείμενα.


Κλάση: "Καφετιέρα"
Χαρακτηριστικά και λειτουργίες της κλάσης "Καφετιέρα"
Αντικείμενα της κλάσης Καφετιέρα
Η καφετιέρα του σπιτιού μου Philips-M32
Η καφετιέρα του γραφείου μου Ziemens-A4
Η καφετιέρα του φίλου μου AEG-CM
Η καφετιέρα της φίλης μου Miele-MC
Αντικείμενα της κλάσης Καφετιέρα
Κλάση: "Καφετιέρα"
Χαρακτηριστικά
Λειτουργίες
Χρώμα
Μέγεθος
Νερό
Ζάχαρη
Καφές
Γάλα
Σκέτος
Γλυκός
Μέτριος
Γέμισμα
Ελεγχος
Philips-M32
Χαρακτηριστικά
Λειτουργίες
Χρώμα Ροζ
Μέγεθος M
Νερό 1000
Ζάχαρη 100
Καφές 200
Γάλα 80
Σκέτος
Γλυκός
Μέτριος
Γέμισμα
Ελεγχος
Ziemens-A4
Χαρακτηριστικά
Λειτουργίες
Χρώμα Γκρι
Μέγεθος L
Νερό 800
Ζάχαρη 80
Καφές 100
Γάλα 50
Σκέτος
Γλυκός
Μέτριος
Γέμισμα
Ελεγχος

# Ορολογία (συνέχεια)

**Συναρτήσεις (Functions)**

**Στον προγραματισμό ο κώδικας για μία συγκεκριμένη ενέργεια ονομάζεται συνάρτηση.**

**Στον αντικειμενοστραφή προγραμματισμό την αλληλεπίδραση μεταξύ αντικειμένων ή αντικειμένων και του έξω κόσμου, την χειρίζονται οι συναρτήσεις. Αυτό ονομάζεται συμπεριφορά.**

**Μία αντικειμενοστραφής συνάρτηση είναι παρόμοια με μία συνηθισμένη συνάρτηση αλλά γενικά οι αντικειμενοστραφείς συναρτήσεις τείνουν να είναι μικρότερες και απλούστερες.**

# Ορολογία (συνέχεια)

**Ενθυλάκωση (Encapsulation)**

- Αυτή ορίζει το βαθμό προσπελασιμότητας που μία κλάση επιτρέπει σε άλλες κλάσεις. *Δίνει τη δυνατότητα στις γλώσσες αντικειμενοστραφούς προγραμματισμού να ομαδοποιούν και να αποκρύπτουν δεδομένα και διαδικασίες των αντικειμένων.*
- Κάποιες πληροφορίες είναι προσπελάσιμες μόνο μέσα σε μία κλάση.
- Κάποιες πληροφορίες είναι προσπελάσιμες έξω από την κλάση αλλά μόνο σε συγγενείς κλάσεις.
- Κάποιες πληροφορίες είναι προσπελάσιμες γενικά.
- Ο καλός αντικειμενοστραφής προγραμματισμός κρατά τις πληροφορίες όσο πιο ιδιωτικές γίνεται.

**Philips-M32**


Υπάρχει πρόσβαση σε όλα τα χαρακτηριστικά και τις λειτουργίες του αντικειμένου

**Philips-M32**

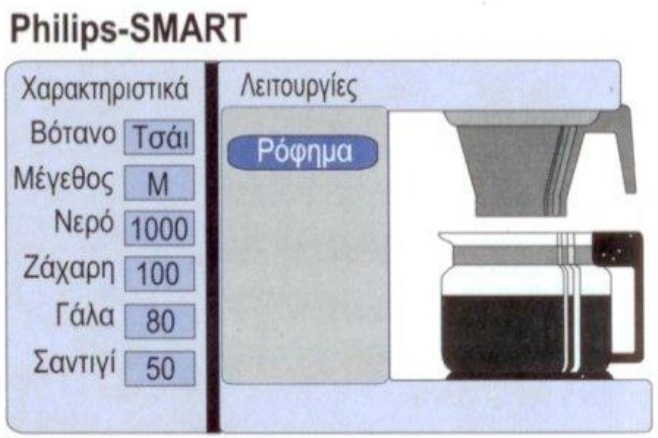
# Ορολογία (συνέχεια)

**Πολυμορφισμός (Polymorphism)**

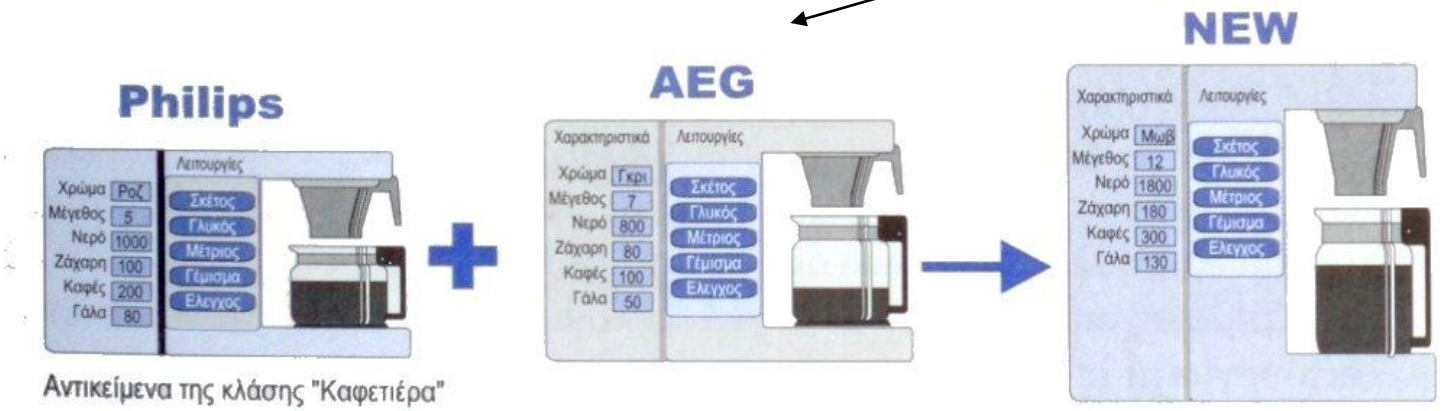
- **Αυτός αναφέρεται στην αλληλεπίδραση μεταξύ αντικειμένων. Τα αντικείμενα σχετίζονται μέσα στο σύστημα και με τον έξω κόσμο με τις ενέργειες.**

- **Η ίδια ενέργεια μπορεί να παρασταθεί με μία σειρά από διαφορετικούς τρόπους.**

- ***Ο πολυμορφισμός είναι χαρακτηριστικό των ενεργειών, όχι των αντικειμένων. Είναι η δυνατότητα που παρέχουν οι αντικειμενοστραφείς γλώσσες προγραμματισμού στα αντικείμενα να συμπεριφέρονται διαφορετικά, ανάλογα με τον τρόπο με τον οποίο χρησιμοποιούνται.***




**Υπερφόρτωση του τελεστή +: Η υπερφόρτωση είναι χαρακτηριστικό του πολυμορφισμού.**



Αντικείμενα της κλάσης "Καφετιέρα"

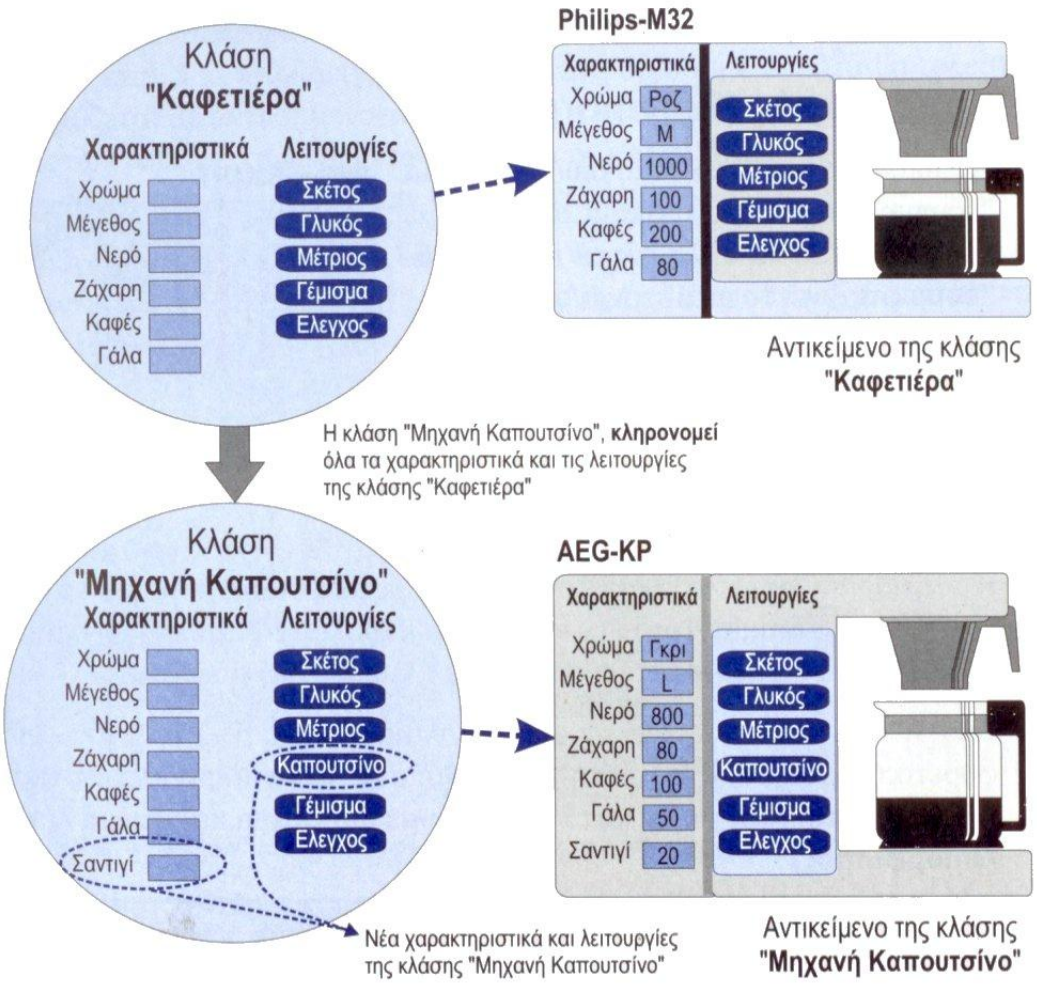
# Ορολογία (συνέχεια)

**Κληρονομικότητα (Inheritance)**

- **Είμαστε εξοικειωμένοι με το πώς κάποια χαρακτηριστικά περνούν από τους γονείς στα παιδιά.**

- **Στον αντικειμενοστραφή προγραμματισμό οι κλάσεις μπορούν να ομαδοποιηθούν σε ιεραρχίες, όπου τα κατώτερα επίπεδα της ιεραρχίας (τα παιδιά) μοιράζονται τα ίδια χαρακτηριστικά με τα ανώτερα επίπεδα (οι γονείς).**

- ***Η κληρονομικότητα είναι η δυνατότητα παραγωγής μίας νέας κλάσης από μία υπάρχουσα (βασική). Η νέα κλάση (παράγωγη) κληρονομεί όλα τα χαρακτηριστικά και τις λειτουργίες της βασικής κλάσης, αλλά ταυτόχρονα μπορεί να ορίσει τα δικά της επιπλέον χαρακτηριστικά και λειτουργίες.***


Κλάση
"Καφετιέρα"
Χαρακτηριστικά
Λειτουργίες
Χρώμα
Μέγεθος
Νερό
Ζάχαρη
Καφές
Γάλα
Σκέτος
Γλυκός
Μέτριος
Γέμισμα
Ελεγχος
Philips-M32
Χαρακτηριστικά
Λειτουργίες
Χρώμα Ροζ
Μέγεθος M
Νερό 1000
Ζάχαρη 100
Καφές 200
Γάλα 80
Σκέτος
Γλυκός
Μέτριος
Γέμισμα
Ελεγχος
Αντικείμενο της κλάσης
"Καφετιέρα"
Η κλάση "Μηχανή Καπουτσίνο", κληρονομεί
όλα τα χαρακτηριστικά και τις λειτουργίες
της κλάσης "Καφετιέρα"
Κλάση
"Μηχανή Καπουτσίνο"
Χαρακτηριστικά
Λειτουργίες
Χρώμα
Μέγεθος
Νερό
Ζάχαρη
Καφές
Γάλα
Σαντιγί
Σκέτος
Γλυκός
Μέτριος
Καπουτσίνο
Γέμισμα
Ελεγχος
AEG-KP
Χαρακτηριστικά
Λειτουργίες
Χρώμα Γκρι
Μέγεθος L
Νερό 800
Ζάχαρη 80
Καφές 100
Γάλα 50
Σαντιγί 20
Σκέτος
Γλυκός
Μέτριος
Καπουτσίνο
Γέμισμα
Ελεγχος
Νέα χαρακτηριστικά και λειτουργίες
της κλάσης "Μηχανή Καπουτσίνο"
Αντικείμενο της κλάσης
"Μηχανή Καπουτσίνο"

# Ορολογία (συνέχεια)

## Επαναχρησιμοποίηση (reusability)

- Μία κλάση , αφού δημιουργηθεί, μπορεί να διανεμηθεί για να χρησιμοποιηθεί σε πολλά προγράμματα. Αυτό καλείται επαναχρησιμοποίηση και είναι σαν τις βιβλιοθήκες συναρτήσεων που χρησιμοποιούν οι διαδικαστικές γλώσσες.
- Στον αντικειμενοστραφή προγραμματισμό, με την έννοια της κληρονομικότητας δίνεται η δυνατότητα να επεκταθεί η έννοια της επαναχρησιμοποίησης. Μπορούμε να πάρουμε μία υπάρχουσα κλάση και χωρίς να την τροποποιήσουμε, να προσθέσουμε σ' αυτήν επιπλέον χαρακτηριστικά και δυνατότητες.

# Η Γλώσσα C++

**Δομή προγράμματος:**

```
#include <iostream>
using namespace std;
main()
{
    cout << "Hello world";
}
```

**Σε παλαιότερους μεταγλωττιστές γράφουμε <iostream.h>, χωρίς τη χρήση namespace.**

- Το πρόγραμμα αποτελείται από μία συνάρτηση, τη *main()*.
- Ένα πρόγραμμα C++ μπορεί να αποτελείται από πολλές συναρτήσεις, κλάσεις και άλλα στοιχεία προγράμματος, αλλά όταν εκτελείται, ο έλεγχος πάντα μεταβιβάζεται στη *main()*. Η συνάρτηση *main()* με τη σειρά της, μπορεί να περιέχει κλήσεις προς άλλες ανεξάρτητες συναρτήσεις.
- Η πρώτη γραμμή του προγράμματος είναι μία οδηγία προ-επεξεργαστή (*preprocessor directive*), δηλαδή μία εντολή προς τον μεταγλωττιστή για να παρεμβάλει ένα άλλο αρχείο στο πηγαίο πρόγραμμα.

# Έξοδος δεδομένων

Το αναγνωριστικό *cout* είναι στην ουσία ένα αντικείμενο. Έχει προκαθορισθεί να αντιστοιχεί στο ρεύμα καθιερωμένης εξόδου (standard output stream). Το ρεύμα αναφέρεται στη ροή δεδομένων. Το ρεύμα καθιερωμένης εξόδου κανονικά κατευθύνεται στην οθόνη, αν και μπορεί να κατευθυνθεί και προς άλλες συσκευές εξόδου.

Ο τελεστής << ονομάζεται τελεστής παρεμβολής (insertion) ή τοποθέτησης (put to). Κατευθύνει τα περιεχόμενα της μεταβλητής που είναι στα δεξιά του, προς το αντικείμενο που είναι στα αριστερά του.

# Σχόλια

**Μπορούν να χρησιμοποιηθούν σχόλια σε ένα πρόγραμμα με δύο τρόπους:**

```
// σχόλια μίας γραμμής

/* σχόλια
   πολλών
   γραμμών */
```

# Είσοδος δεδομένων

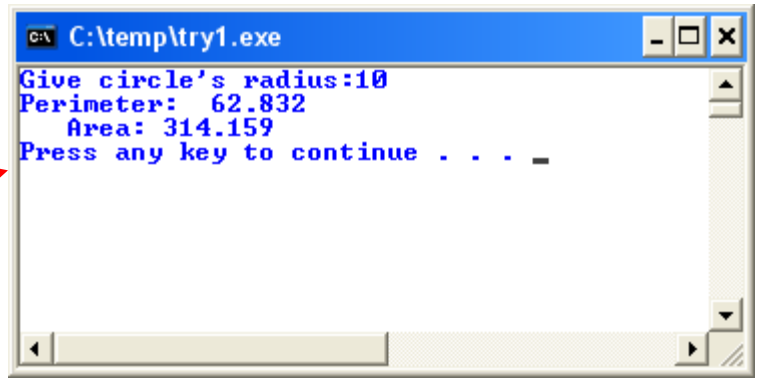
```
#include <iostream>
using namespace std;
main()
{
    int qty, price, value;
    cout << “Δώσε ποσότητα:”;
    cin >> qty;
    cout << “Δώσε τιμή μονάδος:”;
    cin >> price;
    value = qty * price;
    cout << “Η αξία του προϊόντος είναι:” << value;
}
```

- *Η πρόταση cin >> qty υποχρεώνει το πρόγραμμα να περιμένει από τον χρήστη να πληκτρολογήσει έναν αριθμό. Ο αριθμός που δίνεται τοποθετείται στη μεταβλητή qty.*
- *Η δεσμευμένη λέξη cin είναι ένα αντικείμενο, προκαθορισμένο στη C++ να αντιστοιχεί στο ρεύμα καθιερωμένης εισόδου. Αυτό το ρεύμα αναπαριστά δεδομένα που έρχονται συνήθως από το πληκτρολόγιο.*
- *Το >> είναι ο τελεστής που φέρνει την τιμή που βρίσκεται αριστερά του και την τοποθετεί στη μεταβλητή δεξιά του.*
- *Επειδή το πρόγραμμα δε διαβάζει την είσοδο παρά μόνο μετά την πληκτρολόγηση του return, ο χρήστης μπορεί να διορθώσει ενδεχόμενο λάθος κατά την εισαγωγή τιμής.*

# Χειριστές

**Είναι τελεστές που χρησιμοποιούνται με τον τελεστή << για να τροποποιούν ή να χειρίζονται τα δεδομένα ως προς τον τρόπο που θα εμφανιστούν.**

```
#include <cstdlib>        // system
#include <iostream>
#include <iomanip>        // std::setw
using namespace std;
main()
{
   float  radius, perim, area;
   const float PI = 3.14159;
   cout<<"Give circle's radius:";
   cin >> radius;
   perim = 2 * PI *radius;
   area = PI * radius * radius;
   cout<<setw(8)<<"Perimeter:"<<setw(8)<<setprecision(3)<<perim<<endl;
   cout<<setw(8)<<"Area:"<<setw(8)<<setprecision(3)<<area<<endl;
   system("PAUSE");
}
```

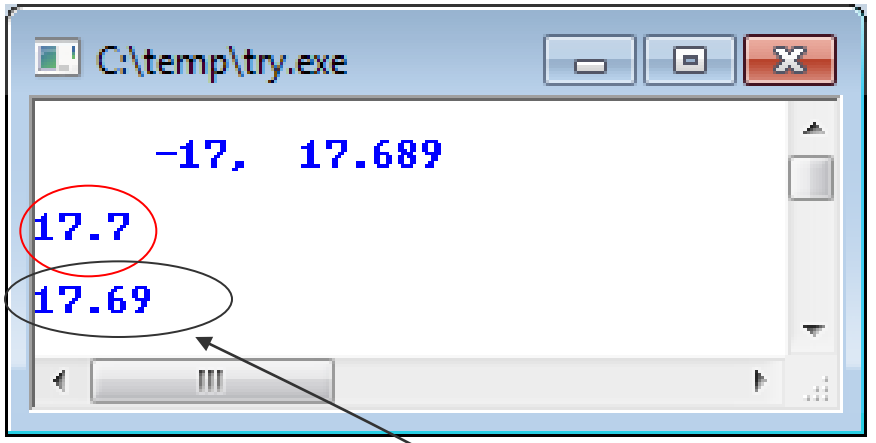



*Καθορισμός του μήκους πεδίου*

# Χειριστές (συνέχεια)

Ο χειριστής setprecision(n) καθορίζει ως n τον αριθμό των ψηφίων σε τιμές μεταβλητών κινητής υποδιαστολής. Εάν όμως προηγηθεί ο τελεστής fixed, ο χειριστής setprecision(n) καθορίζει ως n τον αριθμό των δεκαδικών ψηφίων σε τιμές μεταβλητών κινητής υποδιαστολής:

```
#include <iostream>
#include <iomanip>
using namespace std;
main()
{
    int i=-17;
    float x=17.689;
    cout << endl << setw(8) << i << "," << setw(8) << x;
    cout << endl << endl << setprecision(3) << x << endl;
    cout << fixed << endl << setprecision(2) << x;
}
```



# Τύποι δεδομένων

| *Τύπος* | *Από* | *Έως* | *Ψηφία ακρίβειας* | *Bytes μνήμης* |
|---|---|---|---|---|
| char | -128 | 127 | | 1 |
| int | -32768 | 32767 | | 2 |
| long | -2147483648 | 2147483647 | | 4 |
| float | $3.4*10^{-38}$ | $3.4*10^{38}$ | 7 | 4 |
| double | $1.7*10^{-308}$ | $1.7*10^{308}$ | 15 | 8 |
| long double | $3.4*10^{-4932}$ | $1.1*10^{4932}$ | 19 | 10 |

# Μετατροπή τύπων

Όταν δύο τελεσταίοι διαφορετικού τύπου εμφανίζονται στην ίδια παράσταση, η μεταβλητή κατώτερου τύπου μετατρέπεται στον τύπο της μεταβλητής ανώτερου τύπου, δηλαδή:

| *Τύπος δεδομένων* | *Σειρά* |
|---|---|
| long double | Ανώτερος |
| double | |
| float | |
| long | |
| int | |
| char | Κατώτερος |

# Προσαρμογές τύπων (casting)

Αναφέρεται στις μετατροπές δεδομένων που καθορίζονται από τον προγραμματιστή, σε αντίθεση με τις αυτόματες μετατροπές δεδομένων που έχουν περιγραφεί.

```
#include <iostream>
using namespace std;
main()
{
   int  x = 25000;
   x = (x * 10) / 10;      // χωρίς προσαρμογή
   cout << "x = " << x << endl;
   x = 25000 ;
   x = (long(x) * 10) / 10 ;    // προσαρμογή σε τύπο long
   cout << "x = " << x << endl;
}
```

Η έξοδος του προγράμματος θα είναι:

x = -1214
x = 25000

Στην πρώτη παράσταση χωρίς προσαρμογή, το γινόμενο x*10 είναι μεγάλο για να αποθηκευτεί σε μεταβλητή τύπου int.
Στη δεύτερη παράσταση ο τύπος της μεταβλητής μετατρέπεται σε long και έτσι το γινόμενο 250000 μπορεί να αποθηκευτεί στη μεταβλητή.

# Τελεστές

***Αριθμητικοί***
**+, -, /, *, \ ακέραιο πηλίκο, % ακέραιο υπόλοιπο**

***Αύξησης***
**++**

***Μείωσης***
**--**

***Απόδοσης τιμής***
**=, +=, -=, *=, /=, %=**

***Συσχετιστικοί***
**<, >, <=, >=, ==, !=**

***Λογικοί***
**&& (and), || (or), ! (not)**

# Βρόχοι και αποφάσεις

***Δομή επιλογής***

- ***Απλή επιλογή***
- ***Σύνθετη επιλογή***
- ***Εμφωλευμένη επιλογή***
- ***Πολλαπλή επιλογή***

***Δομή Επανάληψης***

- ***Επαναληπτικό σχήμα με αρχικό έλεγχο επανάληψης***
- ***Επαναληπτικό σχήμα με τελικό έλεγχο επανάληψης***
- ***Επαναληπτικό σχήμα συγκεκριμένου αριθμού επαναλήψεων***

# Δομές επιλογής

***Απλή επιλογή:***

```
if  (x > 0)
     cout << “x is positive”;
```

***Σύνθετη επιλογή:***

```
if  (x > 0)
     cout << “x is positive”;
else
     cout << “x is negative or zero”;
```

***Εμφωλευμένη επιλογή:***

```
if  (x > 0)
     cout << “x is positive”;
else
     if (x == 0)
          cout << “x is zero”;
     else
          cout << “x is negative”;
```

# Δομές επιλογής

***Πολλαπλή επιλογή***

***1η έκδοση (με εμφωλευμένες if):***

```
if  (grade > 8)
     cout << “excellent”;
else
     if (grade > 6)
          cout << “very good”;
     else
          if (grade == 6)
               cout << “good”;
          else
               if (grade == 5)
                    cout << “pass”;
               else
                    cout << “fail”;
```

# Δομές επιλογής

*2η έκδοση (με την εντολή switch):*

```
switch (grade)
{
   case 9, 10: cout << “excellent”;
               break;
   case 7,   8: cout << “very good”;
               break;
   case      6: cout << “good”;
               break;
   case      5: cout << “pass”;
               break;
   default    : cout << “fail”;
               break;
}
```

# Δομές επανάληψης

***Επαναληπτικό σχήμα με αρχικό έλεγχο επανάληψης:***

```
sum = 0;
cin >> x;
while (x != 0)
{
    sum = sum + x;
    cin >> x;
}
cout << sum;
```

# Δομές επανάληψης

***Επαναληπτικό σχήμα με έλεγχο επανάληψης στο τέλος:***

```
do
{
   cout << "1. ΕΙΣΑΓΩΓΗ" << endl;
   cout << "2. ΔΙΑΓΡΑΦΗ" << endl;
   cout << "3. ΕΜΦΑΝΙΣΗ" << endl;
   cout << "4. ΕΞΟΔΟΣ" << endl;
   cout << "Δώσε επιλογή:";
   cin >> x;
   switch (x)
   {
       case 1: cout << "ΕΙΣΑΓΩΓΗ";
               break;
       case 2: cout << "ΔΙΑΓΡΑΦΗ";
               break;
       case 3: cout << "ΕΜΦΑΝΙΣΗ";
               break;
   }
} while (x != 4);
```

# Δομές επανάληψης

***Επαναληπτικό σχήμα με συγκεκριμένο αριθμό επαναλήψεων:***

```
sum = 0;
for (i=0; i<10; i++)
{
    cin >> x;
    sum = sum + x;
}
cout << sum;
```

# Συναρτήσεις

**Η χρήση μίας συνάρτησης σε ένα πρόγραμμα περιλαμβάνει τρία στάδιο:**

- **Δήλωση**
- **Ορισμός**
- **Κλήση**

- **Στο στάδιο της *δήλωσης* μίας συνάρτησης, δηλώνουμε στο μεταγλωττιστή ότι θα χρησιμοποιήσουμε στο πρόγραμμά μας τη συνάρτηση.**

- **Στο στάδιο του *ορισμού* περιγράφουμε τη λειτουργία της συνάρτησης (δηλαδή γράφουμε τον κώδικά της).**

- **Στο στάδιο της *κλήσης* χρησιμοποιούμε τη συνάρτηση (δηλαδή εκτελούμε τον κώδικά της).**

# Ορισμός συνάρτησης

```
<τύπος δεδομένων επιστροφής>  <όνομα συνάρτησης>  <λίστα παραμέτρων>
{
   <δήλωση τοπικών μεταβλητών>
   <κώδικας συνάρτησης>
}
```

- Όλες οι συναρτήσεις τελειώνουν και επιστρέφουν αυτόματα στη διαδικασία από την οποία κλήθηκαν, όταν συναντήσουν το τελευταίο τους άγκιστρο. Μαζί τους επιστρέφουν - συνήθως - και μία τιμή, η οποία περικλείεται στην εντολή return.
- Όταν όμως δεν επιθυμούμε μία συνάρτηση να επιστρέφει τιμή, τότε πρέπει να γράψουμε τη δεσμευμένη λέξη void αντί για τον τύπο δεδομένων επιστροφής.

# Κλήση συνάρτησης

- κλήση κατ' αξία
  *(call by value)*
- κλήση κατ' αναφορά
  *(call by reference)*
- κλήση με χρήση δεικτών, υποπερίπτωση της κλήσης κατ' αναφορά
  *(call using pointers)*

# Κλήση κατ' αξία (*call by value*), συνάρτηση με επιστροφή τιμής

```
int  athroisma(int x, int y)
{
    int  z;    // τοπική μεταβλητή
    z = x + y;
    return  z ;
}

// κλήση συνάρτησης
 c = athroisma(a,b);
```

# Κλήση κατ' αξία (*call by value*), συνάρτηση χωρίς επιστροφή τιμής

```
void  athroisma(int x, int y)
{
   int  z;    // τοπική μεταβλητή
   z = x + y;
   cout << "Το άθροισμα των" << x << " και " << y << "
   είναι " << z;
   return  0;
}

// κλήση συνάρτησης
  athroisma(a,b);
```

# Παράδειγμα με τοπικές μεταβλητές

```
#include <iostream>

using namespace std;    float square (float x);
main () {
    float in,out;

     in = -4.0;
     out = square(in);
     cout << in << " squared is " << out << endl;

}

float  square (float x)
{
    float out;
     out = 24.5;
     return (x*x);
```

Ίδια ονόματα τοπικών μεταβλητών

Αποτέλεσμα στην οθόνη out=16.0

out=24.5 μέσα στην square

# Παράδειγμα με καθολικές μεταβλητές

```
#include <iostream>
using namespace std;    float glob;      // καθολική μεταβλητή
float square (float x);
main ()  {
  float in;
  glob = 2.0;
  in = square(glob);
  cout << glob << " squared is " << in << endl;
  in = square(glob);
  cout << glob << " squared is " << in << endl;
}
float  square (float x)  {
  glob=glob+1.0;
  return (x*x); }
```

**Ζητά από τη *square()* το τετράγωνο της glob,
δηλαδή το τετράγωνο του 2.0**

**Τώρα ζητά από τη *square()* το τετράγωνο τη νέας τιμής της glob, δηλαδή το τετράγωνο του 3.0**

**Η glob γίνεται 3.0**

# Εμβέλεια μεταβλητών (scope)

- **Εμβέλεια προγράμματος:** μεταβλητές αυτής της εμβέλειας είναι οι καθολικές. Είναι ορατές από όλες τις συναρτήσεις του πρόγραμματος, έστω κι αν βρίσκονται σε διαφορετικά αρχεία πηγαίου κώδικα.
- **Εμβέλεια αρχείου:** μεταβλητές αυτής της εμβέλειας είναι ορατές μόνο στο αρχείο που δηλώνονται και μάλιστα από το σημείο της δήλωσής τους και κάτω. Μεταβλητή που δηλώνεται με τη λέξη κλειδί *static* πριν από τον τύπο, έχει εμβέλεια αρχείου, π.χ. static int velocity.
- **Εμβέλεια συνάρτησης:** Προσδιορίζει την ορατότητα του ονόματος από την αρχή της συνάρτησης έως το τέλος της. Εμβέλεια συνάρτησης έχουν μόνο οι goto ετικέτες.
- **Εμβέλεια μπλοκ:** Προσδιορίζει την ορατότητα από το σημείο δήλωσης έως το τέλος του μπλοκ στο οποίο δηλώνεται. Μπλοκ είναι ένα σύνολο από προτάσεις, οι οποίες περικλείονται σε άγκιστρα. Μπλοκ είναι η σύνθετη πρόταση αλλά και το σώμα συνάρτησης. Εμβέλεια μπλοκ έχουν και τα τυπικά ορίσματα των συναρτήσεων.

# Εμβέλεια μεταβλητών (συνέχεια)

**Η C++ επιτρέπει τη χρήση ενός ονόματος για την αναφορά σε διαφορετικά αντικείμενα, με την προϋπόθεση ότι αυτά έχουν διαφορετική εμβέλεια ώστε να αποφεύγεται η *σύγκρουση ονομάτων* (name conflict). Εάν οι περιοχές εμβέλειας έχουν επικάλυψη, τότε το όνομα με τη μικρότερη εμβέλεια αποκρύπτει (hides) το όνομα με τη μεγαλύτερη.**

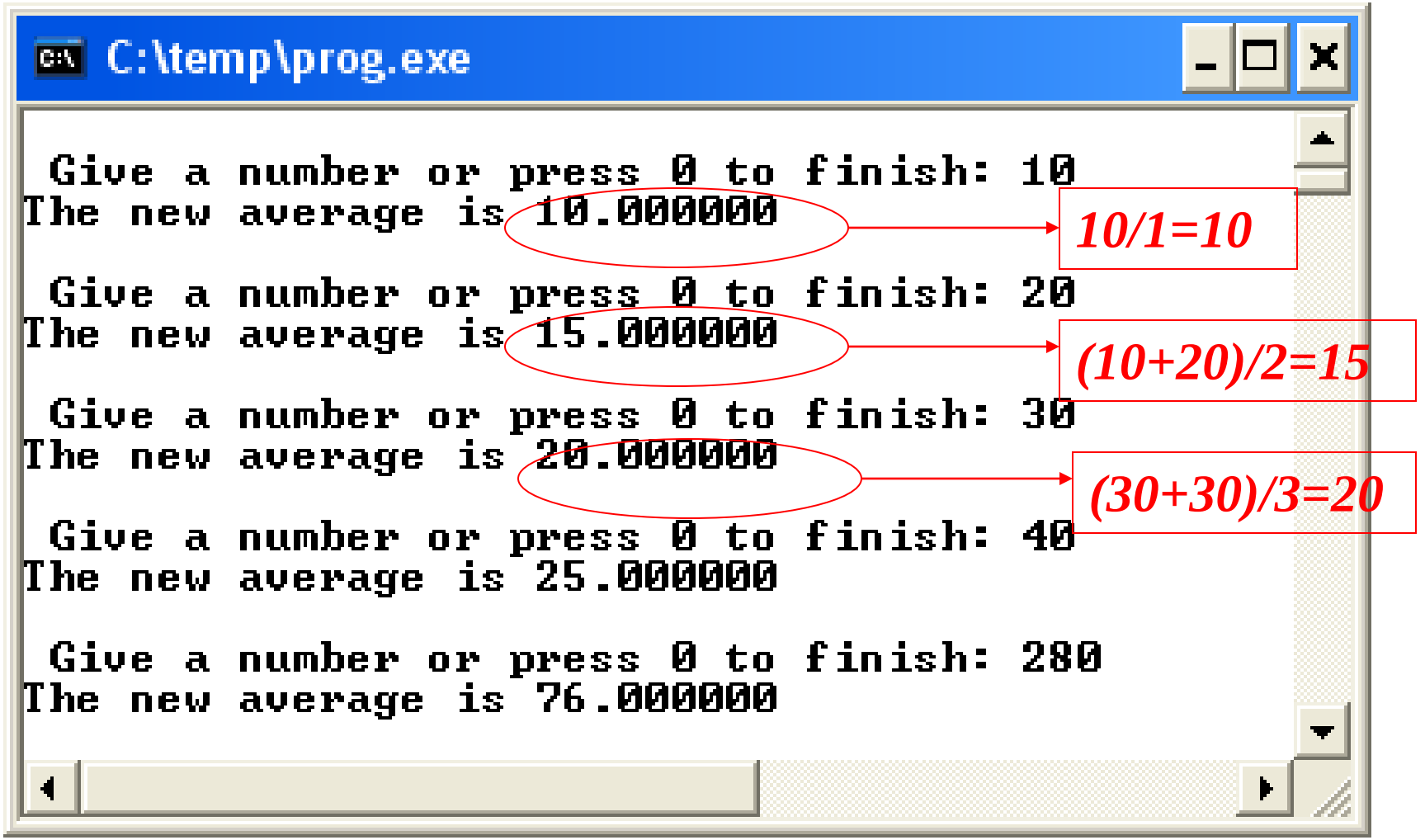
***Παράδειγμα στατικών μεταβλητών:***

```
#include <iostream>
using namespace std;
float get_average(float newdata);   // δήλωση συνάρτησης
main()
{
  float data=1.0;
  float average;
  while (data!=0)
  {
    cout << "Give a number or press 0 to finish: "
    cin >> data;
    average=get_average(data);
    cout << endl << "The new average is " << average;
  }
}  // τέλος της main, συνέχεια στην επόμενη διαφάνεια
```

```
float get_average(float newdata)
{
  static float total=0.0;
  static int count=0;
  count++;
  total=total+newdata;
  return(total/count);
}  //end of get_average
```

*Εκτελούνται μόνο την πρώτη φορά. Τις επόμενες διατηρούν το αποτέλεσμα της προηγούμενης κλήσης και σε αυτό προστίθενται στη μεν total το newdata, στη δε count η μονάδα.*

# Αποτέλεσμα:


C:\temp\prog.exe
Give a number or press 0 to finish: 10
The new average is 10.000000
Give a number or press 0 to finish: 20
The new average is 15.000000
Give a number or press 0 to finish: 30
The new average is 20.000000
Give a number or press 0 to finish: 40
The new average is 25.000000
Give a number or press 0 to finish: 280
The new average is 76.000000
10/1=10
(10+20)/2=15
(30+30)/3=20

# Αναδρομικότητα (recursion)

*Μία συνάρτηση ονομάζεται αναδρομική όταν μία εντολή του σώματος της συνάρτησης καλεί τον ίδιο της τον εαυτό. Η αναδρομή είναι μία διαδικασία με την οποία ορίζουμε κάτι μέσω του ίδιου του οριζόμενου.*

***Παράδειγμα:*** Να ορισθεί συνάρτηση που υπολογίζει το άθροισμα των αριθμών από 1 έως n. *(επεξηγήσεις στην επόμενη διαφάνεια)*

***1ος τρόπος:***

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

***2ος τρόπος (με αναδρομή):***

```
int sum(int n)
{
  if (n<=1) return(n);
  else return(sum(n-1)+n);
}
```

**Μέσα στη *sum()* καλείται ο εαυτός της.**

***Επεξηγήσεις:***

```
if (n<=1) return(n);
else return(sum(n-1)+n);
```

Εάν το n είναι ίσο με 1, τότε το άθροισμα ταυτίζεται με το n (οριακή περίπτωση). Στη γενική περίπτωση, θεωρούμε ότι ο υπολογισμός του αθροίσματος n μπορεί να θεωρηθεί ως υπολογισμός του αθροίσματος των αριθμών από το 1 έως το n-1 συν το n. Αντίστοιχα, ο υπολογισμός του αθροίσματος n-1 μπορεί να θεωρηθεί ως υπολογισμός του αθροίσματος των αριθμών από το 1 έως το n-2 συν το n-1. Ακολουθώντας την παραπάνω διαδικασία, μπορούμε να ορίσουμε τα εξής:

1+…+n = (1+…+(n-1)) + n

(1+…+(n-1)) = (1+…+(n-2)) + (n-1)

(1+…+(n-2)) = (1+…+(n-3)) + (n-2)

(1+…+(n-3)) = (1+…+(n-4)) + (n-3)

**κ.ο.κ.**

***Από τα παραπάνω προκύπτει ότι κάθε σχέση είναι ίδια με την προηγούμενη, με απλή αλλαγή των ορισμάτων.***

*Τι σημαίνει όμως αυτό;*

**Την πρώτη φορά καλείται η συνάρτηση *sum()* με όρισμα n. Με την πρόταση *return(sum(n-1)+n)* η *sum()* καλεί τον ίδιο της τον εαυτό με διαφορετικό όμως όρισμα (n-1). Η ενεργοποίηση αυτή θα προκαλέσει με τη σειρά της νέα ενεργοποίηση και αυτό θα συνεχισθεί έως ότου προκληθεί διακοπή. Η διακοπή είναι αποκλειστική ευθύνη του προγραμματιστή. Στο συγκεκριμένο παράδειγμα η διακοπή προκαλείται με την πρόταση *if (n<=1) return(n)* ,που σημαίνει ότι όταν το n φθάσει να γίνει 1 υπάρχει πλέον αποτέλεσμα. Έτσι οι διαδοχικές κλήσεις για n=4 είναι:**

*sum(4) καλεί τη sum(3)*

*sum(3) καλεί τη sum(2)*

*sum(2) καλεί τη sum(1)*

*η sum(1) δίνει αποτέλεσμα 1 και το επιστρέφει στη sum(2)*

*η sum(2) δίνει αποτέλεσμα 1+2=3 και το επιστρέφει στη sum(3)*

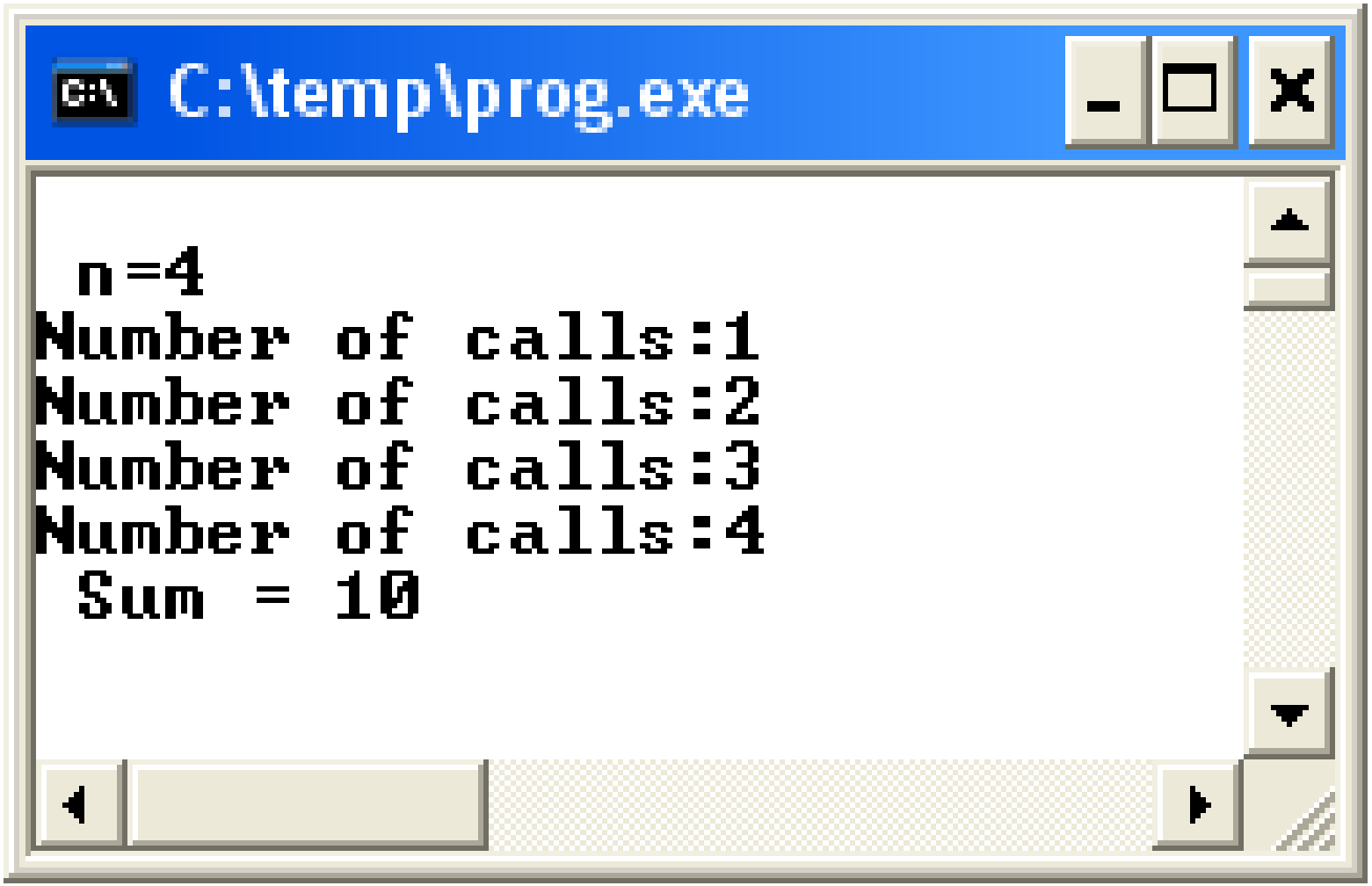
*η sum(3) δίνει αποτέλεσμα 3+3=6 και το επιστρέφει στη sum(4)*

*η sum(4) δίνει αποτέλεσμα 6+4=10, το οποίο είναι και το τελικό*

**Το πλήρες πρόγραμμα έχει την ακόλουθη μορφή:**

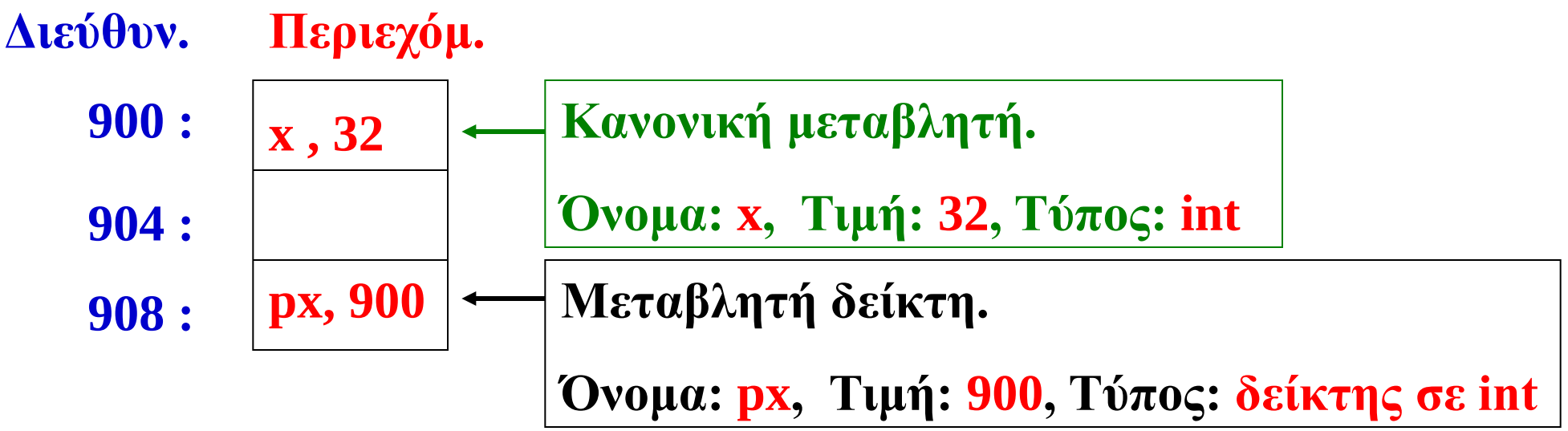
```
#include <iostream>
# include <cstdlib>
using namespace std;
int sum(int n);              // δήλωση της συνάρτησης sum
int number_of_calls=0;
void main(){
  int n=4;                   // ανάθεση n=4
  cout << endl << " n= " n;
  cout << endl << " Sum = " << sum(n);
  cout << endl << endl;
  system("PAUSE");
}   // τέλος της main
int sum(int n){              // ορισμός της συνάρτησης sum
  if (n<=1){
    number_of_calls++;
    cout << endl << "Number of calls:" << number_of_calls;
    return(n);
  }
  else{
    number_of_calls++;
    cout << endl << "Number of calls:" << number_of_calls;
    return(sum(n-1)+n);
  }
}  // τέλος της sum
```

# *Αποτέλεσμα:*


C:\temp\prog.exe
n=4
Number of calls:1
Number of calls:2
Number of calls:3
Number of calls:4
Sum = 10

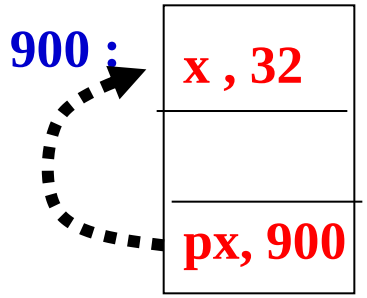
# Δήλωση δείκτη

**Δείκτης:** μία μεταβλητή που κρατά **μία διεύθυνση**.



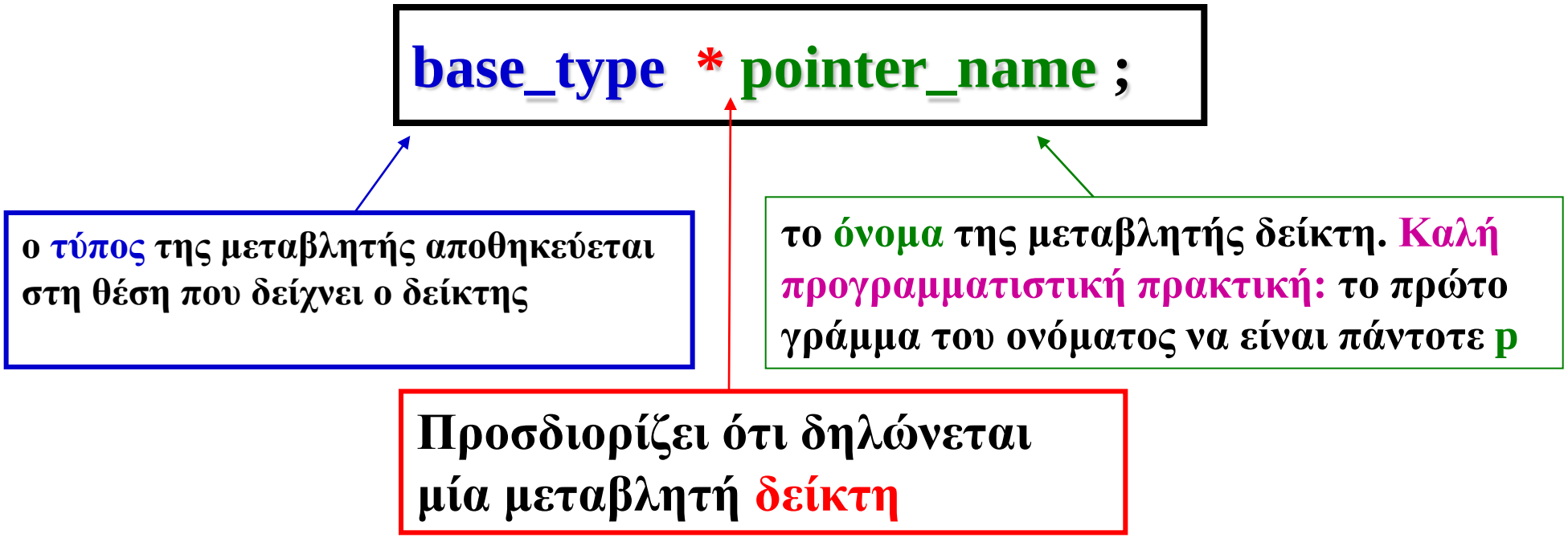

**Λέμε ότι ο px δείχνει στην x**

**Φανταστείτε ένα τόξο από τη μεταβλητή δείκτη στην κανονική μεταβλητή, το οποίο δείχνει πού "δείχνει" ο δείκτης.**



# Δήλωση δείκτη

Ο δείκτης πρέπει να δηλωθεί:



# Δήλωση δείκτη

**Γιατί πρέπει να δηλωθεί ο τύπος της κανονικής μεταβλητής;**

**Γιατί όταν δηλώνεται μία (κανονική) μεταβλητή, δεσμεύεται συγκεκριμένη μνήμη, π.χ. 8 bytes για double, 4 bytes για int.**

**Ο δείκτης αναφέρεται σε μία διεύθυνση, στην οποία αποθηκεύεται η τιμή μίας κανονικής μεταβλητής.**

**Ο δείκτης χρησιμοποιείται για να γίνεται έμμεση αναφορά σ' αυτήν την τιμή. Έτσι, πρέπει να γνωρίζουμε πόση ακριβώς μνήμη καταλαμβάνει αυτή η τιμή.**

**Πόση μνήμη καταλαμβάνει η ίδια η μεταβλητή δείκτη;**

**Η διεύθυνση είναι ένας ακέραιος, έτσι ο δείκτης καταλαμβάνει 4 bytes, ανεξάρτητα από τον τύπο της κανονικής μεταβλητής που δείχνει.**

# Δήλωση δείκτη

**Ο αστερίσκος συνδέεται με το όνομα κι όχι με τον τύπο:**

```
int *pcount;   // δείκτης σε ακεραίους, με ονομασία pcount
```

```
int *pcount, *pnum;   // δείκτες σε ακεραίους, με ονομασίες pcount και pnum
```

```
int *pcount,  number;  /* ένας δείκτης σε ακέραιο, με ονομασία pcount και ένας ακέραιος με ονομασία number */
```

# Δήλωση δείκτη

**Πώς επιλέγεται το όνομα ενός δείκτη;**

Οι ίδιες συμβάσεις που ισχύουν στις κανονικές μεταβλητές.

Ωστόσο, συνήθως ο αρχικός χαρακτήρας του ονόματος δείκτη είναι το **p**, έτσι ώστε το πρόγραμμα να καθίσταται περισσότερο ευανάγνωστο, καθώς με τον πρώτο χαρακτήρα φαίνεται εάν μία μεταβλητή είναι δείκτης ή όχι. Εναλλακτικά, μπορούμε να προσθέτουμε την κατάληξη **_ptr**.

***Παράδειγμα***:

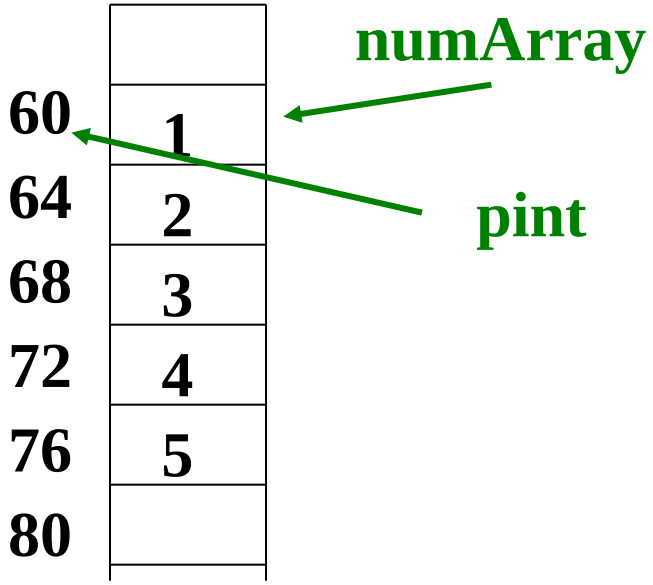
```
int *pcount, *count_ptr;     // δείκτες σε ακεραίους
char *pword, *word_ptr;      // δείκτες σε χαρακτήρες
```

# Αρχικοποίηση δεικτών

1) Χρησιμοποιώντας πίνακα

(Υπενθύμιση:το όνομα ενός πίνακα είναι μία διεύθυνση.

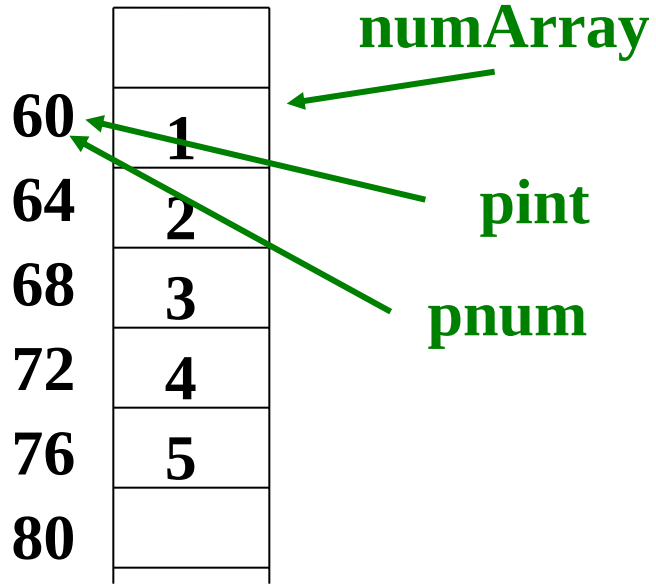
```
int  numArray[5] = {1,2,3,4,5};

int *pint;

pint = numArray;
```



# Αρχικοποίηση δεικτών

## 2) Χρησιμοποιώντας άλλους δείκτες ίδιου τύπου

```
int  numArray[5] = {1,2,3,4,5};
int *pint, *pnum;
pint = numArray;
pnum = pint;
```

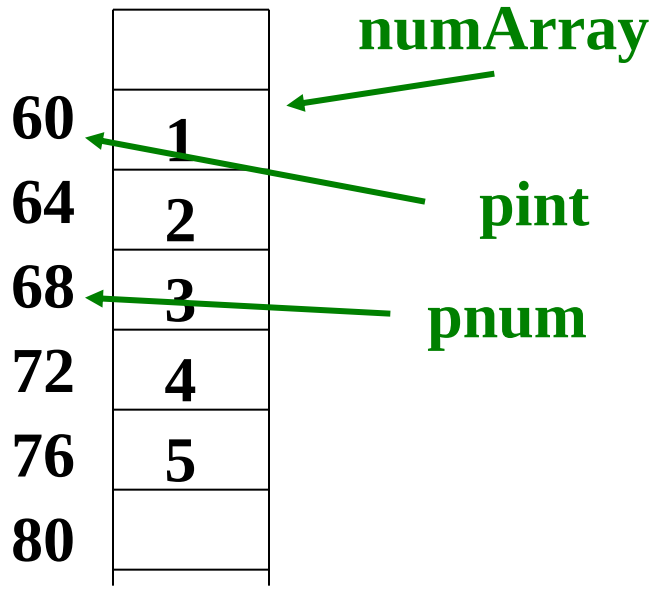


# Αρχικοποίηση δεικτών

## 3) Χρησιμοποιώντας αριθμητική δεικτών

```
int  numArray[5] = {1,2,3,4,5};
int *pint, *pnum;
pint = numArray;
pnum = pint+2;
```

**Πήγαινε δύο θέσεις πιο κάτω (με περιεχόμενο ακεραίου)**



# Αρχικοποίηση δεικτών

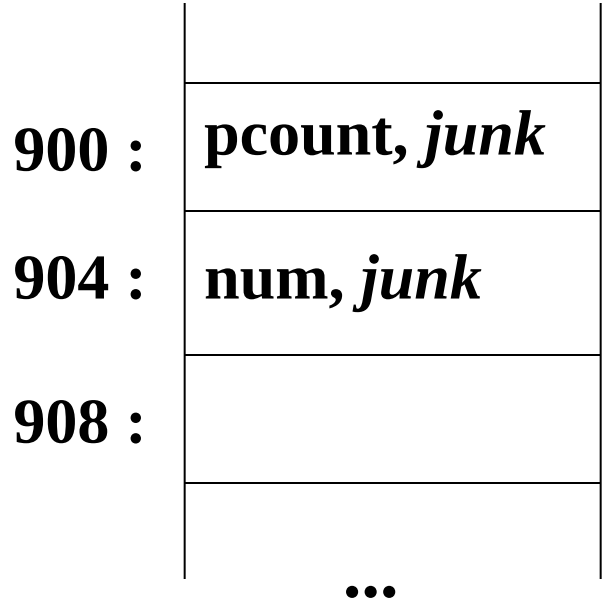
4) Χρησιμοποιώντας τον τελεστή διεύθυνσης & (address-of operator)

```
int *pnum;

int count;

pnum = &count;
```

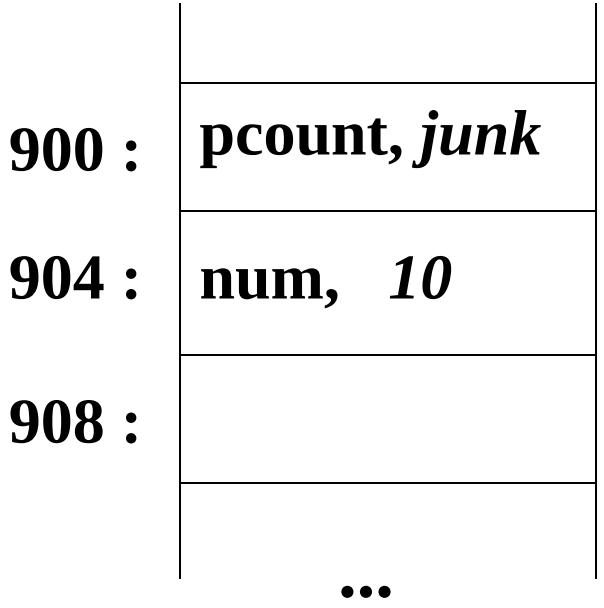
Η γραμμή αυτή αναφέρει: ο pnum να λάβει ως τιμή τη διεύθυνση της μεταβλητής count, δηλαδή ο δείκτης pnum να "δείχνει" στη μεταβλητή count.

*Ακολούθως φαίνεται πώς μπορούμε να προσπελάσουμε την τιμή μίας μεταβλητής με τη χρήση δείκτη.*

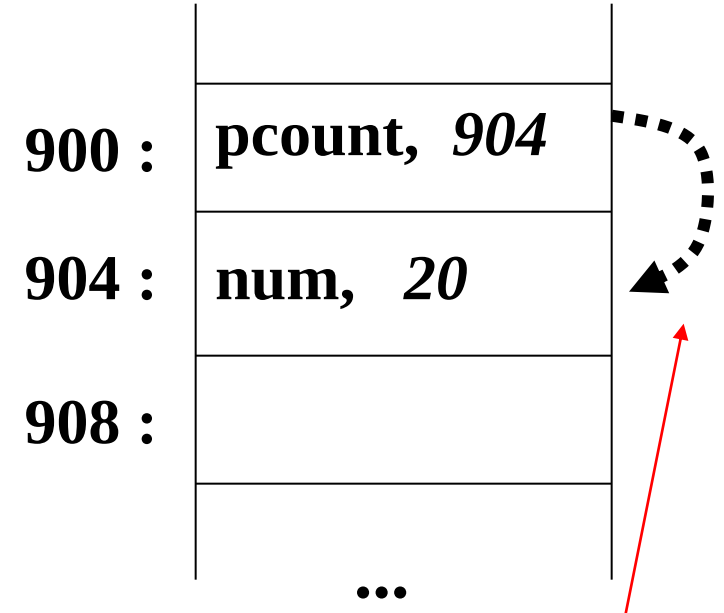
```
int *pcount, num;
num = 10;
pcount = &num;
*pcount = 20;
```



```
int *pcount, num;
num = 10;
pcount = &num;
*pcount = 20;
```



```
int *pcount, num;

num = 10;

pcount = &num;

*pcount = 20;
```




Ο αστερίσκος υποδηλώνει ότι πρέπει να ακολουθηθεί το βέλος για να προσπελασθούν τα δεδομένα της θέσης, στην οποία δείχνει.

***pcount = 20;**

**Ο αστερίσκος ονομάζεται *τελεστής περιεχομένου* (dereferencing operator).**

**Διαβάζεται *"στη διεύθυνση"*.**

**Χρησιμοποιείται για να προσπελαύνει τα περιεχόμενα της θέσης μνήμης στην οποία δείχνει ο δείκτης.**

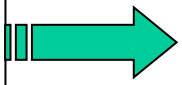
**Δε θα πρέπει να συγχέεται με τον αστερίσκο της δήλωσης δείκτη.**

# Εφαρμογή δεικτών

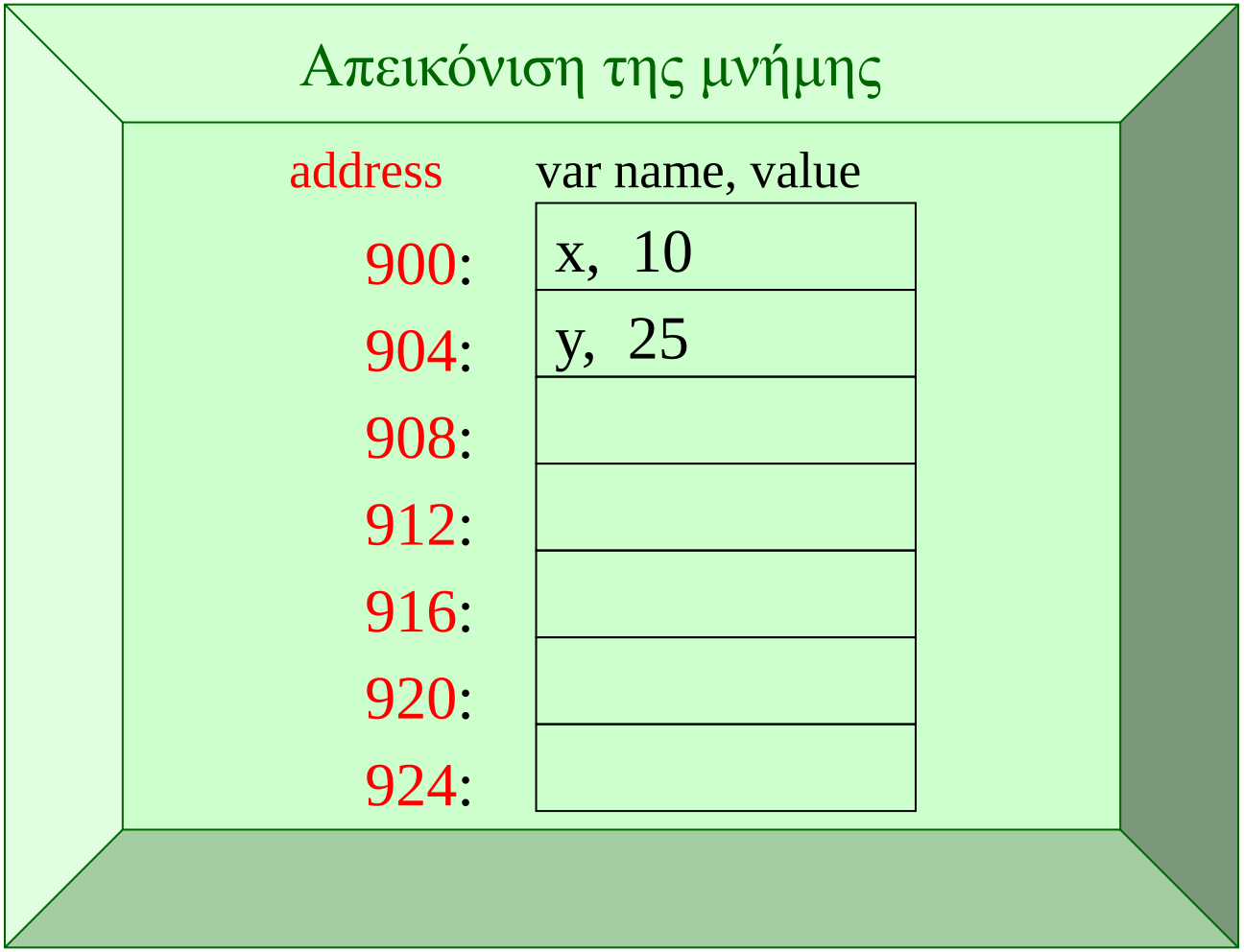
```
void main()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

Τα ορίσματα της συνάρτησης είναι δείκτες σε ακέραιες μεταβλητές.

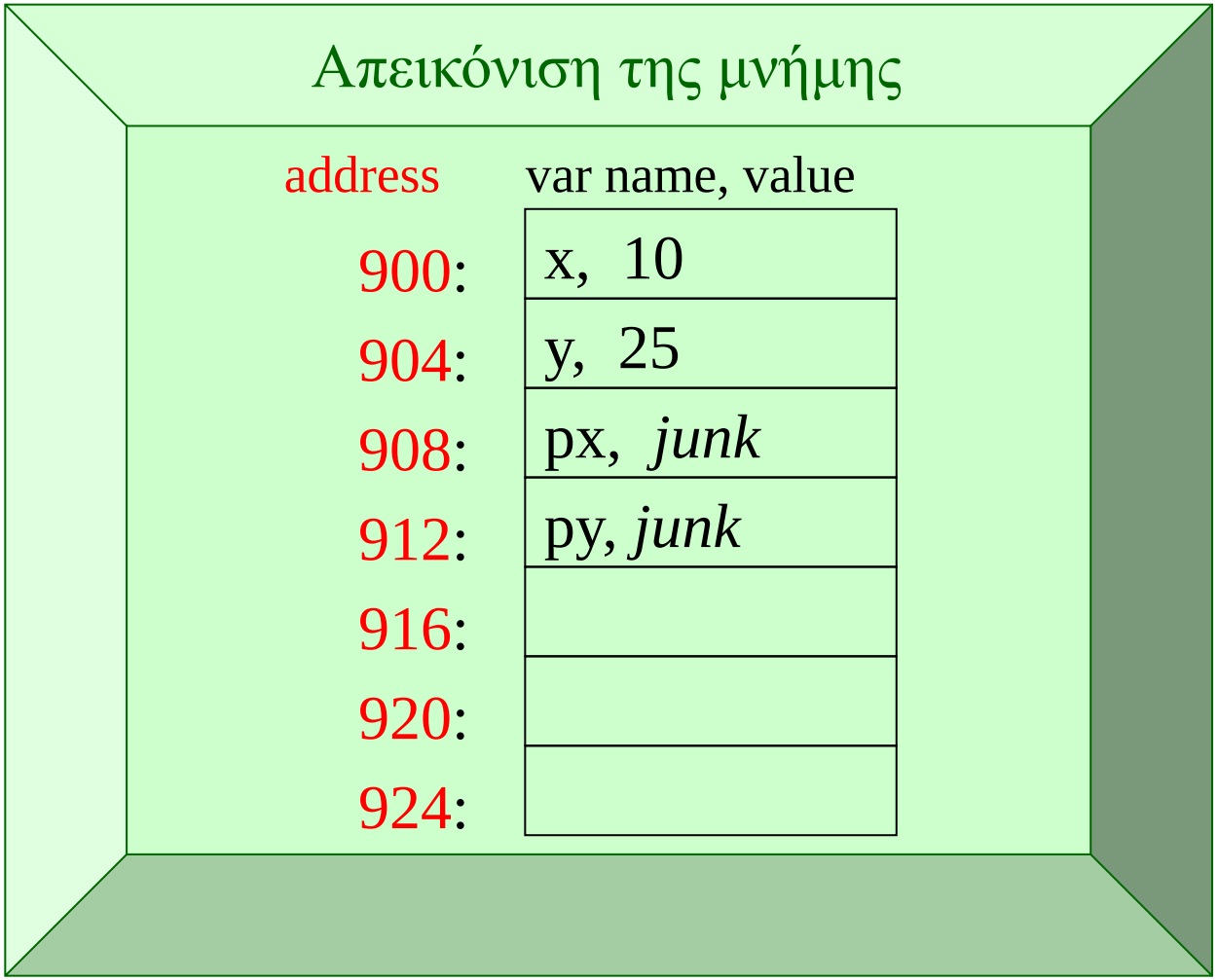
*Θα χρησιμοποιήσουμε μαύρο χρώμα για τις τοπικές μεταβλητές της main() και μπλε για τις τοπικές μεταβλητές της swap().*



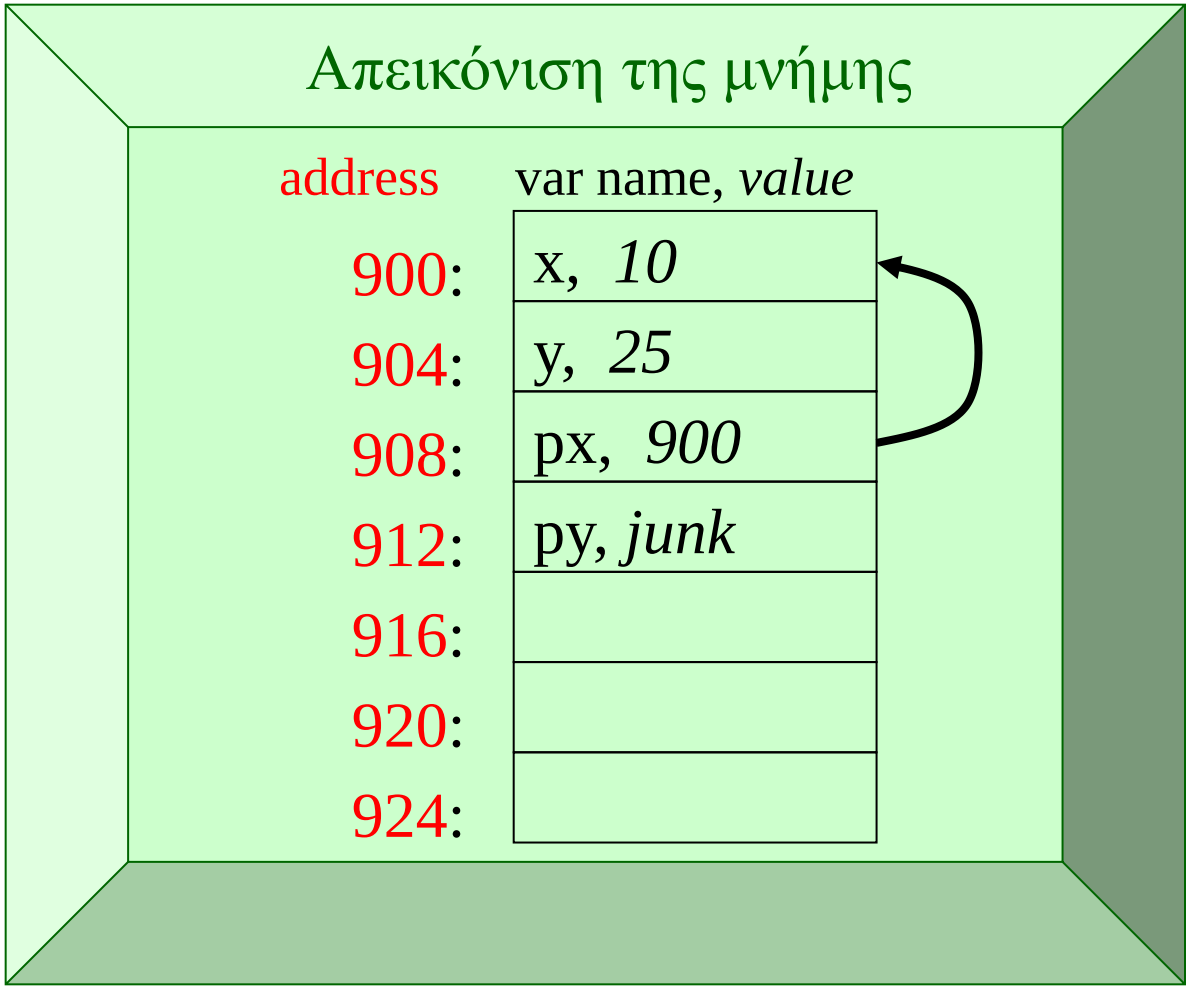
```
void main()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



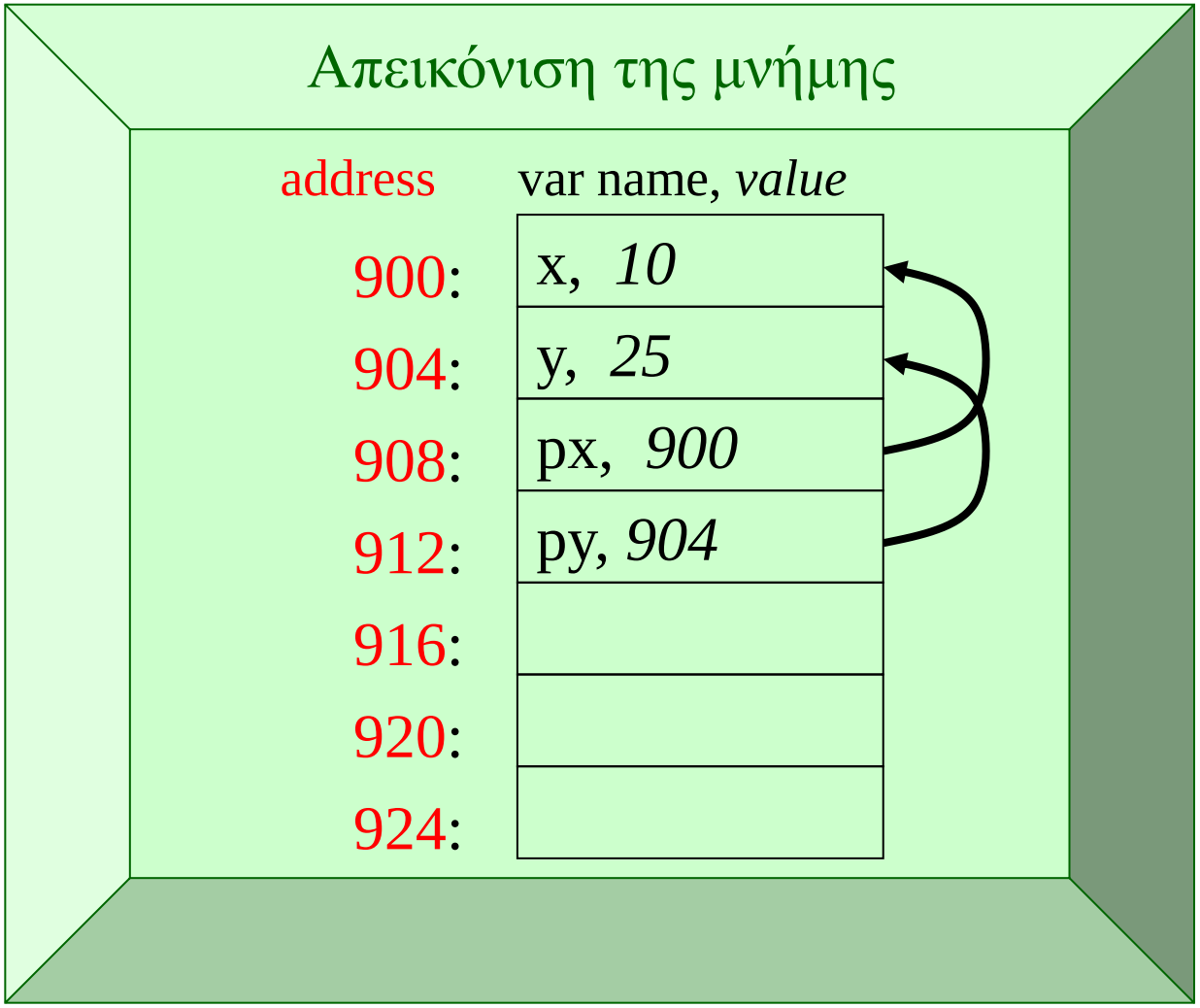
```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

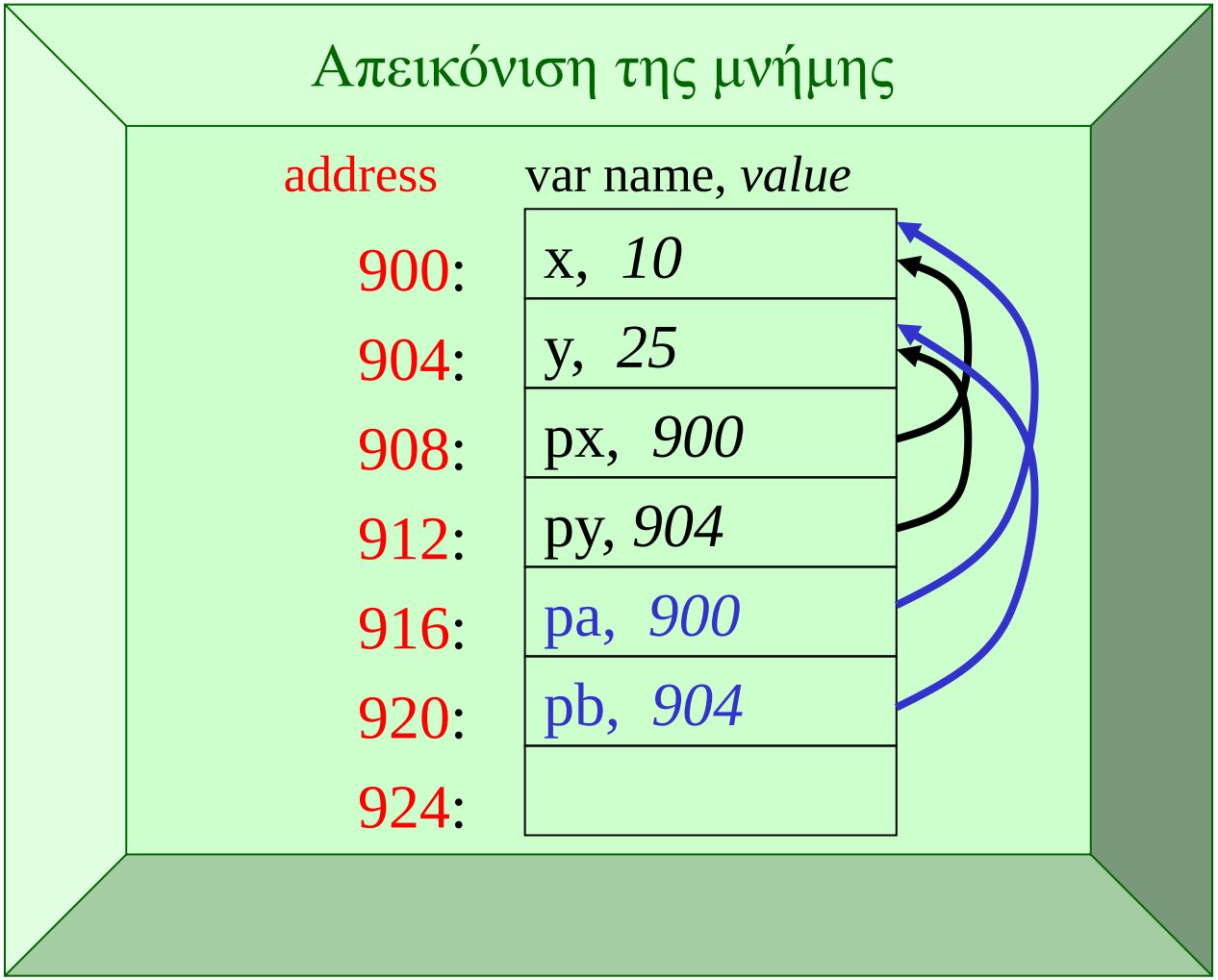


```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

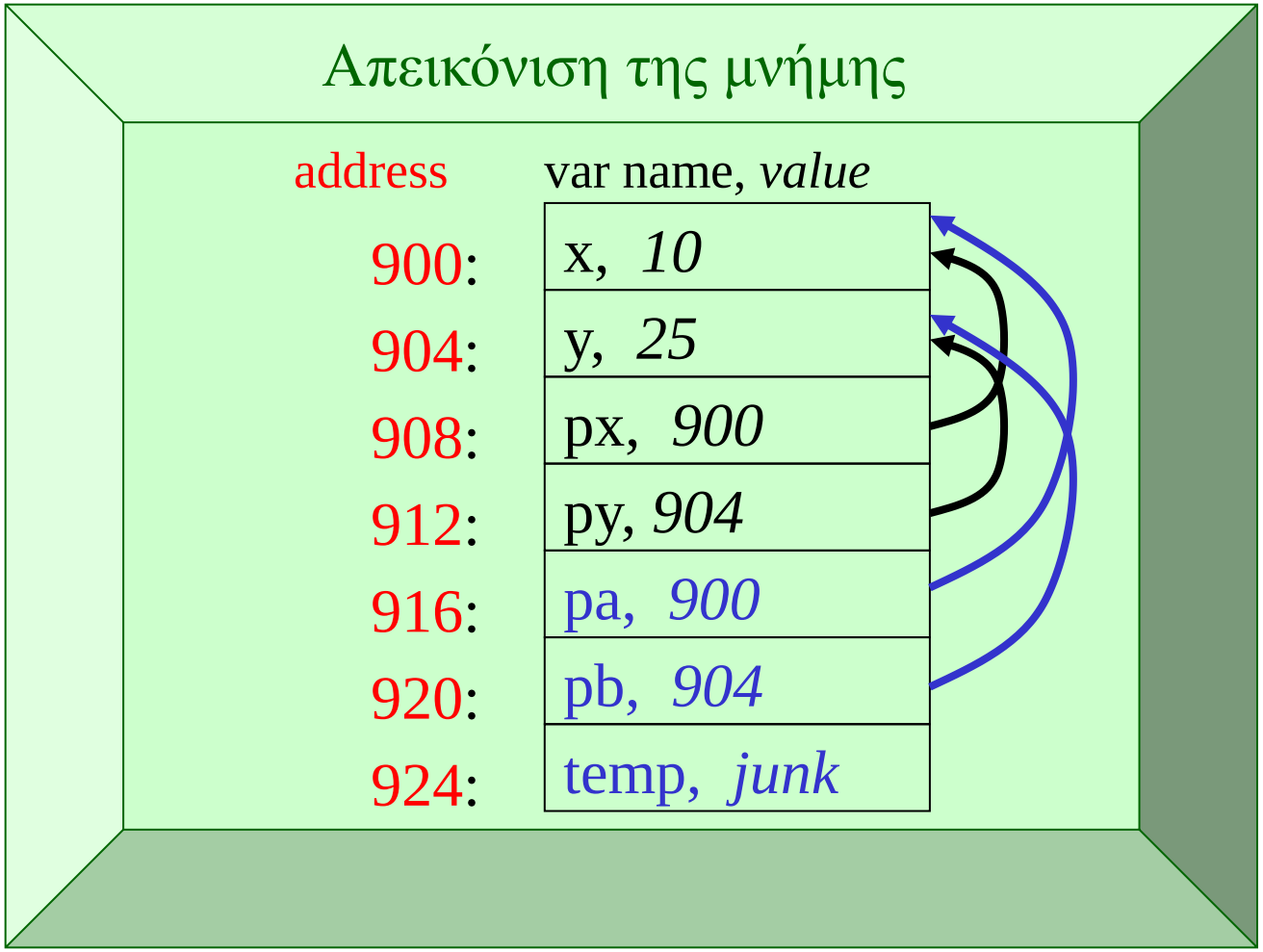


```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

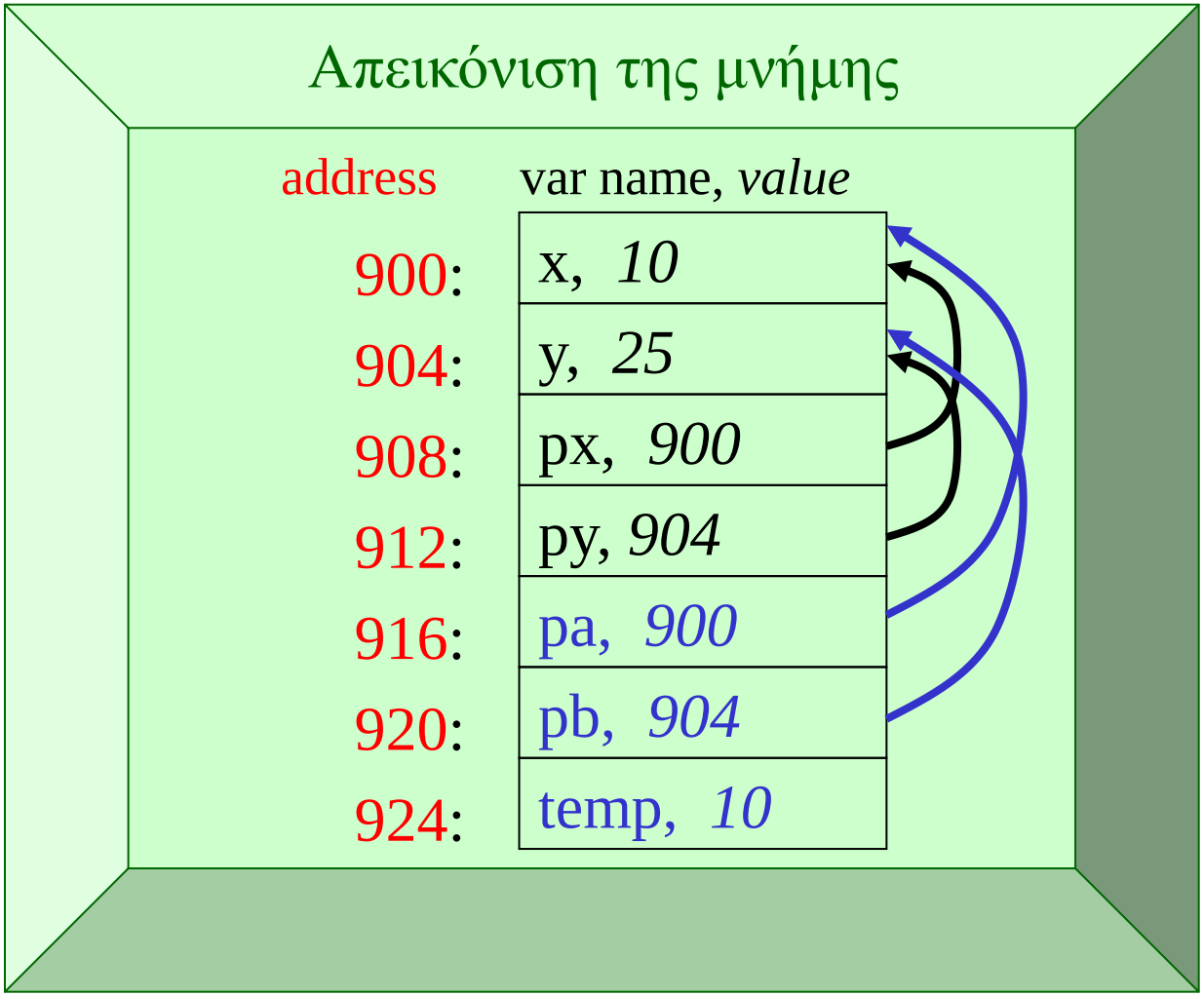
Αντίγραφο της τιμής της ***py***



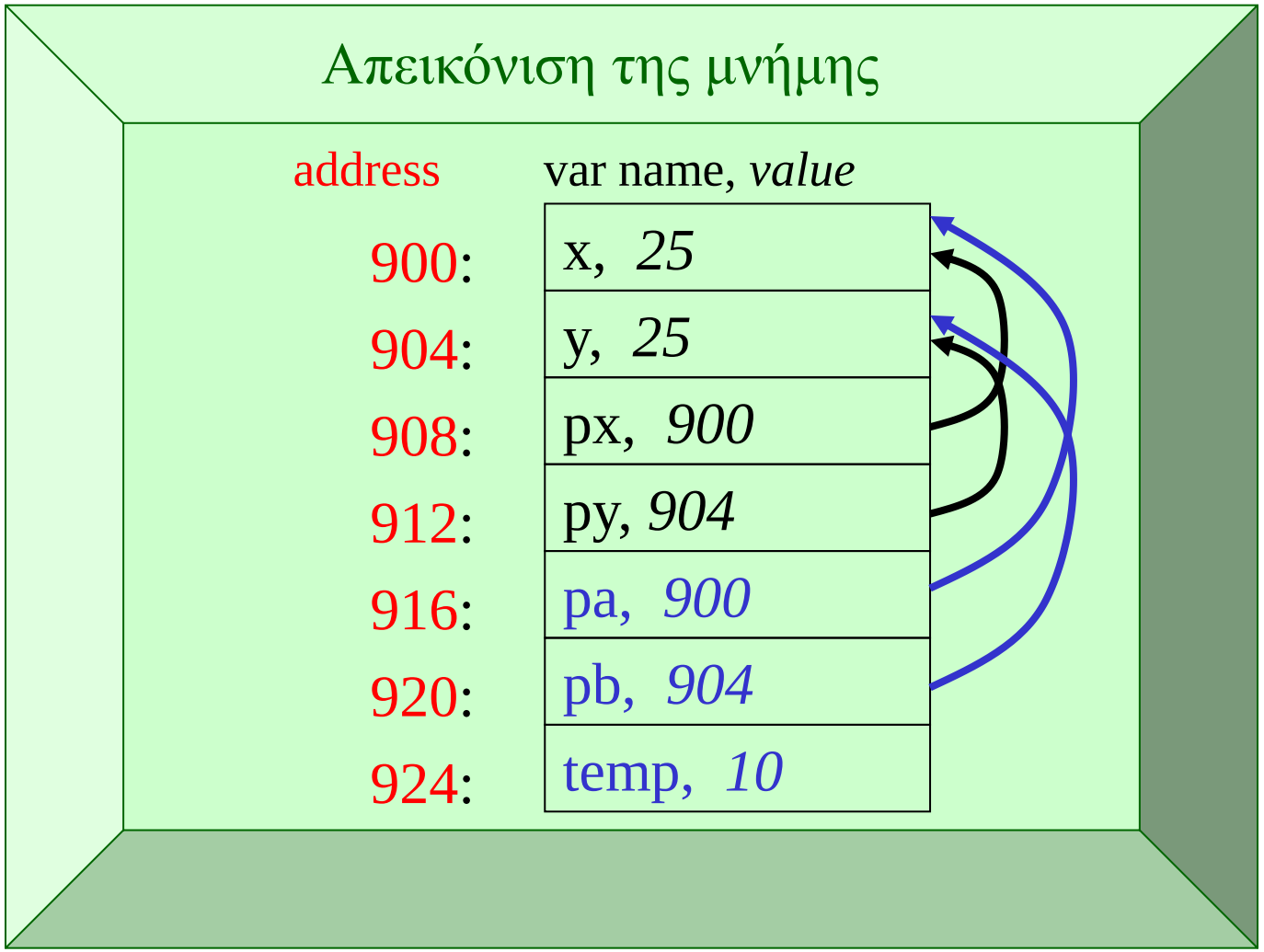
```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



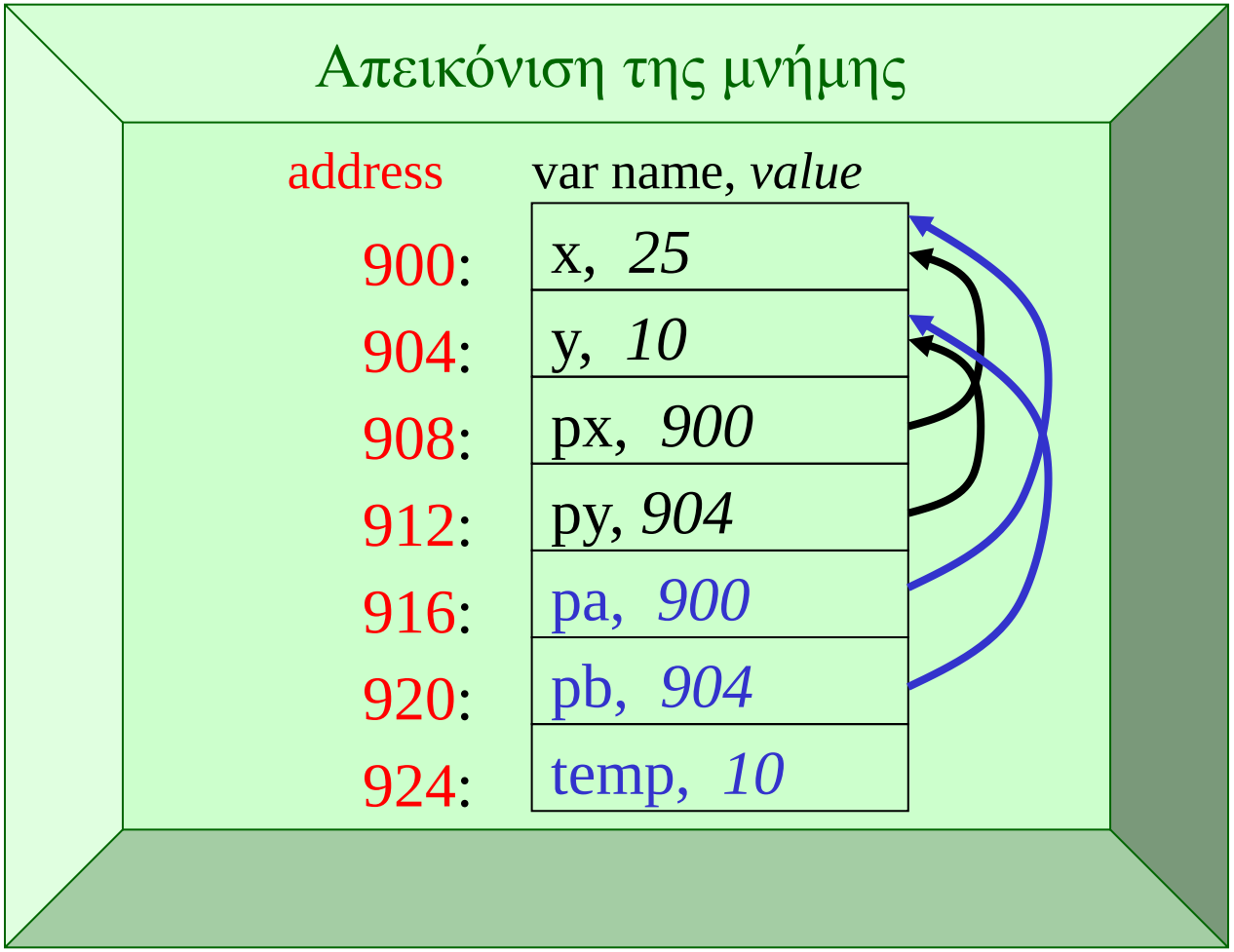
```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



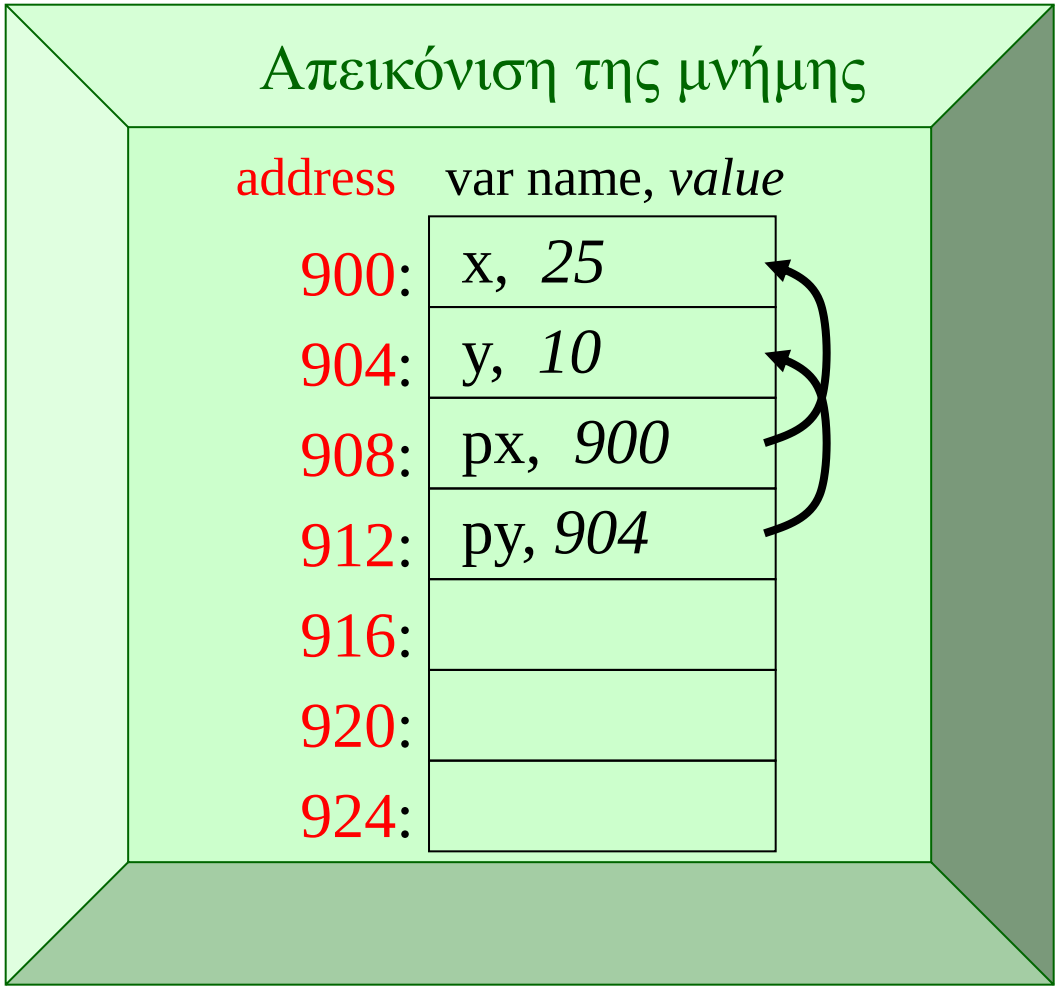
```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



```
void main ()
{
   int x=10, y=25;
   int *px, *py;
   px = &x;
   py = &y;
   swap(px, py);
}

void swap (int *pa, int *pb) {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```



# Επεξηγήσεις:

Αν και η συνάρτηση *swap()* δεν επιστρέφει τίποτε άμεσα στη *main()*, έχει μία παρενέργεια (side effect).

Όταν καλείται η *swap()*, τα ορίσματά της είναι οι δείκτες px και py, οι οποίες σχετίζονται με τις διευθύνσεις των x και y, αντίστοιχα.

Παρατήρηση: δε χρειάζεται να δηλώσουμε τους δείκτες px και py. Το μόνο που απαιτείται είναι να περασθούν οι διευθύνσεις των x και y στη *swap()*, όπως φαίνεται ακολούθως:

```
void main ()
{
   int x=10, y=25;
   swap(&x, &y);
}
```

```
void swap (int *pa, int *pb)
 {
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}
```

*Λειτουργεί με τον ίδιο τρόπο που λειτουργούσε το πρόγραμμα με τους* **px** *και* **py**.

# Δείκτες και συναρτήσεις

Έως τώρα περνούσαμε τους δείκτες ως ορίσματα σε συναρτήσεις.

Μπορούμε όμως να επιστρέφουμε δείκτες;

- Όταν επιστρέφεται ένας δείκτης, θα πρέπει να δείχνει σε δεδομένα της καλούσας συνάρτησης.
- Δεν πρέπει ποτέ να επιστρέφετε δείκτη που δείχνει σε τοπική μεταβλητή της καλούμενης συνάρτησης, γιατί όταν τερματισθεί η συνάρτηση οι τοπικές μεταβλητές εξαφανίζονται.
- Μπορούμε να χρησιμοποιήσουμε δείκτη για να αλλάξουμε το περιεχόμενο της θέσης στην οποία δείχνει, αλλά δεν πρέπει να αλλάξουμε τον ίδιο το δείκτη μέσα στην καλούμενη συνάρτηση.

```
void main () {
    int *pscore, num;
    num = 32;
    pscore = &num;
    print(pscore);
}
```

Αντιγράφει την τιμή του ***pscore*** (δηλαδή τη διεύθυνση στην οποία δείχνει) στον ***ptr***

```
void print(int *ptr) {
    printf("%d", *ptr);
    ptr=ptr+1;
}
```

**ΚΑΚΟ**! Όταν τυπώνει η συνάρτηση τελειώνει και ο ***ptr*** εξαφανίζεται.

Ο ***pscore*** είναι ό,τι ήταν και πριν την κλήση της συνάρτησης ***print***.

```
void main () {
    int *pscore, num;
    num = 32;
    pscore = incr(num);

}
```

```
int *incr(int x) {
    x = x+10;
    return &x;
}
```

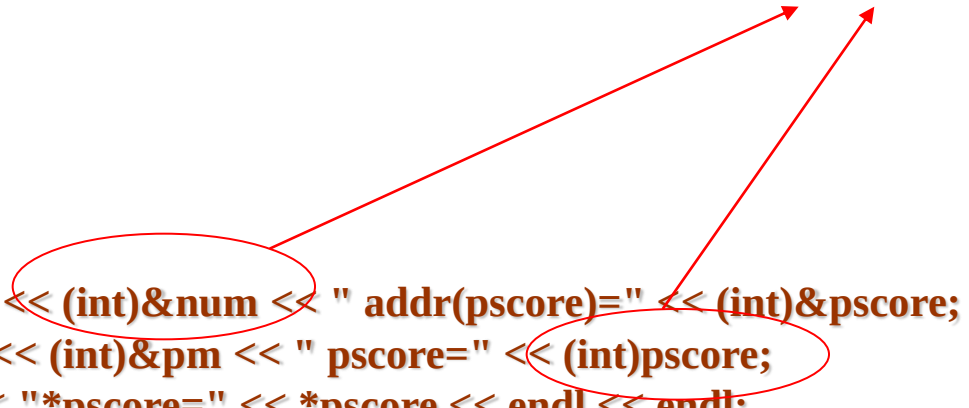
**ΛΑΘΟΣ**! Όταν τελειώνει η ***incr***, η ***x*** εξαφανίζεται και η τιμή της χάνεται.

Ωστόσο, πίσω στη συνάρτηση ο ***pscore*** θα δείχνει στη διεύθυνση που επέστρεψε από τη συνάρτηση αλλά θα υπάρχει «σκουπίδι» (junk) σ' αυτή τη διεύθυνση, εφόσον το ***x*** έχει εξαφανισθεί.

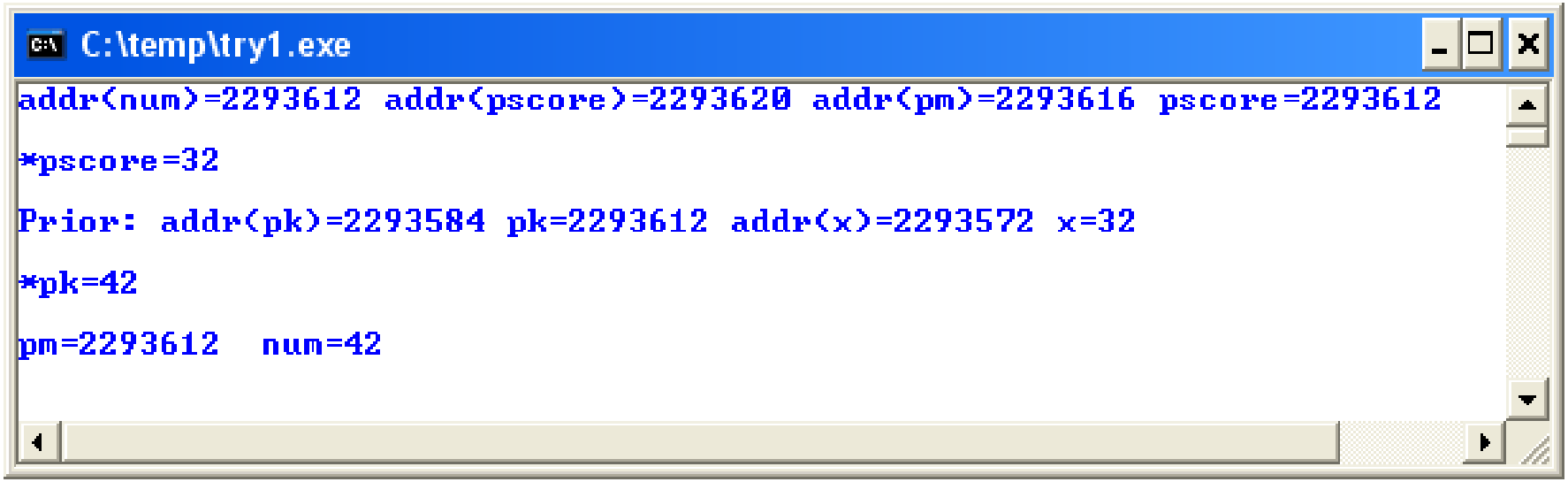
***Σωστή χρήση της επιστρεφόμενης τιμής διεύθυνσης***

```
#include <iostream>
#include <cstdlib>
using namespace std;
int *incr(int *pkitsos);
main()  {
  int *pscore, *pm, num;
  num=32;
  pscore=&num;
  cout << "addr(num)=" << (int)&num << " addr(pscore)=" << (int)&pscore;
  cout << " addr(pm)=" << (int)&pm << " pscore=" << (int)pscore;
  cout << endl << endl << "*pscore=" << *pscore << endl << endl;

  pm=incr(pscore);

  cout << "pm=" << (int)pm << "  num=" << num << endl << endl << endl;

}
```

*Για να εμφανισθεί η διεύθυνση σε ακέραια μορφή*



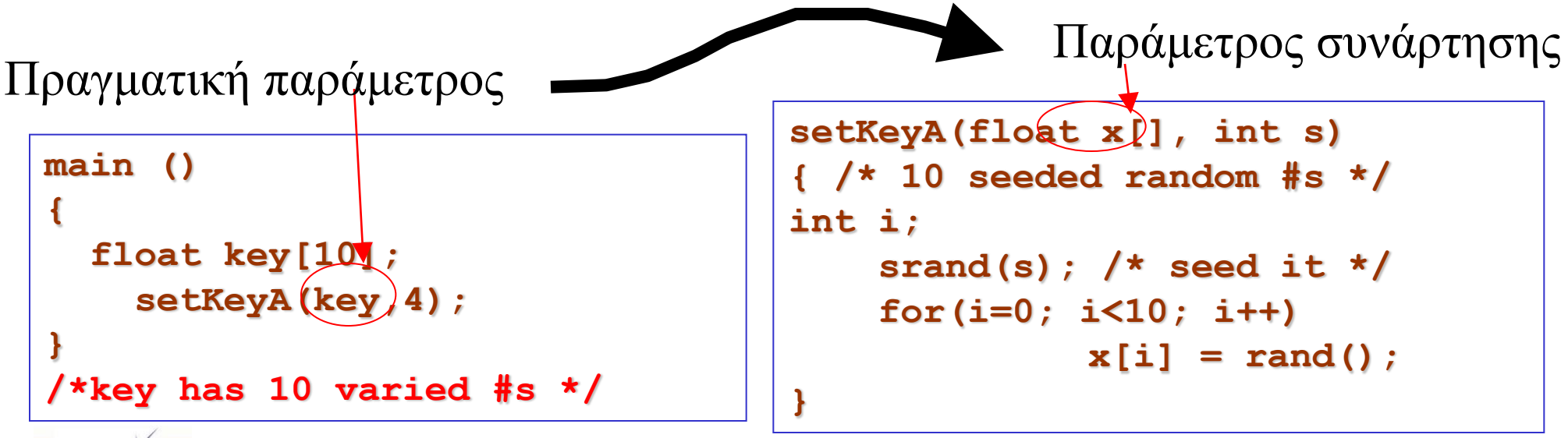
```
int *incr(int *pk)  {
  int x;
  x=*pk;

  cout << "Prior: addr(pk)=" << (int)&pk << " pk=" << (int)pk;
  cout << " addr(x)=" << (int)&x << " x=" << x << endl << endl;
  x=x+10;

  *pk=x;
  cout << "*pk=" << *pk << endl << endl;

  return(pk);
}
```



# (*Υπενθύμιση*) Συνάρτηση και Πίνακες

Εάν η πραγματική παράμετρος είναι όνομα πίνακα (πχ. ***key***), στέλνει τη διεύθυνση του πρώτου byte του πίνακα.

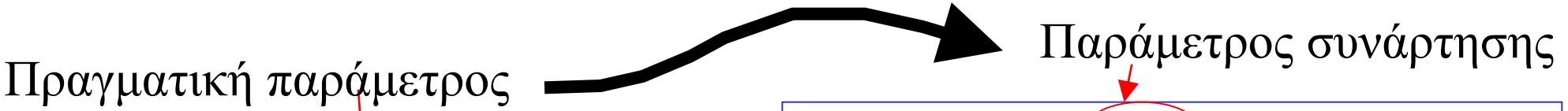
Η παράμετρος στη δήλωση της συνάρτησης είναι ένα όνομα **τοπικού** πίνακα (π.χ. ***x***). Κρατά ένα αντίγραφο της ίδιας διεύθυνσης αλλά χρησιμοποιεί διαφορετικό όνομα.



# Κλήση κατ' αναφορά

Όταν τα ορίσματα είναι πίνακες, περνιούνται στις συναρτήσεις **'κατ' αναφορά' (call by reference)** – και μπορεί να είναι ΤΟΣΟ είσοδοι ΟΣΟ και έξοδοι.

(Τόσο η **main()** όσο και η **setKeyA()** μπορούν να θέσουν τιμές σε στοιχεία πινάκων).




```
main ()
{
     float key[10];
     setKeyA(key, 4);
}
/*key has 10 varied #s */
```

```
setKeyA(float x[], int s)
{ /* 10 seeded random #s */
     int i;
     srand(s); /* seed it */
     for(i=0; i<10; i++)
                x[i] = rand();
}
```

# Κλήση κατ' αναφορά

## Οι δείκτες επιτρέπουν να περνάμε ΟΠΟΙΟΔΗΠΟΤΕ δεδομένο κατ'αναφορά

Παράδειγμα: αλλάζουμε την ***setKeyA*** έτσι ώστε να δέχεται ως ορίσματα μόνο δείκτες και διευρύνονται οι δυνατότητες. **(αλλάζουμε το όνομα σε *setKeyP*)**

Πραγματική παράμετρος

```
main ()
{ /*double-width key */
   int seed0 = 4; seed1 = 89;
   float key2[20];
   setKeyP(&key2[0], &seed0);
   setKeyP(&key2[10],&seed1);
}
```

Παράμετρος συνάρτησης

```
setKeyP(float *pK, int *pS)
{ /* 10 seeded random #s */
    int i;
    srand(pS[0]); /* seed...*/
    for(i=0; i<10; i++)
                  pK[i] = rand();
    pS[0]=0;      /* clear it*/
}
```

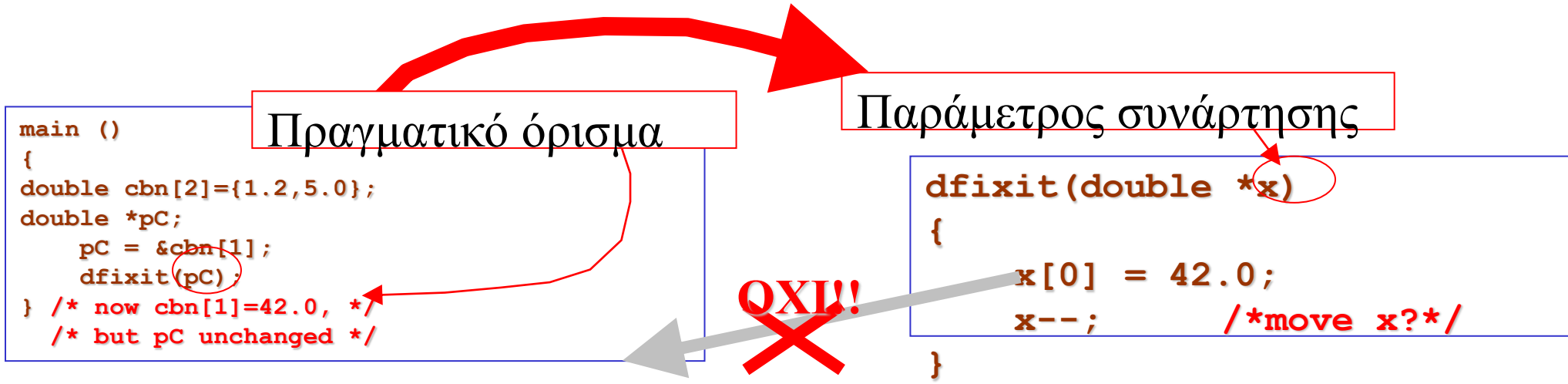
# Κλήση κατ' αναφορά

```
main ()
{ /*double-width key */
   int seed0 = 4; seed1 = 89;
   float key2[20];
   setKeyP(&key2[0], &seed0);
   setKeyP(&key2[10],&seed1);
} /* now seed0,seed1 are 0 */
```

```
setKeyP(float *pK, int *pS)
{ /* 10 seeded random #s */
    int i;
    srand(pS[0]); /* seed...*/
    for(i=0; i<10; i++)
                pK[i] = rand();
   pS[0]=0;      /* clear it*/
}
```

# Συναρτήσεις και Δείκτες

Οι πραγματικές παράμετροι που είναι δείκτες αντιγράφουν μία ***διεύθυνση στις*** παραμέτρους της συνάρτησης, αλλά εάν αλλαχθεί η παράμετρος στη συνάρτηση (δηλ. η διεύθυνση) ΔΕ θα αλλαχθεί η πραγματική παράμετρος !!



# Συναρτήσεις με τύπο επιστροφής δείκτη

Οι συναρτήσεις των οποίων ο τύπος επιστροφής είναι δείκτης;

ΠΩΣ: **η δήλωση της συνάρτησης μοιάζει:**

```
char* findNextVowel( char* str);
```

**→!!ΚΙΝΔΥΝΟΣ!!←**

Όλες οι μεταβλητές της συνάρτησης είναι τοπικές και προσωρινές. Μπορεί να εξαφανισθούν όταν φύγουμε από τη συνάρτηση.

Μην επιστρέφετε δείκτες σε μη ορισμένες μεταβλητές!

**Να μην επιστρέφετε ποτέ δείκτες ΕΚΤΟΣ εάν χρησιμοποιείτε τη λέξη κλειδί "static"**

# Παράδειγμα *static*

```
void main()
{
   float* pKey;

   pKey = setKey(0);
   cout << "keys 0,7 are " << pKey[0] << pKey[7] << endl;
   ... (κώδικας που δεν αλλάζει τον pKey) ...
   cout << "keys 0,7 are " << pKey[0] << pKey[7] << endl;
}
```

**ΣΦΑΛΜΑ**

```
float* setKey(int s)  /* make a cryptographic key */
{
   float keep[10];
   int i;
   srand(s);          /* set rand's seed */
   for(i=0; i<10; i++) keep[i] = rand();
   return(keep);
}
```

# Παράδειγμα *static*

```
void main()
{
  float* pKey;

  pKey = setKey(0);
  cout << "keys 0,7 are " << pKey[0] << pKey[7] << endl;
  ... (code that doesn't change pKey) ...
  cout << "keys 0,7 are " << pKey[0] << pKey[7] << endl;
}
```

**ΣΦΑΛΜΑ**

```
float* setKey(int s)  /* make a cryptogra
{
  float keep[10];
  int i;
  srand(s); /* set rand's seed */
  for(i=0; i<10; i++) keep[i] = rand();
  return(keep);
}
```

**ΔΙΟΤΙ:**
Στην έξοδο η προσωρινή μεταβλητή ***i*** και ο πίνακας ***keep*** είναι ακαθόριστοι

# Παράδειγμα *static*

```
void main()
{
    float* pKey;

   pKey = setKey(0);
   cout << "keys 0,7 are " << pKey[0] << pKey[7] << endl;
   ... (code that doesn't change pKey) ...
 cout << "keys 0,7 are " << pKey[0] << pKey[7] << endl;
}
```

**ΣΩΣΤΟ**

Η λέξη κλειδί '**static**' διατηρεί τον πίνακα ***keep***, ακόμη και μετά την έξοδο από τη συνάρτηση.

```
float* setKey(int s)  /* make a cryptographic key */
{
    static float keep[10];
    int i;
    srand(s);  /* set rand's seed */
    for(i=0; i<10; i++) keep[i] = rand();
    return(keep);
}
```

**επιστροφή(δείκτης)→→Πάντοτε** έλεγξε για ***static***!

# Τέλος Ενότητας