ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

Η γλώσσα C++ σε βάθος

2η αναθεωρημένη έκδοση

Γνωρίστε τη φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού μέσα από τη γλώσσα C++

Νίκος Μ. Χατζηγιαννάκης

Ενδεικτικές σελίδες από κάθε κεφάλαιο του βιβλίου

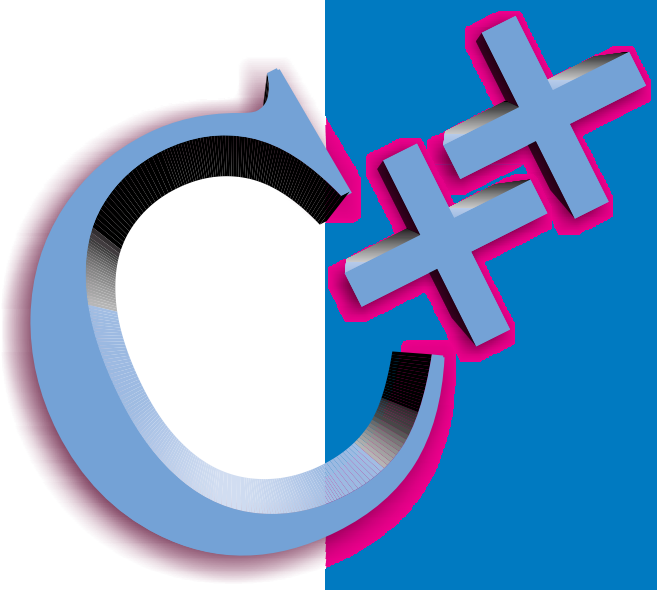
Περιέχει δωρεάν CD-ROM

Αποσπώμενη Κάρτα Αναφοράς

ΚΛΕΙΔΑΡΙΘΜΟΣ

Κεφάλαιο 1

Εισαγωγή



Εισαγωγή

Ξεκίνησα τη "θητεία" μου ως προγραμματιστής όταν ακόμα τα προγράμματα γράφονταν σε διατρητικές μηχανές και όχι με το πληκτρολόγιο. Σε όλα αυτά τα χρόνια ασχολήθηκα με διάφορες γλώσσες προγραμματισμού, ξεκινώντας από τις γλώσσες της εποχής εκείνης, όπως η θρυλική FORTRAN, η ακαδημαϊκή PASCAL, η επαναστατική ALGOL και η συναρπαστική γλώσσα μηχανής ASSEMBLY. Αργότερα, ως μεταπτυχιακός φοιτητής πια, χρησιμοποίησα τη BCPL και την κλασική C για τον προγραμματισμό συστημάτων, ενώ αργότερα ασχολήθηκα με γλώσσες οπτικού προγραμματισμού, όπως η VISUAL BASIC, αλλά και με γλώσσες 4ης γενιάς, όπως η PL-SQL.

Δεν πέρασε πολύς καιρός και βρέθηκα αντιμέτωπος με μια νέα πρόκληση: μια γλώσσα αντικειμενοστρεφούς προγραμματισμού η οποία δεν ήταν άλλη από τη C++.

Δεν άργησα να καταλάβω ότι θα έπρεπε να ξεχάσω όλα όσα ήξερα, να μπω σε μια διαφορετική λογική και να αλλάξω τη φιλοσοφία του σχεδιασμού και της ανάπτυξης των προγραμμάτων μου.

Η πρόκληση ήταν μεγάλη, η προσπάθεια συνεχής, αλλά το αποτέλεσμα πάντα το ίδιο: επέστρεφα πάντα στις «κλασικές» γλώσσες προγραμματισμού αρνούμενος να υιοθετήσω τη νέα αυτή λογική.

Τελικά βρήκα τη λύση στη φράση που γνωρίζει πολύ καλά ο κάθε εκπαιδευτικός: "Αν θες να πιστέψεις σε κάτι και να το μάθεις σε βάθος, θα πρέπει πρώτα να το διδάξεις". Έτσι λοιπόν άρχισα να διδάσκω τη C++ και τις αρχές του αντικειμενοστρεφούς προγραμματισμού, χωρίς μέχρι τότε να τις έχω υιοθετήσει ούτε εγώ ο ίδιος.

Πολύ γρήγορα συνειδητοποίησα ότι ο αντικειμενοστρεφής προγραμματισμός είναι η πιο συναρπαστική, αλλά και μυστηριώδης, καινοτομία στην ανάπτυξη λογισμικού στην ιστορία της επιστήμης των υπολογιστών. Παρόλο που διατηρεί αρκετά από τα χαρακτηριστικά των "παραδοσιακών" γλωσσών προγραμματισμού, εισάγει πρωτόγνωρες έννοιες και τεχνικές, οι οποίες όμως τις περισσότερες φορές δύσκολα γίνονται κατανοητές.

Ο αρχικός δισταγμός μου και η δυσκολία που αντιμετώπισα στην πρώτη μου επαφή με μια αντικειμενοστρεφή γλώσσα ήταν το βασικό μου κίνητρο για την συγγραφή αυτού του βιβλίου.

Το βιβλίο αυτό δεν προϋποθέτει γνώσεις προγραμματισμού. Απευθύνεται τόσο στον αρχάριο σπουδαστή όσο και στον έμπειρο προγραμματιστή που θέλει να γνωρίσει τις αρχές και τη φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού, μέσα από μια ευέλικτη και χωρίς όρια γλώσσα όπως η C++.

Αν όμως ο αναγνώστης γνωρίζει μια άλλη γλώσσα προγραμματισμού, και ιδιαίτερα τη C, θα μπορέσει να κατανοήσει γρηγορότερα τις βασικές έννοιες ώστε να επικεντρωθεί στα αντικειμενοστρεφή χαρακτηριστικά της γλώσσας.

Μεγάλο βάρος έχει δοθεί στη διδακτική σειρά αυτού του βιβλίου ώστε η ανάγνωση και η κατανόηση ενός κεφαλαίου να προϋποθέτει **μόνο** τις γνώσεις που αποκτήθηκαν στα προηγούμενα κεφάλαια.

Ο στόχος του βιβλίου και σε ποιους απευθύνεται

Μέσα από αυτό το βιβλίο προσπαθώ να δώσω στον ανυποψίαστο αναγνώστη αυτό που θα ήθελα να είχα εγώ όταν ήλθα για πρώτη φορά σε επαφή με μια γλώσσα αντικειμενοστρεφούς προγραμματισμού: Ένα βιβλίο το οποίο, χωρίς να πλατειάζει με τις αναμφισβήτητα αμέτρητες δυνατότητες μιας γλώσσας, να εισάγει με απλό και κατανοητό τρόπο τον αναγνώστη στις βασικές αρχές της φιλοσοφίας και της χρήσης του αντικειμενοστρεφούς προγραμματισμού.

> Το παρόν βιβλίο δεν αποτελεί ένα ακόμη εγχειρίδιο της γλώσσας, αλλά ένα εργαλείο εκπαίδευσης στις αρχές και τη φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού μέσα από την πιο δημοφιλή σήμερα γλώσσα, τη C++.

Το βιβλίο απευθύνεται κυρίως σε όσους διδάσκονται τη C++ ως πρώτη ή ως μεταγενέστερη γλώσσα προγραμματισμού σε εκπαιδευτικά ιδρύματα κάθε βαθμίδας, αλλά και σε όσους θα ήθελαν να μάθουν και να κατανοήσουν μόνοι τους τη γλώσσα C++ και τη φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού.

Πώς να διαβάσετε αυτό το βιβλίο

Αν γνωρίζετε οποιαδήποτε άλλη γλώσσα προγραμματισμού, μπορείτε να ρίξετε μια γρήγορη ματιά σε αυτό το κεφάλαιο και μετά να προχωρήσετε κατευθείαν στο επόμενο. Η C++, όπως υποδηλώνει και το όνομά της αλλά και όπως θα γίνει αντιληπτό και αργότερα, έχει στενή συγγένεια με τη γλώσσα C. Η C++ έχει δανειστεί πολλά από τα χαρακτηριστικά της C, οπότε ένας αναγνώστης που έχει έστω και μικρή γνώση της γλώσσας C ξεκινά με σαφές πλεονέκτημα.

Το Κεφάλαιο 2 είναι μια μικρή "περιοδεία" στη γλώσσα. Γίνεται μια πρώτη γνωριμία, ώστε να αποκτήσετε μια σφαιρική εικόνα για τη δομή και τα χαρακτηριστικά της για να μπορέσετε να δημιουργήσετε τα πρώτα σας απλά προγράμματα. Τα Κεφάλαια 3 έως 12 αναλύουν τα μη αντικειμενοστρεφή χαρακτηριστικά της γλώσσας, δίνοντας έμφαση στην παρουσίαση των βασικών, αλλά και των περισσότερο πολύπλοκων, εννοιών, με απλό, εποπτικό και κατανοητό τρόπο.

Αν έχετε λίγες γνώσεις της γλώσσας C, μπορείτε να κάνετε μια γρήγορη ανάγνωση των Κεφαλαίων 3-6 και να δώσετε βάρος στα επόμενα κεφάλαια. Αν είστε έμπειρος χρήστης της γλώσσας C, μπορείτε να κάνετε μια γρήγορη ανάγνωση των Κεφαλαίων 3-11 ώστε

να γνωρίσετε τα λιγοστά αντικειμενοστρεφή στοιχεία της γλώσσας C++ που χρησιμοποιούνται σε αυτά τα κεφάλαια.

Αρκετές ενότητες σε αυτά τα πρώτα κεφάλαια του βιβλίου είναι παρόμοιες με τις αντίστοιχες ενότητες του βιβλίου μου *Η γλώσσα C σε βάθος*. Αυτό είναι μάλλον αναμενόμενο αφού αναφέρονται σε κοινές δυνατότητες των δύο γλωσσών. Στόχος του παρόντος βιβλίου είναι να ρίξει το βάρος στα αντικειμενοστρεφή χαρακτηριστικά της C++ και στη φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού. Έτσι, σε αρκετά σημεία του βιβλίου όπου απαιτείται περισσότερη εμβάθυνση σε έννοιες και τεχνικές διαδικαστικού προγραμματισμού, παρέχονται παραπομπές προς το βιβλίο *Η γλώσσα C σε βάθος*.

Τα επόμενα κεφάλαια, από το 13 και μετά, είναι μια βαθιά "βουτιά" στα αντικειμενοστρεφή χαρακτηριστικά της γλώσσας, σε τεχνικές προγραμματισμού και σε δομές δεδομένων.

Το Παράρτημα Α περιέχει μια συνοπτική αναφορά στις πιο συχνά χρησιμοποιούμενες συναρτήσεις της C++. Στο Παράρτημα Β θα βρείτε λεπτομέρειες για το περιβάλλον ανάπτυξης **Code::Blocks**[1]. Το **Code::Blocks** είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment – IDE) για τις γλώσσες C και C++, το οποίο διατίθεται κάτω από τη γενική άδεια χρήσης GNU[2] GPL v3.0 το οποίο σημαίνει ότι διατίθεται δωρεάν στον οποιονδήποτε. Όλα τα προγράμματα που υπάρχουν σε αυτό το βιβλίο έχουν δοκιμαστεί στο περιβάλλον του **Code::Blocks**.

Στο συνοδευτικό CD θα βρείτε το πρόγραμμα εγκατάστασης για το ολοκληρωμένο περιβάλλον του **Code::Blocks**, τον κώδικα των περισσοτέρων προγραμμάτων που βρίσκονται σε αυτό το βιβλίο, καθώς και τις απαντήσεις όλων των ασκήσεων. επίσης, αποκλειστικά για το βιβλίο αυτό, έχει δημιουργηθεί ένας δικτυακός τόπος στη διεύθυνση **http://cpp.bytes.gr** με πλήθος αναφορών για τον αντικειμενοστρεφή προγραμματισμό και τη γλώσσα C++. Σας συνιστώ να εγκαταστήσετε άμεσα το περιβάλλον του **Code::Blocks** ώστε να μπορείτε να δοκιμάζετε τα παραδείγματα του βιβλίου, αλλά και τα δικά σας προγράμματα.

Το σύμβολο ⊙ δίπλα από τον τίτλο μιας ενότητας σημαίνει ότι η ενότητα αυτή περιγράφει εξειδικευμένα θέματα και μπορεί να παραλειφθεί αρχικά αν δεν επιθυμούμε ιδιαίτερη εμβάθυνση.

[1] http://www.codeblocks.org

[2] http://www.gnu.org

Γιατί να μάθω C++;

Όταν αποφασίσει κάποιος να ασχοληθεί με τον αντικειμενοστρεφή προγραμματισμό (ΑΠ), ή όταν αποφασίσει να τον διδάξει, έρχεται αντιμέτωπος με ένα δίλημμα: ποια γλώσσα να επιλέξει! Οι πιο διαδεδομένες αντικειμενοστρεφείς γλώσσες που χρησιμοποιούνται σήμερα είναι η C++, η Java και η C#. Γιατί όμως να επιλέξει τη C++ και όχι κάποια από τις άλλες δύο;

Τόσο η Java όσο και η C# δεν είναι γλώσσες που παράγουν ένα άμεσα εκτελέσιμο πρόγραμμα, αλλά έναν ενδιάμεσο κώδικα ο οποίος εκτελείται από μια εικονική μηχανή (virtual machine). Αυτό τις κάνει πολύ φορητές και ιδανικές για διαδικτυακές εφαρμογές και εφαρμογές μικροσυσκευών, αλλά παράγουν προγράμματα που εκτελούνται πιο αργά και διαθέτουν λιγότερες δυνατότητες ελέγχου στο σύστημα όπου εκτελούνται.

Αν επομένως θέλουμε προγράμματα στα οποία μας ενδιαφέρει η υψηλή απόδοση και ο πλήρης έλεγχος του συστήματος, η C++ αποτελεί μονόδρομο. Εξάλλου, ένα μεγάλο μέρος των δημοφιλέστερων προγραμμάτων και λειτουργικών συστημάτων που χρησιμοποιούμε σήμερα, όπως τα Windows και όλες οι εφαρμογές του Office, είναι γραμμένα σε C++[3].

Ας έλθουμε τώρα στην εκπαίδευση του αντικειμενοστρεφούς προγραμματισμού. Εκεί, κατά τη γνώμη μου, η C++ έχει αδιαφιλονίκητο προβάδισμα. Η C++ είναι μια «συσκευασία» δύο γλωσσών σε μία! Διαθέτει όλα τα διαδικαστικά χαρακτηριστικά της C και επιπλέον δικά της αντικειμενοστρεφή χαρακτηριστικά. Με τη C++ μπορούμε να ξεκινήσουμε φτιάχνοντας απλές διαδικαστικές εφαρμογές και να προσθέτουμε βήμα-βήμα τα αντικειμενοστρεφή στοιχεία της γλώσσας. Αυτό κάνει τη διαδικασία μάθησης της αντικειμενοστρεφούς λογικής σταδιακή και λιγότερο επίπονη.

Τόσο η Java όσο και η C# μας ρίχνουν κατευθείαν σε βαθιά νερά. Για να φτιάξουμε το απλούστερο προγραμματάκι θα πρέπει να έχουμε κατανοήσει πολύπλοκες αρχές του αντικειμενοστρεφούς προγραμματισμού. Είναι σαν να μας ρίχνουν στο λιμάνι για να μάθουμε να κολυμπάμε. Το πιθανότερο είναι ότι, αν δεν πνιγούμε, θα πάρουμε από φόβο το νερό και δεν θα μάθουμε ποτέ σωστό κολύμπι.

Αρκετά εκπαιδευτικά ιδρύματα του εξωτερικού, σε μια προσπάθεια να επιβάλουν από την αρχή την αντικειμενοστρεφή λογική στον σχεδιασμό προγραμμάτων, έβαλαν στο πρόγραμμα σπουδών τους ως πρώτο μάθημα προγραμματισμού τη γλώσσα Java. Τα αποτελέσματα ήταν απογοητευτικά και τα περισσότερα ιδρύματα γύρισαν πίσω στις πιο κλασικές γλώσσες.

> Η C++ βοηθάει στη σταδιακή κατανόηση των αρχών του αντικειμενοστρεφούς προγραμματισμού, που είναι απαραίτητες για τη μετέπειτα μετάβαση σε πιο «καθαρόαιμες» γλώσσες αντικειμενοστρεφούς προγραμματισμού, όπως η Java και η C#.

[3] Πηγή: The Programming Languages Beacon, http://www.lextrait.com/vincent/implementations.html

Ο δομημένος διαδικαστικός προγραμματισμός επιτρέπει την απόκρυψη των δεδομένων από διαφορετικά τμήματα του προγράμματος. Μια διαδικασία μπορεί να έχει τα δικά της τοπικά δεδομένα στα οποία έχει πρόσβαση μόνο η ίδια, ενώ στα καθολικά δεδομένα έχουν άμεση πρόσβαση όλες οι διαδικασίες του προγράμματος. Όταν το κύριο πρόγραμμα ή μια διαδικασία καλεί μια άλλη διαδικασία, έχει τη δυνατότητα να της μεταβιβάσει δεδομένα (στα οποία δε θα είχε πρόσβαση) κατά την ώρα της κλήσης της. Επίσης, μια διαδικασία μπορεί να επιστρέφει επεξεργασμένα δεδομένα στο πρόγραμμα που την κάλεσε.

Τμηματικός προγραμματισμός

Στον τμηματικό (ή αρθρωτό) προγραμματισμό, το πρόγραμμα χωρίζεται σε ξεχωριστά τμήματα (ή αρθρώματα – modules), τα οποία βρίσκονται σε διαφορετικά αρχεία πηγαίου κώδικα, και κάθε ένα μπορεί να περιέχει πολλές διαδικασίες. Κάθε τμήμα (module) έχει τα δικά του καθολικά δεδομένα στα οποία έχουν πρόσβαση μόνο οι διαδικασίες του συγκεκριμένου τμήματος. Με αυτόν τον τρόπο έχουμε μεγαλύτερο βαθμό **αφαιρετικότητας** και τα δεδομένα είναι προσπελάσιμα μόνο από τις διαδικασίες που τα χρειάζονται. Τόσο στον δομημένο διαδικαστικό προγραμματισμό, όσο και στον τμηματικό προγραμματισμό, τα δεδομένα και οι διαδικασίες αποτελούν διαφορετικές οντότητες. Οι διαδικασίες πρέπει να έχουν πρόσβαση στα δεδομένα, αλλά τα δεδομένα θα πρέπει ταυτόχρονα να προστατεύονται από μη εξουσιοδοτημένη χρήση. Η αφαιρετικότητα βοηθάει, αλλά δεν λύνει τα προβλήματα του διαχωρισμού δεδομένων και διαδικασιών.

Αντικειμενοστρεφής προγραμματισμός

Ο αντικειμενοστρεφής προγραμματισμός συγκεντρώνει όλα τα στοιχεία του δομημένου και του τμηματικού προγραμματισμού, καταργώντας το χάσμα που υπάρχει μεταξύ δεδομένων και διαδικασιών, εισάγοντας παράλληλα νέες έννοιες και μια διαφορετική φιλοσοφία όσον αφορά τον σχεδιασμό και την ανάπτυξη των προγραμμάτων.

Μια από τις καθημερινές μου συνήθειες είναι να φτιάχνω τον πρωινό μου καφέ. Ας προσπαθήσουμε να αναλύσουμε λίγο αυτή τη διαδικασία. Η πρώτη ύλη που διαθέτουμε (τα δεδομένα μας) είναι ο καφές, η ζάχαρη, το νερό και το γάλα, αποθηκευμένα στα ντουλάπια και στο ψυγείο της κουζίνας μας. Εμείς παίρνουμε τα υλικά που χρειαζόμαστε με απευθείας πρόσβαση στους αποθηκευτικούς μας χώρους και κάνουμε την απαραίτητη επεξεργασία για να φτιάξουμε τον καφέ μας. Ο συγκάτοικός μας έχει και αυτός πρόσβαση στα ίδια υλικά και φτιάχνει και αυτός τον καφέ του ακολουθώντας την ίδια ή διαφορετική διαδικασία.

Σύμφωνα με την παραπάνω διαδικασία

- Τα υλικά μας (δεδομένα) είναι εκτεθειμένα στον οποιονδήποτε που έχει πρόσβαση στους αποθηκευτικούς μας χώρους (ντουλάπια, ψυγείο). Έτσι μπορεί να γίνει αλόγιστη χρήση των υλικών ακόμη και για διαφορετικό σκοπό (π.χ. να ξοδέψουμε ζάχαρη και γάλα για να φτιάξουμε ένα κέικ).

- Η επεξεργασία των υλικών για την κατασκευή του καφέ δεν είναι αυτοματοποιημένη και ο καθένας μπορεί να ακολουθεί μια εντελώς διαφορετική διαδικασία.
- Αν αλλάξει ο χώρος που αποθηκεύουμε τα υλικά (π.χ. αν τα βάλουμε σε άλλο ντουλάπι), πρέπει να αλλάξουμε και τη διαδικασία επεξεργασίας ώστε να τα παίρνουμε από το νέο αποθηκευτικό χώρο.
- Στην περίπτωση που θέλουμε να φτιάχνουμε διαφορετικά είδη καφέ (π.χ. καπουτσίνο, φραπέ, εσπρέσο), θα πρέπει κάθε φορά να χρησιμοποιούμε διαφορετική διαδικασία επεξεργασίας.
- Αν θελήσουμε να φτιάξουμε καφέ όχι στο σπίτι μας αλλά σε έναν άλλο χώρο, θα πρέπει αρχικά να μάθουμε πού είναι τα υλικά μας (τα δεδομένα μας), ενώ ίσως χρειαστεί να τροποποιήσουμε τη διαδικασία κατασκευής του καφέ ανάλογα με τις δυνατότητες του χώρου στον οποίο βρισκόμαστε. Για παράδειγμα, ανάλογα με το αν ο χώρος διαθέτει υγραέριο, ηλεκτρική κουζίνα, ή φούρνο μικροκυμάτων θα ακολουθήσουμε διαφορετική διαδικασία για να ζεστάνουμε το νερό.

Μετά από αρκετή υπομονή, η κατάσταση δεν πήγαινε άλλο. Πότε ανακάλυπτα την τελευταία στιγμή ότι τελείωνε ο καφές γιατί τον είχε χρησιμοποιήσει ο συγκάτοικος μου, πότε η ζάχαρη γιατί έκανε γλυκό η φίλη μου, πότε δεν έβρισκα καθόλου τα υλικά γιατί η καθαρίστρια αποφάσιζε να τα βάλει σε διαφορετικό ντουλάπι. Και δε φτάνουν όλα αυτά, κάθε φορά ο καφές ήταν διαφορετικός, είτε γιατί έπεφτε λίγο περισσότερη ζάχαρη είτε γιατί δεν υπολόγιζα σωστά το γάλα! Σκεφτόμουν, λοιπόν, μήπως υπάρχει κάποιο αντικείμενο που να φτιάχνει καφέ και να με βγάλει από όλη αυτή τη διαδικασία. Έτσι αποφάσισα να αγοράσω μια μηχανή για καφέ και αμέσως λύθηκαν όλα μου τα προβλήματα:

- Τα υλικά πλέον ήταν μέσα στη μηχανή. Κανείς δεν είχε πρόσβαση σε αυτά και κανείς δε διανοείτο να διαρρήξει τη μηχανή για να βγάλει λίγο καφέ ή ζάχαρη.
- Ποτέ πια δε χρειάστηκε να ψάχνω πού είναι ο καφές, η ζάχαρη, ή το γάλα· ενώ η ίδια η μηχανή με ειδοποιούσε όταν κόντευαν να τελειώσουν τα υλικά που περείχε.
- Το μόνο που χρειαζόταν να κάνω ήταν να πατώ ένα κουμπί, οπότε ο καφές φτιαχνόταν χωρίς ιδιαίτερο κόπο και, το κυριότερο, πάντα με τις ίδιες αναλογίες.
- Η μηχανή είχε τη δυνατότητα να φτιάχνει καφέ διαφόρων ειδών. Εγώ απλώς πατούσα το σωστό κουμπί χωρίς να με απασχολεί η διαφορετική διαδικασία που ακολουθούσε η μηχανή για να φτιάξει τον καφέ που ζήτησα.
- Και, φυσικά, όταν πήγαινα σε κάποιο άλλο χώρο, απλώς έπαιρνα τη μηχανή μαζί μου ή φρόντιζα να υπάρχει μια ακριβώς ίδια.

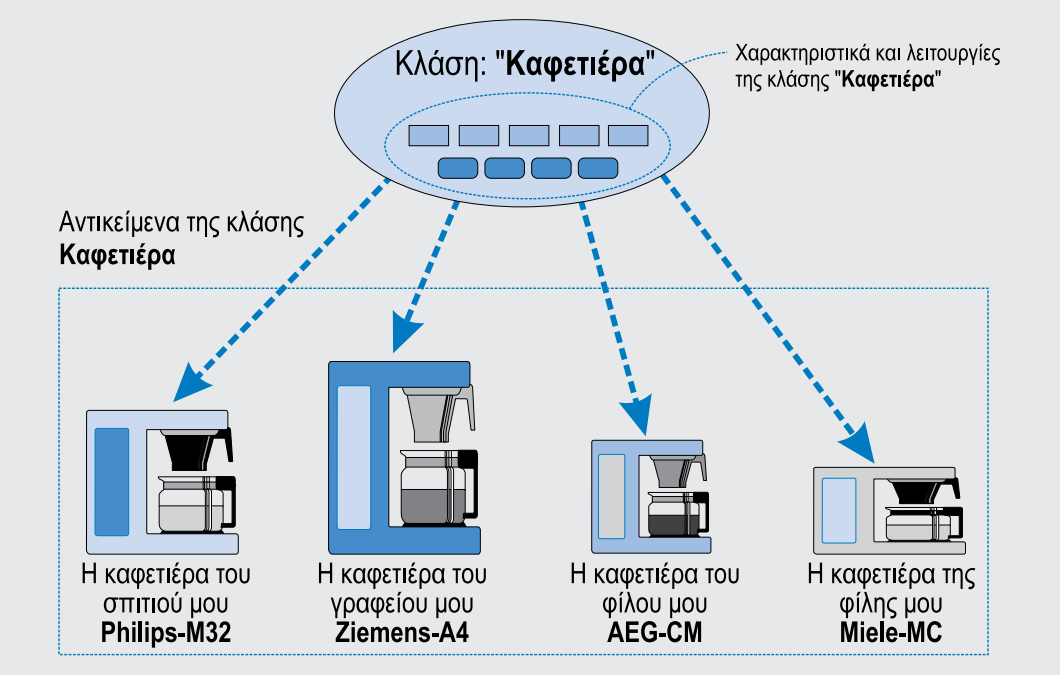
Συγχαρητήρια, μόλις αντιληφθήκατε τις βασικές αρχές και τα πλεονεκτήματα του αντικειμενοστρεφούς προγραμματισμού.

Κλάσεις και αντικείμενα

Παρατηρούμε ότι, από τη στιγμή που αποφασίσαμε να κατασκευάσουμε (ή να αγοράσουμε) ένα αντικείμενο (καφετιέρα) το οποίο έκανε τη εργασία που μέχρι εκείνη τη στιγμή διεκπεραιώναμε με ξεχωριστές διαδικασίες, λύθηκαν τα περισσότερα προβλήματά μας. Μήπως λοιπόν η καλύτερη προσέγγιση και στην κατασκευή των προγραμμάτων βρίσκεται στη χρήση **αντικειμένων**; Τα προγράμματα με τις παραδοσιακές διαδικαστικές τεχνικές προγραμματισμού βασίζονται περισσότερο στις διαδικασίες παρά στα δεδομένα. Ας επισημάνουμε επίσης ότι ο φυσικός και ο τεχνητός κόσμος στον οποίο ζούμε είναι πιο κοντά στη φιλοσοφία του **αντικειμενοστρεφούς** παρά του διαδικαστικού προγραμματισμού. Οι **κλάσεις** (classes) και τα **αντικείμενα** (objects) είναι έννοιες πάνω στις οποίες βασίζεται όλη η φιλοσοφία και τα χαρακτηριστικά των αντικειμενοστρεφών γλωσσών προγραμματισμού.

> Η **κλάση** (class) είναι μια έννοια η οποία προσδιορίζει μια κατηγορία αντικειμένων και ταυτόχρονα περιγράφει τα χαρακτηριστικά και τις λειτουργίες τους.

Για παράδειγμα, η κλάση «Καφετιέρα» περιγράφει μια κατηγορία συσκευών για την κατασκευή καφέ (Σχήμα 1.4).



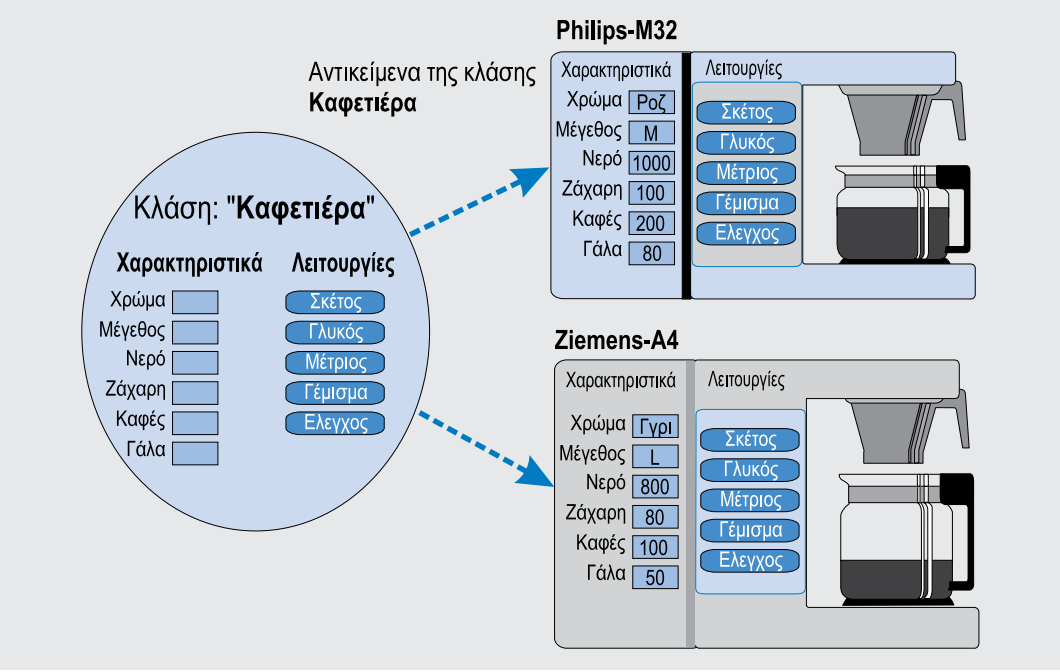
Σχήμα 1.4 *Παράδειγμα κλάσης: η κλάση «Καφετιέρα»*

Όλα τα αντικείμενα αυτής της κλάσης έχουν κάποια κοινά χαρακτηριστικά και λειτουργίες. Π.χ. όλες οι καφετιέρες έχουν χώρο για νερό, καφέ, γάλα και ζάχαρη. Επίσης όλες διαθέτουν λειτουργία για να φτιάχνουν καφέ σκέτο, γλυκό ή μέτριο.

Ένα **αντικείμενο** (object) μιας κλάσης έχει όλα τα χαρακτηριστικά και τις λειτουργίες της κλάσης στην οποία ανήκει. Ένα αντικείμενο λέμε ότι αποτελεί ένα **στιγμιότυπο** (instance) μιας κλάσης.

Μια κλάση δεν αναφέρεται σε κανένα συγκεκριμένο αντικείμενο, αλλά είναι μια γενική έννοια που προσδιορίζει μια κατηγορία αντικειμένων. Π.χ. η κλάση "Αυτοκίνητο" δεν αναφέρεται σε κανένα συγκεκριμένο αυτοκίνητο, αλλά αποτελεί μια έννοια που προσδιορίζει μια κατηγορία οχημάτων. Στο Σχήμα 1.5 απεικονίζεται πιο αναλυτικά η κλάση «Καφετιέρα» και δύο αντικείμενα που ανήκουν σε αυτή την κλάση.

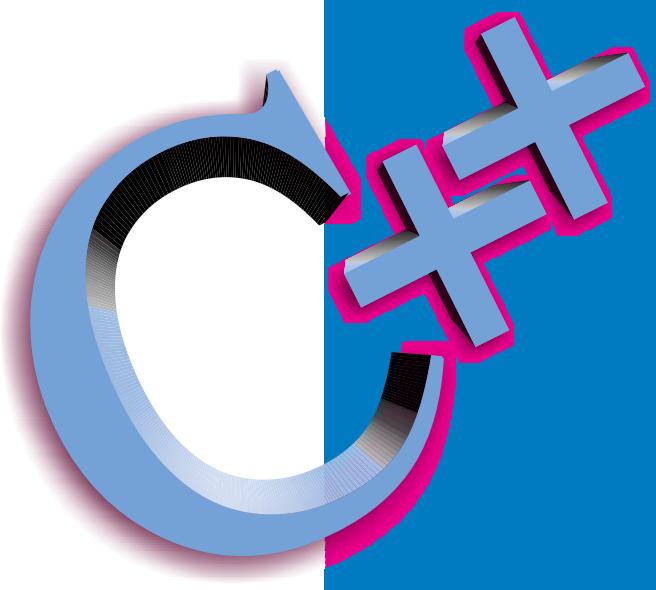
Στη συγκεκριμένη περίπτωση, η κλάση προσδιορίζει τα χαρακτηριστικά, όπως το χρώμα, το μέγεθος, τις ποσότητες νερού, ζάχαρης, καφέ και γάλακτος, που θα έχουν τα αντικείμενα χωρίς όμως να αναφέρονται συγκεκριμένες τιμές. Παρατηρούμε ότι τα δύο αντικείμενα της κλάσης έχουν όλα τα παραπάνω χαρακτηριστικά, **αλλά με συγκεκριμένες τιμές το καθένα**.



Σχήμα 1.5 *Αντικείμενα μιας κλάσης: αντικείμενα της κλάσης «Καφετιέρα»*

Κεφάλαιο 2

Μια πρώτη ματιά στη C++



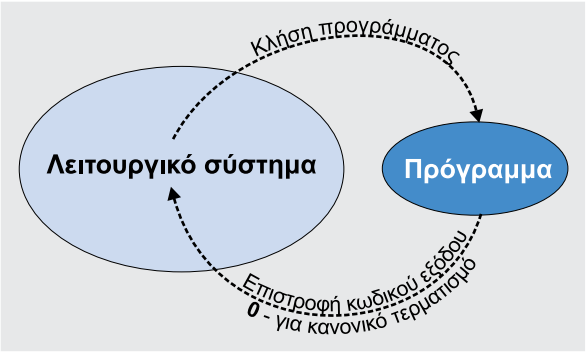
Μια πρώτη ματιά στη C++

Στο κεφάλαιο αυτό ξεκινά μια μικρή περιοδεία στη γλώσσα C++. Γίνεται μια πρώτη γνωριμία ώστε ο αναγνώστης να αποκτήσει μια σφαιρική εικόνα της δομής και των βασικών χαρακτηριστικών της γλώσσας.

Τα περισσότερα από όσα αναφέρονται στο παρόν κεφάλαιο θα αναπτυχθούν εκτενέστερα στα επόμενα κεφάλαια.

Πριν ακόμα προχωρήσουμε σε αυτή τη πρώτη γνωριμία, ας δούμε τη διαδικασία με την οποία εκτελείται ένα οποιοδήποτε πρόγραμμα στον Η/Υ μας. Η διαδικασία αυτή θα μας βοηθήσει να κατανοήσουμε κάποιες έννοιες που θα συναντήσουμε αμέσως μετά.

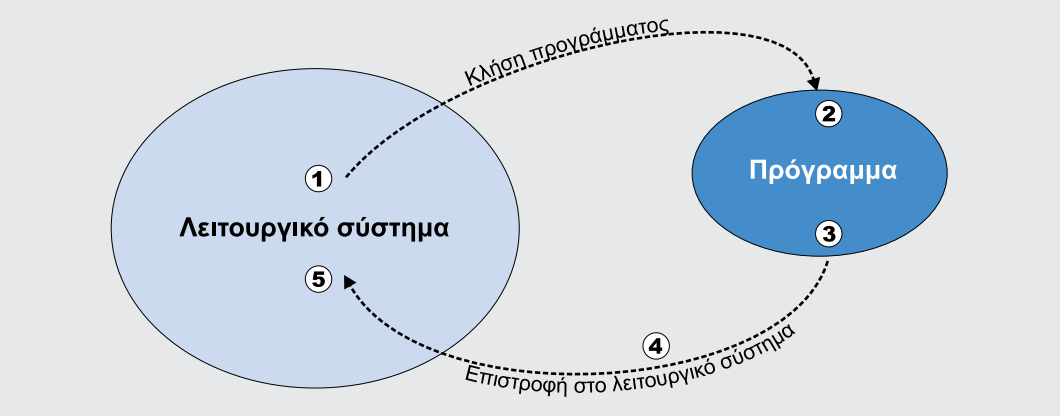
Γνωρίζουμε ότι οποιοδήποτε σύστημα Η/Υ βρίσκεται υπό τον έλεγχο του λειτουργικού του συστήματος (π.χ. Windows, Linux, OS X, κ.λπ.). Η εκτέλεση ενός προγράμματος συνήθως ξεκινάει από μια ενέργεια του χρήστη (π.χ. με ένα κλικ του ποντικιού), και μετά το λειτουργικό σύστημα φορτώνει το πρόγραμμα στη μνήμη του Η/Υ και μας βάζει στο περιβάλλον του. Με τον τερματισμό του προγράμματος επιστρέφουμε πίσω στο περιβάλλον του λειτουργικού συστήματος. Συνήθως, το πρόγραμμα μετά τον τερματισμό του επιστρέφει στο λειτουργικό σύστημα έναν κωδικό (κωδικός εξόδου), ο οποίος είναι απλώς ένας ακέραιος αριθμός. Ο **κωδικός εξόδου** δείχνει αν το πρόγραμμα τερματίστηκε κανονικά ή παρουσίασε κάποιο πρόβλημα.



Ο κωδικός **0** σημαίνει ότι το πρόγραμμα τερματίστηκε κανονικά. Οποιοσδήποτε άλλος αριθμός υποδηλώνει πρόβλημα και ανορθόδοξο τερματισμό του προγράμματος! Αυτός ο κωδικός αργότερα ελέγχεται από το λειτουργικό σύστημα, το οποίο με αυτόν τον τρόπο γνωρίζει αν το πρόγραμμα εκτελέστηκε κανονικά. Στην περίπτωση προβλήματος (κωδικός εξόδου διάφορος του μηδενός), το λειτουργικό σύστημα μπορεί να κάνει κάποια ενέργεια (π.χ. να εμφανίσει κάποιο μήνυμα).

Αναλυτικά, τα στάδια της παραπάνω διαδικασίας είναι τα ακόλουθα πέντε (Σχήμα 2.1):

1. Κάποια ενέργεια του χρήστη υποδηλώνει την εκτέλεση ενός προγράμματος (π.χ. ένα διπλό κλικ επάνω σε κάποιο εικονίδιο).
2. Το λειτουργικό σύστημα εκτελεί το πρόγραμμα και μας βάζει στο περιβάλλον του.
3. Κάποια ενέργεια του χρήστη έχει αποτέλεσμα τον τερματισμό του προγράμματος (π.χ. η εντολή **Έξοδος** από το μενού **Αρχείο**).



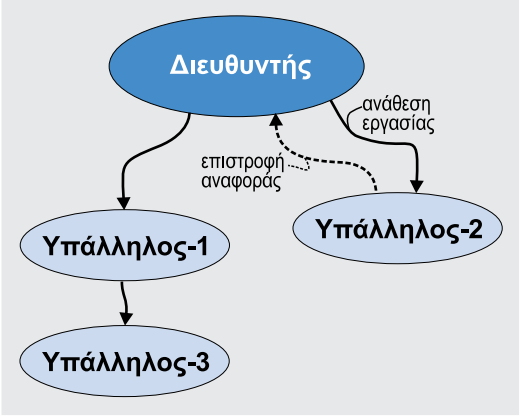
Σχήμα 2.1 *Τα στάδια της κλήσης ενός προγράμματος από το λειτουργικό σύστημα*

4. Το πρόγραμμα επιστρέφει στο λειτουργικό σύστημα τον κωδικό εξόδου, με τη λογική που αναφέρθηκε προηγουμένως.
5. Το λειτουργικό σύστημα ελέγχει τον κωδικό που επέστρεψε το πρόγραμμα και αντιδρά ανάλογα (π.χ. εμφανίζει ένα μήνυμα λάθους).

Ο **κωδικός εξόδου** είναι ένας ακέραιος αριθμός τον οποίο επιστρέφει ένα πρόγραμμα, μετά από τον τερματισμό του, στο λειτουργικό σύστημα. Ο κωδικός εξόδου εξαρτάται από τον προγραμματιστή. Αυτός αποφασίζει τους κωδικούς εξόδου που θα επιστρέφει το πρόγραμμά του, έχοντας πάντα υπόψη ότι σε περίπτωση ομαλού τερματισμού το πρόγραμμα θα πρέπει να επιστρέφει κωδικό εξόδου 0.

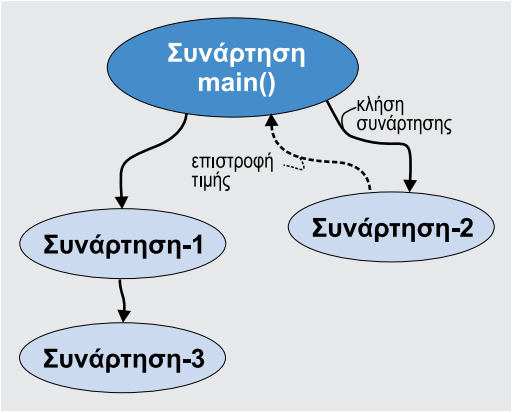
Η δομή ενός προγράμματος της C++

Σκεφτείτε τον τρόπο με τον οποίο λειτουργεί μια εταιρεία. Υπάρχει ο διευθυντής και οι υπάλληλοι που κάνουν συγκεκριμένες δουλειές. Όσο μεγαλώνει μια εταιρεία, τόσο περισσότερους υπαλλήλους χρειάζεται. Αν βέβαια η εταιρεία είναι πολύ μικρή, τότε ο μοναδικός εργαζόμενος θα μπορούσε να είναι ο ίδιος ο διευθυντής. Για να κάνει όμως μια εργασία κάποιος υπάλληλος, θα πρέπει να του την αναθέσει ο διευθυντής ή κάποιος άλλος υπάλληλος (π.χ. ο προϊστάμενός του). Στο παράδειγμα του



παραπάνω σχήματος, ο διευθυντής αναθέτει μια εργασία στον υπάλληλο-1 και στον υπάλληλο-2, ενώ ο υπάλληλος-1 αναθέτει μια εργασία στον υπάλληλο-3. Τα πάντα όμως ξεκινούν από τις εντολές του διευθυντή!

Ένας υπάλληλος μπορεί να κάνει τη δουλειά του χωρίς να επιστρέφει κάτι σε αυτόν που του την ανέθεσε, αλλά μπορεί να επιστρέψει πίσω και κάποια αναφορά. Για παράδειγμα, αν ανατεθεί σε έναν υπάλληλο να ταχυδρομήσει κάποιες επιστολές, δεν χρειάζεται να επιστρέψει κάποια αναφορά σε αυτόν που του ανέθεσε την εργασία. Αν όμως του ανατεθεί να ελέγξει τα διαθέσιμα χρήματα που υπάρχουν στο ταμείο, σίγουρα τότε θα πρέπει να επιστρέψει μια αναφορά με το ποσό που βρήκε.



Όπως έχει ήδη αναφερθεί, ένα πρόγραμμα στη C++ χωρίζεται συνήθως σε τμήματα (υπαλλήλους) κάθε ένα από τα οποία εκτελεί μια συγκεκριμένη εργασία. Τα τμήματα αυτά λέγονται **συναρτήσεις** και η κάθε συνάρτηση διαθέτει τον δικό της κώδικα εντολών. Η **συνάρτηση-διευθυντής** έχει συγκεκριμένο όνομα, και λέγεται **main()**. Για να εκτελεστεί μια συνάρτηση και να διεκπεραιώσει την εργασία για την οποία φτιάχτηκε, πρέπει να κληθεί είτε από τη main(), είτε από κάποια άλλη συνάρτηση. Στο παράδειγμα του προηγούμενου σχήματος, η main() καλεί τη συνάρτηση-1 και τη συνάρτηση-2, ενώ η συνάρτηση-1 με τη σειρά της καλεί τη συνάρτηση-3. Τα πάντα όμως ξεκινούν από τη συνάρτηση main()! Αν η συνάρτηση main() δεν καλέσει καμία άλλη συνάρτηση, τότε ο μοναδικός κώδικας που εκτελείται είναι αυτός που βρίσκεται μέσα στη main().

Τα πολύ μικρά προγράμματα, τα οποία δεν χρειάζεται να χωριστούν σε τμήματα, διαθέτουν μόνο μία συνάρτηση, τη συνάρτηση **main()**. Κάθε πρόγραμμα στη C++ διαθέτει **υποχρεωτικά μία** συνάρτηση με όνομα **main()**.

Η συνάρτηση **main()** είναι αυτή που εκτελείται **πρώτη**. Κάθε πρόγραμμα της C++, πρέπει να έχει **μία και μόνο μία** συνάρτηση main(). Η συνάρτηση main(), όπως και κάθε συνάρτηση της C++, έχει την ακόλουθη μορφή:

```
main()
{
   .....
   προτάσεις;
   .....
}
```

Δεν ξεχνάμε το ερωτηματικό στο τέλος κάθε πρότασης.

Μετά από το όνομα της συνάρτησης, ακολουθούν μέσα σε καλλιγραφικές αγκύλες ({}) οι προτάσεις, δηλαδή ο κώδικας της συνάρτησης. Κάθε πρόταση της main() τερματίζεται με **ερωτηματικό** (;).

Πέρα από τις συναρτήσεις που γράφουμε εμείς οι ίδιοι (δηλαδή τα ξεχωριστά τμήματα ενός προγράμματος), υπάρχουν και **έτοιμες συναρτήσεις** που διαθέτει η ίδια η γλώσσα, τις οποίες μπορούμε να χρησιμοποιούμε στα προγράμματά μας. Αυτές οι έτοιμες συναρτήσεις καλούνται **συναρτήσεις βιβλιοθήκης**.

Οι συναρτήσεις βιβλιοθήκης επιτελούν διάφορες εργασίες. Υπάρχουν συναρτήσεις για την έξοδο πληροφοριών στην οθόνη, για την είσοδο δεδομένων από το πληκτρολόγιο, για τον υπολογισμό μαθηματικών εννοιών, κ.λπ. Η γλώσσα C++, ως μια αντικειμενοστρεφής γλώσσα, διαθέτει βιβλιοθήκες οι οποίες πέρα από συναρτήσεις διαθέτουν επίσης έτοιμες κλάσεις και αντικείμενα.

Η **καθιερωμένη βιβλιοθήκη της C++** (C++ standard library) περιλαμβάνει εκατοντάδες έτοιμες συναρτήσεις, κλάσεις και αντικείμενα και αποτελεί αναπόσπαστο μέρος της γλώσσας.

Το πρώτο σας πρόγραμμα στη C++

Έφτασε η ώρα να γράψετε το πρώτο σας πρόγραμμα στη C++, το οποίο δεν θα μπορούσε να είναι άλλο από το κλασικό "Hello world".

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

Το πρόγραμμα αποτελείται μόνο από μία συνάρτηση, τη **main()**. Το πρόγραμμα εμφανίζει στην οθόνη τη φράση "Hello world".

Ας εξηγήσουμε μία-μία τις προτάσεις του προγράμματος.

```
#include <iostream>
```

Η πρόταση αυτή αναγκάζει τον μεταγλωττιστή της C++ να συμπεριλάβει το αρχείο **iostream** κατά τη διαδικασία της μεταγλώττισης. Το αρχείο αυτό περιέχει τις δηλώσεις των αντικειμένων και των συναρτήσεων εισόδου/εξόδου της καθιερωμένης βιβλιοθήκης της C++. Για να μπορέσουμε να χρησιμοποιήσουμε οποιαδήποτε από αυτές τις λειτουργίες εισόδου/εξόδου στο πρόγραμμά μας (π.χ. το αντικείμενο cout), θα πρέπει οπωσδήποτε να συμπεριληφθεί (**#include**) το συγκεκριμένο αρχείο. Η χρήση της οδηγίας **#include** επεξηγείται αργότερα, και αναλύεται πλήρως στα Κεφάλαια 7 και 23.

```
using namespace std;
```

Η πρόταση αυτή ενημερώνει τον μεταγλωττιστή της C++, ότι τα ονόματα που θα χρησιμοποιούνται (συναρτήσεις, αντικείμενα, κ.λπ) από εκείνο το σημείο και μετά, θα ανήκουν στον χώρο ονομάτων **std**. Η χρήση χώρων ονομάτων δίνει τη δυνατότητα στη C++ να χρησιμοποιεί οντότητες (μεταβλητές, συναρτήσεις, αντικείμενα, κ.λπ.) που έχουν το ίδιο όνομα, αλλά εκτελούν διαφορετική λειτουργία ανάλογα με το όνομα του χώρου στον οποίο ανήκουν. Αν όλα αυτά σας φαίνονται κάπως δυσνόητα, κάντε υπομονή μέχρι να εξηγηθούν οι χώροι ονομάτων, και μέχρι τότε απλά να χρησιμοποιείτε αυτή την πρόταση σε κάθε σας πρόγραμμα.

```
main()
```

Η συνάρτηση main() είναι υποχρεωτική σε κάθε πρόγραμμα της C++. Στην περίπτωση που το πρόγραμμά μας έχει μόνο μία συνάρτηση, αυτή πρέπει να είναι η main(). Αν το πρόγραμμά μας περιέχει περισσότερες από μία συναρτήσεις, η main() είναι η πρώτη που εκτελείται.

```
{
```

Η αριστερή αγκύλη σηματοδοτεί την αρχή των προτάσεων της συνάρτησης.

```
cout << "Hello world";
```

Το αντικείμενο **cout** χρησιμοποιείται για την έξοδο πληροφοριών στην οθόνη. Στη συγκεκριμένη περίπτωση, θα εμφανίσει το κείμενο "Hello world".

```
}
```

Η δεξιά αγκύλη σηματοδοτεί το τέλος των προτάσεων της συνάρτησης main().

Ας ξαναγράψουμε το πρώτο μας πρόγραμμα

Ας θυμηθούμε λίγο την εικόνα του Σχήματος 2.1 και τον τρόπο κλήσης ενός προγράμματος από το λειτουργικό σύστημα του Η/Υ μας. Έχουμε ήδη πει ότι το πρόγραμμα επιστρέφει στο λειτουργικό σύστημα έναν κωδικό εξόδου, ο οποίος είναι ένας ακέραιος αριθμός. Στην περίπτωση κανονικής λειτουργίας του προγράμματος, ο κωδικός αυτός πρέπει να είναι ο αριθμός 0. Η συνάρτηση η οποία περιέχει τον βασικό κορμό ενός προγράμματος της C++ είναι η συνάρτηση main(). Όταν τερματιστεί η εκτέλεση της main(), τερματίζεται και το πρόγραμμά μας. Επομένως, την ευθύνη της επιστροφής του κωδικού εξόδου την έχει η συνάρτηση main().

> Συνοψίζοντας, έχουμε τα εξής: α) Η συνάρτηση main() πρέπει να επιστρέφει έναν ακέραιο αριθμό (τον κωδικό εξόδου) και β) Σε περίπτωση κανονικής εκτέλεσης του προγράμματος, ο αριθμός αυτός πρέπει να είναι το 0.

Για να είμαστε λοιπόν και τυπικά σωστοί (σύμφωνα με τα τελευταία πρότυπα της γλώσσας), θα πρέπει να ξαναγράψουμε το πρώτο μας πρόγραμμα λαμβάνοντας υπόψη τα παραπάνω. Έτσι το πρόγραμμα διαμορφώνεται ως εξής:

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Hello world";
   return 0;
}
```

Το πρόθεμα **int** πριν από το όνομα της συνάρτησης **main()**, υποδηλώνει ότι ο επιστρεφόμενος **τύπος** της συνάρτησης είναι **int** και επομένως επιστρέφει έναν ακέραιο αριθμό.

Με την πρόταση **return 0;** η συνάρτηση main() **επιστρέφει**, ως **κωδικό εξόδου**, το **0**.

Η συνάρτηση main() θα πρέπει να επιστρέφει έναν ακέραιο αριθμό (τον κωδικό εξόδου), επομένως ο τύπος της θα πρέπει να είναι int.

Η τελευταία πρόταση return 0; αναγκάζει τη συνάρτηση main() να επιστρέψει τη τιμή 0 η οποία, όπως αναφέρθηκε και προηγουμένως, σημαίνει κανονικό τερματισμό του προγράμματος[5].

Από τώρα και στο εξής θα χρησιμοποιούμε την παραπάνω σύνταξη στη main().

Προγράμματα με περισσότερες συναρτήσεις

Ένα πιο σύνθετο πρόγραμμα στη C++, πέρα από τη συνάρτηση main(), μπορεί να περιλαμβάνει περισσότερες συναρτήσεις:

```
int main()
{
   προτάσεις της συνάρτησης main();
   .....
}
τύπος function1()
{
   προτάσεις της συνάρτησης function1();
   .....
}
τύπος function2()
{
   προτάσεις της συνάρτησης function2();
   .....
}
```

Ο **τύπος** μιας συνάρτησης καθορίζει το είδος της τιμής που θα επιστρέψει η συνάρτηση. Επομένως, μια συνάρτηση μπορεί να είναι τύπου **int**, **float**, **char** κ.λπ. Όταν μια συνάρτηση δεν επιστρέφει τιμή, δηλώνεται ως **void**.

[5] Το πρότυπο της C++ αναφέρει ότι η χρήση της **return 0** δεν είναι υποχρεωτική (αντίθετα από το πρότυπο της C στην οποία είναι υποχρεωτική). Αν την παραλείψουμε, η **main()** θα επιστρέψει αυτόματα την τιμή 0. Καλό είναι όμως για λόγους σαφήνειας και συμβατότητας να τη χρησιμοποιούμε.

Παραδείγματα κώδικα με παραστάσεις

Στα παραδείγματα που ακολουθούν θα χρησιμοποιήσουμε ένα αντικείμενο και μια συνάρτηση βιβλιοθήκης της C++: το αντικείμενο cout και τη συνάρτηση rand().

Δεδομένου ότι η C++ δεν έχει ενσωματωμένες εντολές για είσοδο και έξοδο πληροφοριών, τις λειτουργίες αυτές αναλαμβάνουν αντικείμενα και συναρτήσεις βιβλιοθήκης της γλώσσας. Το αντικείμενο cout χρησιμοποιείται για την εμφάνιση πληροφοριών στην οθόνη. Θα χρησιμοποιήσουμε την πιο απλή μορφή του αντικειμένου, η οποία απλώς εμφανίζει ένα σύνολο χαρακτήρων στην οθόνη:

cout << "Αυτό εμφανίζεται στην οθόνη";

Η χρήση του αντικειμένου cout με την παραπάνω σύνταξη έχει ως αποτέλεσμα την εμφάνιση του κειμένου μέσα στα εισαγωγικά στην οθόνη. Η συνάρτηση rand() δεν δέχεται ορίσματα και επιστρέφει ως τιμή έναν τυχαίο ακέραιο αριθμό. Το επόμενο πρόγραμμα υπολογίζει τον μέσο όρο των περιεχομένων των μεταβλητών a και b.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
   int a,b;
   float c;
   a = rand();
   b = 18;
   c = (a + b)/2.0;
   cout << "Τέλος";
   return 0;
}
```

Η συνάρτηση **rand()** δηλώνεται στο αρχείο κεφαλίδας **cstdlib** το οποίο θα πρέπει να συμπεριληφθεί στο πρόγραμμά μας αν θέλουμε να χρησιμοποιήσουμε τη συνάρτηση,

Στη μεταβλητή **a** καταχωρίζεται ο τυχαίος ακέραιος αριθμός που επιστρέφει η συνάρτηση **rand()**.

Οι δύο πρώτες προτάσεις είναι δηλωτικές και ενημερώνουν τον μεταγλωττιστή για την ύπαρξη των τριών μεταβλητών a, b και c. Στις μεταβλητές a και b μπορούν να καταχωριστούν ακέραιοι αριθμοί, ενώ η c μπορεί να αποθηκεύσει έναν αριθμό κινητής υποδιαστολής (με δεκαδικά ψηφία). Σε άλλες γλώσσες προγραμματισμού, οι επόμενες τρεις εκτελέσιμες προτάσεις μπορεί να θεωρηθούν εντολές ανάθεσης για τις μεταβλητές a, b και c αντίστοιχα. Στη C++ καμία από αυτές δεν θεωρείται εντολή (με την ακριβή έννοια του όρου), αλλά είναι παραστάσεις που απλώς περιέχουν τον τελεστή ανάθεσης ίσον (=).

Όπως αναφέραμε προηγουμένως, μια πρόταση σε ένα πρόγραμμα της C++ μπορεί να είναι δηλωτική ή εκτελέσιμη πρόταση. Οι παραστάσεις είναι εκτελέσιμες προτάσεις. Είναι λοιπόν συντακτικά σωστό το πρόγραμμα που ακολουθεί;

```
#include <iostream>
using namespace std;
int main()
```

```
{
   int a,b;
   5+3;
   a = 6;
   8;
   return 0;
}
```

Και όμως, είναι !!!

Η δεύτερη πρόταση αποτελείται από την παράσταση 5+3, η οποία απλώς υπολογίζεται χωρίς το αποτέλεσμά της να χρησιμοποιείται (ή να καταχωρίζεται) πουθενά. Η πρόταση αυτή δεν επηρεάζει τη λειτουργία του προγράμματος και, παρόλο που είναι εντελώς άχρηστη, είναι συντακτικά σωστή, όπως και η τελευταία (το «σκέτο» 8).

Μετατροπή τύπου κατά την ανάθεση τιμής σε μεταβλητή

Ο τελεστής ανάθεσης = καταχωρίζει μια τιμή σε μια μεταβλητή. Η τιμή μπορεί να είναι μια σταθερά ή το αποτέλεσμα μιας παράστασης. Τι γίνεται όμως στην περίπτωση που ο τύπος της τιμής είναι διαφορετικός από τον τύπο της μεταβλητής;

Ας μελετήσουμε το παρακάτω τμήμα κώδικα:

```
int a,b;
float k=7.5,l=2;
a=12.456;
b=k/l;
k=2;
```

Παρατηρούμε ότι στην πρόταση a=12.456 η τιμή είναι τύπου double ενώ η μεταβλητή τύπου int. Στην περίπτωση αυτή, η C++ μετατρέπει τη δεκαδική τιμή στον τύπο int (δηλαδή σε ακέραιο αριθμό), αποκόπτοντας έτσι τα δεκαδικά ψηφία, οπότε στη μεταβλητή a θα καταχωριστεί το 12. Στην επόμενη πρόταση, b=k/l, γίνεται κάτι ανάλογο: η τιμή της παράστασης k/l είναι ο αριθμός 3.75 τύπου float. Στη μεταβλητή b θα καταχωριστεί το ακέραιο μέρος του αριθμού, δηλαδή το 3.

Στην τελευταία πρόταση k=2 γίνεται το αντίστροφο. Η τιμή της παράστασης δεξιά του = (το 2) είναι τύπου int, ενώ η μεταβλητή k είναι τύπου float. Στη περίπτωση αυτή η C++ θα μετατρέψει την ακέραια τιμή 2 στην αντίστοιχη τιμή τύπου float (2.0) και θα την καταχωρίσει στη μεταβλητή k χωρίς καμία απώλεια πληροφοριών.

> Στην περίπτωση που η παράσταση δεξιά του τελεστή ανάθεσης = είναι διαφορετικού τύπου από τη μεταβλητή αριστερά του τελεστή, η C++ μετατρέπει την τιμή της παράστασης στον τύπο της μεταβλητής. Στην περίπτωση που η τιμή της παράστασης έχει τύπο υψηλότερης ιεραρχίας από τον τύπο της μεταβλητής, τότε υπάρχει περίπτωση απώλειας πληροφοριών.

Η μετατροπή τύπου θα αναλυθεί διεξοδικά στο Κεφάλαιο 9. Στο κεφάλαιο αυτό θα δούμε ότι, πέρα από την αυτόματη μετατροπή τύπου, μπορούμε να επιβάλουμε και ρητή μετατροπή τύπου (type casting).

Η C++ και οι αγκύλες της

Μέχρι τώρα περιοριστήκαμε στη χρήση των αγκυλών {} για την ένδειξη της αρχής (με την αριστερή αγκύλη {) και του τέλους (με τη δεξιά αγκύλη }) του συνόλου των προτάσεων μιας συνάρτησης:

```
int main()
{            ← Αρχή προτάσεων
   .........
   .........
   .........
}            ← Τέλος προτάσεων
```

Όμως οι αγκύλες δεν έχουν μόνο αυτή τη χρήση. Αντίθετα, αποτελούν ένα βασικό κομμάτι της φιλοσοφίας της C++ για τη δημιουργία δομημένων και ευανάγνωστων προγραμμάτων. Επίσης, οι αγκύλες χρησιμοποιούνται για τη δημιουργία σύνθετων προτάσεων, δηλαδή προτάσεων που αποτελούνται από περισσότερες προτάσεις, αλλά αντιμετωπίζονται από τη C++ ως μία.

Σύνθετη πρόταση (compound statement)

Σύνθετη πρόταση είναι μια ομάδα από απλές προτάσεις που περιέχονται μέσα σε αγκύλες:

```
{
   a=4;
   b=8;
   cout << "Αυτή είναι μία σύνθετη πρόταση";
}
```

☞ Μια σύνθετη πρόταση μπορεί να περιέχει οποιοδήποτε πλήθος απλών προτάσεων, αλλά και άλλες σύνθετες προτάσεις.

```
{
   a=4;
   {
      m=8;
      cout << "Αυτή είναι μία σύνθετη πρόταση";
   }
   cout << "Αυτή είναι μία ακόμη σύνθετη πρόταση";
}
```

☞ Οι απλές προτάσεις τελειώνουν πάντα με ελληνικό ερωτηματικό (;), κάτι που δεν ισχύει για τις σύνθετες.

☞ Ο μεταγλωττιστής της C++ μεταχειρίζεται μια σύνθετη πρόταση όπως οποιαδήποτε απλή πρόταση. Έτσι, με τον όρο «πρόταση» εννοείται είτε μια απλή είτε μια σύνθετη πρόταση.

☞ Όπως θα δούμε αργότερα, μέσα σε μια σύνθετη πρόταση μπορούν να δηλωθούν μεταβλητές οι οποίες όμως έχουν ισχύ μόνο μέσα στη σύνθετη πρόταση που δηλώθηκαν.

Ο προ-μεταγλωττιστής της C++

Μέσα στον πηγαίο κώδικα του προγράμματός μας είναι δυνατόν να συμπεριλάβουμε οδηγίες προς τον μεταγλωττιστή της C++. Οι οδηγίες που συναντάμε συνήθως σε ένα πρόγραμμα της C++ είναι οι οδηγίες **#include** και **#define**.

Η οδηγία #include

Η οδηγία #include αναγκάζει τον μεταγλωττιστή της C++ να συμπεριλάβει κατά τη διαδικασία της μεταγλώττισης και κάποιο άλλο πηγαίο αρχείο, στο οποίο συνήθως δηλώνονται οντότητες (αντικείμενα, συναρτήσεις, κ.λπ.) βιβλιοθήκης.

Για παράδειγμα, με την πρόταση

#include <iostream>

ο μεταγλωττιστής θα συμπεριλάβει κατά τη διαδικασία της μεταγλώττισης και το αρχείο iostream, στο οποίο δηλώνονται οι περισσότερες συναρτήσεις και τα αντικείμενα εισόδου και εξόδου της C++. Η χρήση της #include εξηγείται στο επόμενο κεφάλαιο και αναλύεται στο Κεφάλαιο 7.

☞ Προς το παρόν αρκεί να γνωρίζουμε ότι, για να χρησιμοποιήσουμε μια συνάρτηση ή κάποιο αντικείμενο βιβλιοθήκης, θα πρέπει να χρησιμοποιήσουμε την οδηγία #include για να συμπεριλάβουμε το αρχείο στο οποίο δηλώνεται.

☞ ΠΡΟΣΟΧΗ: Η οδηγία #include, όπως και όλες οι οδηγίες της C++, δεν τερματίζεται με ερωτηματικό (;).

Η οδηγία #define

Η οδηγία #define ορίζει ένα αναγνωριστικό και ένα σύνολο χαρακτήρων που θα αντικαταστήσει αυτό το αναγνωριστικό κατά τη διαδικασία της μεταγλώττισης του πηγαίου κώδικα. Η σύνταξη της οδηγίας είναι:

#define αναγνωριστικό χαρακτήρες

όπου το **αναγνωριστικό** αντικαθίσταται στο στάδιο της μεταγλώττισης από τους **χαρακτήρες**, σε όποιο σημείο του πηγαίου κώδικα και αν βρεθεί. Συνήθως χρησιμοποιούμε την οδηγία #define για να ορίσουμε κάποιες σταθερές του προγράμματός μας:

```
#include <iostream>
using namespace std;
```

```
#define PI 3.141593
int main()
{
   float aktina,emvadon;
   aktina=10;
   embadon=aktina*aktina*PI;
   return 0;
}
```

Το παραπάνω πρόγραμμα υπολογίζει το εμβαδό ενός κύκλου ακτίνας 10 μέτρων. Ο προμεταγλωττιστής της C++ αντικαθιστά κατά τη διαδικασία της μεταγλώττισης το PI με την τιμή 3.141593. Για παράδειγμα, με τις παρακάτω οδηγίες:

```
#define ONE 1
#define TWO 2
```

η πρόταση c=ONE+TWO θα είναι σωστή και θα καταχωρίσει την τιμή 3 στη μεταβλητή c.

☞ Πρέπει να γίνει κατανοητό ότι στα προηγούμενα παραδείγματα τα PI, ONE, και TWO **δεν** είναι μεταβλητές.

☞ ΠΡΟΣΟΧΗ: Η οδηγία #define δεν τερματίζεται με ερωτηματικό (;).

Παραδείγματα

Π2.1 Στο παρακάτω πρόγραμμα υπάρχουν δυο συνηθισμένα συντακτικά λάθη:

```
#include <iostream>
using namespace std;
int main()
{
   int a,b;
   a = 10;
   b = 18;
   c = 10.2;
   float c;
   c = (a + b)/2
   return 0;
}
```

- Η μεταβλητή c στην πρόταση c = 10.2 χρησιμοποιήθηκε πριν από τη δήλωσή της.
- Η πρόταση c = (a + b)/2 δεν τερματίζεται με ερωτηματικό (;).

Π2.2 Στο παρακάτω πρόγραμμα δηλώνεται η μεταβλητή **a** (η οποία δεν χρησιμοποιείται αργότερα) και καλείται η συνάρτηση βιβλιοθήκης sqrt()[7] η οποία υπολογίζει την τετραγωνική ρίζα ενός αριθμού, στη συγκεκριμένη περίπτωση του 25. Όμως η τιμή που επιστρέφει η συνάρτηση (το 5) δεν καταχωρίζεται πουθενά και χάνεται.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   float a;
   sqrt(25);
   return 0;
}
```

Στο πρόγραμμα που ακολουθεί, η τιμή που επιστρέφει η sqrt() καταχωρίζεται στη μεταβλητή a, η οποία μετά από την πρόταση **a=sqrt(25)** έχει την τιμή 5.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   float a;
   a = sqrt(25);
   return 0;
}
```

Π2.3 Στο παρακάτω πρόγραμμα, στις μεταβλητές a, b και c καταχωρίζονται αντίστοιχα το 2, το 1 και το 0.

```
#include <iostream>
using namespace std;
int main()
{
   int a,b,c;
   a = 5/2;
   b = a>0;
   c = b==2;
   return 0;
}
```

Το αποτέλεσμα της παράστασης **5/2** είναι 2 και όχι 2.5, αφού τα μέλη της είναι όλα τύπου **int**.

Η παράσταση **a>0** είναι αληθής με αποτέλεσμα 1 και η **b==2** ψευδής με αποτέλεσμα 0!

☞ Όταν μια παράσταση αποτελείται από μέλη του ίδιου τύπου, τότε το αποτέλεσμά της είναι και αυτό του ίδιου τύπου.

☞ Μια λογική παράσταση έχει πάντα ως αποτέλεσμα το 1 ή το 0.

[7] Η χρήση της συνάρτησης **sqrt()** προϋποθέτει τη συμπερίληψη του αρχείου κεφαλίδας **cmath** με την πρόταση **#include <cmath>**, μέσα στο οποίο δηλώνονται όλες οι μαθηματικές συναρτήσεις της C++.

Συχνά λάθη

- Ξεχνάμε να δηλώσουμε τη main() ως τύπου int και να βάλουμε την εντολή return 0. Αυτό είναι τυπικό (σύμφωνα με τα τελευταία πρότυπα) και όχι ουσιαστικό λάθος. Αρκετοί μεταγλωττιστές δεν θα έχουν πρόβλημα.
- Όλες οι δηλωτικές και εκτελέσιμες προτάσεις στη C++ τερματίζονται με ερωτηματικό (;). Καμιά φορά το ξεχνάμε!
- Κατά λάθος τερματίζουμε τις οδηγίες #include και #define με ερωτηματικό.
- Χρησιμοποιούμε μια μεταβλητή χωρίς προηγουμένως να την έχουμε δηλώσει.
- Κάνουμε διαίρεση με ακέραιους αριθμούς και περιμένουμε το αποτέλεσμα να είναι δεκαδικός. Για παράδειγμα, περιμένουμε ότι το αποτέλεσμα της παράστασης 5/2 θα είναι 2.5, ενώ είναι 2!
- Χρησιμοποιούμε συναρτήσεις βιβλιοθήκης χωρίς να έχουμε συμπεριλάβει (με #include) τα απαραίτητα αρχεία κεφαλίδας.

Ανασκόπηση Κεφαλαίου 2

- Κάθε πρόγραμμα της C++ περιέχει υποχρεωτικά μια συνάρτηση main() η οποία εκτελείται πρώτη. Κάθε πρόγραμμα της C++ μπορεί να περιέχει και άλλες συναρτήσεις εκτός από τη main(), οι οποίες για να εκτελεστούν πρέπει να κληθούν είτε από τη main() είτε από οποιαδήποτε άλλη συνάρτηση.
- Κάθε μεταβλητή που χρησιμοποιείται πρέπει να δηλώνεται. Συνήθως οι δηλωτικές προτάσεις των μεταβλητών τοποθετούνται στην αρχή μιας συνάρτησης.
- Οι βασικοί τύποι μεταβλητών στη C++ είναι ο τύπος int, ο τύπος float, ο τύπος double, ο τύπος char και ο τύπος bool.
- Οι αριθμητικές παραστάσεις είναι πράξεις μεταξύ αριθμητικών δεδομένων και έχουν αποτέλεσμα έναν αριθμό.
- Οι λογικές παραστάσεις χρησιμοποιούν συγκριτικούς και συνδετικούς τελεστές και έχουν αποτέλεσμα **αληθές** (τιμή 1) ή **ψευδές** (τιμή 0).
- Ο τελεστής ανάθεσης = καταχωρίζει το αποτέλεσμα της παράστασης στα δεξιά του στη μεταβλητή που βρίσκεται αριστερά του, π.χ. a=b+3
- Ο συγκριτικός τελεστής = = (δύο ίσον) δεν πρέπει να συγχέεται με τον τελεστή ανάθεσης = (ένα ίσον).
- Μια σύνθετη πρόταση αποτελείται από απλές προτάσεις μέσα σε αριστερή και δεξιά αγκύλη.

Ασκήσεις Κεφαλαίου 2

2.1 Τι θα περιέχουν οι μεταβλητές a,b, και c μετά την εκτέλεση του παρακάτω κώδικα; ★

```
#include <iostream>
using namespace std;
int main()
{
   int a,b,c=3;
   a=b=2;
   a=c+b;
   return 0;
}
```

2.2 Τι θα περιέχουν οι μεταβλητές a,b, και c μετά την εκτέλεση του παρακάτω κώδικα; ★

```
#include <iostream>
#define MM 23
using namespace std;
int main()
{
   const int c=3;
   int a,b;
   a=4+b=2;
   b=c+b+MM;
   return 0;
}
```

2.3 Εντοπίστε τα λάθη στο παρακάτω πρόγραμμα: ★★

```
#include <iostream>
#define MM 23;
using namespace std;
int main()
{
   const int c=3;
   int a,b;
   a=2;
   float d;
   d=4.3
   a=4+(b=2);
   MM=10;
   3=a;
   c=c+b+MM;
   return 0;
}
```

2.4 Τι θα περιέχουν οι μεταβλητές a,b, και c μετά την εκτέλεση του παρακάτω κώδικα; ★★

```
#include <iostream>
using namespace std;
int main()
{
   int a,b,c=3;
   a=b=2;
   a=c>b;
   b=b==1;
   return 0;
}
```

2.5 Ποια από τα παρακάτω αληθεύουν: ★★

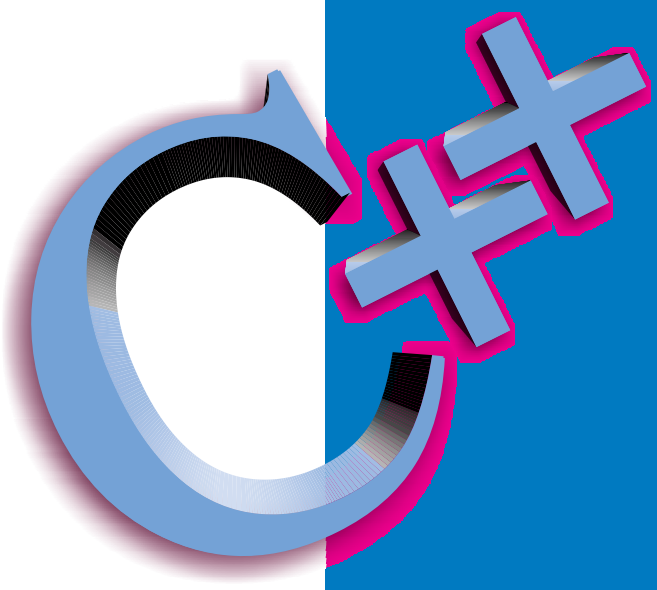
- ❑ Δηλωτικές προτάσεις μπορούν να μπουν σε οποιοδήποτε σημείο του προγράμματος.
- ❑ Ένα πρόγραμμα της C++ μπορεί να χωριστεί σε περισσότερα τμήματα (συναρτήσεις).
- ❑ Μια λογική παράσταση έχει τιμή 1 ή 0.
- ❑ Μια μεταβλητή στη C++, πριν να της δοθεί τιμή, έχει τιμή 0.
- ❑ Η οδηγία #define χρησιμοποιείται για να ορίσει μια σταθερά του προγράμματός μας.
- ❑ Η παράσταση a==5 καταχωρίζει στη μεταβλητή a το 5.
- ❑ Η παράσταση a=5==5 καταχωρίζει στη μεταβλητή a το 1.
- ❑ Η παράσταση a=a!=a καταχωρίζει στη μεταβλητή a το 0.

2.6 Εντοπίστε τα λάθη στον παρακάτω κώδικα: ★★

```
#include <iosteam>
using namespace std;
#define ONE 12
#define TWO 78
#define ΤΕΣΣΕΡΑ 4
int main()
{
   int c=3,a,b,γ;
   float c;
   b=ONE+TWO;
   cout<<"Η γλώσσα C++ σε βάθος";
   cout<<'Τέλος';
   return 0;
}
```

Κεφάλαιο 3

Προετοιμασία για αργότερα



Προετοιμασία για αργότερα

Η C++ και η μνήμη

Όπως αναφέρθηκε και προηγουμένως, η C++ χειρίζεται βασικούς τύπους μεταβλητών όπως: int, float, double, char και bool. Μπορούμε να φανταστούμε την κάθε μεταβλητή ως ένα κουτί με ένα όνομα και κάποιο περιεχόμενο. Οι μεταβλητές καταλαμβάνουν χώρο στη μνήμη RAM του Η/Υ, αλλά γνωρίζουμε ότι κάθε θέση μνήμης RAM έχει μέγεθος 1 byte.

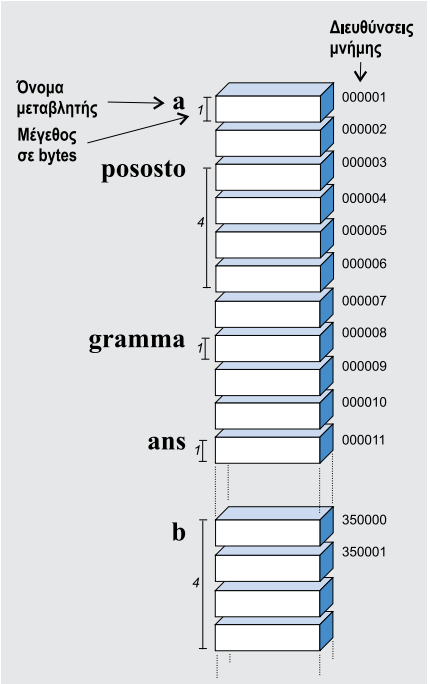
Δηλαδή, κάθε μεταβλητή του προγράμματος καταλαμβάνει ακριβώς τον ίδιο χώρο στη μνήμη του Η/Υ; **ΟΧΙ** βέβαια!

> Κάθε μεταβλητή καταλαμβάνει ένα πλήθος θέσεων μνήμης (byte), το οποίο εξαρτάται από τον τύπο της μεταβλητής, από τον τύπο του συστήματος Η/Υ στο οποίο δουλεύουμε, αλλά και από τον μεταγλωττιστή που χρησιμοποιούμε.

Στα συστήματα προσωπικών Η/Υ που βασίζονται στους επεξεργαστές 32bit της INTEL, οι μεταβλητές τύπου bool και char καταλαμβάνουν ένα byte, οι μεταβλητές int και float καταλαμβάνουν τέσσερα byte, και οι μεταβλητές double καταλαμβάνουν 8 byte. Αργότερα θα δούμε και άλλους τύπους μεταβλητών (που στηρίζονται πάντως στους βασικούς τύπους) με διαφορετικά μεγέθη σε byte.

Ας θεωρήσουμε τέσσερις μεταβλητές, οι οποίες δηλώνονται με τις παρακάτω προτάσεις:

```
char a,gramma;
float pososto;
int b;
bool ans;
```



Στο σχήμα φαίνεται εποπτικά η μνήμη ενός Η/Υ σαν ένα σύνολο από κουτιά (θέσεις μνήμης) το ένα πάνω από το άλλο. Το κάθε κουτί έχει έναν αύξοντα αριθμό (ο αριθμός στα δεξιά) ο οποίος αποτελεί τη **διεύθυνση** του κουτιού μέσα στη μνήμη. Κάθε μεταβλητή που δηλώνεται δεσμεύει κάποιες θέσεις μνήμης ανάλογα με τον τύπο της. Έτσι, η μεταβλητή **a** δεσμεύει μία θέση (την 1), η μεταβλητή **gramma** μία θέση (την 8), η **pososto** τέσσερις (τις 3,4,5, και 6), η **ans** μία θέση (την 11) και η **b** επίσης τέσσερις (τις 350000, 350001, 350002, και 350003).

Δύο αντικείμενα, μια συνάρτηση και μια εντολή

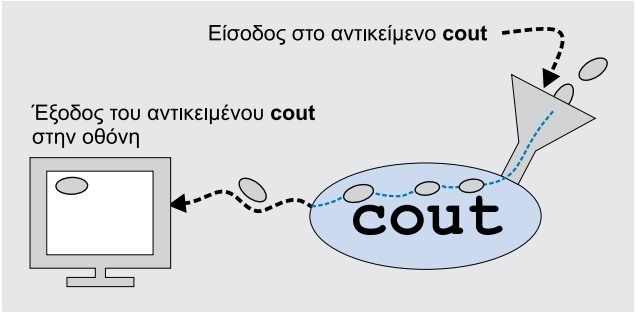
Οι ιδιαιτερότητες της C++ επιβάλλουν τη γνώση κάποιων βασικών εννοιών και τη χρήση κάποιων εντολών, συναρτήσεων και αντικειμένων, μόνο και μόνο για να μπορούμε να συνθέσουμε τα πρώτα μας απλά παραδείγματα. Έτσι, χωρίς ακόμα να εμβαθύνουμε και να αναλύσουμε πλήρως το συντακτικό και τον τρόπο χρήσης τους, θα γνωρίσουμε τα αντικείμενα **cout**, **cin,** τη συνάρτηση **exit()**, καθώς και την εντολή **if**.

Είσοδος/έξοδος

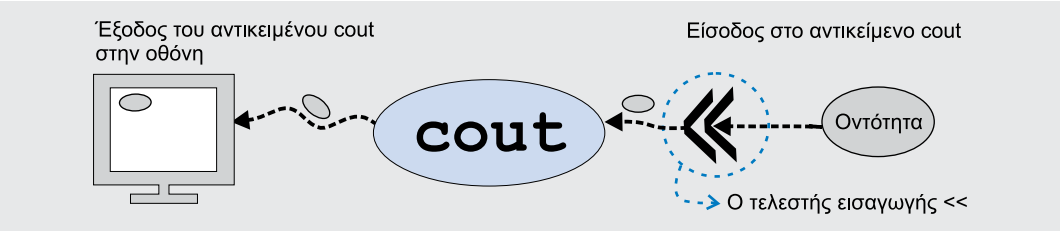
Η γλώσσα C++ δεν διαθέτει εντολές εισόδου και εξόδου από και προς τις περιφερειακές συσκευές του Η/Υ (πληκτρολόγιο, οθόνη, δίσκος, κ.λπ). Οι βασικές λειτουργίες εισόδου/εξόδου υλοποιούνται με τη χρήση προκαθορισμένων αντικειμένων και συναρτήσεων που περιλαμβάνονται στην καθιερωμένη βιβλιοθήκη της γλώσσας. Η έξοδος στην οθόνη γίνεται μέσω του αντικειμένου **cout** και η είσοδος από το πληκτρολόγιο μέσω του αντικειμένου **cin**.

Το αντικείμενο cout

Το αντικείμενο cout είναι συνδεδεμένο με την προκαθορισμένη μονάδα εξόδου που είναι συνήθως η οθόνη. Το αντικείμενο cout έχει μία είσοδο. Οτιδήποτε στείλουμε στην είσοδο του cout προωθείται στην έξοδό του, δηλαδή στην οθόνη. Φανταστείτε το αντικείμενο cout ως μια συσκευή η οποία είναι από τη μια μεριά συνδεδεμένη με την οθόνη και από την άλλη έχει μια τρύπα με ένα χωνί. Οτιδήποτε ρίξουμε μέσα στο χωνί (είσοδος) η συσκευή το εμφανίζει στην οθόνη (έξοδος).



Φυσικά, επειδή το το αντικείμενο cout είναι μια προγραμματιστική έννοια, δεν μπορούμε να χρησιμοποιήσουμε ένα πραγματικό χωνί για να ρίξουμε κάτι μέσα σε αυτό. Έτσι, αντί για χωνί χρησιμοποιούμε τον τελεστή << ο οποίος καλείται τελεστής **εισαγωγής**.

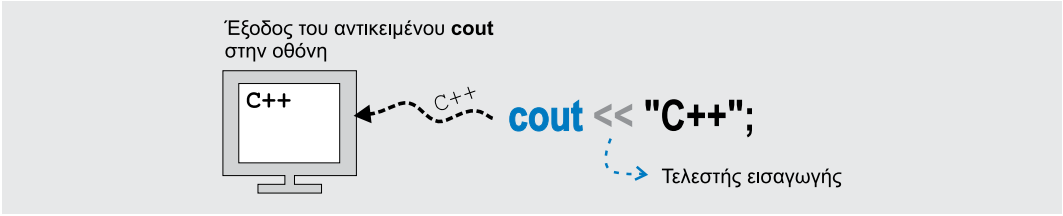


Το αντικείμενο cout ακολουθείται από τον **τελεστή εισαγωγής** << και την *οντότητα* την οποία θέλουμε να εισαγάγουμε στο αντικείμενο για να εμφανιστεί στην οθόνη.

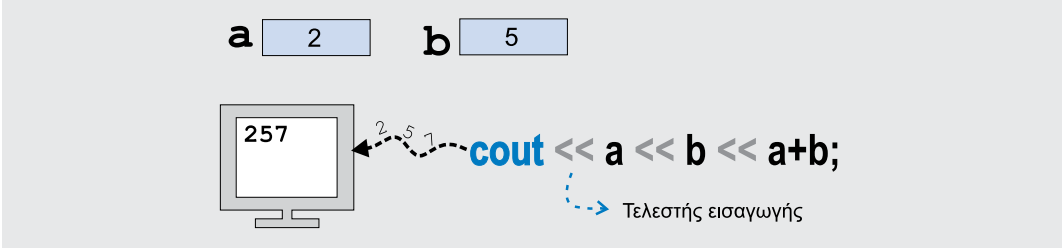
Η *οντότητα*, δηλαδή αυτό που ρίχνουμε μέσα στο αντικείμενο cout, συνήθως είναι:

Σταθερά, οπότε εμφανίζεται στην οθόνη η **τιμή** της όπως είναι
Μεταβλητή, οπότε εμφανίζεται στην οθόνη το **περιεχόμενό** της
Παράσταση, οπότε εμφανίζεται στην οθόνη το **αποτέλεσμά** της
Συνάρτηση, οπότε εμφανίζεται στην οθόνη η **τιμή** που **επιστρέφει**
Χειριστής, ειδική οντότητα η οποία δεν εμφανίζει **τίποτα** στην οθόνη και στην οποία θα αναφερθούμε αργότερα

Στο παρακάτω παράδειγμα, η αλφαριθμητική σταθερά «C++» στέλνεται στην είσοδο του αντικειμένου cout, οπότε εμφανίζεται στην έξοδό του, δηλαδή στην οθόνη.



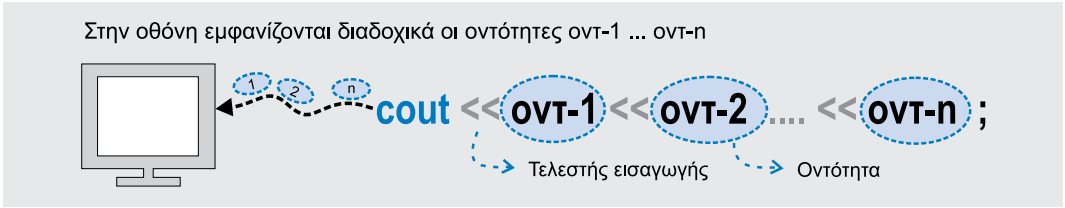
Μπορούμε να στείλουμε περισσότερες οντότητες στο αντικείμενο cout, χρησιμοποιώντας διαδοχικούς τελεστές εισαγωγής. Στην οθόνη εμφανίζεται η κάθε οντότητα σύμφωνα με τη λογική που αναφέρθηκε παραπάνω. Στο παρακάτω παράδειγμα του σχήματος, στέλνονται στην είσοδο του αντικειμένου cout τρεις οντότητες: δύο μεταβλητές και μία παράσταση. Στην οθόνη θα εμφανιστεί το περιεχόμενο των δύο μεταβλητών, καθώς και το αποτέλεσμα της παράστασης.



Παρατηρούμε ότι τα αποτελέσματα εμφανίζονται το ένα δίπλα στο άλλο χωρίς να παρεμβάλλονται κενά διαστήματα μεταξύ τους. Η ακόλουθη παραλλαγή της παραπάνω πρότασης αφήνει από ένα κενό διάστημα ανάμεσα στα αποτελέσματα που εμφανίζονται στην οθόνη.

```
cout<<a<<" "<<b<<" "<<a+b;
```

Το σχήμα που ακολουθεί δείχνει τον γενικό τρόπο χρήσης του αντικειμένου cout.



Όπως ήδη αναφέρθηκε, στο αντικείμενο cout μπορεί να εισαχθεί μια οντότητα που να είναι **χειριστής**. Όταν σταλεί ένας χειριστής στο αντικείμενο cout, τότε δεν εμφανίζεται **τίποτα** στην οθόνη, αλλά ενεργοποιείται μια λειτουργία του αντικειμένου. Π.χ. αλλαγή γραμμής, διαφορετική στοίχιση, καθορισμός πλήθους δεκαδικών ψηφίων, κ.λπ. Το **endl** είναι ο πιο συχνά χρησιμοποιούμενος **χειριστής** του αντικειμένου cout. Η αποστολή του χειριστή endl επιβάλλει μια αλλαγή γραμμής στην οθόνη. Για παράδειγμα η παρακάτω πρόταση:

```
cout <<"Νίκος"<<endl<<"Πάτροκλος"<<endl<<"Δέσποινα";
```

θα εμφανίσει τα τρία ονόματα σε διαφορετική γραμμή το καθένα!

> Ο χειριστής **endl** του αντικειμένου **cout** επιβάλλει μια αλλαγή γραμμής στην οθόνη. Οι υπόλοιποι χειριστές του αντικειμένου cout θα αναφερθούν σε επόμενο κεφάλαιο.

Στην πρόταση

```
cout <<"Τέλος\n";
```

ο χαρακτήρας \n στο τέλος της αλφαριθμητικής σταθεράς επιβάλλει επίσης μια αλλαγή γραμμής μετά το τέλος της εμφάνισης των χαρακτήρων. Το \n είναι ένας χαρακτήρας (και όχι δύο ξεχωριστοί) που ανήκει στους λεγόμενους «χαρακτήρες διαφυγής» (escape characters), οι οποίοι δεν εμφανίζονται στην οθόνη αλλά ελέγχουν μια συγκεκριμένη λειτουργία. Επομένως οι προτάσεις **cout <<"Τέλος\n"** και **cout <<"Τέλος"<<endl** είναι ισοδύναμες. Όλοι οι χαρακτήρες διαφυγής της C++ αναφέρονται στο Κεφάλαιο 4.

Το παρακάτω πρόγραμμα δείχνει τη χρήση του αντικειμένου cout:

3_cout.cpp

```
#include <iostream>
using namespace std;
int main()
{
   int a;
   char c;
   float b;
   a=15;
   b=a/2.0;
   c='A';
   cout <<"a+2="<<a+2<<" b="<<b<<endl<<"c="<<c<<endl<<"Τέλος\n";
   return 0;
}
```

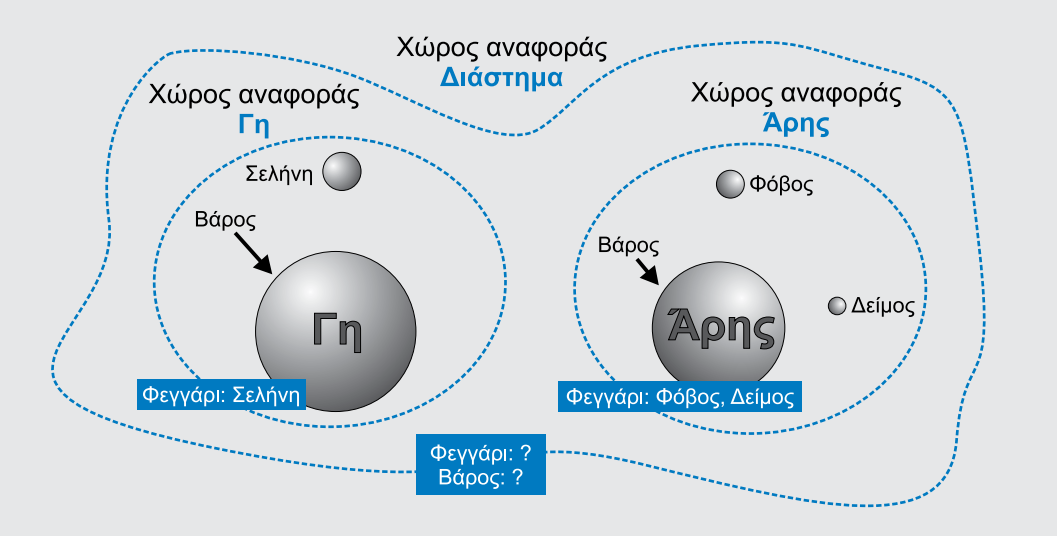
```
a+2=17 b=7.5
c=A
Τέλος
```

Χώροι ονομάτων της C++

Μέχρι τώρα, χρησιμοποιούσαμε την πρόταση **using namespace std;** σε όλα μας τα προγράμματα χωρίς να γνωρίζουμε τι ακριβώς κάνει. Μάλιστα, αν την παραλείψουμε θα διαπιστώσουμε ότι ο μεταγλωττιστής (compiler) δεν θα αναγνωρίσει κάποια από τα στοιχεία της γλώσσας όπως, για παράδειγμα, τα αντικείμενα cout και cin. Ήρθε η ώρα να πάρουμε μια πρώτη γεύση από τους **χώρους ονομάτων** (namespaces) της C++, οι οποίοι θα αναλυθούν διεξοδικά σε επόμενο κεφάλαιο.

Για να καταλάβουμε σε βάθος κάποιες έννοιες, θα ξεκινήσουμε με ένα παράδειγμα του πραγματικού κόσμου (Σχήμα 3.1). Επειδή ζούμε στη Γη, όταν αναφερόμαστε στο φεγγάρι εννοούμε φυσικά τη Σελήνη, και όταν μιλάμε για βάρος εννοούμε το βάρος στη Γη. Αυτό συμβαίνει επειδή θεωρούμε ως χώρο αναφοράς τη Γη. Αν όμως ζούσαμε στον Άρη και είχαμε ως χώρο αναφοράς αυτόν τον πλανήτη, οι δύο παραπάνω έννοιες θα είχαν διαφορετικό νόημα. Το φεγγάρι πια θα ήταν είτε ο Φόβος είτε ο Δείμος (ο Άρης έχει δύο δορυφόρους), και το βάρος θα ήταν το βάρος στον Άρη (αρκετά μικρότερο από το βάρος αντίστοιχης μάζας στη Γη). Αν τώρα ήμασταν διαστημάνθρωποι και ταξιδεύαμε σε όλο το πλανητικό μας σύστημα (χώρος αναφοράς το διάστημα), οι έννοιες *φεγγάρι* και *βάρος* από μόνες τους δεν θα είχαν κανένα νόημα αν δεν αναφερόμασταν στο όνομα κάποιου πλανήτη. Για παράδειγμα: Το φεγγάρι της Γης, το βάρος στον Άρη.

Επομένως, όταν δηλώσουμε τον χώρο αναφοράς μας (π.χ. Γη), όταν μιλάμε για *φεγγάρι* και *βάρος* δεν χρειάζεται να προσδιορίζουμε τον πλανήτη στον οποίο αναφερόμαστε.



Σχήμα 3.1 *Χώροι αναφοράς στον πραγματικό κόσμο*

Όταν όμως δεν έχουμε δηλώσει χώρο αναφοράς, ή όταν θέλουμε να αναφερθούμε στις ίδιες έννοιες αλλά άλλου πλανήτη, θα πρέπει οπωσδήποτε να αναφέρουμε το όνομά του.

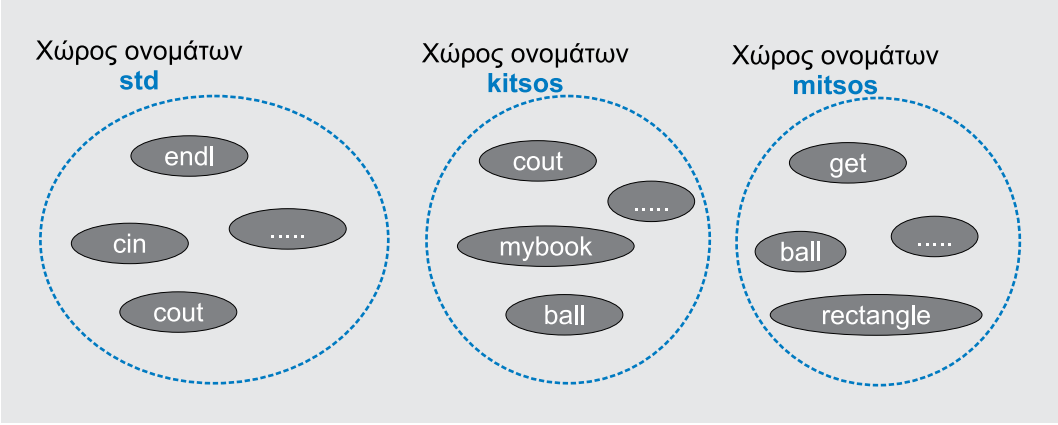
Επομένως, οι **χώροι ονομάτων** (αναφοράς) μας επιτρέπουν να δίνουμε σε **ίδιες** έννοιες **διαφορετικό** νόημα ανάλογα με το **περιβάλλον** στο οποίο τις χρησιμοποιούμε.

Χώροι ονομάτων (namespaces) της C++

Με βάση το προηγούμενο παράδειγμα, είναι πολύ εύκολο να κατανοήσετε τους χώρους ονομάτων (namespaces) της C++, αφού η φιλοσοφία είναι ακριβώς η ίδια.

Γνωρίζουμε ότι η C++ συνοδεύεται από την καθιερωμένη βιβλιοθήκη της (standard library) η οποία περιέχει διάφορες έτοιμες οντότητες όπως συναρτήσεις, κλάσεις, αντικείμενα, κ.ά. Όμως, υπάρχει η δυνατότητα να χρησιμοποιήσουμε και έτοιμες βιβλιοθήκες άλλων κατασκευαστών, ή ακόμα να δημιουργήσουμε και τις δικές μας βιβλιοθήκες. Φανταστείτε κάθε βιβλιοθήκη σαν ένα τσουβάλι με διάφορα παιχνίδια. Κάθε τσουβάλι έχει ένα όνομα αναφοράς, ενώ είναι πιθανόν να υπάρχουν τσουβάλια με παιχνίδια που να έχουν ακριβώς το ίδιο όνομα. Έστω λοιπόν ότι έχουμε δύο τσουβάλια όπου στο καθένα υπάρχει ένα παιχνίδι με όνομα *ball*. Τα παιχνίδια μπορεί να είναι ίδια ή εντελώς διαφορετικά, παρόλο που έχουν το ίδιο όνομα. Τώρα όταν αναφερόμαστε στο παιχνίδι με το όνομά του, σε ποιο από τα δύο παιχνίδια αναφερόμαστε; Προφανώς θα πρέπει να προσδιορίσουμε και το τσουβάλι στο οποίο βρίσκεται.

Κάθε βιβλιοθήκη της C++ διαθέτει ένα όνομα αναφοράς (χώρο ονομάτων). Η καθιερωμένη βιβλιοθήκη της C++ διαθέτει τον χώρο ονομάτων **std**. Άλλες βιβλιοθήκες διαθέτουν χώρους ονομάτων που προσδιορίζονται από τους κατασκευαστές τους.

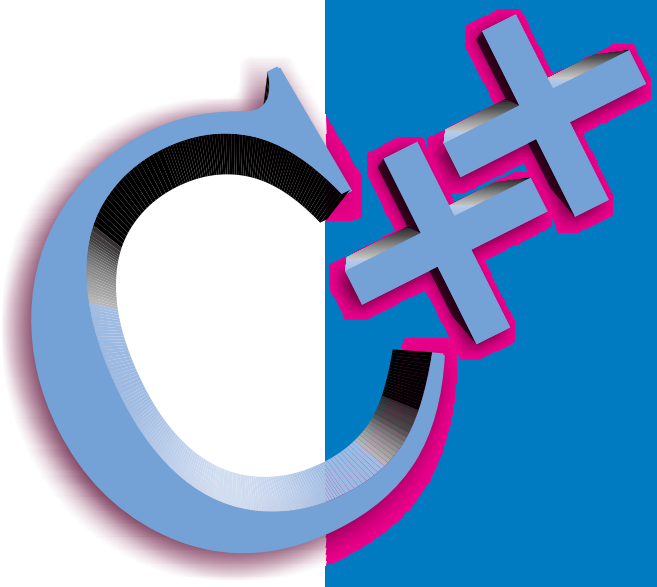


Σχήμα 3.2 *Χώροι ονομάτων στη C++*

Κεφάλαιο

4

Οι βασικοί τύποι δεδομένων της C++



Οι βασικοί τύποι δεδομένων της C++

Το κεφάλαιο αυτό παρουσιάζει και αναλύει τους βασικούς τύπους δεδομένων που υποστηρίζει η γλώσσα C++. Στο κεφάλαιο αυτό εξετάζεται η εσωτερική αναπαράσταση (απεικόνιση) των σταθερών και των μεταβλητών των διαφόρων τύπων, καθώς και οι τελεστές που εφαρμόζονται σε αυτούς.

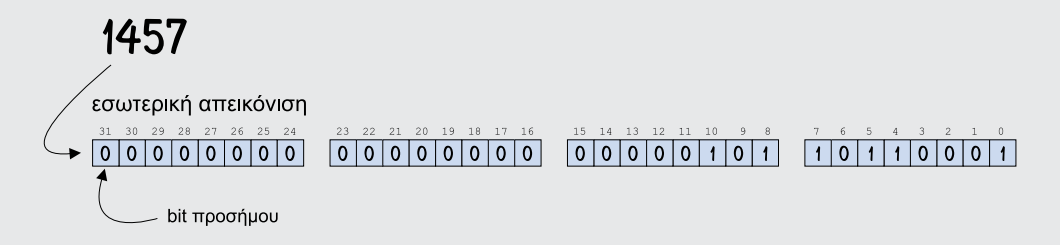
Ο τύπος δεδομένων int

Ο τύπος δεδομένων int αναφέρεται στα ακέραια μεγέθη. Σταθερές, μεταβλητές, παραστάσεις και συναρτήσεις τύπου int αποδίδουν ακέραιες τιμές.

Σταθερές τύπου int

Οι ακέραιες σταθερές δεν είναι τίποτε άλλο από απλοί ακέραιοι αριθμοί. Μια ακέραια σταθερά αποθηκεύεται στη μνήμη αφού μετατραπεί σε δυαδικό αριθμό. Μια σταθερά τύπου int μετατρέπεται σε έναν δυαδικό αριθμό μήκους 4 byte (βλέπε υποσημείωση επόμενης σελίδας).

Με 4 byte μπορεί να αναπαρασταθεί οποιοσδήποτε αριθμός από -2.147.483.648 μέχρι +2.147.483.647. Το 31ο bit (τα 32 bit αριθμούνται από το 0 έως το 31) χρησιμοποιείται για το πρόσημο και τα υπόλοιπα 31 για την αναπαράσταση του αριθμού.



Μεταβλητές τύπου int

Εκτός από τον τύπο int που έχουμε ήδη αναφέρει, η C++ διαθέτει άλλους δύο τύπους ακέραιων μεταβλητών: long int και short int.

Οι μεταβλητές τύπου int δηλώνονται ως εξής:

int ***όνομα_μεταβλητής*****, *όνομα_μεταβλητής*****, ...;**

Οι μεταβλητές τύπου long int δηλώνονται ως εξής:

long int ***όνομα_μεταβλητής*****, *όνομα_μεταβλητής*****, ...;** ή

long ***όνομα_μεταβλητής*****, *όνομα_μεταβλητής*****, ...;**

Στην πρόταση δήλωσης η λέξη int μπορεί να παραλειφθεί, αλλά για λόγους σαφήνειας αυτό δεν συνιστάται.

Μια παραλλαγή του τύπου int είναι ο τύπος short int. Οι μεταβλητές τύπου short int δηλώνονται με παρόμοιο τρόπο:

short int *όνομα_μεταβλητής*, *όνομα_μεταβλητής*, ...; ή

short *όνομα_μεταβλητής*, *όνομα_μεταβλητής*, ...;

Οι μεταβλητές τύπου int και long int δεσμεύουν 4 byte[8] (32 bit) και μπορούν να αποθηκεύσουν έναν ακέραιο αριθμό από -2.147.483.648 μέχρι +2.147.483.647. Οι μεταβλητές τύπου short int δεσμεύουν 2 byte (16 bit) και μπορούν να αποθηκεύσουν έναν ακέραιο αριθμό από -32.768 μέχρι +32.767.

Ακέραιοι αριθμοί χωρίς πρόσημο (unsigned)

Μια μεταβλητή τύπου int (int, long int ή short int) μπορεί να οριστεί ως μη προσημασμένη (δηλαδή χωρίς πρόσημο), με χρήση του προσδιοριστικού unsigned πριν από το όνομα του τύπου, όπως παρακάτω:

unsigned int *όνομα_μεταβλητής*, *όνομα_μεταβλητής*, ...; ή

unsigned long int *όνομα_μεταβλητής*, *όνομα_μεταβλητής*, ...; ή

unsigned short int *όνομα_μεταβλητής*, *όνομα_μεταβλητής*, ...; ή

Οι μεταβλητές τύπου unsigned δεσμεύουν τα ίδια byte με τις αντίστοιχες προσημασμένες μεταβλητές. Όμως, σε μια μεταβλητή τέτοιου τύπου το πρώτο bit του αριθμού δεν χρησιμοποιείται για το πρόσημο, αλλά συμμετέχει στην αναπαράσταση του αριθμού. Επομένως, σε μεταβλητές τύπου unsigned μπορούν να αποθηκευτούν μόνο θετικοί αριθμοί, αλλά με διπλάσια απόλυτη τιμή (από τους αντίστοιχους προσημασμένους).

Αριθμητικοί τελεστές

Εκτός από τους «κλασικούς» αριθμητικούς τελεστές, η C++ (όπως και η C) διαθέτει και μερικούς τελεστές οι οποίοι συναντώνται μόνο στις γλώσσες προγραμματισμού που βασίζονται στη C (Java, C#). Ο διπλανός πίνακας αναφέρει τους διαθέσιμους αριθμητικούς τελεστές στη C++. Δεν θα ασχοληθούμε με τους κλασικούς αριθμητικούς τελεστές, αλλά μόνο με τους τελεστές ++, -- και τον τελεστή υπολοίπου %.

Τελεστής	Λειτουργία
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο ακέραιας διαίρεσης
++	Αύξηση κατά 1
--	Μείωση κατά 1

[8] Το μέγεθος των τύπων δεδομένων int, long int και short int εξαρτάται από τον τύπο του συστήματος Η/Υ. Έτσι, μπορεί να διαφέρει σε συστήματα 16bit, 32bit, ή 64bit. Αυτό ισχύει και για τους υπόλοιπους τύπους δεδομένων.

Οι τελεστές αύξησης ++ και μείωσης -- εφαρμόζονται *μόνο* σε μεταβλητές και μπορεί να προηγούνται ή να ακολουθούν το όνομα μιας μεταβλητής.

π.χ. `++a; a++; --a; a--;`

Ο τελεστής ++ αυξάνει το περιεχόμενο της μεταβλητής κατά 1, ενώ ο τελεστής -- μειώνει το περιεχόμενο της μεταβλητής κατά 1. Η λειτουργία αυτή γίνεται ανεξάρτητα από τη θέση του τελεστή (πριν ή μετά από τη μεταβλητή).

```
int main()
{
   int a,b;
   b=a=5;
   ++a;
   --b;
   cout << "a=" << a << " b=" << b << endl;
   return 0;
}
```

Με την παράσταση **++a** η μεταβλητή **a** αυξάνεται κατά 1 (γίνεται 6) και με την παράσταση **--b** η μεταβλητή **b** μειώνεται κατά 1 (γίνεται 4). Έτσι η **cout** θα εμφανίσει στην οθόνη:
`a=6 b=4`

Όπως κάθε παράσταση, έτσι και οι παραστάσεις με τους τελεστές ++ και -- επιστρέφουν μια τιμή. Η τιμή αυτή εξαρτάται από τη θέση του τελεστή (πριν ή μετά από τη μεταβλητή).

☞ Όταν ο τελεστής βρίσκεται πριν από τη μεταβλητή (προθεματικός), τότε πρώτα γίνεται η πράξη (αύξηση ή μείωση) και μετά επιστρέφεται η νέα τιμή της μεταβλητής (μετά από την αύξηση ή τη μείωση). Επομένως, στην παράσταση χρησιμοποιείται η νέα τιμή της μεταβλητής, μετά από την αύξηση ή τη μείωσή της. Για παράδειγμα, αν υποθέσουμε ότι `a=5;` η παράσταση `++a;` αυξάνει το `a` κατά 1 και επιστρέφει την τιμή 6.

☞ Όταν ο τελεστής βρίσκεται μετά από τη μεταβλητή (επιθηματικός), τότε πρώτα επιστρέφει την τιμή της μεταβλητής (την παλιά τιμή, πριν από την πράξη) και μετά αυξάνει ή μειώνει το περιεχόμενο της μεταβλητής. Επομένως στην παράσταση χρησιμοποιείται η παλιά τιμή της μεταβλητής πριν από την αύξηση της. Για παράδειγμα, αν υποθέσουμε πάλι ότι `a=5;` η παράσταση `a++;` επιστρέφει την τιμή 5 και αυξάνει το `a` κατά 1.

Στα επόμενα παραδείγματα γίνεται πιο σαφής η χρήση των τελεστών ++ και --.

Το επόμενο τμήμα κώδικα εμφανίζει τα αποτελέσματα που φαίνονται στο διπλανό πλαίσιο:

```
int main()
{
   int a,b;
   a=b=5;
   cout << "++a=" << ++a << endl;
   cout << "b++=" << b++ << endl;
   cout << "a=" << a << "b=" << b << endl;
   return 0;
}
```

```
++a=6
b++=5
a=6 b=6
```

Ο τύπος wchar_t

Με βάση τον επεκτεταμένο πίνακα κωδικοποίησης ASCII μπορούν να κωδικοποιηθούν το πολύ 256 χαρακτήρες, συμπεριλαμβανομένων και των χαρακτήρων ελέγχου. Είναι λοιπόν προφανές ότι είναι αδύνατον να συμπεριληφθούν και κωδικοί για αλφάβητα διαφόρων γλωσσών, αλλά και γλωσσών με περισσότερους χαρακτήρες στο αλφάβητό τους (π.χ. κινέζικα). Το πρότυπο Unicode έρχεται να δώσει λύση, συμπεριλαμβάνοντας σε ένα μόνο σετ χαρακτήρων τη δυνατότητα αποθήκευσης πολλών διαφορετικών αλφάβητων. Το πρότυπο Unicode υποστηρίζει διάφορους τρόπους κωδικοποίησης χαρακτήρων οι οποίοι απαιτούν περισσότερα από ένα byte για κάθε χαρακτήρα. Ο προκαθορισμένος τύπος χαρακτήρων char της C++ υποστηρίζει χαρακτήρες που κωδικοποιούνται με ένα μόνο byte. Τι γίνεται λοιπόν με τους χαρακτήρες που απαιτούν περισσότερα byte; Σύμφωνα με τα τελευταία πρότυπα της γλώσσας, η C++ υποστηρίζει χαρακτήρες μεγαλύτερου μήκους 2 και 4 byte με τον τύπο δεδομένων wchar_t. Συνήθως, σε περιβάλλοντα UNIX και LINUX το μέγεθος του τύπου wchar_t είναι 4 byte, ενώ σε περιβάλλον Windows είναι 2 byte.

Επειδή το παρόν βιβλίο είναι προσανατολισμένο στην καθαρά εκπαιδευτική διαδικασία εκμάθησης της C++, δεν θα ασχοληθούμε καθόλου με τον τύπο wchar_t ο οποίος αφορά εξειδικευμένες εφαρμογές σε παραθυρικό περιβάλλον.

Συμβολοσειρές (character strings)

Μια συμβολοσειρά (ή «αλφαριθμητικό χαρακτήρων») ορίζεται στη C++ ως μια ακολουθία χαρακτήρων μέσα σε διπλά εισαγωγικά (""). Στη συνέχεια του βιβλίου οι συμβολοσειρές μπορεί να αναφέρονται και ως ***ακολουθίες χαρακτήρων***, αλλά και ως ***αλφαριθμητικά*** (character strings). Μερικά παραδείγματα συμβολοσειρών είναι:

"ΝΙΚΟΣ" "12" "" "C++ is the Best."

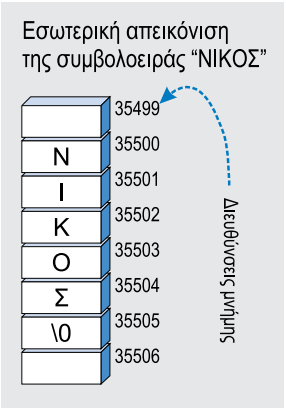
Μια συμβολοσειρά αποθηκεύεται στη μνήμη σε συνεχόμενες θέσεις (μία για κάθε χαρακτήρα), ξεκινώντας από μια συγκεκριμένη διεύθυνση. Ο μεταγλωττιστής της C++ τοποθετεί τον χαρακτήρα τερματισμού '\0' στο τέλος κάθε συμβολοσειράς και σηματοδοτεί το τέλος της συμβολοσειράς.

Οι συμβολοσειρές έχουν τιμή

Όπως κάθε παράσταση στη C++, έτσι και οι συμβολοσειρές έχουν μια τιμή.

> Η τιμή μιας συμβολοσειράς (string) είναι η αρχική διεύθυνση της μνήμης όπου είναι αποθηκευμένη η συμβολοσειρά και όχι οι ίδιοι οι χαρακτήρες της. Η C++ χειρίζεται κάθε συμβολοσειρά ως αριθμό και, πιο συγκεκριμένα, ως μία διεύθυνση.

Το διπλανό σχήμα δείχνει τον τρόπο με τον οποίο αποθηκεύεται η συμβολοσειρά «ΝΙΚΟΣ» στη μνήμη του Η/Υ. Η τιμή της συγκεκριμένης συμβολοσειράς είναι η διεύθυνση της πρώτης από τις θέσεις μνήμης που καταλαμβάνει, δηλαδή η διεύθυνση 35500!



Η C++, όπως και η C, παρέχει στον προγραμματιστή μια σειρά από συναρτήσεις για τον χειρισμό των συμβολοσειρών. Στις συναρτήσεις αυτές μεταβιβάζεται πάντα ως όρισμα η αρχική διεύθυνση της συμβολοσειράς. Η συνάρτηση εντοπίζει το τέλος της συμβολοσειράς ανιχνεύοντας τον χαρακτήρα τερματισμού '\0'.

☞ Αν προσπαθήσουμε να χειριστούμε μια συμβολοσειρά απευθείας, θα βρεθούμε μπροστά σε δυσάρεστες (αλλά όχι τόσο ακατανόητες) εκπλήξεις. Ο χειρισμός των μεγεθών των συμβολοσειρών γίνεται από ειδικές συναρτήσεις, μερικές από τις οποίες θα αναφερθούν στα επόμενα κεφάλαια.

☞ Η κλάση string, την οποία διαθέτει η καθιερωμένη βιβλιοθήκη προτύπων της C++, δίνει επίσης τη δυνατότητα χειρισμού των συμβολοσειρών με ένα διαφορετικό και πιο απλό τρόπο.

Μια πρώτη ματιά στη κλάση string

Η C++, όπως και η C, δεν διαθέτει ενσωματωμένο τύπο για την αποθήκευση συμβολοσειρών. Στη C, για την αποθήκευση συμβολοσειρών χρησιμοποιούνται πίνακες χαρακτήρων. Στη C++, παρά το γεγονός ότι εξακολουθεί να υπάρχει η δυνατότητα χρήσης πινάκων χαρακτήρων (Κεφάλαιο 11), για την αποθήκευση των συμβολοσειρών χρησιμοποιείται η κλάση string η οποία παρέχεται από τη καθιερωμένη βιβλιοθήκη της γλώσσας.

Μέχρι να καταλάβουμε τι ακριβώς είναι μια κλάση και ένα αντικείμενο και πώς αυτά ορίζονται, μπορείτε να θεωρείτε την κλάση string ως έναν τύπο δεδομένων ο οποίος επιτρέπει την αποθήκευση ολόκληρων συμβολοσειρών. Φυσικά, όταν δηλώνουμε μεταβλητές τύπου string στην πραγματικότητα δημιουργούμε αντικείμενα της κλάσης string. Για να είμαστε και τυπικά σωστοί, θα χρησιμοποιούμε αυτή την ορολογία.

Η δήλωση ενός αντικειμένου string γίνεται με μια δηλωτική πρόταση, όπως ακριβώς με τους βασικούς τύπους δεδομένων. Η ακόλουθη πρόταση δηλώνει (και δημιουργεί) ένα αντικείμενο της κλάσης string:

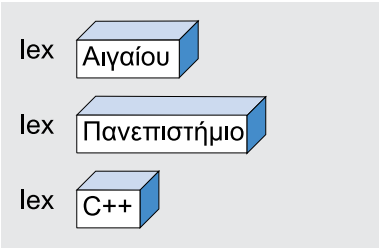
```
string lex;
```

Το αντικείμενο lex είναι ένας αποθηκευτικός χώρος στον οποίο μπορεί να καταχωριστεί μια συμβολοσειρά. Τα αντικείμενα της κλάσης string τα αντιμετωπίζουμε ως μεταβλητές στις οποίες μπορούμε να καταχωρίσουμε συμβολοσειρές με τον τελεστή ανάθεσης =.

Για παράδειγμα, οι παρακάτω προτάσεις καταχωρίζουν διαδοχικά τρεις διαφορετικές συμβολοσειρές στο αντικείμενο lex.

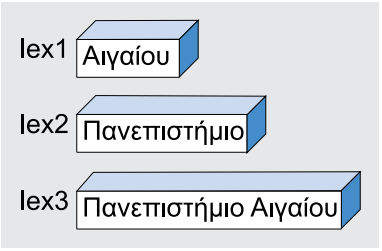
```
lex="Αιγαίου";
lex="Πανεπιστήμιο";
lex="C++";
```

Προφανώς, ένα αντικείμενο της κλάσης string χρησιμοποιεί ένα πλήθος από θέσεις μνήμης (byte) για την αποθήκευση της συμβολοσειράς. Το πλήθος αυτό καθορίζεται από τον αριθμό των χαρακτήρων της συμβολοσειράς και μπορεί να μεταβάλλεται. Έτσι, μπορούμε να παρομοιάσουμε ένα αντικείμενο της κλάσης string με ένα ελαστικό κουτί το μέγεθος του οποίου προσαρμόζεται αυτόματα ανάλογα με το μέγεθος της συμβολοσειράς που περιέχει.



Πράξεις και συγκρίσεις μεταξύ αντικειμένων κλάσης string

Με τα αντικείμενα της κλάσης string μπορούμε να χρησιμοποιήσουμε τον τελεστή + για να προσθέσουμε τους χαρακτήρες του ενός με τους χαρακτήρες του άλλου. Για παράδειγμα, στο παρακάτω τμήμα κώδικα προστίθενται τα περιεχόμενα του lex1 με ένα κενό διάστημα και με τα περιεχόμενα του lex2, και το αποτέλεσμα καταχωρίζεται στο αντικείμενο lex3.




```
string lex1,lex2,lex3;
lex1="Αιγαίου";
lex2="Πανεπιστήμιο";
lex3=lex2+" "+lex1;
```

Επίσης, μπορούμε να χρησιμοποιήσουμε τους συγκριτικούς τελεστές για να συγκρίνουμε αντικείμενα της κλάσης string. Οι συμβολοσειρές συγκρίνονται με απόλυτη αλφαβητική σειρά, δηλαδή χαρακτήρα προς χαρακτήρα. Η σύγκριση των αντίστοιχων χαρακτήρων γίνεται με βάση τον κωδικό τους ASCII. Ένας πρακτικός τρόπος για να βρίσκουμε τη μεγαλύτερη από δύο συμβολοσειρές είναι να τις φανταζόμαστε ως ονόματα σε έναν τηλεφωνικό κατάλογο. Όποιο έρχεται πρώτο είναι το μικρότερο. Για παράδειγμα, η συμβολοσειρά "Αλέξανδρος" είναι μικρότερη από τη συμβολοσειρά "Νίκος".

Τα αντικείμενα της κλάσης string μπορούν να χρησιμοποιούνται σε συνδυασμό με τα αντικείμενα cin και cout, όπως οποιαδήποτε άλλη μεταβλητή.

Το πρόγραμμα που ακολουθεί ζητάει από τον χρήστη να πληκτρολογήσει τρεις διαφορετικές λέξεις και συνθέτει μια φράση που αποτελείται από αυτές τις λέξεις. Ακολούθως

συγκρίνει τις δύο πρώτες λέξεις μεταξύ τους ώστε να διαπιστώσει ποια είναι αλφαβητικά μεγαλύτερη.

4_strings.cpp

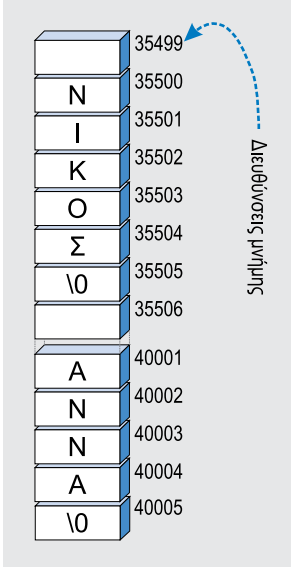
```
#include <iostream>
using namespace std;
int main()
{
   string lex1,lex2,lex3,frasi;
   cout << "Δώσε τρεις λέξεις μιας φράσης χωρισμένες με <enter>:";
   cin >> lex1 >> lex2 >> lex3;
   frasi=lex1+" "+lex2+" "+lex3;
   cout << "Η φράση είναι:" << frasi << endl;
   if (lex1>lex2)
      cout << "Η πρώτη λέξη που εδωσες είναι αλφαβητικά μεγαλύτερη από τη
         δεύτερη"<< endl;
   else if (lex2>lex1)
      cout << "Η δεύτερη λέξη που εδωσες είναι αλφαβητικά μεγαλύτερη από
         τη πρώτη"<< endl;
   else
      cout << "Πρώτη και δεύτερη λέξη είναι ίδιες" << endl;
   return 0;
}
```

ΠΡΟΣΟΧΗ: Παρόλο που μπορούμε να συγκρίνουμε αντικείμενα string μεταξύ τους, αλλά και με συμβολοσειρές, δεν μπορούμε να συγκρίνουμε συμβολοσειρές μεταξύ τους χρησιμοποιώντας τους τελεστές σύγκρισης!

Αυτό συμβαίνει επειδή, όπως ήδη έχουμε αναφέρει, η C++ αντιμετωπίζει μια συμβολοσειρά ως μια διεύθυνση, οπότε με τη σύγκριση δύο συμβολοσειρών ουσιαστικά συγκρίνονται οι διευθύνσεις τους και όχι οι χαρακτήρες που περιέχουν.

Στο παράδειγμα του διπλανού σχήματος φαίνεται πώς είναι καταχωρισμένες οι συμβολοσειρές «ΝΙΚΟΣ» και «ΑΝΝΑ» , στη μνήμη του Η/Υ. Μια παράσταση της μορφής "ΝΙΚΟΣ">"ΑΝΝΑ" θα είναι ψευδής διότι θα συγκρίνει τις διευθύνσεις των δύο συμβολοσειρών (το 35500 με το 40001) και όχι τους χαρακτήρες τους. Φυσικά, αν η σύγκριση γινόταν με απόλυτη αλφαβητική σειρά, η συμβολοσειρά «ΝΙΚΟΣ» είναι αλφαβητικά μεγαλύτερη από τη συμβολοσειρά «ΑΝΝΑ».

Η σύγκριση συμβολοσειρών γίνεται με χρήση ειδικών συναρτήσεων, κληρονομιά από τη C!



Παραδείγματα

Π4.7 Το επόμενο πρόγραμμα διαβάζει έναν χαρακτήρα από το πληκτρολόγιο και τον εμφανίζει στην οθόνη.

4_p4_7.cpp

```
#include <iostream>
using namespace std;
int main()
{
   char c;
   cout << "Πληκτρολογήστε ένα χαρακτήρα:";
   cin >> c;
   cout << "Πληκτρολόγησες το :" << c << endl;
   return 0;
}
```

Στην περίπτωση που πληκτρολογηθούν περισσότεροι χαρακτήρες, θα διαβάζει και θα εμφανίζει μόνο τον πρώτο.

Π4.8 Το επόμενο πρόγραμμα διαβάζει έναν χαρακτήρα και τυπώνει τον κωδικό του ASCII.

4_p4_8.cpp

```
#include <iostream>
using namespace std;
int main()
{
   char c;
   int a;
   cout << "Πληκτρολογήστε ένα χαρακτήρα:";
   cin >> c;
   a = c;
   cout << "ο ASCII κωδικός του "
        << c << " είναι " << a;
   return 0;
}
```

Στη μεταβλητή **a**, καταχωρίζεται ο κωδικός ASCII του χαρακτήρα που βρίσκεται στη μεταβλητή **c**.

Π4.9 Το επόμενο πρόγραμμα διαβάζει ένα χαρακτήρα και ελέγχει αν είναι λατινικός κεφαλαίος ή όχι.

4_p4_9.cpp

```
#include <iostream>
using namespace std;
int main()
{
   char c;
   cout << "Πληκτρολογήστε ένα χαρακτήρα:";
   cin >> c;
   if (c>='A' && c<='Z')
```

Προσθήκες προτύπου C++11

Αυτόματος τύπος δεδομένων

Το πρότυπο C++11 εισήγαγε τον αυτόματο τύπο δεδομένων auto, ο οποίος ουσιαστικά λέει στον μεταγλωττιστή να καθορίσει εκείνος τον τύπο της μεταβλητής ανάλογα με την τιμή που της ανατίθεται ως αρχική τιμή.

Για παράδειγμα η ακόλουθη πρόταση δήλωσης:

```
auto k=3;
```

έχει ως αποτέλεσμα τη δημιουργία μιας μεταβλητής τύπου int, διότι η αρχική τιμή που ανατέθηκε στη μεταβλητή k ήταν ένας ακέραιος αριθμός. Από τώρα και στο εξής η μεταβλητή k θεωρείται μεταβλητή τύπου int.

Στο πρόγραμμα που ακολουθεί δηλώνονται τέσσερις αυτόματες μεταβλητές:

4_auto.cpp

```
#include <iostream>
using namespace std;
int main()
{
   auto a=10,b=5;
   auto mo=(a+b)/2.0;
   auto ch='\n';
   cout << mo << ch;
   return 0;
}
```

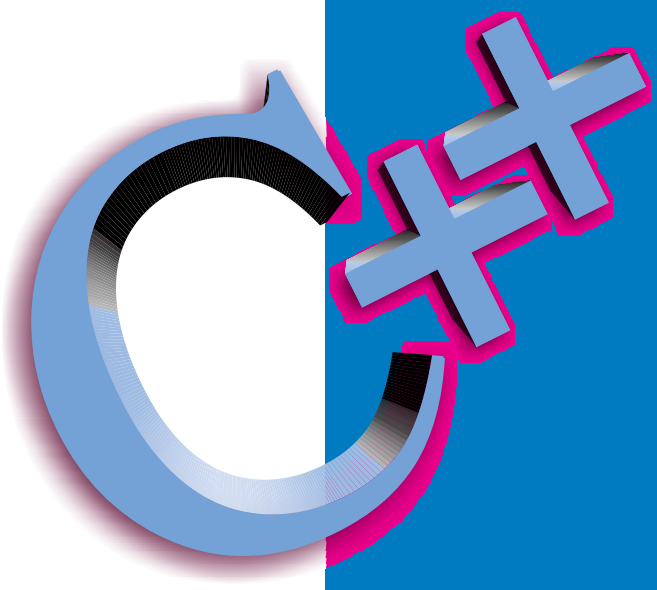
Ο μεταγλωττιστής θεωρεί ότι οι μεταβλητές a και b έχουν τύπο int, επειδή τους δόθηκαν ως αρχικές τιμές δύο ακέραιοι αριθμοί. Τη μεταβλητή mo τη θεωρεί τύπου double διότι αυτός είναι ο τύπος του αποτελέσματος της παράστασης (a+b)/2.0. Τέλος, θεωρεί τη μεταβλητή ch τύπου char, διότι δέχθηκε ως αρχική τιμή έναν χαρακτήρας.

ΠΡΟΣΟΧΗ: Για να μπορέσει να μεταγλωττιστεί σωστά το πρόγραμμα στο περιβάλλον του Code::Blocks, θα πρέπει να έχουμε ενεργοποιήσει τη κατάλληλη επιλογή ώστε ο μεταγλωττιστής να ακολουθεί το πρότυπο C++11. Στο Παράρτημα Β θα βρείτε οδηγίες για το πώς μπορείτε να ενεργοποιήσετε αυτή την επιλογή.

Κεφάλαιο

5

Εντολές συνθήκης



Εντολές συνθήκης

Η C++ διαθέτει δύο εντολές με τις οποίες έχει τη δυνατότητα εκτέλεσης προτάσεων υπό συνθήκη («υπό όρους»). Αυτές είναι η εντολή if και η εντολή switch-case.

Η εντολή if με μια δεύτερη ματιά

Η εντολή if ελέγχει μια συνθήκη και, ανάλογα με το αποτέλεσμα του ελέγχου (αληθές ή ψευδές), εκτελεί (ή δεν εκτελεί) κάποιες προτάσεις.

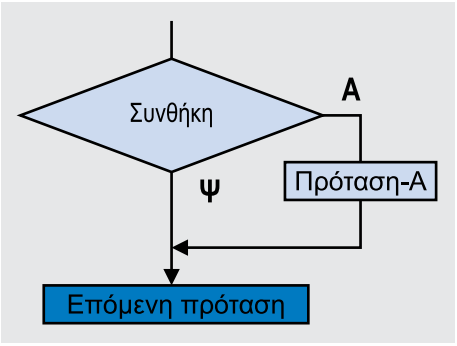
Η if συντάσσεται με τρεις τρόπους:

- ως απλή πρόταση if
- ως συνδυασμός if-else
- ως συνδυασμός if-else if

Η απλή πρόταση if

Έχουμε ήδη δει το συντακτικό της if στην πιο απλή της μορφή:

if (*συνθήκη*) *πρόταση-Α*;



Σε αυτή τη μορφή, η if ελέγχει τη **συνθήκη** μέσα στις παρενθέσεις και, αν είναι αληθής (**Α**), εκτελεί την ***πρόταση-Α*** που ακολουθεί. Αν η συνθήκη είναι ψευδής (**Ψ**) δεν εκτελείται τίποτα και συνεχίζει στην αμέσως απόμενη πρόταση μετά από την εντολή if. Στο λογικό διάγραμμα φαίνεται η λογική της πρότασης if. Η ***πρόταση-Α*** μπορεί να είναι απλή ή σύνθετη πρόταση (compound statement). Ας θυμηθούμε ότι ως *συνθήκη* στη C++ επιτρέπεται να είναι *οποιαδήποτε παράσταση* η οποία επιστρέφει αριθμητική τιμή.

> Η C++ θεωρεί αληθή οποιαδήποτε παράσταση επιστρέφει τιμή διάφορη του μηδενός (ακόμα και αρνητική). Ψευδής θεωρείται μια παράσταση μόνον όταν έχει τιμή 0.

Ακολουθούν ορισμένα παραδείγματα χρήσης της απλής πρότασης if:

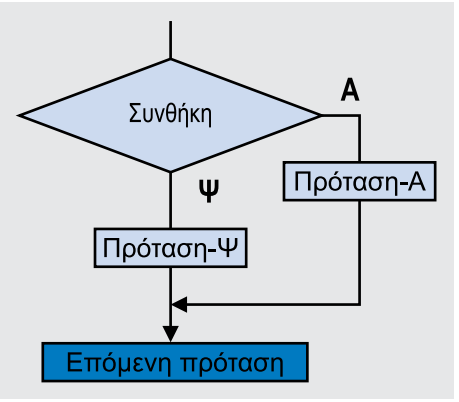
```
if (a==5) cout << "a=5\n";
if (a==5)
   cout << "a=5\n";
if (a==5)
{
   cout << "a=5\n";
   cout << "μία σύνθετη πρόταση\n";
}
```

Σύνθετη πρόταση η οποία θα εκτελεστεί στην περίπτωση που η παράσταση **a==5** είναι **αληθής**.

Η πρόταση if-else

if (*συνθήκη*) *πρόταση-Α*; else *πρόταση-Ψ*;

Σε αυτή τη μορφή, η if ελέγχει τη συνθήκη και, αν είναι αληθής (Α), εκτελεί την ***πρόταση-Α*** που ακολουθεί. Αν η συνθήκη είναι ψευδής (Ψ), εκτελείται η ***πρόταση-Ψ***. Μετά εκτελείται κανονικά η επόμενη από την if πρόταση του προγράμματος.



Στο διπλανό λογικό διάγραμμα φαίνεται η λογική της πρότασης if-else. Οι προτάσεις Α και Ψ μπορεί να είναι απλές ή σύνθετες προτάσεις.

Παραδείγματα χρήσης της if-else:

```
if (a==5) cout <<"Ναι"; else cout <<"Όχι";
if (a==5)
   cout << "Ναι";
else
   cout << "Όχι";
if (a==5)
{
   cout << "Ναι\n";
   cout << "Σύνθετη πρόταση - Α\n";
}
else
{
   cout << "Όχι\n";
   cout << "Σύνθετη πρόταση - Β\n";
}
```

Οι προτάσεις **if** και **else** μπορεί να είναι σύνθετες.

Η πρόταση if-else if

Η πρόταση if-else if κάνει διαδοχικούς ελέγχους. Ελέγχονται με τη σειρά διάφορες συνθήκες και μόλις επαληθευτεί μια συνθήκη, εκτελείται η αντίστοιχη πρόταση (απλή ή σύνθετη). Στη περίπτωση που όλες οι συνθήκες είναι ψευδείς εκτελείται η ***πρόταση-Ψ***.

```
if (Συνθήκη-1)
   Πρόταση-1;
else if (Συνθήκη-2)
   Πρόταση-2;
...
else if (Συνθήκη-N)
   Πρόταση-N;
else
   Πρόταση-Ψ;
```

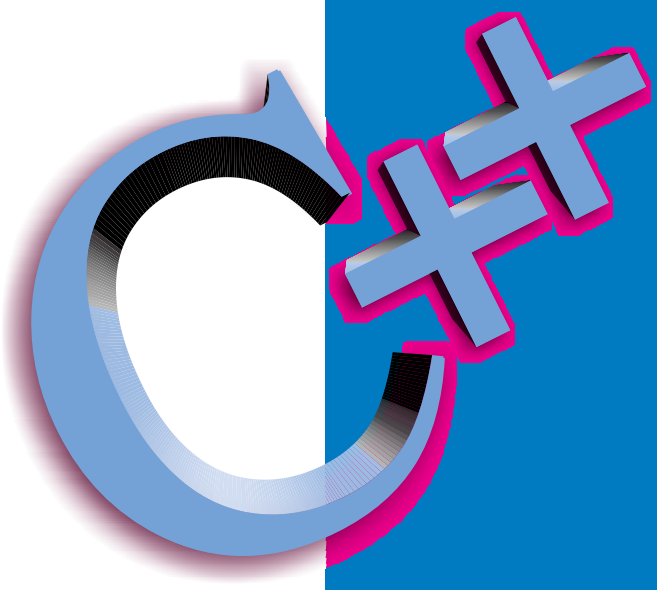
Η **Πρόταση-2** εκτελείται όταν η **Συνθήκη-1** είναι ψευδής και η **Συνθήκη-2** αληθής.

Η **Πρόταση-Ψ** εκτελείται όταν δεν είναι αληθής καμία από τις προηγούμενες συνθήκες.

Κεφάλαιο

6

Δομές επανάληψης



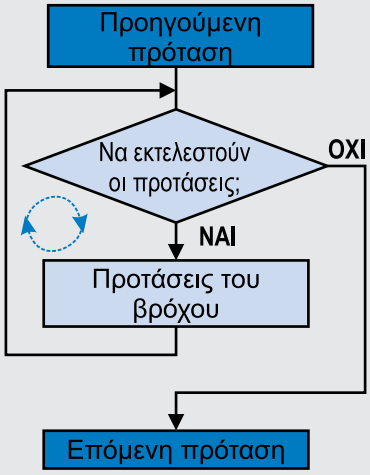
Δομές επανάληψης

Ένα από τα βασικά πλεονεκτήματα της επεξεργασίας στοιχείων μέσω υπολογιστικών συστημάτων είναι η δυνατότητα της επανάληψης των διαδικασιών. Δηλαδή, η δυνατότητα των γλωσσών προγραμματισμού να επιβάλλουν την επαναλαμβανόμενη εκτέλεση ενός τμήματος κώδικα μέσα σε ένα πρόγραμμα. Κάθε γλώσσα προγραμματισμού διαθέτει δομές για την υλοποίηση αυτής της διαδικασίας. Μία δομή επανάληψης καλείται επίσης *βρόχος* (loop) και οι προτάσεις που εκτελούνται αποτελούν το σώμα του βρόχου.

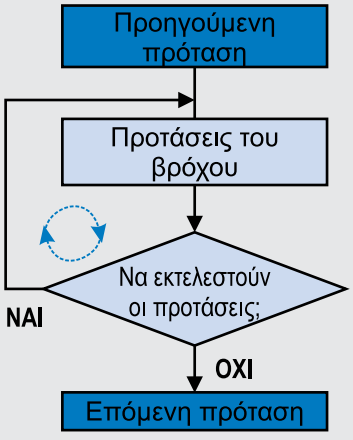
Σε κάθε περίπτωση, η επαναληπτική εκτέλεση των προτάσεων εξαρτάται από τον έλεγχο κάποιας συνθήκης (π.χ. «όσο το a είναι μεγαλύτερο από το 5 επανέλαβε τις παρακάτω προτάσεις»). Ο έλεγχος αυτός σε κάποιες δομές γίνεται πριν από την εκτέλεση της κάθε επανάληψης των προτάσεων του βρόχου (Σχήμα 6.1α) και σε κάποιες δομές μετά από την εκτέλεσή τους (Σχήμα 6.1β).

Κάποιες δομές επανάληψης χρησιμοποιούνται όταν δεν γνωρίζουμε από πριν το πλήθος των επαναλήψεων (π.χ. «μέτρα τα αυτοκίνητα που περνάνε μέχρι να περάσει κάποιο με κόκκινο χρώμα») και κάποιες άλλες όταν το γνωρίζουμε (π.χ. «ζήτα 100 αριθμούς και υπολόγισε το άθροισμά τους»).

Η C++ υποστηρίζει τρεις δομές επανάληψης: while, do-while και for. Οι δομές επανάληψης είναι παρόμοιες μεταξύ τους και στην πραγματικότητα, με λίγες προσθήκες, η λειτουργικότητα μιας δομής μπορεί να υλοποιηθεί από μια άλλη.



(α) Έλεγχος στην αρχή κάθε επανάληψης



(β) Έλεγχος στο τέλος κάθε επανάληψης

Σχήμα 6.1 *Δομές επανάληψης*

Ο βρόχος while

Ο βρόχος (loop) while αποτελεί τη μία από τις τρεις δομές επανάληψης της C++. Με τη χρήση της εντολής while είναι δυνατή η επαναληπτική εκτέλεση μίας ή περισσότερων προτάσεων. Το αν επαναλαμβάνονται οι προτάσεις εξαρτάται από τον έλεγχο μιας συνθήκης (συνήθως μιας λογικής παράστασης). Για παράδειγμα, ο παρακάτω βρόχος while εκτελείται όσο η τιμή της μεταβλητής a είναι μεγαλύτερη από το 0.

```
a=5;
while (a>0)
{
   cout << a << endl;
   --a;
}
```

Ελέγχεται αν η τιμή της μεταβλητής **a** είναι μεγαλύτερη από 0. Αν είναι, συνεχίζεται η εκτέλεση των εντολών του βρόχου.

Η εντολή (ή βρόχος) while επαναλαμβάνει μια πρόταση (απλή ή σύνθετη) ενόσω μια συνθήκη είναι αληθής. Το συντακτικό της while είναι το εξής:

while (*συνθήκη*) *πρόταση*; ή

while (*συνθήκη*)

**   *πρόταση*;** ή

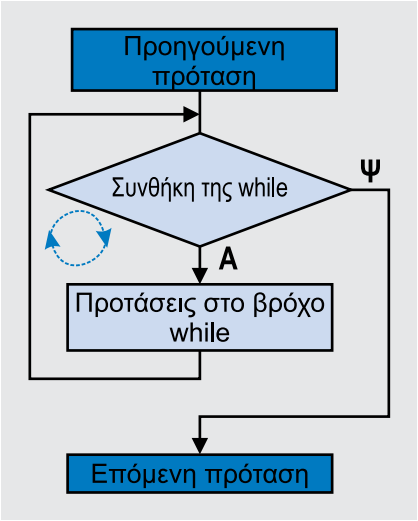
while (*συνθήκη*)

{

**   *πρόταση*;**

**   *πρόταση*;**

}



Το διπλανό λογικό διάγραμμα δείχνει τη φιλοσοφία της while. Υπενθυμίζουμε ότι μια συνθήκη στη C++ μπορεί να είναι μια οποιαδήποτε παράσταση η οποία επιστρέφει αριθμητική τιμή. Αν η τιμή της είναι 0, θεωρείται ψευδής, διαφορετικά θεωρείται αληθής.

Η παρακάτω εντολή while εμφανίζει συνέχεια την πρόταση «C++», αφού η συνθήκη έχει συνέχεια τιμή 1, δηλαδή είναι αληθής.

```
while (1)
   cout << "C++\n";
```

Το πρόγραμμα που ακολουθεί εμφανίζει την τιμή της μεταβλητής a, ενόσω αυτή είναι μεγαλύτερη του μηδενός. Μετά το τέλος του βρόχου, εμφανίζει τη λέξη «ΤΕΛΟΣ».

6_while.cpp

```
#include <iostream>
using namespace std;
int main()
{
   int a,b;
   a=5;
   while (a>0)
   {
      cout << "a=" << a << endl;
      --a;
   }
   cout << "Τέλος"<<endl;
   return 0;
}
```

Η εκτέλεση των προτάσεων θα σταματήσει να επαναλαμβάνεται όταν η τιμή της μεταβλητής **a** γίνει 0.

Ο βρόχος do-while

Η εντολή do-while κάνει παρόμοια δουλειά με τη while, με τη διαφορά ότι ελέγχει τη συνθήκη στο τέλος και όχι στην αρχή του βρόχου.

Το συντακτικό της do-while είναι:

do

***πρόταση*;**

while (*συνθήκη*);

ή

do

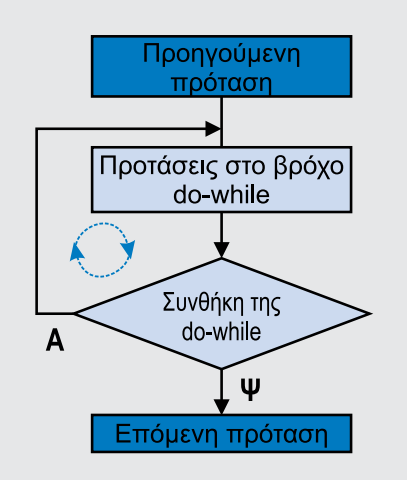
{

***πρόταση*;**

***πρόταση*;**

.....

} while (*συνθήκη*);



Το παραπάνω λογικό διάγραμμα δείχνει τη φιλοσοφία της do-while.

☞ Αντίθετα με τη while, η do-while χρειάζεται ερωτηματικό (;) μετά από τη συνθήκη.

☞ Παρατηρούμε ότι οι προτάσεις μέσα στον βρόχο του do-while θα εκτελεστούν *τουλάχιστον* μία φορά, ανεξάρτητα από την τιμή της συνθήκης.

Στο παρακάτω τμήμα κώδικα, ο χρήστης πληκτρολογεί έναν αριθμό ο οποίος καταχωρίζεται στη μεταβλητή a. Οι προτάσεις του βρόχου do-while εμφανίζουν τη τιμή της μεταβλητής a και ακολούθως τη μειώνουν κατά 1. Η επαναληπτική λειτουργία εκτελείται όσο η τιμή της μεταβλητής a είναι μεγαλύτερη από το 0.

Ένθετοι βρόχοι for

Ο βρόχος for μπορεί να περιέχει οποιαδήποτε πρόταση, επομένως και έναν άλλο βρόχο for.

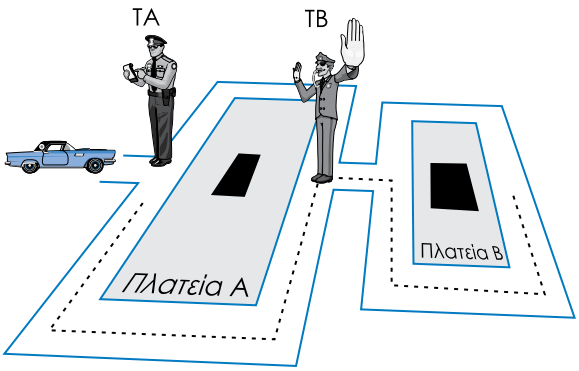
Στον επόμενο κώδικα ο εσωτερικός βρόχος θα εκτελεστεί πέντε (5) φορές λόγω του εξωτερικού for. Έτσι, η πρώτη cout θα εκτελεστεί δεκαπέντε φορές (5 x 3), ενώ η cout << "-----"... θα εκτελεστεί πέντε φορές επειδή βρίσκεται μόνο μέσα στον εξωτερικό βρόχο.

```
for (i=1;i<=5;i++)
{
   for (j=1;j<=3;j++)
   {
      cout << i << " " << j << endl;
   }
   cout << "-----" << endl;
}
```

```
1 1
1 2
1 3
-------
2 1
2 2
2 3
-------
.
.
.

5 1
5 2
5 3
-------
```

Στο επόμενο παράδειγμα γίνεται κατανοητή η λειτουργία των ένθετων βρόχων for με έναν εποπτικό τρόπο:

Υποθέτουμε ότι έχουμε δύο πλατείες, την πλατεία Α και την πλατεία Β. Στην είσοδο κάθε πλατείας υπάρχει ένας τροχονόμος (ΤΑ,ΤΒ).



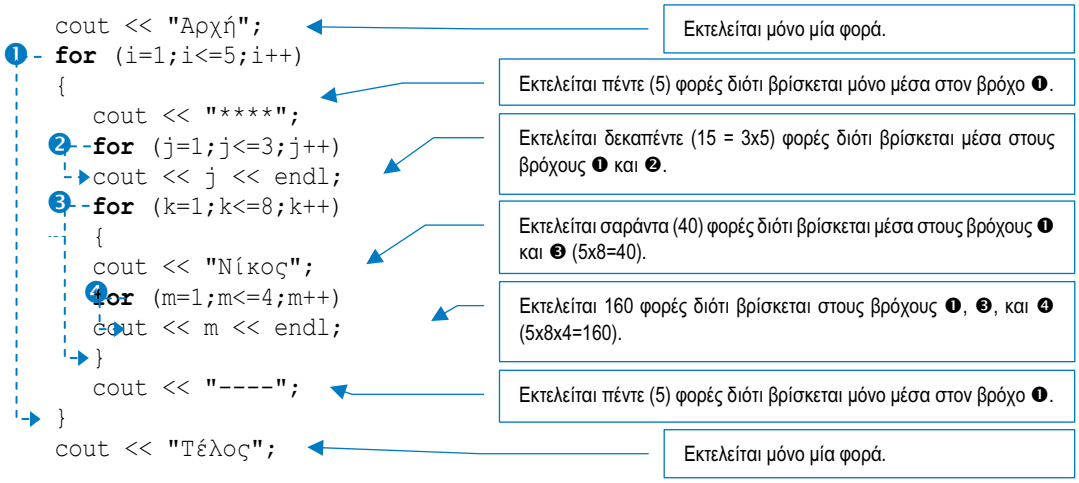
Ο τροχονόμος ΤΑ υποχρεώνει τα αυτοκίνητα που μπαίνουν στην πλατεία Α να κάνουν τον γύρο της πέντε φορές, ενώ ο τροχονόμος ΤΒ υποχρεώνει τα αυτοκίνητα που διέρχονται από την είσοδο της πλατείας Β να κάνουν τον γύρο της τρεις φορές.

Ας υποθέσουμε ότι ένα αυτοκίνητο μπαίνει στην πλατεία Α και περνάει από τον τροχονόμο ΤΑ, ο οποίος σημειώνει πόσες φορές πέρασε το αυτοκίνητο (την πρώτη φορά σημειώνει τον αριθμό 1). Μετά το αυτοκίνητο περνάει από την είσοδο της πλατείας Β, όπου ο τροχονόμος ΤΒ το υποχρεώνει να κάνει τον γύρο της πλατείας Β τρεις φορές. Βγαίνοντας από την πλατεία Β (αφού έχει κάνει τους τρεις γύρους), ξαναπερνά από τον ΤΑ (που σημειώνει τον αριθμό 2, δηλαδή ότι πέρασε για δεύτερη φορά) και ξαναμπαίνει στην πλατεία Β, όπου ο ΤΒ τον υποχρεώνει να κάνει πάλι τρεις γύρους κ.ο.κ. Αυτό θα συνεχιστεί μέχρι να περάσει πέντε φορές από τον ΤΑ, οπότε θα μπορεί να βγει από την πλατεία Α και να συνεχίσει τον δρόμο του. Το αυτοκίνητο συνολικά θα έχει κάνει πέντε φορές τον γύρο της πλατείας Α και δεκαπέντε φορές (3x5) τον γύρο της πλατείας Β.

Αν κάποια πρόταση περικλείεται από πολλούς βρόχους, τότε θα εκτελεστεί τόσες φορές όσο το γινόμενο των εκτελέσεων κάθε βρόχου ξεχωριστά.

Το τμήμα κώδικα που ακολουθεί εξηγεί τη χρήση των ένθετων βρόχων for.

Για παράδειγμα, η πρόταση cout << m << endl του βρόχου 4 θα εκτελεστεί 160 φορές (5*8*4), επειδή περικλείεται από τον βρόχο 1 (θα γίνει 5 φορές), τον βρόχο 3 (θα γίνει 8 φορές), και τον βρόχο 4 (θα γίνει 4 φορές).



Σκεφτείτε ποιες θα είναι οι τιμές των μεταβλητών i, j, k, και m μετά το τέλος του προηγούμενου κώδικα.

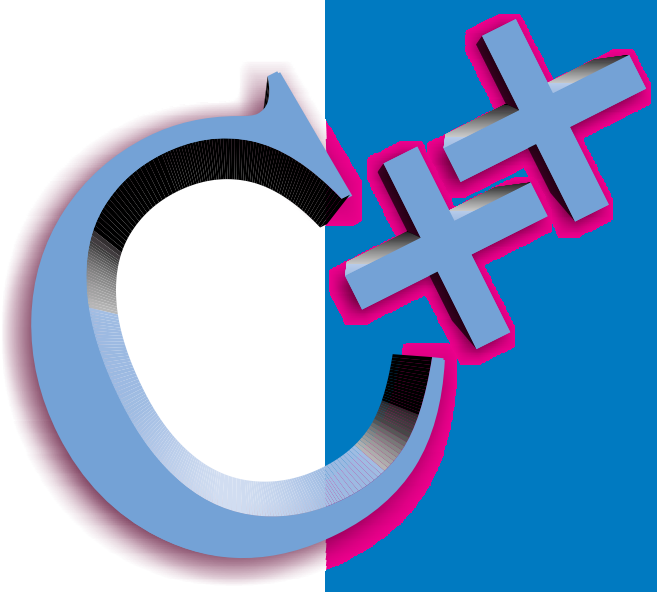
Για να σταματήσει η επαναληπτική διαδικασία ενός βρόχου for, θα πρέπει η συνθήκη του να αποκτήσει τιμή **false** (ψευδής). Επομένως, μετά το τέλος του παραπάνω κώδικα, για να έχουν σταματήσει οι επαναληπτικές διαδικασίες και των τεσσάρων βρόχων for, οι μεταβλητές τους θα πρέπει να έχουν πάρει τέτοιες τιμές ώστε οι συνθήκες τους να έχουν γίνει ψευδείς.

- Για τον βρόχο 1 η τιμή του i θα είναι 6
- Για τον βρόχο 2 η τιμή του j θα είναι 4
- Για τον βρόχο 3 η τιμή του k θα είναι 9
- Για τον βρόχο 4 η τιμή του m θα είναι 5

Έτσι, μετά το τέλος του προηγούμενου κώδικα, οι τιμές των μεταβλητών i, j, k, και m θα είναι 6, 4, 9 και 5 αντίστοιχα.

Κεφάλαιο 7

Συναρτήσεις



Συναρτήσεις

Σε αυτό το κεφάλαιο θα αναφερθούμε στη χρήση των συναρτήσεων στη C++. Μέχρι στιγμής είδαμε τη χρήση της συνάρτησης main(), η οποία είναι απαραίτητη σε κάθε πρόγραμμα της C++. Επίσης συναντήσαμε και ορισμένες συναρτήσεις βιβλιοθήκης, όπως η rand() και η sqrt().

Οι συναρτήσεις βοηθούν τον προγραμματιστή να «κομματιάσει» ένα μεγάλο πρόγραμμα σε μικρότερα τμήματα, κάθε ένα από τα οποία επιτελεί μια συγκεκριμένη λειτουργία. Στις γλώσσες δομημένου προγραμματισμού, οι συναρτήσεις μπορούν να «κρύβουν» κομμάτια κώδικα, με τέτοιο τρόπο ώστε οι μεταβλητές μιας συνάρτησης να μην επηρεάζουν το υπόλοιπο πρόγραμμα. Κάθε συνάρτηση έχει τις δικές της μεταβλητές οι οποίες καλούνται *τοπικές μεταβλητές*. Όπως αναφέραμε στην εισαγωγή, ένα πρόγραμμα στη C++ έχει την ακόλουθη γενική μορφή:

```
main()
{
   προτάσεις της συνάρτησης main();
}
function-1()
{
   προτάσεις της συνάρτησης function-1();
}
function-2()
{
   προτάσεις της συνάρτησης function-2();
}
function-N()
{
   προτάσεις της συνάρτησης function-N();
}
```

Οι συναρτήσεις εκτελούνται μόνο αν κληθούν από μια άλλη συνάρτηση. Η μοναδική συνάρτηση που εκτελείται αυτόματα είναι η συνάρτηση main().

Στη C++ κάθε συνάρτηση επιστρέφει μια τιμή, εκτός αν έχει δηλωθεί ρητά να μην επιστρέφει τιμή. Σε κάθε περίπτωση, μια συνάρτηση έχει έναν τύπο (int, float κ.λπ.) ανάλογα με τον τύπο δεδομένων της επιστρεφόμενης τιμής. Όπως θα γίνει κατανοητό στη συνέχεια του παρόντος κεφαλαίου, ακόμα και αν δεν επιστρέφει κάποια τιμή, θα πρέπει να δηλωθεί με τον κατάλληλο τύπο (void). Επίσης, μια συνάρτηση μπορεί να έχει παραμέτρους μέσω των οποίων ο κώδικας που καλεί τη συνάρτηση μπορεί να της μεταβιβάζει πληροφορίες.

Στη C++ κάθε συνάρτηση ανήκει σε κάποιο τύπο δεδομένων. Μια συνάρτηση μπορεί να έχει καμία, μία ή περισσότερες παραμέτρους.

Ορισμός μιας συνάρτησης

Μια συνάρτηση ορίζεται με τον ακόλουθο τρόπο:

τύπος_συνάρτησης *όνομα_συνάρτησης* (τύπος *παράμετρος1*, τύπος *παράμετρος2*, ...)

{

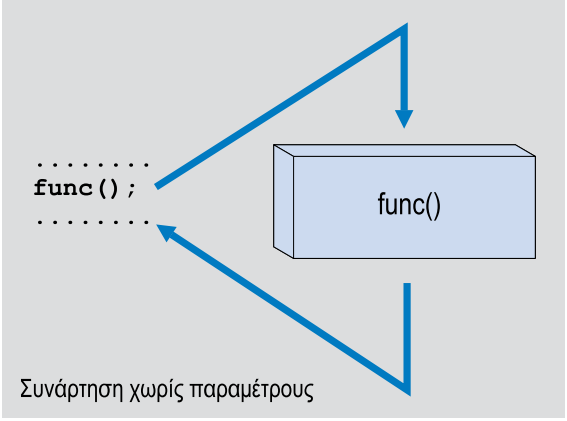
***προτάσεις συνάρτησης*;**

}

- Ο τύπος της συνάρτησης καθορίζει τον τύπο της τιμής που επιστρέφει η συνάρτηση. Στην περίπτωση που η συνάρτηση δεν επιστρέφει τιμή, δηλώνεται ως τύπου void.
- Στη C++ η δήλωση του τύπου μιας συνάρτησης είναι υποχρεωτική.
- Οι παράμετροι της συνάρτησης (αν υπάρχουν) ορίζονται, μαζί με τον τύπο τους, μέσα στις παρενθέσεις που ακολουθούν το όνομα της συνάρτησης.
- Οι παράμετροι μιας συνάρτησης αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης.

Συναρτήσεις χωρίς παραμέτρους

Στη πιο απλή τους μορφή οι συναρτήσεις δεν έχουν παραμέτρους. Σε αυτή την περίπτωση, ο κώδικας που τις καλεί δεν μπορεί να τους μεταβιβάσει πληροφορίες.



Συνάρτηση χωρίς παραμέτρους

Μπορούμε να παρομοιάσουμε μια τέτοια συνάρτηση με ένα ερμητικά κλειστό κουτί το οποίο επιτελεί κάποια λειτουργία, αλλά δεν έχουμε καμία δυνατότητα να τροποποιήσουμε τη λειτουργία του.

Για παράδειγμα, η παρακάτω συνάρτηση εμφανίζει τους αριθμούς από το 5 μέχρι το 10. Δεν επιστρέφει καμία τιμή και για τον λόγο αυτό δηλώνεται ως void. Δεν διαθέτει καμία παράμετρο και, κάθε φορά που καλείται, εκτελεί ακριβώς την ίδια λειτουργία.

```
void display_numbers()
{
   int i;
   for (i=5;i<=10;i++) cout << i <<endl;
}
```

Στο πρόγραμμα που ακολουθεί, η main() καλεί τη συνάρτηση display_numbers() δύο φορές. Κάθε φορά η συνάρτηση εμφανίζει τους ίδιους αριθμούς. Από τη στιγμή που η συνάρτηση δεν έχει παραμέτρους δεν υπάρχει τρόπος να της μεταβιβάσουμε πληροφορίες και να επηρεάσουμε με αυτόν τον τρόπο τη λειτουργία της.

7_synartisi1.cpp

```
#include <iostream>
using namespace std;
void display_numbers()
{
   int i;
   for (i=5;i<=10;i++) cout << i <<endl;
}
int main()
{
   cout << "Πρώτη κλήση της συνάρτησης" << endl;
   display_numbers();
   cout << "Δεύτερη κλήση της συνάρτησης" << endl;
   display_numbers();
   return 0;
}
```

Η συνάρτηση εμφανίζει τους αριθμούς 5, 6, 7, 8, 9, και 10.

Κλήση της συνάρτησης

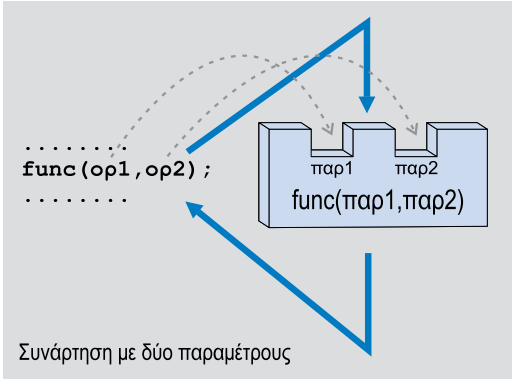
Κλήση της συνάρτησης

Συναρτήσεις με παραμέτρους

Οι παράμετροι είναι ένας τρόπος με τον οποίο ένα πρόγραμμα μεταβιβάζει πληροφορίες σε μια συνάρτηση.

Μια τέτοια συνάρτηση τη φανταζόμαστε σαν ένα κουτί το οποίο διαθέτει εγκοπές στις οποίες μπορούμε να τοποθετήσουμε πληροφορίες. Ο κώδικας της συνάρτησης χρησιμοποιεί αυτές τις πληροφορίες και μπορεί να τροποποιεί ανάλογα τη λειτουργία της. Όταν καλούμε μια συνάρτηση με παραμέτρους, την καλούμε με ισάριθμα ορίσματα.

Το διπλανό σχήμα δείχνει μια συνάρτηση με δύο παραμέτρους func(παρ1,παρ2) σαν ένα κουτί με δύο εγκοπές. Όταν καλούμε τη συνάρτηση func(), πρέπει να χρησιμοποιήσουμε ισάριθμα ορίσματα func(ορ1,ορ2). Η τιμή του πρώτου ορίσματος μεταβιβάζεται στη πρώτη παράμετρο (εγκοπή), και του δεύτερου ορίσματος στη δεύτερη.



Συνάρτηση με δύο παραμέτρους

Στο πρόγραμμα που ακολουθεί η συνάρτηση display_numbers() είναι μια παραλλαγή της συνάρτησης που συναντήσαμε στη προηγούμενη ενότητα, η οποία τώρα διαθέτει δύο παραμέτρους, ώστε ο κώδικας που την καλεί να μπορεί να καθορίζει το εύρος των αριθμών που εμφανίζει.

Ορίσματα και μεταβίβαση παραμέτρων

Οι παράμετροι μιας συνάρτησης χρησιμοποιούνται για τη μεταβίβαση δεδομένων από το πρόγραμμα προς τη συνάρτηση την οποία καλεί.

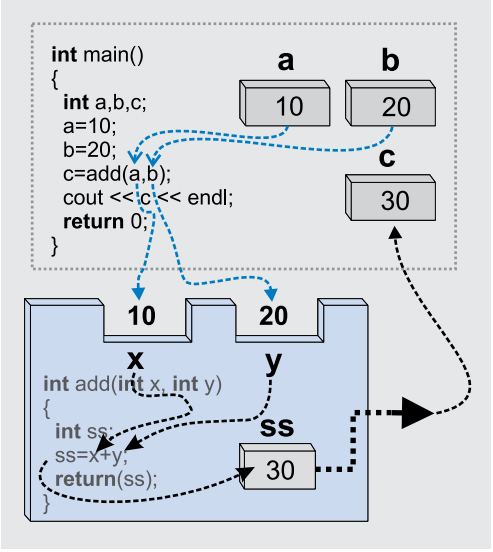
> Οι τιμές με τις οποίες καλείται μια συνάρτηση λέγονται *ορίσματα* κλήσης.

Ας δούμε το επόμενο παράδειγμα κλήσης της συνάρτησης add() από τη συνάρτηση main().

```
int main()
{
   int a,b,c;
   a=10;
   b=20;
   c=add(a,b);
   cout << c << endl;
   return 0;
}
int add(int x, int y)
{
   int ss;
   ss=x+y;
   return(ss);
}
```

Ορίσματα με τα οποία καλείται η συνάρτηση

Παράμετροι συνάρτησης



Η συνάρτηση add() έχει οριστεί με δύο παραμέτρους x και y. Όταν καλείται η συνάρτηση, οι τιμές των ορισμάτων της (το 10 και το 20) αντιγράφονται στις παραμέτρους x και y αντίστοιχα. Η τιμή που επιστρέφει η συνάρτηση καταχωρίζεται στη μεταβλητή c. Το σχήμα δείχνει εποπτικά αυτή τη διαδικασία.

Αυτός ο τρόπος μεταβίβασης ορισμάτων λέγεται **κλήση** ή **μεταβίβαση κατ' αξία** (*by value*), κατά τον οποίο αντιγράφονται οι τιμές των ορισμάτων στις αντίστοιχες παραμέτρους της συνάρτησης.

Οι μεταβλητές των παραμέτρων που ορίζονται σε μια συνάρτηση (στην περίπτωση μας οι x και y) λέγονται **τυπικές παράμετροι** (*formal parameters*) της συνάρτησης.

☞ Όταν καλείται μια συνάρτηση, το πλήθος και ο τύπος των ορισμάτων πρέπει να είναι αντίστοιχα με τις τυπικές παραμέτρους της, όπως αυτές έχουν δηλωθεί στον ορισμό της συνάρτησης.

Έτσι, αν η συνάρτηση add() κληθεί με ορίσματα που δεν αποδίδουν ακέραιες τιμές, τότε το αποτέλεσμα δεν θα είναι αυτό που περιμένουμε.

Υπερφόρτωση συναρτήσεων

Η C++ επιτρέπει τον ορισμό διαφορετικών συναρτήσεων με το ίδιο όνομα. Η τεχνική αυτή καλείται **υπερφόρτωση συνάρτησης** (function overloading) και αποτελεί ένα από τα χαρακτηριστικά του πολυμορφισμού στη C++.

Η υπερφόρτωση μιας συνάρτησης επιτυγχάνεται με τον ορισμό διαφορετικών εκδόσεων της συνάρτησης με το ίδιο όνομα. Κάθε έκδοση πρέπει να έχει διαφορετικό πλήθος ή τύπο παραμέτρων. Όταν καλείται μια υπερφορτωμένη συνάρτηση, εκτελείται η έκδοση της συνάρτησης της οποίας οι παράμετροι ταιριάζουν με τα ορίσματα με τα οποία την καλούμε.

Στο παρακάτω παράδειγμα η συνάρτηση test() υπερφορτώνεται τρεις φορές:

7_function_overloading1.cpp

```
#include <iostream>
using namespace std;
void test(int x, int y);
void test(int x);
void test(double x);
int main()
{
   test(23);
   test(4,10);
   test(34.56);
   return 0;
}
```

Προκαταβολικές δηλώσεις των διαφορετικών εκδόσεων της υπερφορτωμένης συνάρτησης **test()**.

Καλείται η δεύτερη έκδοση της υπερφορτωμένης συνάρτησης **test()**, αφού καλείται με ένα όρισμα τύπου **int**.

Καλείται η πρώτη έκδοση της υπερφορτωμένης συνάρτησης **test()**,αφού καλείται με δύο ορίσματα τύπου **int**.

Καλείται η τρίτη έκδοση της υπερφορτωμένης συνάρτησης αφού καλείται με ένα όρισμα τύπου **double**.

```
// Πρώτη έκδοση της test()
void test(int x, int y)
{
   cout << "Πρώτη έκδοση της test()" << endl;
   cout << "Με δύο παραμέτρους int : " << x << "," << y << endl;
}
// Δεύτερη έκδοση της test()
void test(int x)
{
   cout << "Δεύτερη έκδοση της test()" << endl;
   cout << "Με μια παράμετρο int: " << x << endl;
}
// Τρίτη έκδοση της test()
void test(double x)
{
   cout << "Τρίτη έκδοση της test()" << endl;
   cout << "Με μια παράμετρο double: " << x << endl;
}
```

Ασκήσεις Κεφαλαίου 7

7.1 Βρείτε τα λάθη του επόμενου προγράμματος: ★

```
#include <iostream>
using namespace std;
int main()
{
   float x,y;
   cout << "**************" << endl;
   cin >> x >> y;
   cout << "mo=" << mo(x,y) << endl;
   return 0;
}
float mo(int a, b);
{
   float mesos;
   if (a==0 && b==0)
      return 0;
   else
      mesos=(a+b)/2.0;
   return mesos;
}
```

7.2 Να γραφεί μια συνάρτηση που να κάνει τα εξής:

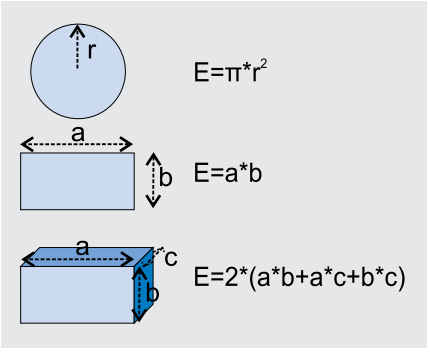
- Να δέχεται τρεις παραμέτρους.
- Αν η πρώτη παράμετρος είναι 1, να επιστρέφει ως τιμή το άθροισμα των δύο άλλων παραμέτρων.
- Αν η πρώτη παράμετρος είναι 2, να επιστρέφει ως τιμή το γινόμενο των δύο άλλων παραμέτρων.
- Αν η πρώτη παράμετρος είναι 3, να επιστρέφει ως τιμή το μέσο όρο των δύο άλλων παραμέτρων.
- Αν η πρώτη παράμετρος δεν είναι ούτε 1, ούτε 2, ούτε 3, να εμφανίζει στην οθόνη το μήνυμα λάθους «Αντικανονική κλήση συνάρτησης» και να σταματάει το πρόγραμμα με κωδικό εξόδου 1.

Να επιλέξετε εσείς τον τύπο της συνάρτησης και τον τύπο των παραμέτρων της. ★★

7.3 Να γραφεί μια συνάρτηση με όνομα total() που να δέχεται ως παράμετρο έναν αριθμό και να επιστρέφει ως τιμή το άθροισμα των αριθμών από το 1 μέχρι την τιμή της παραμέτρου. Για παράδειγμα, η total(1250) πρέπει να επιστρέφει ως τιμή το άθροισμα των αριθμών από το 1 μέχρι το 1250. ★★

```
void do_it(int a)
{
   int i;
   for (i=1;i<=a;i++)
      cout<<"Αιγαίο"<<endl;
}
void do_it()
{
   int i;
   for (i=1;i<=3;i++)
      cout<<"Λέσβος"<<endl;
}
double do_it(double a)
{
   int i;
   cout << a << endl;
   return a/2;
}
```

7.22 Θέλουμε να υπολογίζουμε το εμβαδόν της επιφάνειας ενός κύκλου, ενός ορθογωνίου παραλληλογράμμου και ενός ορθογωνίου παραλληλεπιπέδου. Οι μαθηματικοί τύποι υπολογισμού των εμβαδών αυτών φαίνονται στο διπλανό σχήμα. Ο υπολογισμός θα γίνεται από μια συνάρτηση με όνομα emvadon στην οποία θα μεταβιβάζονται οι διαστάσεις του σχήματος η οποία θα επιστρέφει ως τιμή το εμβαδόν του. Η συνάρτηση θα πρέπει να υπερφορτωθεί τρεις φορές ώστε, όταν καλείται με ένα όρισμα να υπολογίζει το εμβαδόν του κύκλου, όταν καλείται με δύο ορίσματα να υπολογίζει το εμβαδόν του ορθογωνίου παραλληλογράμμου, και όταν καλείται με τρία ορίσματα να υπολογίζει το εμβαδόν του ορθογωνίου παραλληλεπιπέδου. Να κατασκευαστεί ένα πρόγραμμα το οποίο να καλεί και τις τρεις εκδόσεις της συνάρτησης ώστε να υπολογίζει και να εμφανίζει το εμβαδόν ενός κύκλου ακτίνας 20 μέτρων, το εμβαδόν ενός ορθογωνίου παραλληλογράμμου διαστάσεων 5x10 μέτρων και το εμβαδόν ενός ορθογωνίου παραλληλεπιπέδου διαστάσεων 4x6x8 μέτρων. Το π (3.14159) να δηλώνεται ως σταθερά. ★★★

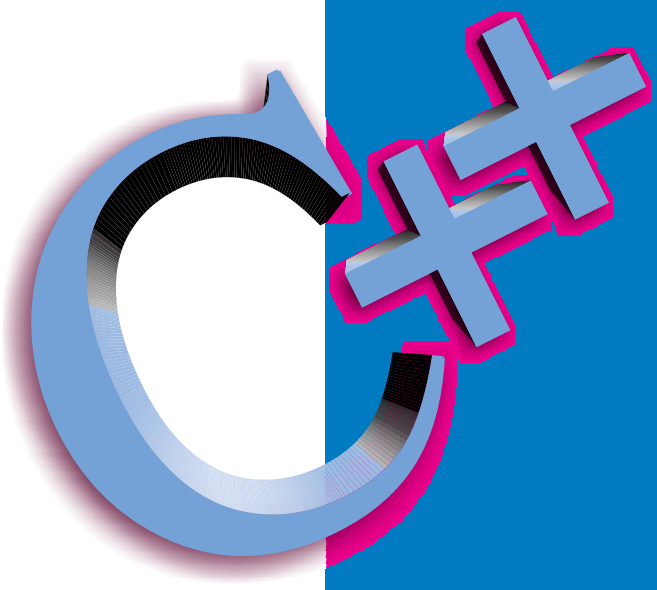


7.23 Τροποποιήστε το πρόγραμμα της προηγούμενης άσκησης χωρίς να υπερφορτώσετε τη συνάρτηση embadon αλλά να επιτύχετε το ίδιο ακριβώς αποτέλεσμα χρησιμοποιώντας προκαθορισμένες τιμές παραμέτρων και αρκετή σκέψη! ★★★

Κεφάλαιο

8

Εμβέλεια μεταβλητών



Εμβέλεια μεταβλητών

Σε ένα πρόγραμμα της C++ μπορεί να δηλωθούν μεταβλητές σε διαφορετικά σημεία του προγράμματος. Η θέση και το είδος της δήλωσης μιας μεταβλητής καθορίζει την *εμβέλειά* της (variable scope).

Εμβέλεια μιας μεταβλητής είναι ο χώρος του προγράμματος μέσα στον οποίο είναι γνωστή η μεταβλητή.

Ανάλογα με τη θέση και τον τρόπο δήλωσής τους, οι μεταβλητές χωρίζονται σε **τοπικές** (local), **καθολικές** (global), και **στατικές** (static). Επίσης, ο *χρόνος ζωής* μιας μεταβλητής (variable life span) διαφέρει ανάλογα με το είδος της μεταβλητής και τη θέση δήλωσής της.

Ο χρόνος ζωής μιας μεταβλητής καθορίζει τον χρόνο κατά τον οποίο υφίσταται η μεταβλητή. Μια μεταβλητή «γεννιέται» κατά την πρόταση δήλωσής της και «πεθαίνει» όταν λήξει ο χρόνος ζωής της. Όταν «πεθάνει» μια μεταβλητή αποδεσμεύονται οι θέσεις μνήμης που καταλάμβανε και χάνονται τα περιεχόμενά της.

Τοπικές μεταβλητές (local variables)

Όπως έχουμε δει μέχρι τώρα, οι μεταβλητές μιας συνάρτησης δηλώνονται σε οποιοδήποτε σημείο της, αρκεί να έχουν δηλωθεί πριν να χρησιμοποιηθούν. Στην περίπτωση που η συνάρτηση έχει τυπικές παραμέτρους, αυτές δηλώνονται μέσα στις παρενθέσεις. Οι τυπικές παράμετροι είναι και αυτές μεταβλητές της συνάρτησης.

Οι μεταβλητές μιας συνάρτησης, καθώς και οι τυπικές παράμετροί της (αν υπάρχουν), καλούνται **τοπικές μεταβλητές** (local variables) και έχουν ισχύ μόνο μέσα στη συνάρτηση που δηλώθηκαν, παραμένοντας άγνωστες στο υπόλοιπο πρόγραμμα.

Με το παράδειγμα που ακολουθεί γίνεται κατανοητή η χρήση των τυπικών παραμέτρων και των τοπικών μεταβλητών.

```
#include <iostream>
using namespace std;
int add(int x, int y);
int main()
{
   int a,b;
   cout << "Δώσε δύο αριθμούς: ";
   cin >> a >> b;
   int sum;
   sum=add(a,b);
   cout << "Άθροισμα=" << sum << endl;
   return 0;
}
```

Προκαταβολική δήλωση της συνάρτησης **add()**.

Τοπικές μεταβλητές της συνάρτησης **main()**. Έχουν εμβέλεια μόνο μέσα στη συνάρτηση, από το σημείο που δηλώθηκαν και μετά.

Τοπική μεταβλητή της **main()**. Έχει εμβέλεια μόνο μέσα στη συνάρτηση, από το σημείο που δηλώθηκε και μετά.

Στην επόμενη συνάρτηση add(), η μεταβλητή ss είναι τοπική μεταβλητή της συνάρτησης και έχει ισχύ μόνο μέσα στη συνάρτηση add(). Επίσης, οι τυπικές παράμετροι x και y αποτελούν και αυτές τοπικές μεταβλητές της συνάρτησης και έχουν ισχύ μόνο μέσα σε αυτή τη συνάρτηση.

```
int add(int x, int y)
{
   int ss;
   ss=x+y;
   return ss;
}
```

Δήλωση τυπικών παραμέτρων της συνάρτησης **add()**. Οι τυπικές παράμετροι παίζουν επίσης τον ρόλο τοπικών μεταβλητών.

Δήλωση τοπικής μεταβλητής **ss**.

Η συνάρτηση **add()** επιστρέφει ως τιμή το περιεχόμενο της μεταβλητής **ss**.

Οι τυπικές παράμετροι παίζουν και αυτές τον ρόλο των τοπικών μεταβλητών, με τη διαφορά ότι μέσα σε αυτές αντιγράφονται οι τιμές των ορισμάτων με τα οποία καλείται η συνάρτηση.

Αν δεν γίνει κατανοητή η χρήση και η «εμβέλεια» των τοπικών μεταβλητών, είναι πολύ εύκολο να δημιουργηθούν σφάλματα στα προγράμματά μας. Στη συνέχεια βλέπετε τον κώδικα του προηγούμενου παραδείγματος γεμάτο με λάθη.

```
int main()
{
   int sum;
   float a,b;
   cout << "Δώσε δύο αριθμούς:";
   cin >> a >> b;
   sum=add(a,b);
   cout << "ss=" << ss << endl;
   cout << "Άθροισμα=" << sum << endl;
   return 0;
}
int add(int x, int y)
{
   int ss;
   ss=a+b;
   return ss;
}
```

Η συνάρτηση **main()** αγνοεί την ύπαρξη της **ss**, η οποία είναι τοπική μεταβλητή της **add()**.

Η συνάρτηση **add()** αγνοεί την ύπαρξη των **a** και **b**, οι οποίες είναι τοπικές μεταβλητές της **main()**.

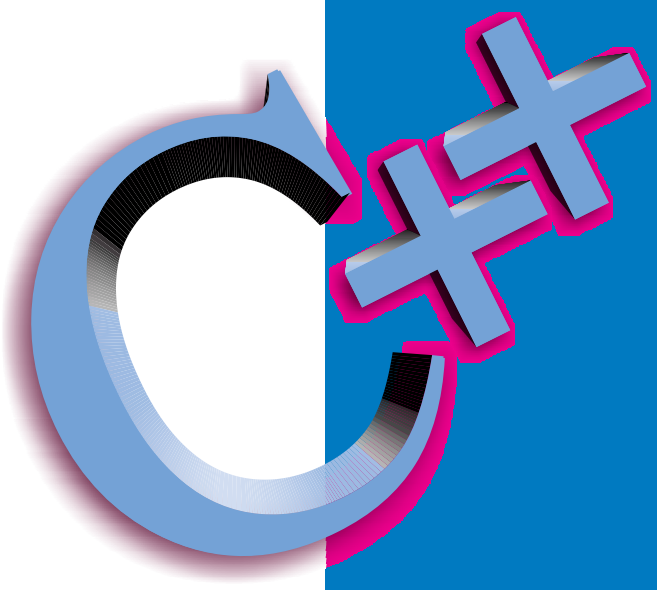
Στη συνέχεια βλέπετε ένα ολοκληρωμένο πρόγραμμα με τέσσερις συναρτήσεις (εκτός από τη main()). Το πρόγραμμα ζητάει δύο ακέραιους αριθμούς, και υπολογίζει το άθροισμα, το γινόμενο, και τον μέσο όρο τους. Τέλος, εμφανίζει τα αποτελέσματα στην οθόνη. Κάθε μία από αυτές τις λειτουργίες υλοποιείται από διαφορετική συνάρτηση.

8_emveleia.cpp

```
#include <iostream>
using namespace std;
```

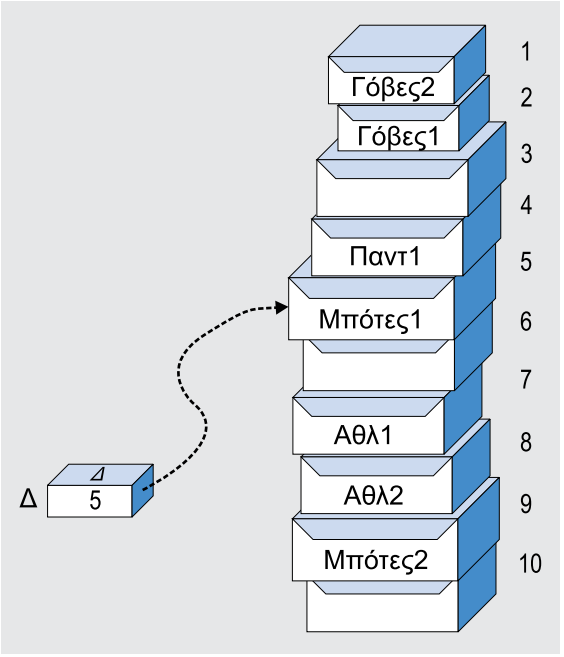
Κεφάλαιο 9

Δείκτες



Δείκτες

Οι δείκτες (pointers) είναι ένα από τα πιο παρεξηγημένα ζητήματα των γλωσσών προγραμματισμού. Οι αρχάριοι στον προγραμματισμό δυσκολεύονται αρκετά να τους κατανοήσουν, ενώ οι πιο πεπειραμένοι «σπαζοκεφαλιάζουν» με τις ιδιοτροπίες τους. Η βασική φιλοσοφία των δεικτών, όμως, είναι πολύ απλή.



Ας φανταστούμε κουτιά διαφόρων μεγεθών, το ένα πάνω από το άλλο, τα οποία περιέχουν παπούτσια. Κάποια κουτιά είναι άδεια και κάποια γεμάτα. Σε κάθε γεμάτο κουτί αναγράφεται ένα ενδεικτικό όνομα ώστε να καταλαβαίνουμε τι περιέχει. Επίσης, κάθε κουτί έχει έναν αύξοντα αριθμό (1,2,3 ....) ο οποίος προσδιορίζει τη σειρά του στη στοίβα. Σε ένα ξεχωριστό κουτί (Δ) βάζουμε τον αύξοντα αριθμό του κουτιού με τα παπούτσια που θα φορέσουμε το βράδυ, π.χ. το 5 αν θέλουμε να βάλουμε το ζευγάρι που βρίσκεται στο κουτί Μπότες1 (με α/α 5).

> Το κουτί Δ δείχνει το κουτί που θα χρησιμοποιήσουμε (το κουτί με α/α 5). Επομένως το κουτί Δ είναι ένας *δείκτης* ο οποίος δείχνει στο κουτί Μπότες1.

Θα μπορούσαμε φυσικά να θυμόμαστε το κουτί που θα χρησιμοποιούσαμε από το όνομά του (Μπότες1). Όμως η χρήση του δείκτη Δ είναι ένας έμμεσος τρόπος πρόσβασης στο κουτί Μπότες1. Έτσι, λοιπόν, για να φορέσουμε τα παπούτσια μας το βράδυ, έχουμε δύο επιλογές:

- Να ανοίξουμε το κουτί Μπότες1 (αν το θυμόμαστε) — ο άμεσος τρόπος.
- Να ανοίξουμε το κουτί Δ, να βρούμε ότι το κουτί που μας ενδιαφέρει είναι το κουτί με α/α 5 και να ανοίξουμε το συγκεκριμένο κουτί. Με αυτόν τον τρόπο δεν χρειάζεται να γνωρίζουμε το όνομα του κουτιού.

Στο συγκεκριμένο παράδειγμα, το κουτί-δείκτης είναι ένα κουτί το οποίο περιέχει τον α/α ενός άλλου κουτιού.

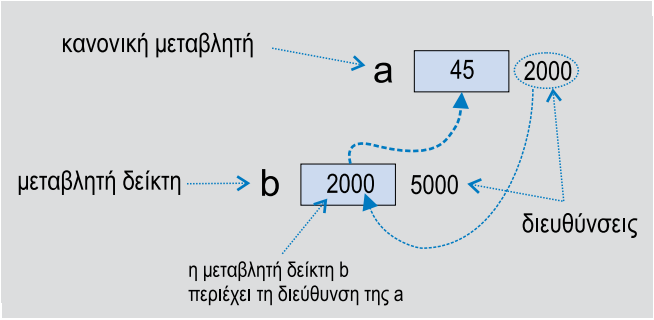
> Το μέγεθος του κουτιού-δείκτη είναι *ανεξάρτητο* από το μέγεθος του κουτιού στο οποίο δείχνει, αφού σε κάθε περίπτωση θα περιέχει μόνο τον α/α του.

Η C++ και οι δείκτες

Η C++ διαθέτει τις ίδιες δυνατότητες στη χρήση των δεικτών όπως και η C. Όμως, τα νέα χαρακτηριστικά που διαθέτει η γλώσσα, πολλές φορές μας γλυτώνουν από τον πονοκέφαλο των δεικτών τον οποίο σίγουρα δεν μπορούμε να αποφύγουμε στη C. Μη νομίζετε όμως ότι θα τη γλυτώσετε, παρά το γεγονός ότι η χρήση των δεικτών στη C++ είναι πιο περιορισμένη δεν παύουν να αποτελούν ένα σημαντικό χαρακτηριστικό της γλώσσας. Σημειώστε ότι άλλες αντικειμενοστρεφείς γλώσσες υποστηρίζουν ελάχιστα (C#) ή και καθόλου (Java) δείκτες. Στη συνέχεια του κεφαλαίου θα εξετάσουμε τη χρήση των δεικτών στη C++ η οποία είναι πανομοιότυπη με αυτή της C.

Μεταβλητές δείκτη (pointer variables)

Οι μεταβλητές δείκτη έχουν ένα ιδιαίτερο χαρακτηριστικό. Μέσα σε μια μεταβλητή δείκτη καταχωρίζεται η διεύθυνση μιας άλλης μεταβλητής. Η φιλοσοφία μιας μεταβλητής δείκτη φαίνεται στο διπλανό σχήμα. Στη μεταβλητή δείκτη b έχει καταχωριστεί η διεύθυνση της μεταβλητής a (b=&a).



Δήλωση μιας μεταβλητής δείκτη

Μια μεταβλητή δείκτη δηλώνεται όπως οποιαδήποτε άλλη μεταβλητή, με τη διαφορά ότι πριν από το όνομά της πρέπει να υπάρχει ο τελεστής *. Για παράδειγμα, η πρόταση:

```
int a,*b;
```

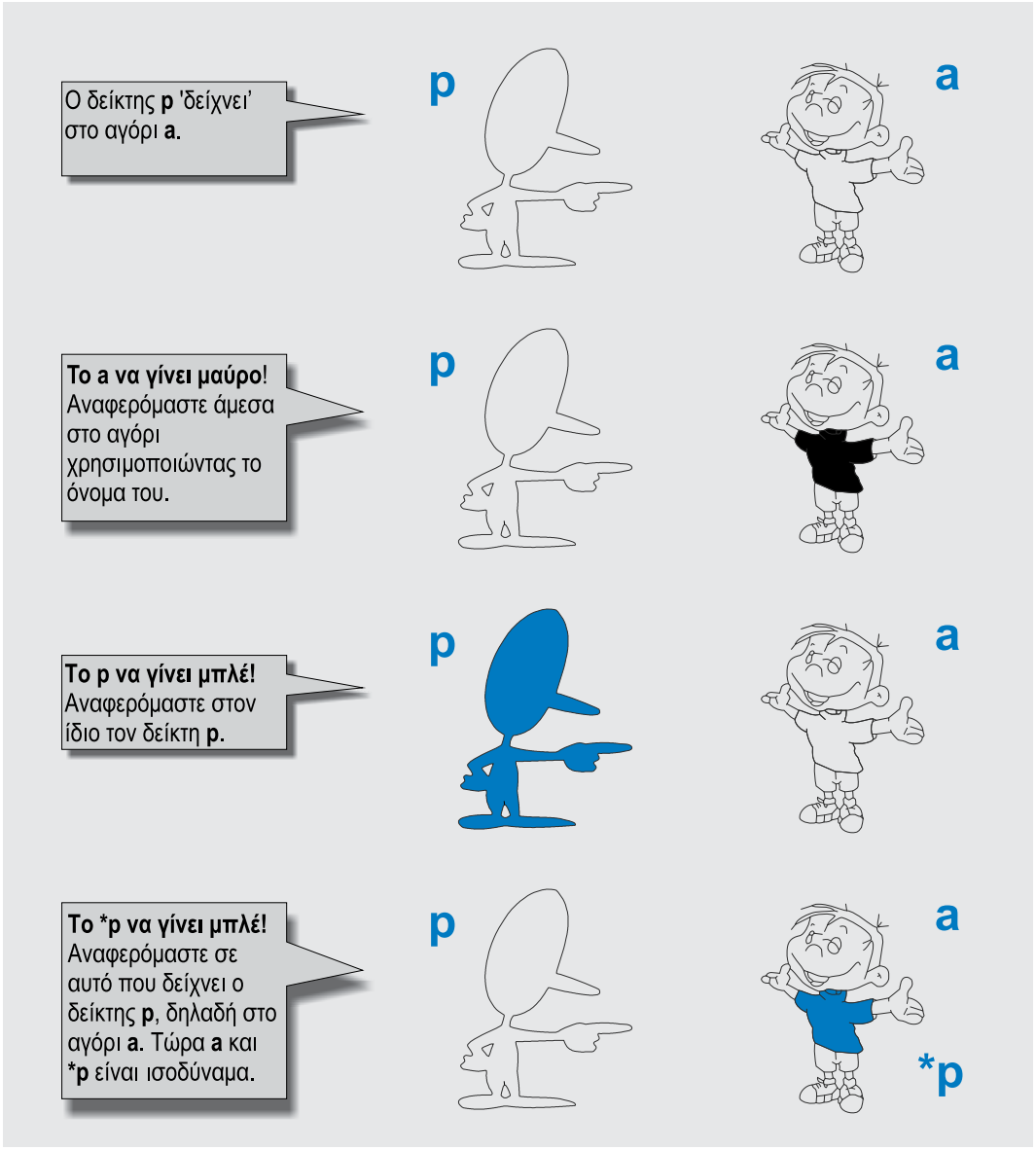
δηλώνει μια μεταβλητή a τύπου int και μια μεταβλητή δείκτη b.

Το γεγονός ότι η συγκεκριμένη μεταβλητή δείκτη δηλώθηκε ως τύπου int προϊδεάζει ότι η μεταβλητή b θα περιέχει τη διεύθυνση μιας μεταβλητής τύπου int.

Όπως θα δούμε αργότερα, ο τύπος της θέσης μνήμης στην οποία «δείχνει» μια μεταβλητή δείκτη παίζει καθοριστικό ρόλο στην αριθμητική δεικτών.

> Προς το παρόν, αρκεί να γνωρίζουμε ότι μια μεταβλητή δείκτη περιέχει τη διεύθυνση μιας άλλης μεταβλητής.

☞ Το μέγεθος μιας μεταβλητής δείκτη *δεν* εξαρτάται από τον τύπο της θέσης μνήμης στην οποία δείχνει η μεταβλητή. Όλες οι μεταβλητές δείκτη, ανεξάρτητα από την πρόταση δήλωσης με την οποία δηλώθηκαν (char, int, double κ.λπ.), έχουν ακριβώς το ίδιο μέγεθος.



Σχήμα 9.1 *Προσπέλαση μιας οντότητας είτε άμεσα με το όνομά της είτε έμμεσα μέσω ενός δείκτη.*

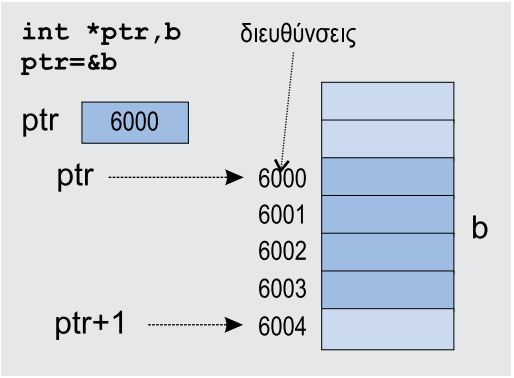
Μέχρι τώρα μάθαμε ότι ...

- Οι δείκτες είναι μεταβλητές που περιέχουν διευθύνσεις μνήμης, συνήθως διευθύνσεις άλλων μεταβλητών.
- Το μέγεθος μιας μεταβλητής δείκτη είναι συγκεκριμένο και ανεξάρτητο από τον τύπο με τον οποίο δηλώθηκε.
- Ο τελεστής & αποδίδει τη διεύθυνση μιας θέσης μνήμης.
- Ο τελεστής αναφοράς * χρησιμοποιείται σε συνδυασμό με μια μεταβλητή δείκτη για να αναφερόμαστε στη μεταβλητή στην οποία δείχνει ο δείκτης.

Αριθμητική δεικτών

Οι δείκτες ακολουθούν τη δική τους λογική στις αριθμητικές πράξεις, το οποίο δεν είναι καθόλου παράδοξο αν εμβαθύνουμε στη βαθύτερη φιλοσοφία τους.

Ας θεωρήσουμε μια μεταβλητή δείκτη με όνομα ptr, η οποία δηλώνεται στο επόμενο πρόγραμμα μαζί με μια μεταβλητή b. Υποθέτουμε ότι η μεταβλητή b έχει διεύθυνση 6000.




```
int *ptr,b;
```

```
ptr=&b;        ◅  η μεταβλητή ptr θα πάρει την τιμή 6000
ptr=ptr+1;     ◅  η μεταβλητή ptr θα πάρει την τιμή 6004
++ptr;         ◅  η μεταβλητή ptr θα πάρει την τιμή 6008
ptr=ptr+3;     ◅  η μεταβλητή ptr θα πάρει την τιμή 6020
```

Το παράδοξο είναι ότι, ενώ αυξάνουμε τη μεταβλητή ptr κατά 1, αυτή αυξάνεται κατά 4, και όταν την αυξήσουμε κατά 3, αυτή αυξάνεται κατά 12!

Πίσω από αυτό το παράδοξο κρύβονται κάποια κρυφά μυστικά των δεικτών.

> Μια μεταβλητή δείκτη δείχνει σε μια μεταβλητή συγκεκριμένου τύπου. Αν αυξήσουμε τη μεταβλητή δείκτη κατά ένα, τότε αυτή θα πάρει τέτοια τιμή ώστε να δείχνει στην επόμενη διεύθυνση του ίδιου τύπου.

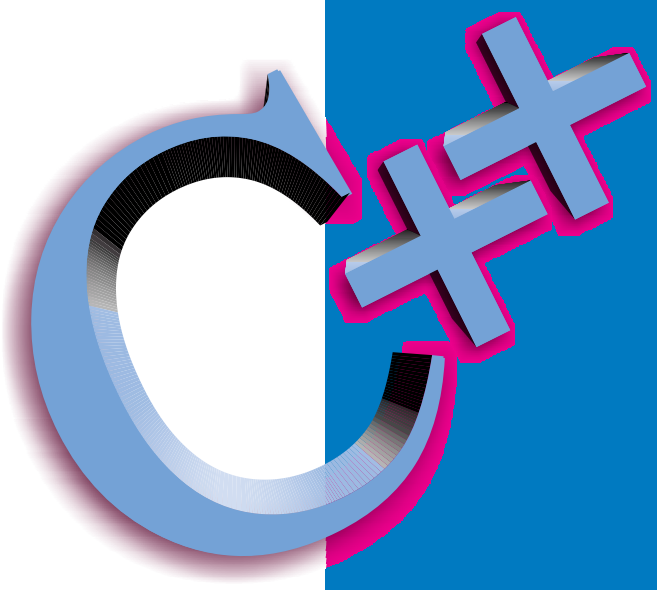
Επομένως, αν αυξήσουμε μια μεταβλητή δείκτη κατά ένα, τότε η διεύθυνση που περιέχει:

- θα αυξηθεί κατά 1 όταν δείχνει σε μεταβλητή τύπου char ή bool
- θα αυξηθεί κατά 4 όταν δείχνει σε μεταβλητή τύπου int

Κεφάλαιο

10

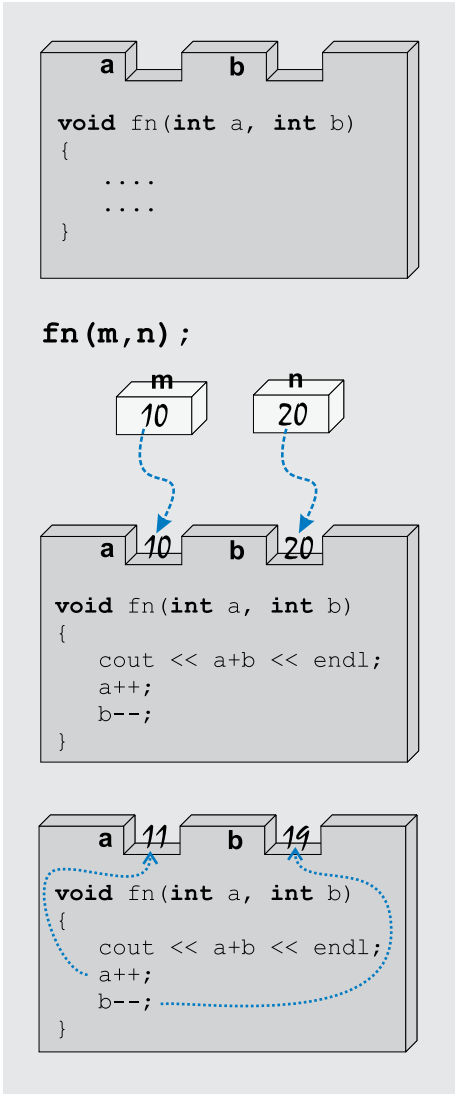
Προχωρημένα θέματα συναρτήσεων



Κλήση συνάρτησης με τιμή, δείκτη και αναφορά

Όπως αναφέρθηκε και στο κεφάλαιο των συναρτήσεων, ο προκαθορισμένος τρόπος κλήσης μιας συνάρτησης είναι κατ' αξία (*call by value*). Δηλαδή, κατά την κλήση της συνάρτησης αντιγράφονται οι τιμές των ορισμάτων στις αντίστοιχες τυπικές παραμέτρους της.

Ας φανταστούμε μια συνάρτηση με τον αγαπημένο μας εποπτικό τρόπο, δηλαδή ως ένα κουτί το οποίο κάνει κάποια συγκεκριμένη λειτουργία. Το κουτί αυτό έχει τόσες εγκοπές όσες οι παράμετροι της συνάρτησης. Μέσω των εγκοπών αυτών ο κώδικας που καλεί τη συνάρτηση μεταβιβάζει πληροφορίες τις οποίες θα χρησιμοποιήσει η συνάρτηση. Το διπλανό σχήμα δείχνει εποπτικά τον τρόπο μεταβίβασης κατ' αξία (*by value*). Η συνάρτηση fn() έχει δύο παραμέτρους, την a και την b. Η κλήση της συνάρτησης με ορίσματα τις μεταβλητές m και n: fn(m,n) έχει ως αποτέλεσμα την αντιγραφή της τιμής του κάθε ορίσματος στην αντίστοιχη «εγκοπή» (παράμετρο) της συνάρτησης. Επομένως αντιγράφονται το 10 και το 20 στις «εγκοπές» a και b αντίστοιχα. Η συνάρτηση χρησιμοποιεί τα περιεχόμενα των «εγκοπών» (παραμέτρων) a και b, και εμφανίζει το άθροισμά τους, το 30. Παρατηρούμε επίσης ότι οι προτάσεις αύξησης a++ και μείωσης b-- επηρεάζουν τις τιμές των τυπικών παραμέτρων a και b (οι οποίες παίζουν και τον ρόλο τοπικών μεταβλητών), αλλά όχι τις τιμές των ορισμάτων m και n με τα οποία κλήθηκε η συνάρτηση.

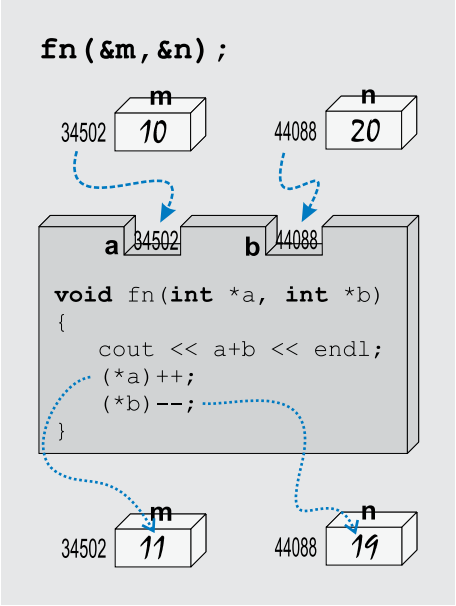


Κατά την κλήση μιας συνάρτησης κατ' αξία (*call by value*), οι τιμές των ορισμάτων αντιγράφονται στις αντίστοιχες τυπικές παραμέτρους της συνάρτησης. Η συνάρτηση δεν μπορεί να τροποποιήσει τις τιμές των μεταβλητών που χρησιμοποιήθηκαν ως ορίσματα κατά την κλήση της.

Κλήση συνάρτησης με δείκτη

Η χρήση των δεικτών είναι ένας έμμεσος τρόπος για να δώσουμε σε μια συνάρτηση τη δυνατότητα να προσπελάζει τις μεταβλητές των ορισμάτων με τα οποία κλήθηκε.

Στο διπλανό παράδειγμα, στη συνάρτηση fn() μεταβιβάζονται οι διευθύνσεις των μεταβλητών m και n στις αντίστοιχες μεταβλητές δείκτη a και b, που αποτελούν τις τυπικές παραμέτρους της συνάρτησης. Η συνάρτηση fn() τώρα γνωρίζει τις διευθύνσεις των m και n και μέσω του τελεστή αναφοράς * έχει πρόσβαση στις μεταβλητές m και n.




```
#include <iostream>
using namespace std;
void fn(int *a, int *b)
{
   cout << a+b << endl;
   (*a)++;
   (*b)--;
}
int main()
{
   int m=10,n=20;
   fn(m,n);
   cout << "m=" << m << ", n=" << n << endl;
   return 0;
}
```

Μέσω των δεικτών **a** και **b**, η συνάρτηση **fn()** έχει πρόσβαση στις μεταβλητές **m** και **n**, δηλαδή στα ορίσματα με τα οποία κλήθηκε.

```
30
m=11 n=19
```

Το παρακάτω πρόγραμμα χρησιμοποιεί τη συνάρτηση swap() για να αντιμεταθέσει τα περιεχόμενα δύο ακέραιων μεταβλητών:

10_swap.cpp

```
#include <iostream>
using namespace std;
void swap(int *a, int *b);
int main()
{
   int x,y;
   cout << "Δώσε δύο αριθμούς :";
   cin >> x >> y;
   cout << "x=" << x << " y=" << y <<endl;
   swap(&x,&y);
   cout << "x=" << x << " y=" << y <<endl;
   return 0;
}
```

Η συνάρτηση **swap()** καλείται με ορίσματα τις διευθύνσεις των μεταβλητών των οποίων τα περιεχόμενα θέλουμε να αντιμεταθέσουμε.

ΠΡΟΣΟΧΗ. Μια αναφορική συνάρτηση πρέπει να επιστρέφει μια αναφορά σε μια θέση μνήμης η οποία να έχει εμβέλεια (να ισχύει) στον χώρο από τον οποίο καλείται η συνάρτηση.

Παραστάσεις αριστερής τιμής (lvalue) και δεξιάς τιμής (rvalue)

Οι παραστάσεις μπορούν να χωριστούν σε δύο κατηγορίες: σε αυτές που μπορούν να τοποθετηθούν αριστερά από τον τελεστή ανάθεσης = και σε αυτές που μπορούν να τοποθετηθούν δεξιά του.

Οι παραστάσεις που μπορούν να βρίσκονται αριστερά του τελεστή = καλούνται *παραστάσεις αριστερής τιμής* (lvalue) και είναι όλες οι παραστάσεις που αποδίδουν αναφορά σε μια θέση μνήμης.

Οι παραστάσεις που δεν είναι αριστερής τιμής (lvalues) καλούνται *παραστάσεις δεξιάς τιμής* (rvalue) και μπορούν να τοποθετηθούν μόνο δεξιά από τον τελεστή =.

Ακολουθούν ορισμένα παραδείγματα παραστάσεων αριστερής τιμής:

```
k = 100;        ◁ όνομα μεταβλητής
*ptr = 100;     ◁ αναφορά με τη χρήση μεταβλητής δείκτη
max() = 100;    ◁ συνάρτηση που επιστρέφει αναφορά
```

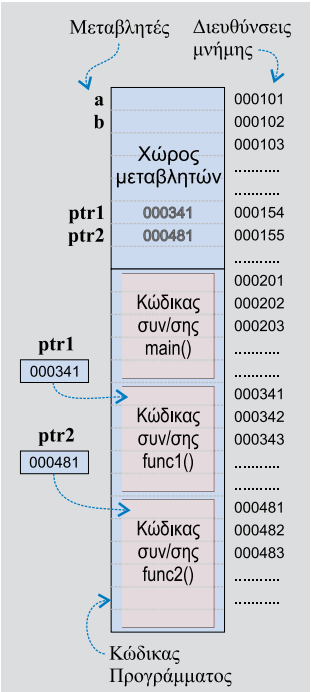
Δείκτες σε συναρτήσεις

Ένα πρόγραμμα καταλαμβάνει συγκεκριμένο χώρο στην κεντρική μνήμη του Η/Υ. Ο χώρος αυτός χρησιμοποιείται τόσο για τις μεταβλητές όσο και για τον μεταγλωττισμένο κώδικα του προγράμματος.

Επομένως, ο κώδικας κάθε συνάρτησης που χρησιμοποιεί το πρόγραμμά μας καταλαμβάνει έναν συγκεκριμένο χώρο στη μνήμη και ξεκινάει από μια συγκεκριμένη διεύθυνση μνήμης. Η διεύθυνση αυτή αποτελεί τη *διεύθυνση της συνάρτησης*.

Στη C++, ένας δείκτης *μπορεί* να δείχνει σε μια συνάρτηση! Ένας δείκτης σε συνάρτηση δεν είναι τίποτα άλλο παρά μια μεταβλητή δείκτη η οποία περιέχει τη διεύθυνση μιας συνάρτησης.

Στο διπλανό σχήμα, οι συναρτήσεις func1() και func2() έχουν διευθύνσεις 000341 και 000481 αντίστοιχα.



Στην περίπτωση που οι διευθύνσεις των συναρτήσεων καταχωριστούν στις μεταβλητές δείκτη ptr1 και ptr2, θα μπορούσε να γίνεται κλήση των συναρτήσεων μέσω των δεικτών αυτών χωρίς τη χρήση του ονόματός τους.

> Αν αποθηκεύσουμε τη διεύθυνση μιας συνάρτησης σε μια μεταβλητή δείκτη, μπορούμε να καλούμε τη συνάρτηση, όχι πια με το όνομά της, αλλά χρησιμοποιώντας τον συγκεκριμένο δείκτη (ο οποίος δείχνει στον κώδικα της συνάρτησης στη μνήμη του Η/Υ).

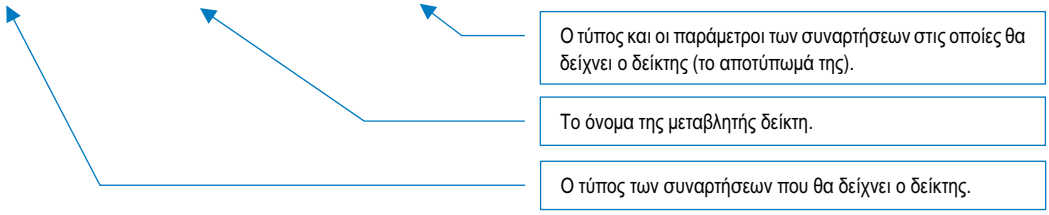
Δήλωση δείκτη σε συνάρτηση

Το παρακάτω παράδειγμα δείχνει τον τρόπο με τον οποίο δηλώνεται μια μεταβλητή δείκτη σε συνάρτηση:

```
int (*ptr) (int x, int y);
```

Η πρόταση αυτή δηλώνει μια μεταβλητή δείκτη με όνομα ptr, ο οποίος θα δείχνει σε συναρτήσεις τύπου int με δύο παραμέτρους επίσης τύπου int. Η γενική σύνταξη της δήλωσης ενός δείκτη σε συνάρτηση είναι το ακόλουθο:

int (όνομα_δείκτη*) (τύπος *παράμετρος1*, τύπος *παράμετρος2*, ...);**



Ένας δείκτης σε συνάρτηση πρέπει να δείχνει σε συναρτήσεις με συγκεκριμένο «αποτύπωμα». Το αποτύπωμα μιας συνάρτησης αποτελείται από τον τύπο της, καθώς και από το πλήθος και τον τύπο των παραμέτρων της.

Ανάθεση τιμής σε μεταβλητή δείκτη σε συνάρτηση

Η ανάθεση τιμής σε δείκτη σε συνάρτηση γίνεται απλά με την ανάθεση της διεύθυνσης της συνάρτησης στη μεταβλητή δείκτη, ή με την ανάθεση του ονόματος της συνάρτησης. Η διεύθυνση μιας συνάρτησης αποδίδεται με τη χρήση του τελεστή & πριν από το όνομά της.

```
ptr = &func1;    // Τρόπος περισσότερο συμβατός με κάθε μεταγλωττιστή της C++
ptr = func1;     // Σύντομος τρόπος, υποστηρίζεται από αρκετούς μεταγλωττιστές
```

***όνομα_δείκτη* = &*όνομα_συνάρτησης*;** ή,
***όνομα_δείκτη* = *όνομα_συνάρτησης*;**

Προσθήκες προτύπου C++11

Η λέξη-κλειδί nullptr

Στο κεφάλαιο των δεικτών μιλήσαμε για τη σταθερά NULL. Η σταθερά NULL ανατίθεται ως τιμή σε έναν δείκτη όταν θέλουμε να δείξουμε ότι αυτός ο δείκτης δεν δείχνει πουθενά. Η αρχικοποίηση ενός δείκτη με τη τιμή NULL μάς προστατεύει από λανθασμένη χρήση του και καταστροφικά αποτελέσματα. Όμως η σταθερά NULL δεν ανήκει σε κάποιο ξεχωριστό τύπο δεδομένων και στην πραγματικότητα δεν είναι τίποτε περισσότερο από τον ακέραιο αριθμό 0.Αυτό σε ορισμένες περιπτώσεις δημιουργεί προβλήματα. Για παράδειγμα, ας δούμε τα πρότυπα των παρακάτω υπερφορτωμένων συναρτήσεων:

```
void test(int a);
void test(char *p);
```

Ποια από τις δύο εκδόσεις θα κληθεί από τη πρόταση test(NULL);

Προφανώς θα θέλαμε να κληθεί η δεύτερη διότι ως όρισμα χρησιμοποιήσαμε έναν δείκτη (έτσι νομίζουμε τουλάχιστον). Στην πραγματικότητα όμως θα κληθεί η πρώτη έκδοση διότι η τιμή NULL είναι όπως είπαμε ένας ακέραιος αριθμός.

Το πρότυπο C++11 εισήγαγε τη λέξη-κλειδί nullptr, η οποία αναφέρεται (όπως και η σταθερά NULL) σε έναν δείκτη που δεν δείχνει πουθενά. Η διαφορά είναι ότι ένας δείκτης nullptr έχει τύπο: οποιονδήποτε τύπο δείκτη!

Στο πρόγραμμα που ακολουθεί γίνεται χρήση της nullptr:

10_nullptr.cpp

```
#include <iostream>
using namespace std;
void test(int a);
void test(char *p);
int main()
{
   int b=20;
   char ch='*';
   test(nullptr);
   return 0;
}
void test(int a)
{
   cout<<a<<endl;
}
void test(char *p)
{
   if (p!=NULL) cout<<*p<<endl;
}
```

Καλείται η δεύτερη έκδοση της υπερφορτωμένης συνάρτησης **test()** δεδομένου ότι το όρισμα **nullptr** θεωρείται δείκτης. Στην περίπτωση που η συνάρτηση καλείτο με όρισμα τη σταθερά NULL, θα καλείτο η πρώτη έκδοσή της διότι η τιμή NULL είναι ένας ακέραιος αριθμός.

Οι δύο υπερφορτωμένες εκδόσεις της συνάρτησης **test()**.

Παραδείγματα

Π10.1 Η παρακάτω συνάρτηση count() μετράει τους χαρακτήρες που βρίσκονται μέσα σε μια συμβολοσειρά της οποίας την αρχική διεύθυνση μεταβιβάζουμε ως παράμετρο στη συνάρτηση.

10_p10_1.cpp

```
#include <iostream>
using namespace std;
int count(char *ptr);
int main()
{
   char *p;
   p="ΑΝΑΝΑΣ";
   cout << "Η λέξη " << p
        << " έχει "
        << count(p)
        << " γράμματα\n";
   return 0;
}
int count(char *ptr)
{
   int c=0;
   while (*ptr != '\0')
   {
      ptr++;
      c++;
   }
   return c;
}
```

```
Η λέξη ΑΝΑΝΑΣ έχει 6 γράμματα
```

Στην **count()** μεταβιβάζεται η διεύθυνση του πρώτου χαρακτήρα της συμβολοσειράς.

Ελέγχει έναν-έναν τους χαρακτήρες της συμβολοσειράς, μέχρι να εντοπίσει τον χαρακτήρα τερματισμού '\0'.

Κάθε φορά αυξάνει τον δείκτη **ptr** ώστε να δείχνει στον επόμενο χαρακτήρα, καθώς και τη μεταβλητή **c** η οποία μετράει το πλήθος των χαρακτήρων.

Επιστρέφει ως τιμή το πλήθος των χαρακτήρων.

Π10.2 Η παρακάτω συνάρτηση to_order() καλείται με αναφορά, και αντιμεταθέτει τα περιεχόμενα των μεταβλητών των ορισμάτων της ώστε να τα τυπώνει με αύξουσα σειρά.

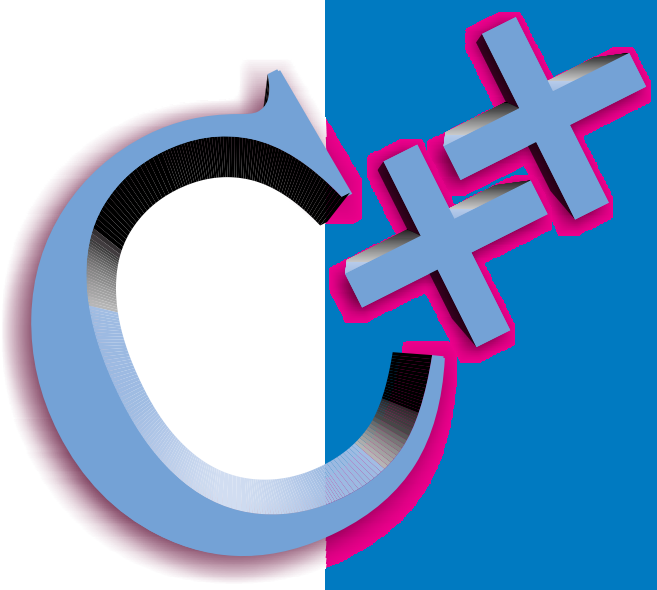
10_p10_2.cpp

```
#include <iostream>
using namespace std;
void to_order(int &x, int &y)
{
   int temp;
   if (x>y)
   {
      temp=x;
      x=y;
      y=temp;
   }
}
```

Η αντιμετάθεση των περιεχομένων των **x** και **y** επηρεάζει τις αντίστοιχες μεταβλητές των ορισμάτων με τα οποία κλήθηκε η συνάρτηση.

Κεφάλαιο 11

Πίνακες



Πίνακες

Ας υποθέσουμε ότι θέλουμε να αποθηκεύσουμε 10 αριθμούς σε ισάριθμες θέσεις μνήμης και κατόπιν να υπολογίσουμε και να εμφανίσουμε το άθροισμά τους. Με όσα έχουμε μάθει μέχρι τώρα, θα έπρεπε να γράψουμε το ακόλουθο πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   int a,b,c,d,e,f,k,l,m,n,sum;
   cout << "Δώσε 10 αριθμούς" << endl;
   cin >>a>>b>>c>>d>>e>>f>>k>>l>>m>>n;
   sum=a+b+c+d+e+f+k+l+m+n;
   cout << "Αθροισμα:" << sum << endl;
   return 0;
}
```

Στις 10 θέσεις μνήμης καταχωρίζονται οι αριθμοί που πληκτρολογεί ο χρήστης.

Υπολογίζεται και εμφανίζεται το άθροισμα των 10 θέσεων μνήμης.

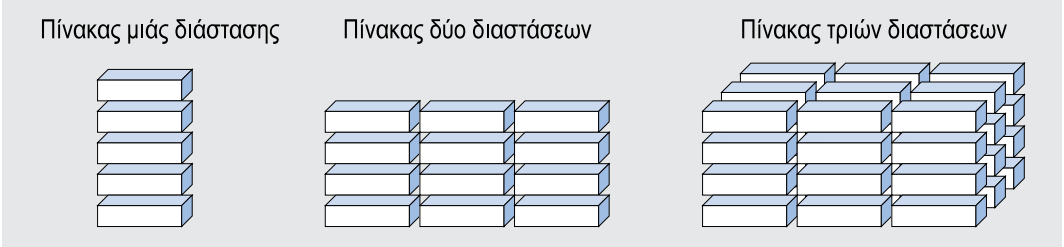
Σκεφτείτε τι θα έπρεπε να κάναμε αν θέλαμε να κάνουμε το ίδιο όχι για 10, αλλά για 100 αριθμούς! Το πρόγραμμα θα γινόταν ιδιαίτερα πολύπλοκο με 100 διαφορετικές μεταβλητές, ενώ για πολύ περισσότερους αριθμούς τα πράγματα θα δυσκόλευαν απελπιστικά. Μη ξεχνάμε ότι μία από τις βασικές δυνατότητες ενός υπολογιστικού συστήματος είναι η επεξεργασία μεγάλου όγκου δεδομένων.

Θα πρέπει λοιπόν να υπάρχει κάποιος καλύτερος και πιο κομψός τρόπος!

Ο τρόπος για να αποθηκεύουμε μεγάλα πλήθη δεδομένων, σε οποιαδήποτε γλώσσα προγραμματισμού, είναι η χρήση **πινάκων** (arrays).

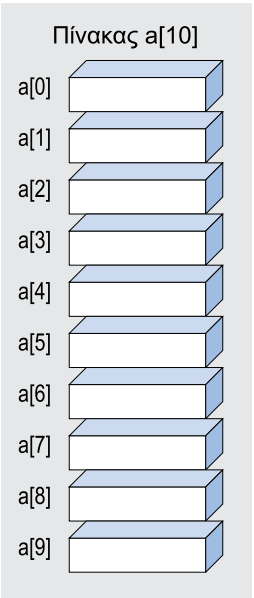
> Οι πίνακες αποτελούν έναν διαφορετικό τρόπο διαχείρισης της μνήμης. Ένας πίνακας είναι μια ομάδα θέσεων μνήμης στις οποίες γίνεται αναφορά με ένα κοινό όνομα. Μια συγκεκριμένη θέση μνήμης αυτής της ομάδας καθορίζεται από έναν ή περισσότερους αριθμοδείκτες.

Οι πίνακες αποτελούνται από θέσεις του ίδιου τύπου και, ανάλογα με τη διάταξή τους, μπορεί να έχουν μία ή περισσότερες διαστάσεις, όπως στο παρακάτω σχήμα:



Πίνακες μίας διάστασης

Έναν πίνακα μιας διάστασης τον φανταζόμαστε σαν μια ομάδα θέσεων μνήμης, τη μια πάνω από την άλλη. Κάθε θέση μνήμης έχει έναν αριθμοδείκτη, η αρίθμηση του οποίου ξεκινάει από το 0 (η πρώτη θέση είναι η θέση 0, η δεύτερη η 1 κ.ο.κ).



Στη C++ η πρόταση δήλωσης ενός πίνακα είναι παρόμοια με τις προτάσεις δήλωσης οποιασδήποτε άλλης μεταβλητής:

τύπος_δεδομένων *όνομα_πίνακα*[*μέγεθος*];

Για παράδειγμα, η πρόταση:

```
int a[10];
```

δηλώνει έναν πίνακα με όνομα a και 10 θέσεις μνήμης.

Στη C++ η πρώτη θέση μνήμης ενός πίνακα έχει αριθμοδείκτη 0. Επομένως, ένας πίνακας int a[10] θα έχει 10 θέσεις μνήμης από την a[0] μέχρι την a[9].

Όλες οι θέσεις μνήμης ενός πίνακα έχουν τον ίδιο τύπο με αυτόν που χρησιμοποιήθηκε στην πρόταση δήλωσης του ίδιου του πίνακα.

Οι θέσεις μνήμης ενός πίνακα χρησιμοποιούνται όπως οι υπόλοιπες μεταβλητές. Για παράδειγμα, οι επόμενες προτάσεις:

```
a[5]=45;
a[8]=67;
```

θα καταχωρίσουν το 45 στην έκτη θέση μνήμης του πίνακα a και το 67 στην ένατη (a[0] είναι η πρώτη θέση μνήμης, a[5] είναι η έκτη, και a[8] η ένατη!). Το μέγεθος ενός πίνακα μπορεί να προσδιορίζεται και από μια μεταβλητή[1]. Επομένως, ο παρακάτω κώδικας

```
int m=20;
int a[m];
cin >> m;
int b[m];
```

Περιμένει να πληκτρολογηθεί ένας αριθμός και τον καταχωρίζει στη μεταβλητή **m**.

θα έχει ως αποτέλεσμα τη δημιουργία ενός πίνακα a 20 θέσεων μνήμης και ενός πίνακα b τόσων θέσεων μνήμης όσες ο αριθμός που πληκτρολογήθηκε.

[1] Οι πίνακες αυτοί λέγονται *πίνακες μεταβλητού μήκους* (variable length arrays – VLAs) και υποστηρίζονται από τη C αλλά όχι από τη C++. Όμως αρκετοί μεταγλωττιστές της C++, όπως ο GCC, διαθέτουν ως επέκταση αυτή τη δυνατότητα.

Συνήθως, για να προσδιορίσουμε μια θέση μνήμης ενός πίνακα μίας διάστασης χρησιμοποιούμε μια μεταβλητή. Αν αλλάξουμε την τιμή της μεταβλητής, μπορούμε να έχουμε πρόσβαση σε μια διαφορετική θέση μνήμης του πίνακα.

Για παράδειγμα, το επόμενο τμήμα προγράμματος

```
i=5;
a[i]=45;
i=8;
a[i]=64;
```

θα καταχωρίσει στη θέση a[5] του πίνακα a τον αριθμό 45 και στη θέση a[8] τον αριθμό 64, ενώ ο ακόλουθος κώδικας

```
for (i=0;i<=9;i++) a[i]=45;
```

θα γεμίσει όλες τις θέσεις μνήμης του πίνακα a με τον αριθμό 45.

Το παρακάτω πρόγραμμα γεμίζει τον πίνακα **a** με 10 τυχαίους αριθμούς και μετά εμφανίζει στην οθόνη μόνον αυτούς που είναι μεγαλύτεροι από 1000:

11_random1.cpp

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
   int i,a[10];
   for (i=0;i<10;i++)
      a[i]=rand();
   for (i=0;i<10;i++)
      if (a[i]>1000) cout << a[i] << endl;
   return 0;
}
```

Στις 10 θέσεις του πίνακα καταχωρίζονται τυχαίοι αριθμοί.

Στην οθόνη εμφανίζονται οι αριθμοί του πίνακα που είναι μεγαλύτεροι από το 1000.

Η διάσχιση ενός πίνακα μίας διάστασης γίνεται συνήθως με μια επαναληπτική δομή for. Η μεταβλητή που χρησιμοποιείται για τον έλεγχο της for χρησιμοποιείται επίσης για να προσδιορίσει τη θέση μνήμης του πίνακα, π.χ. a[i]!

Το επόμενο πρόγραμμα γεμίζει τον πίνακα a με 10 αριθμούς που δίνει ο χρήστης από το πληκτρολόγιο. Στη συνέχεια υπολογίζει και εμφανίζει το άθροισμα των αριθμών.

11_arithmoi1.cpp

```
#include <iostream>
using namespace std;
int main()
{
   int i;
   float a[10],sum;
   for (i=0;i<10;i++)
```

Στις 10 θέσεις του πίνακα καταχωρίζονται οι αριθμοί που πληκτρολογεί ο χρήστης.

Πίνακες μίας διάστασης και δείκτες

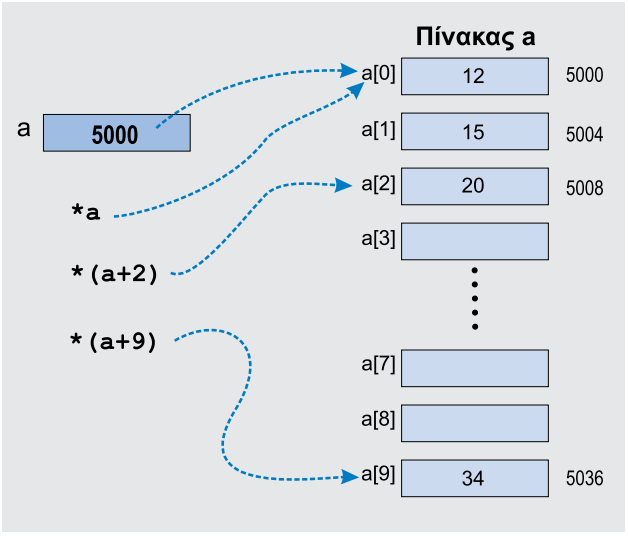
Υπάρχει μια πολύ στενή σχέση μεταξύ πινάκων και δεικτών.

> Με απλά λόγια, το όνομα ενός πίνακα είναι ένας δείκτης ο οποίος περιέχει τη διεύθυνση της πρώτης από τις θέσεις μνήμης του.

Ο δείκτης αυτός δείχνει σε θέσεις μνήμης του τύπου με τον οποίο δηλώθηκε ο πίνακας. Για παράδειγμα, με την πρόταση:

```
int a[10];
```

δημιουργείται ένας πίνακας a 10 θέσεων τύπου int. Αυτόματα δημιουργείται και μια μεταβλητή *δείκτη* τύπου int με όνομα a (βλέπε διπλανό σχήμα), με τιμή τη διεύθυνση της πρώτης από τις 10 θέσεις του πίνακα.



Τις θέσεις μνήμης ενός πίνακα μπορούμε να τις προσπελάζουμε με δύο διαφορετικούς τρόπους: είτε χρησιμοποιώντας τον αριθμοδείκτη τους μέσα στον πίνακα (a[5]), είτε χρησιμοποιώντας τη λογική των δεικτών. Οι ακόλουθες παραστάσεις στα αριστερά είναι απολύτως ισοδύναμες με τις αντίστοιχές τους στα δεξιά:

```
a[0]=12 ⇔ *a=12
a[1]=15 ⇔ *(a+1)=15
a[2]=15 ⇔ *(a+2)=20
a[9]=34 ⇔ *(a+9)=34
```

Για παράδειγμα,, η παράσταση *(a+9) δείχνει στη θέση μνήμης με διεύθυνση 5036 αφού, σύμφωνα με την αριθμητική δεικτών, αν προστεθεί στο a (που έχει τιμή 5000) το 9, το αποτέλεσμα θα είναι 5036! Μη ξεχνάμε ότι η μεταβλητή a είναι ένας δείκτης σε δεδομένα τύπου int, οπότε η πρόσθεση κάθε μονάδας στον δείκτη a έχει ως αποτέλεσμα την αύξηση του κατά 4 (a+9 ⇨ 5000 + 4*9 ⇨ 5036).

Ποια είναι επομένως η διαφορά μεταξύ ενός πίνακα και ενός δείκτη;

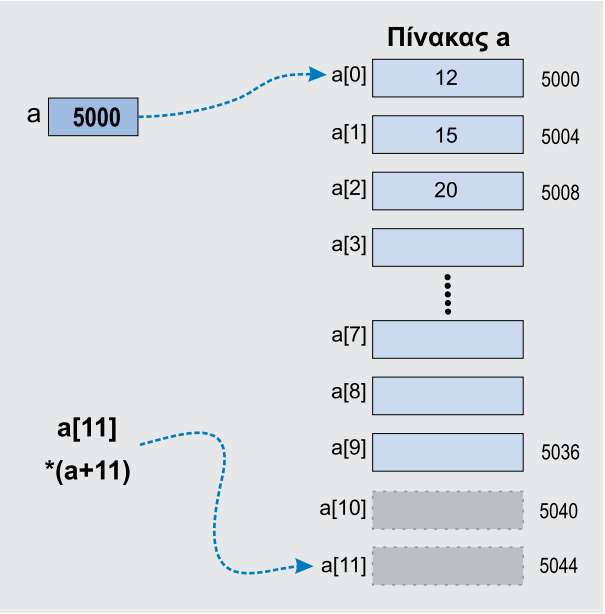
Κάθε μία από τις παρακάτω δύο προτάσεις δημιουργούν έναν δείκτη ptr:

```
int ptr[100];
int *ptr;
```

Είναι όμως ισοδύναμες; Η απάντηση είναι αρνητική, επειδή η πρώτη πρόταση, εκτός του ότι δημιουργεί έναν δείκτη ptr, δεσμεύει επίσης 100 θέσεις μνήμης τύπου int και δίνει στον δείκτη αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης. Η δεύτερη πρόταση απλώς δημιουργεί έναν δείκτη χωρίς να του δίνει αρχική τιμή και χωρίς να δεσμεύει καμία άλλη θέση μνήμης.

Προσοχή:

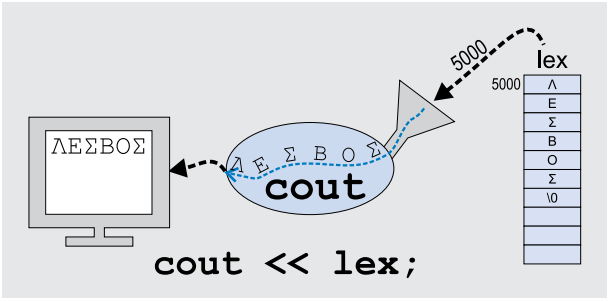
Η C++ *δεν* ελέγχει αν έχει ξεπεραστεί το όριο ενός πίνακα, κάτι που πολλές φορές (σχεδόν πάντα) έχει καταστροφικά αποτελέσματα. Έτσι, στο προηγούμενο παράδειγμα ο πίνακας a έχει δέκα θέσεις, από την a[0] μέχρι την a[9]. Παρόλα αυτά, μπορούμε κάλλιστα να προσπελάσουμε τη θέση a[11] (βλέπε διπλανό σχήμα) χωρίς η C++ να δώσει κανένα μήνυμα λάθους. Το τι θα επακολουθήσει όμως εξαρτάται μόνο από την τύχη μας.



> Στη C++ ο έλεγχος των ορίων ενός πίνακα είναι ευθύνη του προγραμματιστή.

Τα αντικείμενα cout και cin σε περισσότερο ... βάθος

Όταν θέλουμε να εμφανίσουμε μια συμβολοσειρά στην οθόνη, φανταζόμαστε με εποπτικό τρόπο ότι τη «ρίχνουμε» μέσα στο αντικείμενο cout! Στην πραγματικότητα όμως αυτό που «ρίχνουμε» μέσα στο αντικείμενο cout δεν είναι οι χαρακτήρες της συμβολοσειράς αλλά η διεύθυνσή της. Δηλαδή, η διεύθυνση της θέσης μνήμης του πρώτου από τους χαρακτήρες της. Το αντικείμενο cout αναλαμβάνει την εμφάνιση των χαρακτήρων, ξεκινώντας από τη διεύθυνση που του δώσαμε, μέχρι να εντοπίσει τον χαρακτήρα τερματισμού '\0'. Στο παράδειγμα του σχήματος, με τη πρόταση cout << lex; στο αντικείμενο cout «ρίχνουμε» τη διεύθυνση του πίνακα



Αντικείμενα string και πίνακες

Ο προτεινόμενος τρόπος για την διαχείριση συμβολοσειρών στη C++ είναι με αντικείμενα της κλάσης string και όχι με πίνακες χαρακτήρων. Ήδη σε προηγούμενα κεφάλαια έχουμε χρησιμοποιήσει αντικείμενα της κλάσης string για την καταχώριση συμβολοσειρών. Για παράδειγμα, με τις ακόλουθες προτάσεις δημιουργείται ένα αντικείμενο string στο οποίο ακολούθως καταχωρίζεται η φράση "Η γλώσσα C++ σε βάθος".

```
string lex;
lex="Η γλώσσα C++ σε βάθος";
```

Αυτό που δεν γνωρίζαμε μέχρι τώρα είναι ότι μπορούμε να προσπελάσουμε τους χαρακτήρες ενός αντικειμένου string σαν να ήταν ένας πίνακας χαρακτήρων. Έτσι, οι παρακάτω προτάσεις:

```
cout << lex[0];
lex[9]="D";
```

θα έχουν ως αποτέλεσμα α) την εμφάνιση του πρώτου χαρακτήρα της φράσης που είναι καταχωρισμένη στο αντικείμενο lex, και β) την αντικατάσταση του 10ου χαρακτήρα του lex με το γράμμα 'D'. Στα αντικείμενα string, όπως και στους πίνακες χαρακτήρων, η αρίθμηση ξεκινάει από το 0. Μήπως λοιπόν πίσω από ένα αντικείμενο string κρύβεται ένας πίνακας χαρακτήρων; Όχι ακριβώς! Η κλάση string χρησιμοποιεί έναν διαφορετικό μηχανισμό δυναμικής κατανομής μνήμης για την καταχώριση των χαρακτήρων.

> Μπορούμε να φανταζόμαστε ένα αντικείμενο string ως έναν αποθηκευτικό χώρο ο οποίος μπορεί να μεγαλώνει και να μικραίνει ώστε να προσαρμόζεται ακριβώς στο μέγεθος της ακολουθίας χαρακτήρων που περιέχει. Όπως και στις συμβολοσειρές της C, ο τελευταίος χαρακτήρας ενός αντικειμένου string είναι ο χαρακτήρας τερματισμού '\0'.

Ο τρόπος με τον οποίο ένα αντικείμενο string διαχειρίζεται τον αποθηκευτικό χώρο και οι λεπτομέρειες της υλοποίησής του απλώς δεν μας ενδιαφέρουν! Στον αντικειμενοστρεφή προγραμματισμό μας ενδιαφέρει το τι κάνει ένα αντικείμενο και όχι πώς το κάνει. Μας ενδιαφέρει, για παράδειγμα, ότι ένα αυτοκίνητο ξεκινάει, σταματάει, στρίβει και κορνάρει, αλλά δεν μας ενδιαφέρουν οι τεχνικές λεπτομέρειες με τις οποίες υλοποιούνται οι λειτουργίες αυτές.

> Η δυνατότητα των αντικειμενοστρεφών γλωσσών προγραμματισμού να δημιουργούν αντικείμενα τα οποία μπορούμε να χρησιμοποιούμε, χωρίς όμως να χρειάζεται να ασχολούμαστε με όλα τα τεχνικά χαρακτηριστικά και τις λεπτομέρειες υλοποίησής τους, ονομάζεται *αφαιρετικότητα*.

Η συνάρτηση getline()

Έχουμε ήδη χρησιμοποιήσει το αντικείμενο cin για την ανάγνωση συμβολοσειρών από το πληκτρολόγιο. Οι χαρακτήρες που πληκτρολογεί ο χρήστης καταχωρίζονται είτε σε έναν πίνακα χαρακτήρων είτε σε ένα αντικείμενο της κλάσης string. Όμως, όταν το αντικείμενο cin συναντήσει οποιονδήποτε χαρακτήρα λευκού διαστήματος θεωρεί ότι η συμβολοσειρά τελείωσε και την καταχωρίζει μέχρι αυτό το σημείο. Στο ακόλουθο τμήμα κώδικα,

```
string frasi;
cout << "Δώσε μια φράση:";
cin >> frasi;
cout << "Έδωσες:" << frasi;
```

```
Δώσε μια φράση:Άντε γειά
Έδωσες:Άντε
```

υποθέτουμε ότι ο χρήστης πληκτρολογεί μια φράση δύο λέξεων με ένα κενό διάστημα μεταξύ τους. Σύμφωνα με τα παραπάνω, στο αντικείμενο frasi θα καταχωριστεί μόνο η πρώτη λέξη. Οι υπόλοιποι χαρακτήρες παραμένουν στον χώρο προσωρινής αποθήκευσης (buffer) εισόδου για επόμενη ανάγνωση!

Η συνάρτηση getline() λύνει αυτό το πρόβλημα και μας δίνει τη δυνατότητα να διαβάζουμε και να αποθηκεύουμε σύνολα χαρακτήρων μέχρι να πληκτρολογηθεί ο χαρακτήρας <Enter>. Η απλή σύνταξη της συνάρτησης είναι η εξής:

getline(cin, string str)

Η πρώτη παράμετρος cin καθορίζει το ρεύμα εισόδου από το οποίο θα διαβαστούν οι χαρακτήρες, στη συγκεκριμένη περίπτωση το πληκτρολόγιο. Η δεύτερη παράμετρος καθορίζει το αντικείμενο string στο οποίο θα αποθηκευτούν οι χαρακτήρες. Το πρωτότυπο της συνάρτησης, καθώς και το πλήρες συντακτικό της, θα αναλυθούν σε επόμενο κεφάλαιο. Έτσι, η πρόταση

```
getline(cin,frasi);
```

θα έχει ως αποτέλεσμα να διαβαστούν από το πληκτρολόγιο χαρακτήρες, μέχρι να πληκτρολογηθεί το <Enter>, και να καταχωριστούν στο αντικείμενο frasi. Στο τέλος των χαρακτήρων προστίθεται αυτόματα ο χαρακτήρας τερματισμού '\0'.

Το παρακάτω τμήμα κώδικα εμφανίζει όλους τους χαρακτήρες του αντικειμένου frasi, τον καθένα σε ξεχωριστή γραμμή, καθώς και το πλήθος τους.

```
i=0;
while (frasi[i]!='\0')
{
   cout << frasi[i] << endl;
   i++;
}
cout << "Πλήθος χαρακτήρων:" << i << endl;
```

Μεταβίβαση πινάκων δύο διαστάσεων σε συναρτήσεις

Σύμφωνα με τα όσα αναφέρθηκαν στη προηγούμενη παράγραφο, για να μπορέσει η συνάρτηση στην οποία μεταβιβάζεται ένας πίνακας δύο διαστάσεων να υπολογίσει τη διεύθυνση μιας θέσης μνήμης του, πρέπει να γνωρίζει, εκτός από τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα, και την τιμή της δεύτερης διάστασής του.

Στο τμήμα κώδικα που ακολουθεί μεταβιβάζουμε έναν πίνακα δύο διαστάσεων (5x10) σε μια συνάρτηση.

```
int main()
{
   int a[5][10];
   .....
   func(a);
   .....
   return 0;
}
void func(int x[][10])
{
   .....
}
```

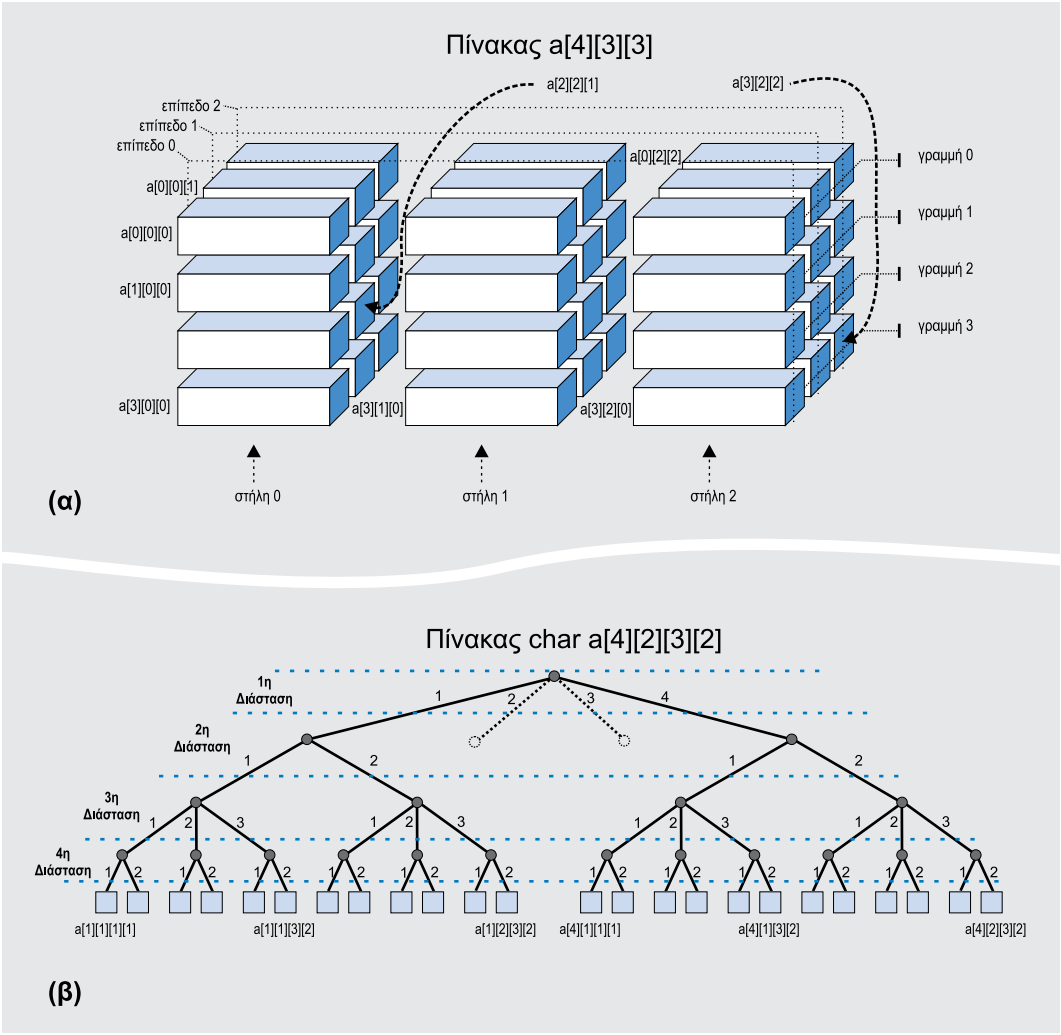
Ο πίνακας a μεταβιβάζεται στην τυπική παράμετρο x της συνάρτησης func(). Στη δήλωση της παραμέτρου x πρέπει οπωσδήποτε να δηλωθεί η τιμή της δεύτερης διάστασης του πίνακα. Στην περίπτωση αυτή, και γενικά στη μεταβίβαση πινάκων δύο διαστάσεων, η αντίστοιχη παράμετρος (η x του παραδείγματος) δεν μπορεί να δηλωθεί απλώς ως δείκτης.

Πίνακες με περισσότερες διαστάσεις

Αν και στην πραγματικότητα οι θέσεις ενός πίνακα πολλών διαστάσεων είναι συνεχόμενες, αρκετές φορές είναι πολύ βολικό να έχουμε στο μυαλό μας μια εποπτική και σίγουρα πιο παραστατική εικόνα ενός πίνακα. Μπορούμε να φανταστούμε έναν πίνακα τριών διαστάσεων σαν θέσεις μνήμης διατεταγμένες σε γραμμές, στήλες, αλλά και επίπεδα. Στο επόμενο παράδειγμα (Σχήμα 11.3α), ο πίνακας a τριών διαστάσεων (a[4][3][3]) απεικονίζεται με τις θέσεις μνήμης του σε διάταξη τεσσάρων γραμμών, τριών στηλών, και τριών επιπέδων.

Για να αναφερθούμε σε μια θέση μνήμης, πρέπει να καθορίσουμε τη γραμμή, τη στήλη, και το επίπεδο στο οποίο βρίσκεται. Με τις τρεις διαστάσεις του χώρου, μπορούμε να απεικονίσουμε πίνακες μέχρι τρεις διαστάσεις. Πώς μπορούμε όμως να φανταστούμε εποπτικά έναν πίνακα περισσότερων διαστάσεων; Είναι δυνατόν να απεικονίσουμε στο χαρτί έναν πίνακα τεσσάρων, πέντε ή και περισσότερων διαστάσεων; Αν και είναι πράγματι δυνατό, για να το κάνουμε αυτό θα πρέπει να ξεχάσουμε τον τριδιάστατο χώρο που χρησιμοποιούσαμε μέχρι τώρα και να σκεφτούμε τις θέσεις μνήμης σαν χωριά τα οποία

μπορούμε να προσεγγίζουμε μόνο αν επιλέγουμε τον σωστό δρόμο σε κάθε σταυροδρόμι. Έτσι φανταζόμαστε έναν πίνακα τεσσάρων διαστάσεων σαν μια συστοιχία δρόμων οι οποίοι ξεκινούν από μια πλατεία. Κάθε δρόμος καταλήγει σε μια άλλη πλατεία κ.ο.κ. Τελικά, για να φτάσουμε σε ένα χωριό (θέση μνήμης), θα πρέπει να επιλέξουμε τον δρόμο που θα πάρουμε σε κάθε μία από τις τέσσερις πλατείες τις οποίες περνάμε με τη σειρά. Έτσι, ο πίνακας char a[4][2][3][2] του Σχήματος 11.3β απεικονίζεται με τέσσερα επίπεδα, σε κάθε ένα από τα οποία (διάσταση) θα πρέπει να επιλέξουμε ποιο δρόμο θα πάρουμε σε κάθε πλατεία. Ο δρόμος που επιλέγουμε σε κάθε πλατεία είναι η τιμή της συγκεκριμένης διάστασης του πίνακα.



Σχήμα 11.3 *Πίνακες πολλών διαστάσεων - Δενδροειδής απεικόνιση πίνακα*

Συχνά λάθη

- Θεωρούμε λογικό ότι σε έναν πίνακα 10 θέσεων, π.χ. a[10], η πρώτη θέση είναι η a[1] και η τελευταία η a[10]. Στη C++ η πρώτη θέση του πίνακα είναι η θέση a[0], και η τελευταία η a[9].
- Δεν έχουμε κατανοήσει πλήρως ότι το όνομα ενός πίνακα είναι ταυτόχρονα ένας *δείκτης* ο οποίος απλώς περιέχει μια διεύθυνση.
- Νομίζουμε ότι όταν μεταβιβάζουμε έναν πίνακα σε μια συνάρτηση, η συνάρτηση αποκτά ένα αντίγραφο του πίνακα. Αυτό δεν είναι αληθές! Η συνάρτηση έχει πρόσβαση στον *ίδιο* πίνακα ο οποίος της μεταβιβάστηκε, αλλά μέσω του ονόματος της παραμέτρου της.
- Προσπαθούμε να καταχωρίσουμε μια συμβολοσειρά σε έναν πίνακα χαρακτήρων χρησιμοποιώντας τον τελεστή ανάθεσης =. Αυτό γίνεται μόνο κατά την πρόταση δήλωσης του πίνακα, όταν του δίνουμε ταυτόχρονα και αρχικές τιμές. Αν αργότερα θέλουμε να καταχωρίσουμε μια συμβολοσειρά στον πίνακα, θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση βιβλιοθήκης strcpy().
- Προσπαθούμε να συγκρίνουμε συμβολοσειρές με τους γνωστούς τελεστές σύγκρισης. Αυτό δεν είναι δυνατόν! Για τη σύγκριση συμβολοσειρών πρέπει να χρησιμοποιούμε τη συνάρτηση βιβλιοθήκης strcmp().
- Χρησιμοποιούμε πίνακες χαρακτήρων αντί για αντικείμενα της κλάσης string. Αυτό είναι μια κακή πρακτική! Από τη στιγμή που μάθουμε να χειριζόμαστε τη κλάση string πρέπει να τη χρησιμοποιούμε *αποκλειστικά* για την αποθήκευση και τον χειρισμό συμβολοσειρών.

Ανασκόπηση Κεφαλαίου 11

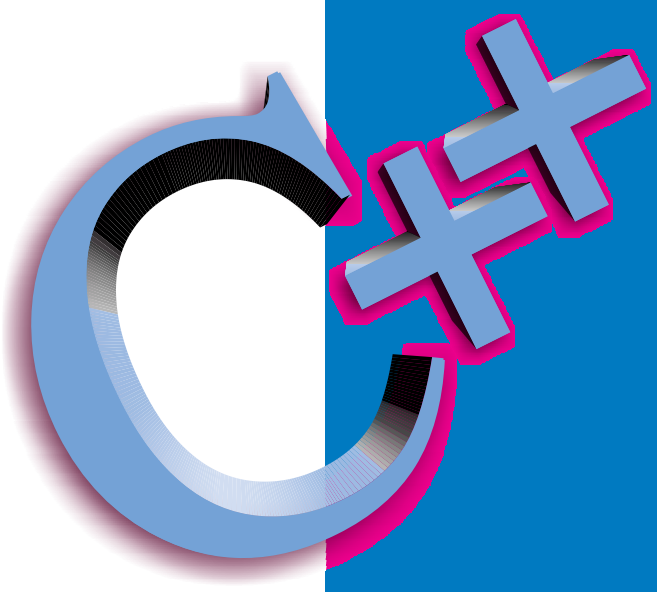
- Ένας πίνακας είναι μια διάταξη θέσεων μνήμης με ένα κοινό όνομα.
- Οι πίνακες που χρησιμοποιούνται πιο συχνά είναι οι πίνακες μίας διάστασης και δύο διαστάσεων. Μπορούμε όμως να ορίσουμε πίνακες περισσότερων διαστάσεων ανάλογα με τις ανάγκες μας.
- Ένας πίνακας ορίζει έμμεσα έναν δείκτη με αρχική τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα.
- Η δήλωση ενός πίνακα γίνεται όπως και για οποιαδήποτε άλλη μεταβλητή και ακολουθείται από το πλήθος και την τιμή των διαστάσεών του. Για παράδειγμα, η int a[4][5] δηλώνει έναν πίνακα a δύο διαστάσεων με είκοσι θέσεις (4x5), με την πρώτη διάσταση από το 0 μέχρι το 3 και τη δεύτερη από το 0 μέχρι το 4.

- Η αναφορά σε μια θέση ενός πίνακα γίνεται με το όνομά του και τους αριθμοδείκτες για την κάθε διάσταση, ώστε να προσδιορίζεται η θέση μνήμης μέσα στον πίνακα. Για παράδειγμα, η a[2][3] αναφέρεται στη θέση μνήμης με αριθμοδείκτες 2 και 3 για την πρώτη και τη δεύτερη διάσταση αντίστοιχα.
- Το όνομα ενός πίνακα αποτελεί ταυτόχρονα και δείκτη με τιμή τη διεύθυνση της πρώτης θέσης μνήμης του πίνακα. Ο χειρισμός ενός πίνακα μπορεί να γίνει και με τη χρήση δεικτών.
- Η C++ επιτρέπει τον χειρισμό συμβολοσειρών, είτε με τον τρόπο που κληρονόμησε από τη C (C-strings), είτε μέσω της κλάσης string.
- Για την αποθήκευση μιας συμβολοσειράς της C χρησιμοποιείται ένας πίνακας χαρακτήρων. Για παράδειγμα, ο πίνακας char lex[30] επιτρέπει την καταχώριση ενός συνόλου το πολύ 30 χαρακτήρων (μαζί με τον χαρακτήρα τερματισμού '\0'). Για τον χειρισμό συμβολοσειρών της C χρησιμοποιούνται οι συναρτήσεις βιβλιοθήκης strcpy(), strcmp(), strlen(), strcat(), κ.λπ., που έχουν κληρονομηθεί από τη C!
- Ο προτεινόμενος τρόπος χειρισμού συμβολοσειρών είναι μέσω της κλάσης string. Σε ένα αντικείμενο string μπορεί να αποθηκευτεί μια συμβολοσειρά οποιουδήποτε μήκους. Η κατανομή της απαιτούμενης μνήμης γίνεται δυναμικά.
- Ο χειρισμός των συμβολοσειρών που είναι αποθηκευμένες σε αντικείμενα string γίνεται μέσω *μεθόδων* της κλάσης string οι οποίες αναλύονται σε επόμενο κεφάλαιο.
- Ο χειρισμός των πινάκων γίνεται συνήθως με τη χρήση βρόχων for. Για να προσπελάζουμε όλες τις θέσεις μνήμης ενός πίνακα, ανάλογα με το πλήθος των διαστάσεών του, χρησιμοποιούμε και αντίστοιχους ένθετους βρόχους for, κάθε ένας από τους οποίους αλλάζει την τιμή του αριθμοδείκτη της κάθε διάστασης.
- Όταν μεταβιβάζουμε έναν πίνακα ως παράμετρο σε μια συνάρτηση, στην ουσία μεταβιβάζουμε τη διεύθυνση της πρώτης θέσης μνήμης του. Η τυπική παράμετρος πρέπει να είναι ένας πίνακας του ίδιου τύπου, όπου υποχρεωτικά πρέπει να δηλώσουμε την τιμή των διαστάσεών του πλην της πρώτης. Για παράδειγμα, a[] για τη μεταβίβαση ενός πίνακα μίας διάστασης, a[][4][5] για τη μεταβίβαση ενός πίνακα τριών διαστάσεων.
- Μπορούμε να έχουμε πίνακες δεικτών, πίνακες δεικτών σε πίνακες, ακόμα και πίνακες δεικτών σε συναρτήσεις!
- Ένας πίνακας δεικτών σε συναρτήσεις περιέχει διευθύνσεις συναρτήσεων. Με αυτόν τον τρόπο μπορούμε να καλούμε διαφορετικές συναρτήσεις, αλλάζοντας μόνο την τιμή του αριθμοδείκτη του πίνακα.

Κεφάλαιο

12

Τύποι δεδομένων οριζόμενοι από τον χρήστη



Τύποι δεδομένων οριζόμενοι από τον χρήστη

Μέχρι τώρα γνωρίσαμε τους πέντε βασικούς τύπους δεδομένων της C++ (int, char, float, double και bool), καθώς και παραλλαγές τους (όπως ο long double). Επίσης, χρησιμοποιήσαμε αντικείμενα string για την αποθήκευση συμβολοσειρών.

Ο προγραμματιστής είναι αναγκασμένος να προσαρμόζει τα δεδομένα του και το πρόγραμμά του στους τύπους δεδομένων που διαθέτει μια γλώσσα. Φανταστείτε να μετακομίζαμε το σπίτι μας και οι μόνοι αποθηκευτικοί χώροι που μας διέθετε η μεταφορική εταιρεία να ήταν κουτιά τριών-τεσσάρων συγκεκριμένων μεγεθών. Σε κάθε κουτί θα έπρεπε να αποθηκεύσουμε ένα μόνο αντικείμενο. Είναι προφανές ότι σε κάποια κουτιά θα περίσσευε χώρος, ενώ κάποια αντικείμενα θα έπρεπε να τα τεμαχίσουμε για να χωρέσουν στα κουτιά που μας διέθεσαν. Από την άλλη πλευρά, θα ήταν πολύ βολικό αν μπορούσαμε να φτιάχναμε τα δικά μας κουτιά, στις διαστάσεις που επιθυμούσαμε, ανάλογα με τα αντικείμενα που θέλαμε να μεταφέρουμε.

Οι βασικοί τύποι δεδομένων που διαθέτει μια γλώσσα παρομοιάζονται με τα κουτιά συγκεκριμένου μεγέθους που διαθέτει η μεταφορική εταιρεία, ενώ όταν φτιάχνουμε τα δικά μας κουτιά είναι σαν να δημιουργούμε δικούς μας τύπους δεδομένων. Η C++ παρέχει διαφορετικούς τρόπους για να δημιουργούμε δικούς μας τύπους δεδομένων (πέρα από τους βασικούς τύπους που έχουμε ήδη γνωρίσει). Οι οριζόμενοι από τον χρήστη τύποι δεδομένων (user defined data types) ανήκουν στις επόμενες κατηγορίες:

- **Δομές** (Structures)

  Οι δομές είναι συλλογές από μεταβλητές διαφόρων τύπων δεδομένων, οι οποίες αναφέρονται με ένα κοινό όνομα.
- **Μέλη εύρους ενός ή περισσότερων bit** (Bit fields)

  Είναι μια παραλλαγή δομής η οποία επιτρέπει την εύκολη προσπέλαση των επιμέρους bit ενός byte.
- **Ενώσεις** (Unions)

  Οι ενώσεις επιτρέπουν σε δύο ή περισσότερες μεταβλητές, ακόμη και διαφορετικού τύπου, να μοιράζονται το ίδιο τμήμα μνήμης.
- **Απαριθμήσεις** (Enumerations)

  Οι απαριθμήσεις είναι λίστες συμβόλων που αντιστοιχούν σε ακέραιους αριθμούς.
- **Συνώνυμα** (χρήση της δεσμευμένης λέξης typedef)

  Η χρήση της typedef δημιουργεί ένα νέο όνομα (συνώνυμο) για έναν υπάρχοντα τύπο δεδομένων. Η typedef δεν δημιουργεί νέο τύπο δεδομένων.
- **Κλάσεις** (Classes)

  Οι κλάσεις αποτελούν τον ακρογωνιαίο λίθο του αντικειμενοστρεφούς προγραμματισμού. Ενθυλακώνουν (ομαδοποιούν) δεδομένα και διαδικασίες σε οντότητες που καλούνται αντικείμενα.

Δομές

Πολλές φορές στα προγράμματά μας έχουμε δεδομένα που σχετίζονται μεταξύ τους. Για παράδειγμα, το ονοματεπώνυμο, το έτος σπουδών, και οι βαθμοί ενός φοιτητή αποτελούν δεδομένα που αναφέρονται στο ίδιο άτομο. Μέχρι τώρα, αυτό που κάναμε ήταν να χρησιμοποιούμε ξεχωριστές μεταβλητές ή πίνακες για την καταχώριση των δεδομένων μας. Για το συγκεκριμένο παράδειγμα θα χρειαζόμασταν έναν πίνακα χαρακτήρων (για το ονοματεπώνυμο), μία ακέραια μεταβλητή (για το έτος σπουδών), και έναν πίνακα πραγματικών αριθμών για τη βαθμολογία του φοιτητή. Η σχέση μεταξύ αυτών των δεδομένων υφίσταται μόνο μέσα στο μυαλό του προγραμματιστή. Στην πραγματικότητα, δεν υπάρχει καμία δέσμευση ότι αυτά τα δεδομένα αφορούν τον ίδιο φοιτητή.

Η λύση στο παραπάνω πρόβλημα είναι η δημιουργία σύνθετων τύπων δεδομένων, οι οποίοι αποτελούνται από απλούστερους βασικούς τύπους. Αυτοί οι σύνθετοι τύποι λέγονται *δομές* (structures). Η δομή στη C++ είναι μια συλλογή από μεταβλητές, πίνακες, ή γενικότερα από οντότητες κάθε τύπου, οι οποίες αναφέρονται με ένα κοινό όνομα. Η δημιουργία μιας δομής επιτρέπει τον ορισμό μεταβλητών αυτού του τύπου.

Μια δομή αποτελείται από επιμέρους πληροφορίες, οι οποίες καλούνται *μέλη* (ή *πεδία*).

Η δομή είναι συνήθως μια λογική ενότητα από πληροφορίες που σχετίζονται μεταξύ τους. Για παράδειγμα, το όνομα, η διεύθυνση, το τηλέφωνο, και η ηλικία ενός ατόμου είναι ένα χαρακτηριστικό παράδειγμα μιας δομής με τέσσερα μέλη:

- το μέλος του ονόματος,
- το μέλος της διεύθυνσης,
- το μέλος του τηλεφώνου, και
- το μέλος της ηλικίας

Μια δομή ορίζεται με την πρόταση struct. Το επόμενο παράδειγμα ορίζει μία δομή με όνομα stoixeia με τα τέσσερα παραπάνω μέλη:

```
struct stoixeia
{
   char onoma[15];
   char address[20];
   char thl[13];
   int ilikia;
};
```

☞ Με την προηγούμενη πρόταση *δεν* δημιουργήθηκε καμία μεταβλητή, ούτε δεσμεύτηκε καθόλου μνήμη. Απλώς ορίστηκε ένας νέος τύπος δεδομένων με όνομα stoixeia. **ΠΡΟΣΟΧΗ:** η δομή stoixeia δεν είναι μεταβλητή, αλλά ένας τύπος δεδομένων όπως ο char, ο int κ.λπ.

☞ Το τέλος της δήλωσης struct οριοθετείται με ελληνικό ερωτηματικό (;).

Μεταβλητές της δομής stoixeia μπορούν να δηλωθούν με προτάσεις της μορφής:

```
stoixeia pelatis,filos;
```

όπου δηλώνονται δύο μεταβλητές του τύπου δομής stoixeia, με ονόματα **pelatis** και **filos** αντίστοιχα. Η δήλωση μεταβλητών μπορεί να γίνει επίσης μαζί με τον ορισμό της δομής:

```
struct stoixeia
{
   char onoma[15];
   char address[20];
   char thl[13];
   int ilikia;
} pelatis,filos;
```

όπου πάλι δηλώνονται οι δύο μεταβλητές δομής pelatis και filos.

Μπορούμε ακόμα να δηλώσουμε μεταβλητές δομής χωρίς να ορίσουμε κάποιο όνομα για τον τύπο της δομής. Για παράδειγμα, στην παρακάτω πρόταση:

```
struct
{
   char onoma[15];
   char address[20];
   char thl[13];
   int ilikia;
} pelatis,filos;
```

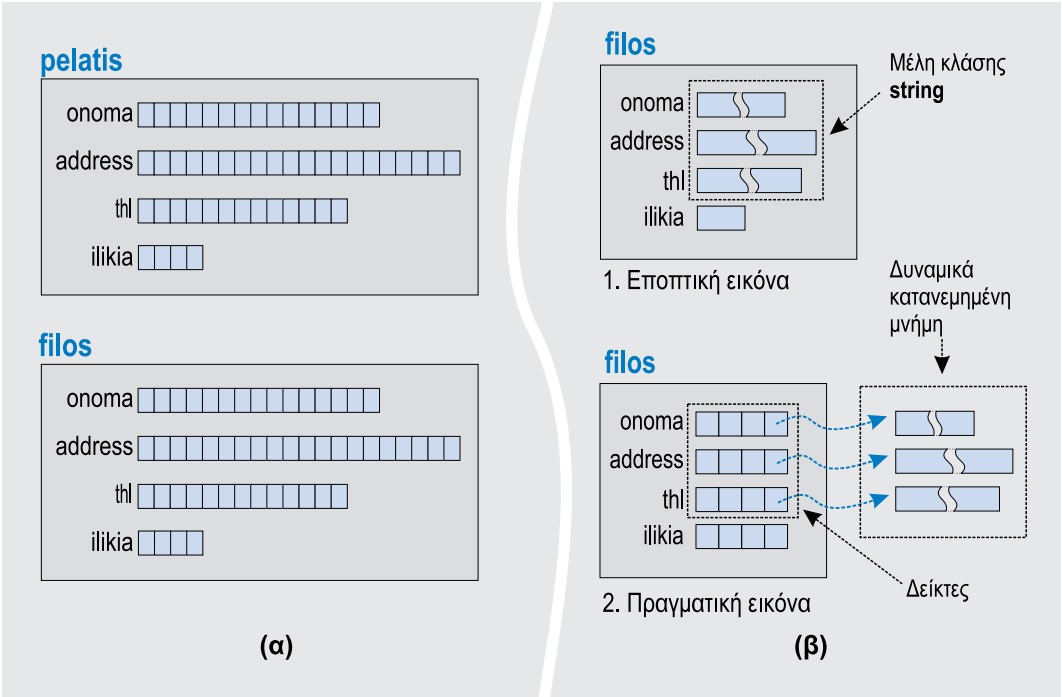
δηλώνονται οι δύο μεταβλητές, με την προηγούμενη δομή, αλλά δεν ορίζεται κάποιο όνομα για το νέο τύπο δεδομένων (όπως το όνομα stoixeia). Αυτό σημαίνει ότι δεν έχουμε τη δυνατότητα να δηλώσουμε αργότερα, με μια ξεχωριστή πρόταση δήλωσης, μεταβλητές με την ίδια δομή. Γι' αυτόν το λόγο, αυτός ο τρόπος δήλωσης δεν χρησιμοποιείται συχνά.

Με όποιον τρόπο όμως και αν δηλωθούν, η κάθε μία από τις μεταβλητές pelatis και filos καταλαμβάνει χώρο 52 byte στη μνήμη (15+20+13+4), και έχουν τη διάταξη που φαίνεται στο Σχήμα 12.1α της επόμενης σελίδας. Ο τελεστής sizeof αποδίδει το συνολικό μέγεθος μιας δομής (σε byte) – όπως ακριβώς και με τους βασικούς τύπους δεδομένων. Στα προηγούμενα παραδείγματα χρησιμοποιήθηκαν μέλη με πίνακες χαρακτήρων για την αποθήκευση συμβολοσειρών. Όπως έχουμε ήδη αναφέρει, στη C++ πρέπει να προτιμάται η χρήση αντικειμένων string.

Επομένως, η δομή stoixeia των προηγούμενων παραδειγμάτων διαμορφώνεται ως εξής:

```
struct stoixeia
{
   string onoma;
   string address;
   string thl;
   int ilikia;
} filos;
```

Δεδομένου ότι τα αντικείμενα string δεν δεσμεύουν συγκεκριμένο χώρο αλλά χρησιμοποιούν τόσο χώρο όσο χρειάζονται, είναι λογικό να αναρωτηθούμε: Πόσο είναι τώρα το μέγεθος σε byte μιας μεταβλητής δομής stoixeia; Στο Σχήμα 12.1β (1) φαίνεται η μορφή της μεταβλητής δομής filos που δηλώθηκε μαζί με τη δομή stoixeia. Η πρώτη μας σκέψη λογικά θα είναι ότι το μέγεθος της μεταβλητής θα είναι ανάλογο με το πλήθος των αποθηκευμένων χαρακτήρων στα μέλη της. Όμως, η πραγματική εικόνα της μεταβλητής filos είναι αυτή που φαίνεται στη δεύτερη περίπτωση του Σχήματος 12.1β (2). Τα τρία πρώτα μέλη της μεταβλητής στη πραγματικότητα είναι δείκτες προς τους χώρους που έχει δεσμεύσει δυναμικά η κλάση string για την αποθήκευση των συμβολοσειρών. Η αποθήκευση γίνεται σε ξεχωριστό τμήμα της μνήμης, το οποίο είναι ανεξάρτητο από τον χώρο που καταλαμβάνει η μεταβλητή δομής.

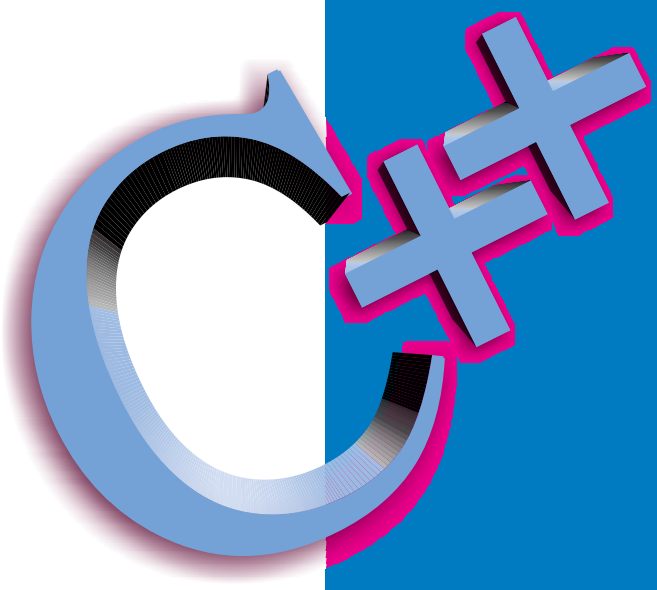


Σχήμα 12.1 *Δομές με πεδία πίνακες χαρακτήρων ή αντικείμενα string*

Κεφάλαιο

13

Κλάσεις και αντικείμενα



Κλάσεις και αντικείμενα

Οι κλάσεις και τα αντικείμενα είναι έννοιες επάνω στις οποίες βασίζεται όλη η φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού.

Θα ήθελα να χαλαρώσετε, να φτιάξετε ένα καφεδάκι, καλού-κακού πάρτε και μια ασπιρίνη και ετοιμαστείτε να ξεχάσετε όλα όσα μάθατε μέχρι τώρα.

Η αλήθεια είναι ότι δεν πρέπει να πανικοβάλλεστε, αφού οι εντολές που μάθατε, οι συναρτήσεις, τα αντικείμενα που χρησιμοποιήσατε, αλλά και ο διαδικαστικός τρόπος προγραμματισμού θα εξακολουθούν να είναι χρήσιμα. Όμως, θα χρειαστεί όλα αυτά να τα εντάξουμε σε μια διαφορετική φιλοσοφία προγραμματισμού και σε μια διαφορετική φιλοσοφία σκέψης και σχεδιασμού των προγραμμάτων μας.

> Ο αντικειμενοστρεφής προγραμματισμός δεν περιορίζεται απλώς στη χρήση μιας διαφορετικής γλώσσας προγραμματισμού με κάποιες νέες δυνατότητες, αλλά είναι μια άλλη φιλοσοφία σκέψης και σχεδιασμού εφαρμογών.

Σε αυτό το σημείο, θα σας παρακαλέσω να ξαναδιαβάσετε με *πολλή* προσοχή την ενότητα «Αντικειμενοστρεφής Προγραμματισμός» του Κεφαλαίου 1 (σελίδες 40-41), προτού συνεχίσετε την ανάγνωση του παρόντος κεφαλαίου.

Παίζοντας με αντικείμενα

Ο άνθρωπος πριν αρχίζει να κατασκευάζει τα δικά του αντικείμενα πρώτα εξοικειώνεται με τη χρήση τους παίζοντας με αυτά. Ένας πιτσιρικάς παίζει με μπάλες πολύ προτού ενηλικιωθεί και φτιάξει μια βιοτεχνία που να τις κατασκευάζει!

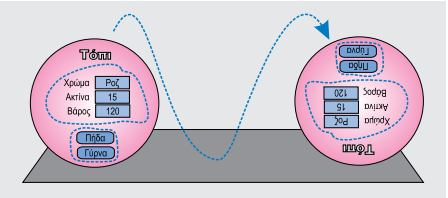
Τι είναι όμως ένα αντικείμενο; Μπορούμε να δώσουμε έναν ορισμό;

> Αντικείμενο είναι μια οντότητα η οποία διαθέτει χαρακτηριστικά τα οποία προσδιορίζουν την κατάστασή του, και ενέργειες (μεθόδους) οι οποίες καθορίζουν τις διάφορες συμπεριφορές του.

Για παράδειγμα, ένα από τα πρώτα αντικείμενα με τα οποία έρχεται σε επαφή ο άνθρωπος είναι ένα τόπι! Το τόπι έχει χαρακτηριστικά όπως το χρώμα του, η ακτίνα που προσδιορίζει το μέγεθός του και το βάρος του. Επίσης, το τόπι διαθέτει ενέργειες που το κάνουν να αναπηδήσει ή να γυρίσει (Σχήμα 13.1α).

Στο παράδειγμα του Σχήματος 13.1, το αντικείμενό μας ονομάζεται **Τόπι**, διαθέτει τα χαρακτηριστικά που αναφέρθηκαν προηγουμένως και τις ενέργειες **Πήδα** και **Γύρνα**, που υποθέτουμε ότι το αναγκάζουν να αναπηδήσει μια φορά και να περιστραφεί κατά 180^{o} αντίστοιχα.

Οι παρακάτω προτάσεις θέτουν τις τιμές που φαίνονται στο Σχήμα 13.1 στα χαρακτηριστικά του αντικειμένου Τόπι, το κάνουν να αναπηδήσει δύο φορές και να περιστραφεί κατά 180°:

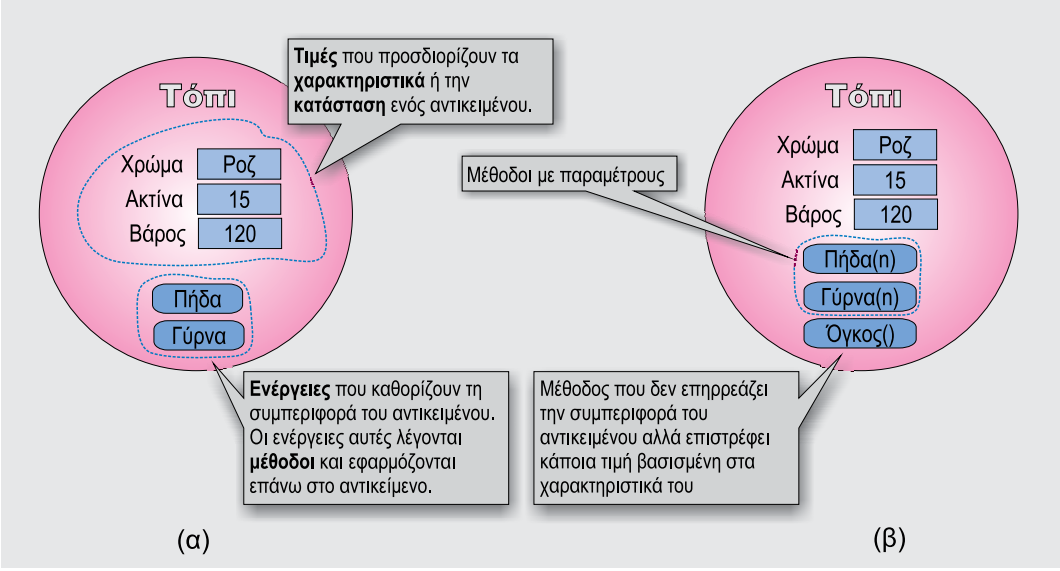



```
Τόπι.Χρώμα="Ροζ";
Τόπι.Ακτίνα=15;
Τόπι.Βάρος=120;
Τόπι.Πήδα;
Τόπι.Πήδα;
Τόπι.Γύρνα;
```

Παρατηρούμε ότι η πρόσβαση στα χαρακτηριστικά και στις μεθόδους του αντικειμένου γίνεται με χρήση του τελεστή τελείας, όπως ακριβώς και στα μέλη μιας δομής.

Στο συγκεκριμένο παράδειγμα, οι μέθοδοι που διαθέτει το αντικείμενο Τόπι εκτελούν την ίδια ακριβώς λειτουργία κάθε φορά που εφαρμόζονται. Στο Σχήμα 13.1β φαίνεται μια νέα εκδοχή του αντικειμένου Τόπι, όπου οι δύο μέθοδοι διαθέτουν μια παράμετρο η οποία προσδιορίζει τον αριθμό των αναπηδήσεων και τη γωνία περιστροφής αντίστοιχα.

Επίσης, το αντικείμενο Τόπι τώρα διαθέτει και μια επιπλέον μέθοδο, την Όγκος(), η οποία δεν καθορίζει καμία συμπεριφορά του αντικειμένου αλλά επιστρέφει ως τιμή τον όγκο που διαθέτει το τόπι ο οποίος υπολογίζεται[1] με βάση την ακτίνα του.



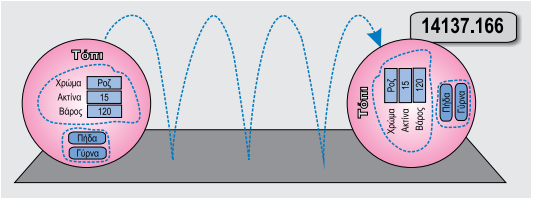
Σχήμα 13.1 *Αντικείμενα – Χαρακτηριστικά και μέθοδοι*

[1] Ο όγκος μιας σφαίρας υπολογίζεται από τον τύπο **V=4/3*π*r³**. Όπου **π** είναι η σταθερά 3.141592653589... και **r** η ακτίνα της σφαίρας.

Αν το τόπι μας ήταν ένα αντικείμενο της C++, ο ακόλουθος ψευδοκώδικας:

```
Τόπι.Πήδα(4);
Τόπι.Γύρνα(270);
cout << Τόπι.Όγκος();
```

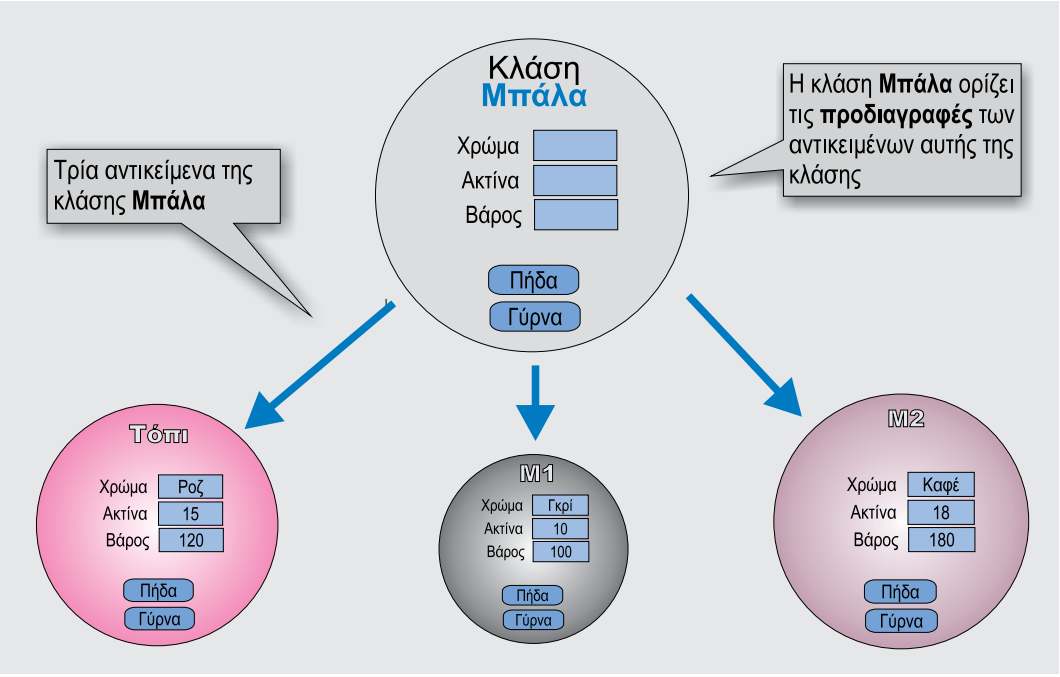
θα το έκανε να αναπηδήσει 4 φορές, να περιστραφεί κατά 270° και στο τέλος να εμφανίσει στην οθόνη τον όγκο του.



Τι είναι κλάση και τι αντικείμενο

Όλες οι μπάλες που παίζαμε όταν ήμασταν μικροί, παρά το γεγονός ότι δεν ήταν ίδιες, είχαν κοινά χαρακτηριστικά που τις περιέγραφαν και κοινές λειτουργίες. Όλες είχαν χρώμα, βάρος και μέγεθος και όλες αναπηδούσαν και γυρνούσαν.

Όπως έχει αναφερθεί και στην εισαγωγή, μια κλάση είναι μια έννοια η οποία προσδιορίζει μια κατηγορία αντικειμένων και περιγράφει τα κοινά χαρακτηριστικά τους. Για παράδειγμα, η κλάση «Μπάλα» του Σχήματος 13.2 περιγράφει μια κατηγορία παιχνιδιών, τις μπάλες, με κοινά χαρακτηριστικά και λειτουργίες.



Σχήμα 13.2 *Κλάσεις και αντικείμενα*

Η **κλάση** (class) είναι μια έννοια η οποία προσδιορίζει μια κατηγορία αντικειμένων και ταυτόχρονα περιγράφει τα κοινά χαρακτηριστικά τους.

Μια κλάση δεν αναφέρεται σε κανένα συγκεκριμένο αντικείμενο, αλλά είναι μια γενική έννοια που προσδιορίζει μια κατηγορία αντικειμένων. Για παράδειγμα, η κλάση «Ορθογώνιο παραλληλόγραμμο» δεν αναφέρεται σε κανένα συγκεκριμένο ορθογώνιο, αλλά αποτελεί μια έννοια που προσδιορίζει μια κατηγορία σχημάτων.

Ο ορισμός μιας κλάσης περιλαμβάνει χαρακτηριστικά και λειτουργίες οι οποίες προσδιορίζουν τα αντικείμενα αυτής της κλάσης. Μπορούμε να πούμε ότι μια κλάση είναι το σύνολο των *προδιαγραφών* για την κατασκευή αντικειμένων με συγκεκριμένα χαρακτηριστικά και λειτουργίες.

Ένα αντικείμενο είναι μια πραγματική υπόσταση μιας κλάσης. Για παράδειγμα, όταν αναφερόμαστε στην κλάση «Μπάλα», αναφερόμαστε σε μια γενική έννοια που προσδιορίζει κάποια χαρακτηριστικά και λειτουργίες. Όμως όταν κατασκευάσουμε μια μπάλα με αυτά τα χαρακτηριστικά και τις λειτουργίες, έχουμε ένα *αντικείμενο* αυτής της κλάσης.

Στο παραπάνω σχήμα τα αντικείμενα **Τόπι**, **Μ1** και **Μ2** ανήκουν στην κλάση **Μπάλα**, έχουν κοινά χαρακτηριστικά, αλλά με διαφορετικές τιμές το καθένα, και ίδιες λειτουργίες. Τα αντικείμενα μιας κλάσης καλούνται και *στιγμιότυπα* της κλάσης. Όλα τα αντικείμενα που ανήκουν σε μια κλάση έχουν κοινά χαρακτηριστικά, πιθανώς με διαφορετικές τιμές το καθένα, και κοινές λειτουργίες.

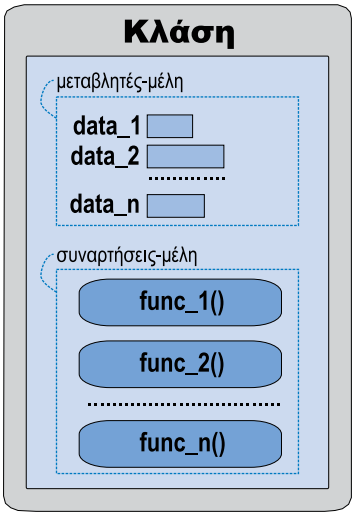
Ένα **αντικείμενο** (object) μιας κλάσης έχει όλα τα χαρακτηριστικά και τις λειτουργίες της κλάσης στην οποία ανήκει. Ένα αντικείμενο λέμε ότι αποτελεί ένα **στιγμιότυπο** (instance) μιας κλάσης.

Μια κλάση μπορεί να οριστεί και να μην υπάρχει κανένα αντικείμενο που να ανήκει σε αυτή την κλάση. Όταν φτιάχνουμε τα σχέδια για ένα σπίτι στην ουσία έχουμε μια κλάση αλλά κανένα αντικείμενο. Όταν θα κατασκευάσουμε το πρώτο σπίτι με βάση τα σχέδια που έχουμε κάνει, τότε θα έχουμε το πρώτο μας αντικείμενο.

Ένας πιτσιρίκος μπορεί να έχει πολλές μπάλες οι οποίες, ακόμα και αν είναι ακριβώς ίδιες (χρώμα, μέγεθος, βάρος), αποτελούν διαφορετικά αντικείμενα με δική τους ταυτότητα.

Κάθε αντικείμενο έχει **ταυτότητα** (identity) η οποία το διαφοροποιεί από τα υπόλοιπα αντικείμενα, ακόμα και αν τα χαρακτηριστικά τους έχουν ακριβώς τις ίδιες τιμές.

Ορισμός μιας κλάσης



Μια κλάση στη C++ έχει την ίδια μορφή με μια δομή και ορίζεται με τον ίδιο τρόπο. Μια κλάση αποτελείται από τις **μεταβλητές-μέλη** που προσδιορίζουν τα χαρακτηριστικά και την κατάσταση των αντικειμένων της κλάσης, και τις **συναρτήσεις-μέλη** που προσδιορίζουν τις λειτουργίες που μπορούν να επιτελούν.

Ο ορισμός μιας κλάσης γίνεται όπως και ο ορισμός μιας δομής, με τη διαφορά ότι χρησιμοποιείται το προσδιοριστικό **class**.

class ***όνομα_κλάσης***

{

Δηλώσεις μεταβλητών-μελών

Δηλώσεις συναρτήσεων-μελών

};

Το σώμα μιας κλάσης αποτελείται από τις δηλώσεις των μεταβλητών-μελών στις οποίες αποθηκεύονται τα χαρακτηριστικά ή η κατάσταση των αντικειμένων της κλάσης, και από τις δηλώσεις των συναρτήσεων-μελών που επενεργούν στα αντικείμενα της κλάσης.

Παρατηρούμε επίσης ότι, όπως και στις δομές, ο ορισμός μιας κλάσης τερματίζεται με το ελληνικό ερωτηματικό (;).

Ας φανταστούμε ένα ορθογώνιο παραλληλόγραμμο, το οποίο προσδιορίζεται από τις διαστάσεις των δύο πλευρών του και το χρώμα του.

Ο παρακάτω κώδικας ορίζει μία κλάση με όνομα rectangle με τρεις μεταβλητές-μέλη και δύο συναρτήσεις-μέλη. Η κλάση αυτή θα χρησιμοποιηθεί για τη δημιουργία και την επεξεργασία αντικειμένων ορθογωνίων παραλληλογράμμων.

```
class rectangle
{
public:
   float plevra_a;
   float plevra_b;
   string xroma;
   float emvado()
   {
      return plevra_a * plevra_b;
   }
```

Ορισμός της κλάσης **rectangle**. Η κλάση περιγράφει αντικείμενα που αντιστοιχούν σε ορθογώνια παραλληλόγραμμα.

Μεταβλητές-μέλη της κλάσης **rectangle**. Στις μεταβλητές-μέλη αποθηκεύονται οι διαστάσεις και το χρώμα των αντικειμένων (ορθογωνίων παραλληλογράμμων) της κλάσης.

Συνάρτηση-μέλος της κλάσης **rectangle** η οποία υπολογίζει και επιστρέφει το εμβαδό του ορθογωνίου παραλληλογράμμου για το οποίο καλείται.

```
   void set_ab(float a, float b)
   {
      plevra_a = a;
      plevra_b = b;
   }
};
```

Η συνάρτηση-μέλος **set_ab()** καταχωρίζει τις τιμές των ορισμάτων με τις οποίες καλείται στις μεταβλητές μέλη **plevra_a** και **plevra_b** του ορθογωνίου παραλληλογράμμου για το οποίο καλείται.

Παρατηρούμε για πρώτη φορά το προσδιοριστικό **public:** πριν από τους ορισμούς των μελών της κλάσης. Προς το παρόν αγνοήστε το. Στις επόμενες ενότητες θα περιγραφεί αναλυτικά η σημασία και η χρήση του.

Οι συναρτήσεις-μέλη μπορούν επίσης να οριστούν και έξω από το σώμα μιας κλάσης. Σε αυτή την περίπτωση απλώς δηλώνονται μέσα στην κλάση, αλλά ορίζονται έξω από αυτή:

```
class rectangle
{
public:
   float plevra_a;
   float plevra_b;
   string xroma;
   float emvado();
   void set_ab(float a, float b);
};
float rectangle::emvado()
{
   return plevra_a * plevra_b;
}
void rectangle::set_ab(float a, float b)
{
   plevra_a = a;
   plevra_b = b;
}
```

Δηλώσεις των συναρτήσεων-μελών οι οποίες ορίζονται αργότερα.

Ορισμός της συνάρτησης **emvado()**. Το πρόθεμα **rectangle**:: υποδηλώνει οτι η συνάρτηση **emvado()** είναι συνάρτηση-μέλος της κλάσης **rectangle**.

Ορισμός της συνάρτησης **set_ab()**. Το πρόθεμα **rectangle**:: υποδηλώνει ότι και η συνάρτηση **set_ab()** είναι συνάρτηση-μέλος της κλάσης **rectangle**.

Όταν μια συνάρτηση-μέλος μιας κλάσης ορίζεται έξω από την κλάση, τότε πριν από το όνομά της πρέπει να αναφέρεται το όνομα της κλάσης στην οποία ανήκει: **όνομα_κλάσης::όνομα_συνάρτησης().** Ο τελεστής **::** ονομάζεται τελεστής **επίλυσης εμβέλειας** και χρησιμοποιείται όταν θέλουμε να αναφερθούμε σε μια οντότητα η οποία ορίζεται σε έναν διαφορετικό χώρο εμβέλειας.

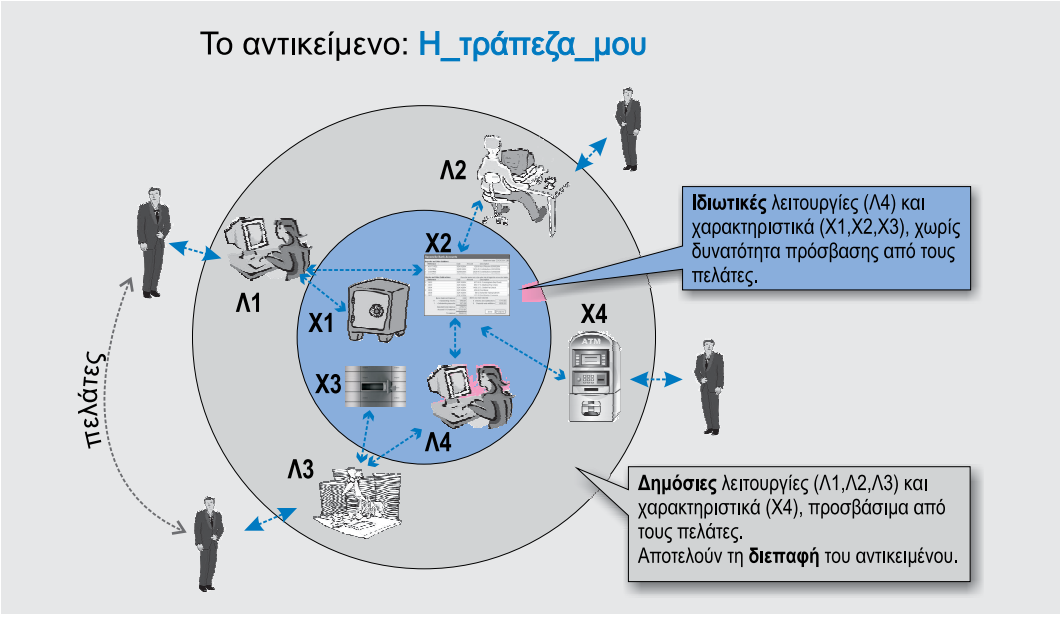
Ορισμός και χρήση αντικειμένων

Ο κυριότερος λόγος για την ύπαρξη των κλάσεων είναι η δημιουργία αντικειμένων τα οποία αποτελούν τη βάση για τη δημιουργία αντικειμενοστρεφών εφαρμογών.

Αν υπάρχουν ακόμα αμφιβολίες για το τι είναι διεπαφή ενός αντικειμένου, μελετήστε ένα παράδειγμα της καθημερινότητας που απεικονίζεται στο Σχήμα 13.5. Ας θεωρήσουμε μια συγκεκριμένη τράπεζα ως ένα αντικείμενο το οποίο περιλαμβάνει χαρακτηριστικά (Χ1-Χ2-Χ3-Χ4) που είναι οι αποθηκευτικοί της χώροι, και λειτουργίες (Λ1-Λ2-Λ3-Λ4) που υλοποιούνται από τους υπαλλήλους της. Χρήστες αυτού του αντικειμένου είναι οι πελάτες της τράπεζας. Κάποια χαρακτηριστικά της τράπεζας είναι άμεσα προσπελάσιμα από τους πελάτες, όπως το ATM, (Χ4) ενώ το χρηματοκιβώτιο (Χ1), οι λογαριασμοί πελατών (Χ2) και οι θυρίδες (Χ3) δεν είναι άμεσα προσπελάσιμα από τους πελάτες αλλά μόνο από μερικούς υπαλλήλους της τράπεζας. Επίσης, σε κάποιες λειτουργίες της τράπεζας έχουν πρόσβαση οι πελάτες μέσω των υπαλλήλων της (Λ1,Λ2 και Λ3). Υπάρχουν όμως και εσωτερικές λειτουργίες της τράπεζας που υλοποιούνται πάλι από υπαλλήλους (π.χ. Λ4) οι οποίοι όμως δεν έρχονται σε καμία επαφή με τους πελάτες.

Ο εσωτερικός μπλε κύκλος περιλαμβάνει τα «προστατευμένα» ιδιωτικά χαρακτηριστικά και λειτουργίες της τράπεζας, ενώ ο εξωτερικός γκρίζος κύκλος περιλαμβάνει αυτά που είναι δημόσια και προσπελάσιμα από τους πελάτες της. Παρατηρούμε ότι τα ιδιωτικά μέλη της τράπεζας μπορεί να μην είναι προσπελάσιμα από τους πελάτες (π.χ. τα Χ3 και Λ4), αλλά είναι από κάποια δημόσια μέλη της (π.χ. από το Λ3).

Επομένως, η *διεπαφή* της τράπεζας είναι ο εξωτερικός γρίζος κύκλος, με τον οποίον μπορούν να έρχονται σε επαφή οι πελάτες της.

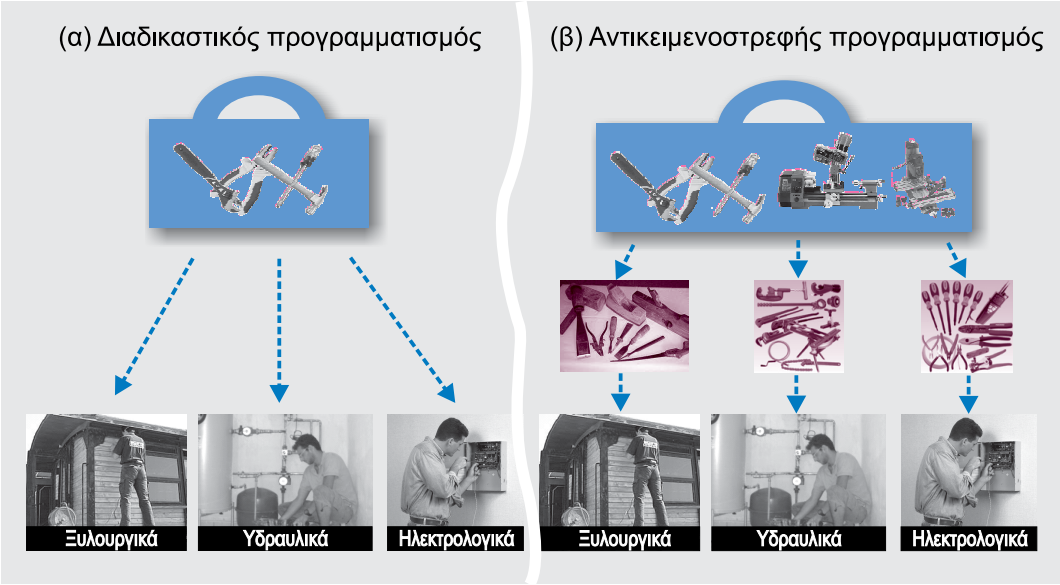


Σχήμα 13.5 *Το σύστημα διεπαφής ενός αντικειμένου-τράπεζα*

Σκεφτείτε αντικειμενοστρεφώς

Από την αρχή του παρόντος κεφαλαίου επεσήμανα ότι η αντικειμενοστρεφής αντιμετώπιση ενός προβλήματος επιφέρει ριζική αλλαγή στον τρόπο που πρέπει σκεφτόμαστε και να σχεδιάζουμε τα προγράμματά μας. Ένα πρόβλημα του πραγματικού κόσμου έχει δεδομένα και διαδικασίες οι οποίες τα επεξεργάζονται. Στον προγραμματισμό, αυτά τα δεδομένα θα πρέπει να μοντελοποιηθούν, δηλαδή να αντιστοιχιστούν σε κατάλληλες μεταβλητές (ή άλλου τύπου δομές) στις οποίες μπορούμε να τα αποθηκεύσουμε. Επίσης, η επεξεργασία στην οποία υπόκεινται θα πρέπει να υλοποιηθεί από συγκεκριμένες συναρτήσεις του προγράμματος. Οι κλάσεις και τα αντικείμενα χρησιμοποιούνται από τους προγραμματιστές για να «μοντελοποιήσουν» δεδομένα και διαδικασίες του πραγματικού κόσμου.

Στο Σχήμα 13.6 φαίνεται η διαφορετική αντιμετώπιση των προβλημάτων στον διαδικαστικό και στον αντικειμενοστρεφή προγραμματισμό. Μια γλώσσα προγραμματισμού δεν είναι τίποτε άλλο παρά μια εργαλειοθήκη γεμάτη εργαλεία τα οποία χρησιμοποιούμε για τη μοντελοποίηση και λύση των προβλημάτων του πραγματικού κόσμου. Οι διαδικαστικές γλώσσες προγραμματισμού διαθέτουν έναν περιορισμένο αριθμό εργαλείων στα οποία πρέπει να προσαρμοστούμε και με αυτά είμαστε υποχρεωμένοι να λύσουμε οποιοδήποτε πρόβλημα. Για παράδειγμα, αν η εργαλειοθήκη μας περιέχει μαχαίρι και όχι πριόνι, είμαστε αναγκασμένοι να κόβουμε τα ξύλα με το μαχαίρι!



Σχήμα 13.6 *Διαδικαστική και αντικειμενοστρεφής αντιμετώπιση ενός προβλήματος*

Οι αντικειμενοστρεφείς γλώσσες, πέρα από τα κλασικά εργαλεία περιέχουν και εργαλεία με τα οποία μπορούμε να φτιάχνουμε άλλα εργαλεία! Έτσι, αν η εργαλειοθήκη μας δεν διαθέτει πριόνι τότε απλώς θα φτιάξουμε ένα! Ένα πρόγραμμα μπορεί κάλλιστα να γραφεί με μια διαδικαστική γλώσσα προγραμματισμού. Αυτό που προσφέρουν οι αντικειμενοστρεφείς γλώσσες είναι ότι ο σχεδιασμός και η μοντελοποίηση του προγράμματος είναι πολύ πιο κοντά στην πραγματική υπόσταση του προβλήματος.

> Αντί να προσαρμόσουμε το πρόβλημα στη γλώσσα προγραμματισμού που χρησιμοποιούμε, προσαρμόζουμε τη γλώσσα στο πρόβλημα που θέλουμε να λύσουμε κατασκευάζοντας τα κατάλληλα εργαλεία.

Διαδικαστική αντιμετώπιση

Ας υποθέσουμε ότι θέλουμε να κρατάμε τις διαστάσεις και τα χρώματα δύο σχημάτων ορθογωνίων παραλληλογράμμων και δύο κύκλων. Επίσης θέλουμε να μπορούμε να υπολογίζουμε και το εμβαδόν αυτών των σχημάτων. Το πρόγραμμα που ακολουθεί δείχνει τη διαδικαστική αντιμετώπιση του προβλήματος. Χρησιμοποιούνται οι μεταβλητές p1a, p1b, p2a και p2b για την καταχώριση των πλευρών των δύο παραλληλογράμμων και οι k1r και k2r για την καταχώριση των ακτίνων των κύκλων. Στα αντικείμενα xrp1, xrp2, xrk1 και xrk2 τύπου string καταχωρίζονται τα χρώματα των τεσσάρων σχημάτων και υπάρχουν οι συναρτήσεις emvado_p() και emvado_k() για τον υπολογισμό του εμβαδού ορθογωνίου παραλληλογράμμου και κύκλου αντίστοιχα.

13_functional.cpp

```
#include <iostream>
#include <string>
using namespace std;
float emvado_p(float x, float y);
float emvado_k(float ak);
int main()
{
   // Οι μεταβλητές στις οποίες αποθηκεύονται τα μήκη των πλευρών των παραλληλογράμμων
   float p1a=10,p1b=5,p2a=7,p2b=8;
   // Οι μεταβλητές στις οποίες αποθηκεύονται τα μήκη των ακτίνων των κύκλων
   float k1r=5,k2r=4;
   // Αποθήκευση των χρωμάτων των σχημάτων
   string xrp1="Πράσινο",xrp2="Κόκκινο",xrk1="Γαλάζιο",xrk2="Κίτρινο";
   float emp1, emk1;
   emp1=emvado_k(p1a,p1b);  // Υπολογισμός του εμβαδού του ενός παραλληλογράμμου
   emk2=emvado_p(k2r);      // Υπολογισμός του εμβαδού του ενός κύκλου
   // Εμφανίζει τα στοιχεία του πρώτου παραλληλογράμμου
   cout << "Ορθογώνιο "<< xrp1 <<" "<<p1a<<"x"<<p1b<<" E="<<emp1<<endl;
   // Εμφανίζει τα στοιχεία του δεύτερου κύκλου
   cout << "Κύκλος "<< xrk2 <<" R="<<k2r<<" E="<<emk1<<endl;
   return 0;
}
```

```
float emvado_p(float x, float y)
{
   return x*y;
}
float emvado_k(float ak)
{
   return 3.14*ak*ak;
}
```

Συνάρτηση υπολογισμού του εμβαδού ορθογωνίου παραλληλογράμμου.

Συνάρτηση υπολογισμού του εμβαδού κύκλου.

Το γεγονός ότι οι μεταβλητές p1a, p1b και xrp1 αφορούν χαρακτηριστικά του ίδιου σχήματος (δηλαδή του πρώτου ορθογώνιου παραλληλογράμμου) είναι μόνο μέσα στο μυαλό μας. Το μόνο που προδίδει τη συσχέτιση των μεταβλητών μεταξύ τους είναι η έξυπνη επιλογή των ονομάτων τους. Για το πρόγραμμά μας είναι ανεξάρτητες μεταβλητές χωρίς καμία απολύτως συσχέτιση, όπως ακριβώς και οι υπόλοιπες που αφορούν τα υπόλοιπα σχήματα.

Αντικειμενοστρεφής αντιμετώπιση

Το πρόγραμμα που ακολουθεί δείχνει την αντικειμενοστρεφή προσέγγιση του ίδιου προβλήματος. Τώρα για τη διαχείριση των σχημάτων χρησιμοποιούνται οι κλάσεις rectangle και circle. Στις μεταβλητές-μέλη των κλάσεων αποθηκεύονται οι διαστάσεις και το χρώμα των σχημάτων. Κάθε κλάση επίσης διαθέτει μια μέθοδο για τον υπολογισμό του εμβαδού του σχήματος. Επομένως το πρόβλημα τώρα ανάγεται στη δημιουργία και στη χρήση αντικειμένων αυτών των κλάσεων. Μελετήστε το με προσοχή!

13_OOP.cpp

```
#include <iostream>
#include <string>
using namespace std;
class rectangle
{
public:
   float plevra_a;
   float plevra_b;
   string xroma;
   float emvado()
   {
      return plevra_a * plevra_b;
   }
   void display()
   {
      cout << "Ορθογώνιο : "<<xroma<<" "<<plevra_a<<"x"<<plevra_b<<endl;
   }
};
class circle
{
public:
   float aktina;
```

Ορισμός της κλάσης **rectangle** για τη διαχείριση αντικειμένων ορθογωνίων παραλληλογράμμων.

Μέθοδος υπολογισμού του εμβαδού αντικειμένων ορθογωνίων παραλληλογράμμων.

Μέθοδος εμφάνισης στοιχείων αντικειμένων ορθογωνίων παραλληλογράμμων.

Ορισμός της κλάσης **circle** για τη διαχείριση αντικειμένων κύκλων.

```
   string xroma;
   float emvado()
   {
      return aktina*aktina*3.14;
   }
   void display()
   {
      cout << "Κύκλος : " << xroma << " R=" << aktina << endl;
   }
};
int main()
{
   int a,b;
   rectangle rec1,rec2;
   circle c1,c2;
   rec1.plevra_a=10;
   rec1.plevra_b=5;
   rec1.xroma="Πράσινο";
   rec2.plevra_a=7;
   rec2.plevra_b=8;
   rec2.xroma="Κόκκινο";
   c1.aktina=5;
   c1.xroma="Γαλάζιο";
   c2.aktina=4;
   c2.xroma="Κίτρινο";
   rec1.display();
   rec2.display();
   c1.display();
   c2.display();
   cout<<rec1.emvado()<<endl;
   cout<<c2.emvado()<<endl;
   return 0;
}
```

Μέθοδος υπολογισμού του εμβαδού αντικειμένων κύκλων.

Μέθοδος εμφάνισης στοιχείων αντικειμένων κύκλων.

Δημιουργία δύο αντικειμένων ορθογωνίων παραλληλογράμμων και δύο αντικειμένων κύκλων.

Ανάθεση διαστάσεων και χρωμάτων στα τέσσερα αντικείμενα

Εμφάνιση των στοιχείων των τεσσάρων αντικειμένων.

Εμφάνιση του εμβαδού του ενός ορθωγωνίου παραλληλογράμου και του ενός κύκλου.

Παρατηρούμε ότι, από τη στιγμή που ορίσαμε τις δύο κλάσεις, το κύριο σώμα του προγράμματος που βρίσκεται στη συνάρτηση main() είναι πολύ λιτό, δομημένο και κατανοητό. Επίσης, η προσέγγιση αυτή μας προφυλάσσει από λανθασμένη ή κακή χρήση του κώδικα. Για παράδειγμα, η μέθοδος emvado() δεν μπορεί να χρησιμοποιηθεί αυτόνομα παρά μόνο σε συνδυασμό με κάποιο αντικείμενο των δύο αυτών κλάσεων, ενώ στη διαδικαστική αντιμετώπιση του προβλήματος ο ακόλουθος κώδικας θα ήταν απόλυτα σωστός:

```
cout << "Δώσε ηλικία:";
cin >> ilikia;
cout embado_k(ilikia);
```

Δηλαδή, ζητάμε από τον χρήστη να πληκτρολογήσει μια ηλικία και μετά τη χρησιμοποιούμε ως ακτίνα ενός κύκλου του οποίου υπολογίζουμε το εμβαδό του! Αυτό προφανώς είναι λάθος του προγραμματιστή, αλλά η γλώσσα μας αφήνει να το κάνουμε.

Απεικόνιση κλάσεων με τη UML

Όπως αναφέρθηκε στην εισαγωγή, η UML (Unified Modeling Language – Ενοποιημένη Γλώσσα Μοντελοποίησης) είναι μια γλώσσα μοντελοποίησης, η οποία μέσω συγκεκριμένων στοιχείων και διαγραμμάτων μας δίνει τη δυνατότητα να απεικονίσουμε συστήματα αντικειμενοστρεφών εφαρμογών.

Ένα από τα εργαλεία μοντελοποίησης που χρησιμοποιεί η UML είναι τα *διαγράμματα κλάσεων*. Τα διαγράμματα αυτά περιγράφουν τη δομή των κλάσεων μιας εφαρμογής καθώς και τη συσχέτιση των κλάσεων μεταξύ τους.

Μια κλάση σε ένα διάγραμμα κλάσεων απεικονίζεται ως ένα ορθογώνιο που αποτελείται από τρία οριζόντια τμήματα (Σχήμα 13.7α). Στο πάνω τμήμα αναφέρεται το όνομα της κλάσης, στο μεσαίο τμήμα τα χαρακτηριστικά της (μεταβλητές-μέλη) και στο κάτω τμήμα οι λειτουργίες της (μέθοδοι).

Κάθε μεταβλητή-μέλος αναφέρεται ως ακολούθως:

```
±όνομα:τύπος=αρχική_τιμή
```

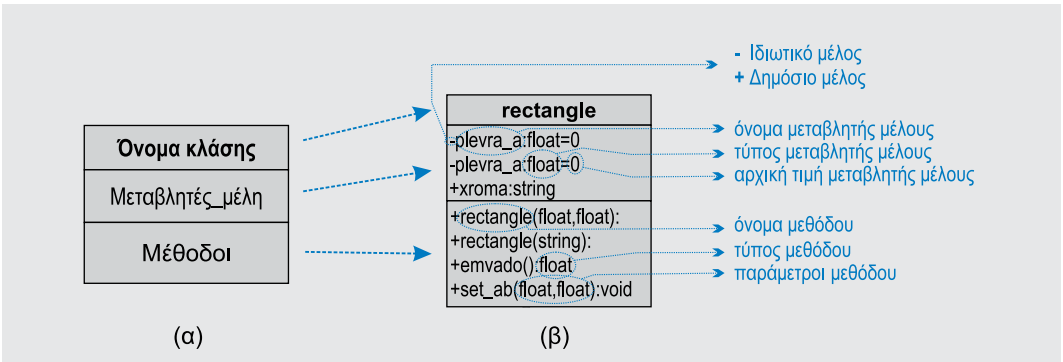
Το πρόθεμα + υποδεικνύει δημόσιο μέλος και το - ιδιωτικό, ***όνομα*** είναι το όνομα της μεταβλητής-μέλους, ***τύπος*** είναι ο τύπος της και ***αρχική_τιμή*** είναι η αρχική της τιμή στην περίπτωση που ορίζεται κάποια.

Κάθε μέθοδος αναφέρεται ως ακολούθως:

```
±όνομα(τύπος_παρ1,τύπος_παρ2,....):τύπος
```

Όνομα είναι το όνομα μεθόδου, ***τύπος_παρ1***, ***τύπος_παρ2***, ... είναι ο τύπος κάθε παραμέτρου της μεθόδου και ***τύπος*** είναι ο τύπος της μεθόδου.

Στο Σχήμα 13.7β εμφανίζεται το διάγραμμα κλάσης UML για τη κλάση rectangle, με τρεις μεταβλητές-μέλη από τις οποίες οι δύο είναι ιδιωτικές, και τέσσερις μεθόδους από τις οποίες οι δύο είναι μέθοδοι δόμησης.



Σχήμα 13.7 *Διάγραμμα κλάσης UML*

Ασκήσεις Κεφαλαίου 13

13.1 Δίνεται η ακόλουθη κλάση: ★

```
class stoixeia
{
   int ilikia;
public:
   string onoma;
   float varos;
   void set(int a)
   {
      ilikia=a;
   }
};
```

Γράψτε ένα πρόγραμμα το οποίο να δημιουργεί ένα αντικείμενο της παραπάνω κλάσης και να καταχωρίζει στις μεταβλητές-μέλη του τα δικά σας στοιχεία (ηλικία, όνομα, βάρος). ★

13.2 Τι πρόβλημα θα μας δημιουργήσει η χρήση της παρακάτω κλάσης: ★

```
class circle
{
   float aktina;
   circle(){aktina=0;}
   float emvado(){return 3.14*aktina*aktina;}
};
```

13.3 Δίνεται η παρακάτω κλάση:

```
class test
{
   int ar;
public:
   void show() {cout << ar << endl;}
};
```

Να συμπληρωθεί κατάλληλα, ώστε:

- όταν δημιουργείται ένα νέο αντικείμενο, να καταχωρίζει στο μέλος ar έναν τυχαίο αριθμό και να εμφανίζει τη φράση «Δημιουργήθηκε το #» όπου το # είναι ο τυχαίος αριθμός.
- Όταν καταστρέφεται ένα αντικείμενο, να εμφανίζει τη φράση «Καταστράφηκε το #», όπου το # είναι ο αριθμός που περιέχεται στο αντικείμενο. ★★

13.4 Ποια από τα επόμενα αληθεύουν: ★

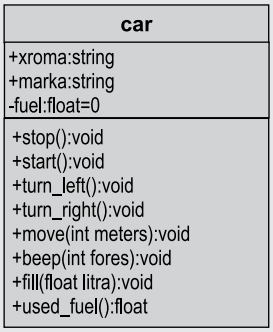
- ❑ Τα ιδιωτικά μέλη μιας κλάσης είναι προσπελάσιμα μόνο από τις μεθόδους της κλάσης.
- ❑ Με τη δήλωση μιας κλάσης δεσμεύεται χώρος μνήμης για τις μεταβλητές-μέλη της κλάσης.
- ❑ Όταν έχουμε δύο αντικείμενα της ίδιας κλάσης, μπορούμε να καταχωρίσουμε όλα τα στοιχεία του ενός στο άλλο με τον τελεστή =.
- ❑ Αν μια κλάση δεν έχει μέθοδο δόμησης, δεν μπορούν να δημιουργηθούν αντικείμενα σε αυτή την κλάση.
- ❑ Μια κλάση μπορεί να έχει πολλές μεθόδους δόμησης.
- ❑ Κάθε αντικείμενο της κλάσης έχει το δικό του αντίγραφο κώδικα για κάθε μέθοδο της κλάσης.

13.5 Εντοπίστε τα λάθη στο παρακάτω πρόγραμμα: ★

```
#include <iostream>
using namespace std;

class circle
{
   float aktina;
public
   circle(float r);
   float emvado();
}
float circle::emvado()
{
   return pi*aktina*aktina;
}
void circle::set_r(float r)
{
   aktina=r;
}
circle::circle(float r)
{
   aktina=r;
}

int main()
{
   circle cir1,cir2(20);
   cir1.aktina=12;
   return 0;
}
```

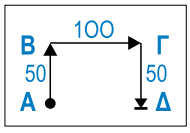
13.15 Να σχεδιάσετε τα διαγράμματα κλάσεων UML για τις κλάσεις που εμφανίζονται στις ασκήσεις 13.1, 13.6, 13.7 και 13.10. ★★

13.16 Στο διπλανό διάγραμμα κλάσης UML εμφανίζονται τα χαρακτηριστικά και οι μέθοδοι μιας κλάσης car η οποία θα χρησιμοποιηθεί για την προσομοίωση της λειτουργίας αυτοκινήτων. Το μέλος xroma προσδιορίζει το χρώμα ενός αυτοκινήτου, το μέλος marka τη μάρκα του και το μέλος fuel την ποσότητα (σε λίτρα) της βενζίνης που διαθέτει το ντεπόζιτό του. Η μέθοδος start() βάζει μπροστά τη μηχανή ενώ η stop() τη σβήνει. Η μέθοδος turn_left() στρίβει το αυτοκίνητο 90° αριστερά, ενώ η turn_right() το στρίβει 90° δεξιά. Η μέθοδος move() αναγκάζει το αυτοκίνητο να κινηθεί μπροστά τόσα μέτρα όση είναι η τιμή της παραμέτρου της. Η μέθοδος beep() αναγκάζει το αυτοκίνητο να κορνάρει τόσες φορές όση είναι η τιμή της παραμέτρου της. Η μέθοδος fill() προσθέτει στο ντεπόζιτο του αυτοκινήτου τόσα λίτρα βενζίνης όση είναι η τιμή της παραμέτρου της. Τέλος, η μέθοδος used_fuel() επιστρέφει ως τιμή την ποσότητα (σε λίτρα) της βενζίνης που κατανάλωσε το αυτοκίνητο: ★★★



Να γραφεί μόνο ο απαραίτητος κώδικας ο οποίος:

- Να δημιουργεί ένα αντικείμενο της κλάσης car με όνομα που θα αποφασίσετε εσείς.
- Να ορίζει ως χρώμα του το *κόκκινο*, ως μάρκα το *Toyota* και ως χωρητικότητα ντεπόζιτου τα 50 λίτρα βενζίνης.
- Τώρα θα πρέπει να γράψετε τις κατάλληλες προτάσεις ώστε το αυτοκίνητό μας να κάνει τη διαδρομή που φαίνεται στο διπλανό σχήμα. Στο σημείο **Α** θα το βάλουμε μπροστά και στο σημείο **Δ** θα το σβήσουμε. Οι αριθμοί δείχνουν τις αποστάσεις σε μέτρα μεταξύ των σημείων. Επίσης, στο σημείο **Β** το αυτοκίνητο πρέπει να κορνάρει 2 φορές και στο σημείο **Γ** μία φορά.



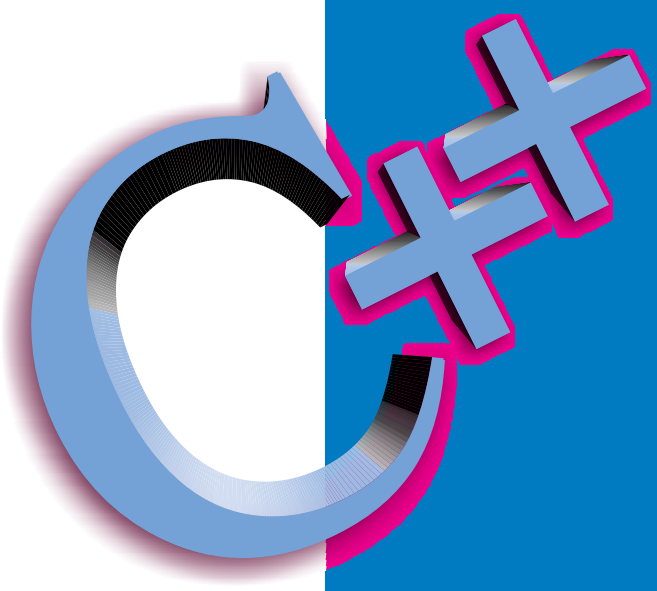
- Τέλος, να γραφεί μια πρόταση η οποία, χρησιμοποιώντας την κατάλληλη μέθοδο, να εμφανίζει στην οθόνη τα λίτρα της βενζίνης που κατανάλωσε το αυτοκίνητο.

ΠΡΟΣΟΧΗ: *Δεν* πρέπει να γράψετε ένα ολοκληρωμένο πρόγραμμα, *ούτε* να φτιάξετε την κλάση car, αλλά να αναφέρετε *μόνο* τις απαραίτητες προτάσεις για να υλοποιηθούν τα παραπάνω.

Κεφάλαιο

14

Επιπρόσθετα θέματα κλάσεων και αντικειμένων



Μεταβίβαση αντικειμένων σε συναρτήσεις

Η μεταβίβαση ενός αντικειμένου σε μια συνάρτηση γίνεται με ακριβώς τον ίδιο τρόπο όπως με οποιονδήποτε άλλο τύπο μεταβλητής.

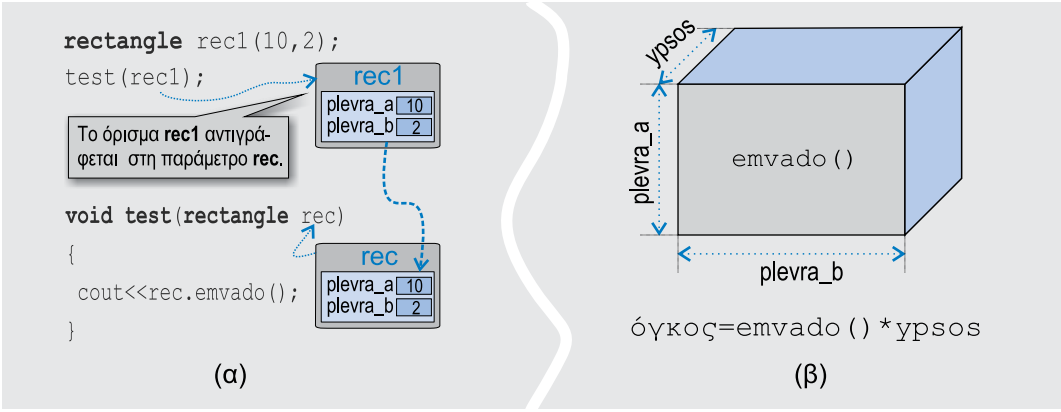
Μεταβίβαση κατ' αξία

Ο προκαθορισμένος τρόπος μεταβίβασης παραμέτρων στη C++ είναι κατ' αξία (by value).

> Όταν μεταβιβάζουμε ένα αντικείμενο σε μια συνάρτηση, η αντίστοιχη παράμετρος πρέπει να είναι ένα αντικείμενο της ίδιας κλάσης. Οι τιμές των μεταβλητών-μελών του ορίσματος αντιγράφονται στις αντίστοιχες μεταβλητές-μέλη της παραμέτρου.

Η διαδικασία μεταβίβασης είναι παρόμοια με τη μεταβίβαση μεταβλητών τύπου δομής σε συναρτήσεις, όπως αναφέρθηκε αναλυτικά στο προηγούμενο κεφάλαιο. Το Σχήμα 14.1α δείχνει εποπτικά τη διαδικασία μεταβίβασης του ορίσματος rec1 στην παράμετρο rec της συνάρτησης test(). Οι τιμές των μεταβλητών-μελών του ορίσματος rec1 αντιγράφονται στις αντίστοιχες μεταβλητές-μέλη της παραμέτρου rec.

Ας υποθέσουμε τώρα ότι θέλουμε να υπολογίζουμε τον όγκο ενός ορθογώνιου παραλληλεπιπέδου όπου η μία του πλευρά αποτελείται από ένα αντικείμενο κλάσης rectangle με διαστάσεις plevra_a και plevra_b (Σχήμα 14.1β). Ο όγκος του ορθογώνιου παραλληλεπιπέδου υπολογίζεται από τον πολλαπλασιασμό του εμβαδού αυτής της πλευράς του επί το ύψος.



Σχήμα 14.1 *Μεταβίβαση αντικειμένων σε συναρτήσεις*

Στην παρακάτω συνάρτηση ogos(), μεταβιβάζεται ένα αντικείμενο-ορθογώνιο παραλληλόγραμμο, το οποίο αντιστοιχεί στη μια πλευρά του παραλληλεπιπέδου, καθώς και το ύψος του. Η συνάρτηση επιστρέφει ως τιμή τον όγκο του παραλληλεπιπέδου.

```
........
float ogos(rectangle rec, float ypsos)
{
   float ogos_rec;
   ogos_rec = rec.emvado()*ypsos;
   return ogos_rec;
}

int main()
{
   rectangle rec1(10,2);
   cout << "Εμβαδό παραλληλογράμμου = "
        << rec1.emvado() << endl;
   cout << "Όγκος παραλληλεπιπέδου με ύψος 5 = "
        << ogos(rec1,5) << endl;
   return 0;
}
```

Στην παράμετρο **rec** θα αντιγραφεί το αντικείμενο του αντίστοιχου ορίσματος κατά την κλήση της συνάρτησης.

Υπολογίζεται ο όγκος του παραλληλεπιπέδου με την εφαρμογή της μεθόδου **emvado()** στο αντικείμενο της παραμέτρου **rec**.

Δημιουργείται ένα αντικείμενο-ορθογώνιο παραλληλόγραμμο **rec1** με διαστάσεις 10x2.

Το αντικείμενο **rec1** αντιγράφεται στην αντίστοιχη παράμετρο **rec** της συνάρτησης **ogos()**.

```
Εμβαδό παραλληλογράμμου = 20
Όγκος παραλληλεπιπέδου με ύψος 5 = 100
```

Η κλήση της συνάρτησης ogos(rec1,5) με ορίσματα rec1 και 5 έχει ως αποτέλεσμα το αντικείμενο-όρισμα rec1 να αντιγραφεί στο αντικείμενο rec που αποτελεί την αντίστοιχη τυπική παράμετρο της συνάρτησης. Οι τιμές των μεταβλητών-μελών plevra_a και plevra_b του αντικειμένου rec1 αντιγράφονται στις αντίστοιχες μεταβλητές-μέλη του αντικειμένου rec. Επίσης, στην τυπική παράμετρο ypsos μεταβιβάζεται η τιμή του δεύτερου ορίσματος, το 5.

Στην παραπάνω έκδοση της συνάρτησης ogos(), αφού πρώτα υπολογιστεί ο όγκος του παραλληλεπιπέδου, με την εφαρμογή της μεθόδου set_ab() αλλάζουν οι διαστάσεις του αντικειμένου rec σε 7x7.

```
float ogos(rectangle rec, float ypsos)
{
   float ogos_rec;
   ogos_rec = rec.emvado()*ypsos;
   rec.set_ab(7,7);
   return ogos_rec;
}
```

Η αλλαγή στις διαστάσεις του αντικειμένου **rec** δεν επηρεάζει το αντικείμενο-όρισμα με το οποίο καλείται η συνάρτηση.

Μια μέθοδος δόμησης αντιγράφου χρησιμοποιείται για να αρχικοποιήσει ένα νέο αντικείμενο (δηλαδή να του δώσει αρχική τιμή) από ένα υπάρχον αντικείμενο.

Η C++ μας δίνει τη δυνατότητα να ορίσουμε τις δικές μας μεθόδους δόμησης αντιγράφου για μια κλάση.

Επειδή, όπως αναφέρθηκε προηγουμένως, η χρήση μιας εξειδικευμένης μεθόδου δόμησης αντιγράφου έχει νόημα συνήθως όταν έχουμε δυναμική κατανομή μνήμης σε ένα αντικείμενο, θα εξετάσουμε τις μεθόδους αυτές στο Κεφάλαιο 21, το οποίο αναφέρεται στη δυναμική διαχείριση μνήμης από τη C++.

Στατικά μέλη μιας κλάσης

Τόσο οι μεταβλητές-μέλη όσο και οι μέθοδοι μιας κλάσης μπορούν να δηλωθούν ως *στατικές* με το προσδιοριστικό static στην αρχή της πρότασης δήλωσής τους.

Τα στατικά μέλη μιας κλάσης διαφοροποιούνται από τα υπόλοιπα ως προς τον τρόπο με τον οποίο κατανέμεται μνήμη για αυτά, αλλά και στον τρόπο χρήσης τους.

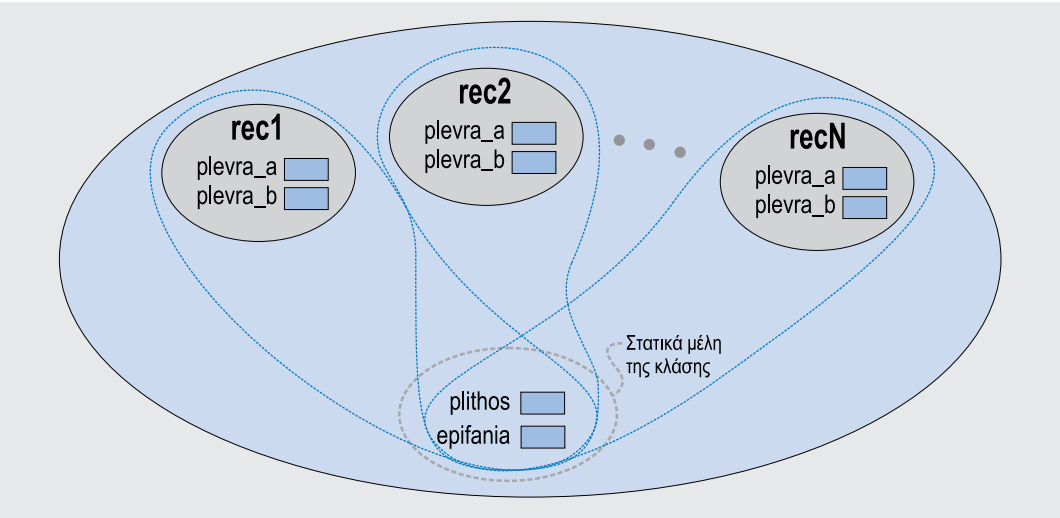
Στατικές μεταβλητές-μέλη

Οι στατικές μεταβλητές-μέλη μιας κλάσης δεσμεύουν μνήμη μόνο μία φορά και είναι «κοινόχρηστες» σε όλα τα αντικείμενα αυτής της κλάσης. Οι στατικές μεταβλητές-μέλη μπορεί να είναι είτε δημόσιες είτε ιδιωτικές.

Η δήλωση μιας στατικής μεταβλητής-μέλος μέσα στην κλάση δεν συνεπάγεται και τη δημιουργία της. Η μεταβλητή-μέλος πρέπει να ξαναδηλωθεί εκτός της κλάσης (ως καθολική μεταβλητή της κλάσης) με χρήση του τελεστή επίλυσης εμβέλειας (::), ώστε να γίνει σαφές για ποια κλάση ορίζεται.

Στις στατικές μεταβλητές-μέλη αποθηκεύουμε συνήθως καθολικές πληροφορίες που αφορούν όλα τα αντικείμενα της κλάσης.

Ας υποθέσουμε ότι στην κλάση rectangle θέλουμε να τηρούμε το πλήθος των αντικειμένων-ορθογωνίων παραλληλογράμμων που έχουμε δημιουργήσει, αλλά και τη συνολική τους επιφάνεια. Δεν θα είχε νόημα αν χρησιμοποιούσαμε δύο κανονικές μεταβλητές-μέλη, αφού αυτές θα επαναλαμβάνονταν σε κάθε αντικείμενο. Για αυτό τον σκοπό δηλώνουμε τις δύο στατικές μεταβλητές-μέλη plithos και epifania αντίστοιχα. Οι μεταβλητές αυτές καταλαμβάνουν χώρο στη μνήμη *μόνο μία φορά* για ολόκληρη την κλάση, και είναι «κοινόχρηστες» από όλα τα αντικείμενα της κλάσης (βλέπε Σχήμα 14.3).



Σχήμα 14.3 *Στατικά μέλη μιας κλάσης*

Κάθε αντικείμενο της κλάσης rectangle έχει πρόσβαση στα στατικά μέλη plithos και epifania, όπως και στις κανονικές μεταβλητές-μέλη. Η διαφορά είναι ότι με τις προτάσεις rec1.plithos και rec2.plithos αναφερόμαστε στην ίδια θέση μνήμης της στατικής μεταβλητής-μέλους plithos που είναι κοινή για όλη την κλάση. Στο πρόγραμμα που ακολουθεί φαίνεται η υλοποίηση της παραπάνω ιδέας. Η κλάση rectangle αποκτά τα δύο δημόσια στατικά-μέλη plithos και epifania. Οι μέθοδοι δόμησης και αποδόμησης τροποποιούνται κατάλληλα ώστε να ενημερώνουν ανάλογα αυτές τις στατικές μεταβλητές-μέλη.

14_rectangle_static.cpp

```
#include <iostream>
#include <string>
using namespace std;
class rectangle
{
   float plevra_a;
   float plevra_b;
public:
   string xroma;
   static int plithos;
   static float epifania;
   rectangle(float a, float b);
   rectangle();
   ~rectangle();
   float emvado() {return plevra_a * plevra_b;}
   void set_ab(float a, float b) {plevra_a=a; plevra_b=b;}
};
rectangle::rectangle(float a, float b)
```

Δήλωση των στατικών μεταβλητών-μελών της κλάσης. Οι μεταβλητές αυτές θα ξαναδηλωθούν και εκτός της κλάσης **rectangle** ως καθολικές μεταβλητές της κλάσης και μόνον τότε δεσμεύεται η μνήμη που χρησιμοποιούν.

Η κλάση string με μια δεύτερη ματιά

Από τα πρώτα ήδη κεφάλαια έχουμε γνωρίσει τη κλάση string και χρησιμοποιήσαμε αντικείμενά της για την αποθήκευση συμβολοσειρών. Τώρα όμως που γνωρίσαμε τις βασικές έννοιες των κλάσεων και αντικειμένων, μπορούμε να μελετήσουμε τη κλάση string πιο αναλυτικά.

> Η κλάση string διατίθεται από την καθιερωμένη βιβλιοθήκη της C++ (STL) και δίνει τη δυνατότητα στη C++ να χειρίζεται τις συμβολοσειρές ως αντικείμενα.

Ένα αντικείμενο της κλάσης string μπορεί να αποθηκεύσει σύνολα χαρακτήρων με δυναμικό τρόπο. Το μέγεθος ενός τέτοιου αντικειμένου καθορίζεται κατά το στάδιο της εκτέλεσης (run time), σε αντίθεση με τις κλασικές συμβολοσειρές της C (C-strings) το μέγεθος των οποίων καθορίζεται στο στάδιο της μεταγλώττισης (compile time). Επιπλέον, με τα αντικείμενα της κλάσης string μπορούν να χρησιμοποιηθούν οι τελεστές ανάθεσης, σύγκρισης και πρόσθεσης, με αποτέλεσμα ο χειρισμός τους να γίνεται πολύ πιο εύκολος.

Στον κώδικα που ακολουθεί δηλώνονται τρία αντικείμενα της κλάσης string στα οποία καταχωρίζονται σύνολα χαρακτήρων με διαφορετικούς τρόπους:

```
#include <string>
.....
string s1 = "Η γλώσσα";
string s2,s3;
s2 = "C++ σε βάθος";
s3 = s1 + " " + s2;
cout << s3 << endl;
.....
```

Δημιουργεί ένα αντικείμενο **s1** κλάσης **string** στο οποίο καταχωρίζει τη συμβολοσειρά "Η γλώσσα".

Δημιουργεί δύο αντικείμενα κλάσης **string**, **s2** και **s3.**

Καταχωρίζει στο αντικείμενο **s2** τη συμβολοσειρά "C++ σε βάθος". Στο αντικείμενο **s3** καταχωρίζεται η συνένωση του **s1,** ενός κενού διαστήματος και του **s2**.

Στο αντικείμενο s1 καταχωρίζεται τιμή στην πρόταση δήλωσης του, ενώ στο αντικείμενο s2 καταχωρίζεται τιμή με τον τελεστή ανάθεσης =, όπως και στο αντικείμενο s3. Ο τελεστής + επιτελεί συνένωση των συνόλων χαρακτήρων. Επομένως, η παράσταση s1 + " " + s2 θα έχει ως αποτέλεσμα το σύνολο χαρακτήρων "Η γλώσσα C++ σε βάθος" το οποίο τελικά καταχωρίζεται στο αντικείμενο s3.

Η καταχώριση αρχικής τιμής σε ένα αντικείμενο της κλάσης string μπορεί να γίνει και με κλήση της μεθόδου δόμησης της κλάσης. Η παρακάτω πρόταση, δημιουργεί ένα αντικείμενο s1 και ταυτόχρονα καλεί τη μέθοδο δόμησης της κλάσης με παράμετρο ένα σύνολο χαρακτήρων:

```
string s1("Η γλώσσα");
```

Η κλήση αυτή έχει ως αποτέλεσμα την καταχώριση των χαρακτήρων της παραμέτρου στο αντικείμενο s1.

Πέρα από τον τελεστή +, μπορούν να χρησιμοποιηθούν και οι τελεστές σύγκρισης, μέσω των οποίων συγκρίνονται, με απόλυτη αλφαβητική σειρά, οι αποθηκευμένες συμβολοσειρές στα αντικείμενα της κλάσης.

Για παράδειγμα, το παρακάτω τμήμα κώδικα:

```
string s1 = "δεζ", s2 = "αβγδε";
if (s1>s2) cout<<"NAI"<<endl;
if (s1=="δεζ") cout<<"NAI"<<endl;
```

```
NAI
NAI
```

συγκρίνει τα περιεχόμενα των αντικειμένων s1 και s2. Και στις δύο περιπτώσεις το αποτέλεσμα των λογικών παραστάσεων είναι η τιμή true, οπότε εμφανίζει τη λέξη NAI στην οθόνη. Στη συγκεκριμένη περίπτωση, το περιεχόμενο του s2 είναι αλφαβητικά μεγαλύτερο από το περιεχόμενο του s1.

Η χρήση των τελεστών ανάθεσης, πρόσθεσης και σύγκρισης, από αντικείμενα της κλάσης string, είναι εφικτή διότι οι τελεστές αυτοί έχουν **υπερφορτωθεί** για αυτή την κλάση ώστε να επιτελούν τις συγκεκριμένες λειτουργίες. Την υπερφόρτωση τελεστών θα τη συναντήσουμε στο επόμενο κεφάλαιο.

Διαχείριση αντικειμένων της κλάσης string

Η διαχείριση των αντικειμένων της κλάσης string γίνεται μέσω των μεθόδων αυτής της κλάσης. Στον πίνακα που ακολουθεί, αναφέρονται διάφορες λειτουργίες που αφορούν αντικείμενα της κλάσης string και τον τρόπο με τον οποίο υλοποιούνται. Υποθέτουμε ότι ισχύουν οι παρακάτω δηλώσεις:

```
string s, s1, s2;  // αντικείμενα της κλάσης string
char c, *cs;       // το cs είναι δείκτης σε συμβολοσειρά C-string π.χ. ένας πίνακας χαρακτήρων
int i, start, len, newSize;
bool b;
.....
```

Τύπος[1]	Σύνταξη	Περιγραφή
Δημιουργία αντικειμένων κλάσης string		
	`string s;`	Δημιουργεί ένα αντικείμενο **s** κλάσης **string**.
	`string s(s1);`	Δημιουργεί ένα αντικείμενο **s** κλάσης **string**, με αρχική τιμή το **string s1**.
	`string s(cs);` `string s("C++");`	Δημιουργεί ένα αντικείμενο **s** κλάσης **string**, με αρχική τιμή τη συμβολοσειρά της C **cs** ή τη συμβολοσειρά "C++".

[1] Προσδιορίζει τον τύπο δεδομένων που επιστρέφει η παράσταση. Για παράδειγμα, η ένδειξη **int** υποδεικνύει ότι επιστρέφει τιμή τύπου **int,** ενώ η ένδειξη ***char** ένα δείκτη σε συμβολοσειρά, δηλαδή **C-string**. Παράλειψη του τύπου σημαίνει ότι είτε δεν επιστρέφεται κάποια τιμή είτε ότι δεν έχει ιδιαίτερη χρησιμότητα.

Καταχώριση, αλλαγή στοιχείων		
	`s1 = s2;`	Καταχωρίζει το **s2** στο **s1**.
	`s1 = cs;`	Καταχωρίζει τη συμβολοσειρά της C **cs** στο **s1**.
	`s1 = c;`	Καταχωρίζει τον χαρακτήρα **c** στο **s1**.
	`s[i] = c;`	Καταχωρίζει τον χαρακτήρα **c** στην **i** θέση του συνόλου χαρακτήρων του **s**. Η πρώτη θέση ενός συνόλου χαρακτήρων είναι η θέση 0.
	`s.at(i) = c;`	Όπως η προηγούμενη, με τη διαφορά ότι στην περίπτωση που ο δείκτης **i** είναι εκτός των ορίων του string **s**, δημιουργεί την εξαίρεση **out_of_range**.
	`s.append(s2);`	Προσθέτει το string **s2** στο τέλος του **s**. Ίδιο αποτέλεσμα με την πρόταση **s = s + s2**;
	`s.append(cs);`	Προσθέτει το string **cs** στο τέλος του **s**. Ίδιο αποτέλεσμα με την πρόταση **s = s + cs**;
	`s.assign(s2,start,len);`	Καταχωρίζει στο string **s**, **len** χαρακτήρες από το **s2** ξεκινώντας από τη θέση **start**.
	`s.clear();`	Διαγράφει όλους τους χαρακτήρες από το **s**.
	`s.replace(start,len,s1);`	Αντικαθιστά **len** χαρακτήρες του string **s**, ξεκινώντας από τη θέση **start**, με τους χαρακτήρες του **s1**.
	`s.insert(start,s2)`	Παρεμβάλλει τους χαρακτήρας του **s2** μέσα στο **s** ξεκινώντας από τη θέση **start**.
	`s.erase(start,len);`	Απομακρύνει **len** χαρακτήρες του string **s**, ξεκινώντας από τη θέση **start**.
	`s.erase(start);`	Απομακρύνει όλους τους χαρακτήρες του string **s**, ξεκινώντας από τη θέση **start**.
	`s1.swap(s2)`	Αντιμεταθέτει τα περιεχόμενα των **s1** και **s2**.
Προσπέλαση		
`*char`	`s.c_str();`	Επιστρέφει τους ίδιους χαρακτήρες όπως το string **s**, τερματισμένους με τον χαρακτήρα **null**. Δηλαδή η αντίστοιχη συμβολοσειρά της C.
`string`	`s.substr(start,len);`	Επιστρέφει **len** χαρακτήρες από το **s** ξεκινώντας από τη θέση **start**. Στην περίπτωση που παραληφθεί η δεύτερη παράμετρος επιστρέφει όλους τους χαρακτήρες από τη θέση **start** μέχρι το τέλος.
`char`	`s[i];`	Επιστρέφει τον χαρακτήρα της **i** θέσης του **s**.
`char`	`s.at(i);`	Επιστρέφει τον χαρακτήρα της **i** θέσης, με τη διαφορά ότι, στην περίπτωση που ο δείκτης **i** είναι εκτός των ορίων του string **s**, δημιουργεί την εξαίρεση **out_of_range**.

Μέγεθος		
`int`	`s.length();`	Επιστρέφει το πλήθος των χαρακτήρων του αντικειμένου **s**.
`int`	`s.size();`	Ίδια λειτουργία όπως η **s.length()**.
`int`	`s.capacity();`	Επιστρέφει το πλήθος χαρακτήρων που μπορεί να αποθηκεύσει το string **s** χωρίς ανακατανομή μνήμης.
`bool`	`s.empty();`	Επιστρέφει τιμή true αν το αντικείμενο **s** δεν περιέχει κανένα χαρακτήρα.
`int`	`s.resize(newSize,c);`	Αλλάζει το μέγεθος του **s** σε **newsize**, συμπληρώνοντας τα κενά (αν χρειάζεται) με τον χαρακτήρα **c**.
Εύρεση – Επιστρέφει ως τιμή τη θέση του εντοπισμού. Σε περίπτωση μη εντοπισμού επιστρέφει τιμή string::npos		
`int`	`s.find(c);`	Εντοπίζει την πρώτη εμφάνιση του χαρακτήρα **c** μέσα στο **s**.
`int`	`s.find(s1);`	Εντοπίζει την πρώτη εμφάνιση του συνόλου χαρακτήρων **s1** μέσα στο **s**.
`int`	`s.rfind(s1);`	Εντοπίζει την τελευταία εμφάνιση του συνόλου χαρακτήρων **s1** μέσα στο **s**.
`int`	`s.find_first_of(s1);`	Εντοπίζει τον πρώτο χαρακτήρα του **s** ο οποίος περιλαμβάνεται στο σύνολο χαρακτήρων του **s1**.
`int`	`s.find_first_not_of(s1);`	Εντοπίζει τον πρώτο χαρακτήρα του **s** ο οποίος δεν περιλαμβάνεται στο σύνολο χαρακτήρων του **s1**.
`int`	`s.find_last_of(s1);`	Εντοπίζει τον τελευταίο χαρακτήρα του **s** ο οποίος περιλαμβάνεται στο σύνολο χαρακτήρων του **s1**.
`int`	`s.find_last_not_of(s1);`	Εντοπίζει τον τελευταίο χαρακτήρα του **s** ο οποίος δεν περιλαμβάνεται στο σύνολο χαρακτήρων του **s1**.
Σύγκριση		
`int`	`s1.compare(s2);`	Συγκρίνει αλφαβητικά τα string **s1** και **s2**. Επιστρέφει -1 εάν **s1<s2**, 0 εάν **s1==s2**, 1 εάν **s1>s2**.
`bool`	`s1 == s2` επίσης: `> < >= <= !=`	Οι τελεστές σύγκρισης έχουν υπερφορτωθεί για την κλάση **string** και μπορούν να χρησιμοποιηθούν για τη σύγκριση αντικειμένων της.
Είσοδος / Έξοδος		
	`cin >> s;`	Καταχωρίζει στο string **s** το σύνολο χαρακτήρων που εισάγεται από το προκαθορισμένο ρεύμα εισόδου (συνήθως το πληκτρολόγιο). Η εισαγωγή τερματίζεται όταν πληκτρολογηθεί «λευκός χαρακτήρας».
	`getline(cin,s);`	Διαβάζει από το προκαθορισμένο ρεύμα εισόδου οτιδήποτε μέχρι τον επόμενο χαρακτήρα αλλαγής γραμμής και το καταχωρίζει στο **s**.
	`cout << s;`	Στέλνει τα περιεχόμενα του **s** στο προκαθορισμένο ρεύμα εξόδου (συνήθως την οθόνη).

Το παρακάτω πρόγραμμα δείχνει τον τρόπο διαχείρισης των αντικειμένων της κλάσης string μέσω των μεθόδων που διαθέτει. Παρατηρήστε το πρόγραμμα με προσοχή ώστε να κατανοήσετε πλήρως τον τρόπο λειτουργίας των μεθόδων που εφαρμόζονται σε αντικείμενα της κλάσης string.

14_string.cpp

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string s("C++ ");
   string s1="123456789", s2;
   int i;
   cout<<"Δώσε το όνομά σου:";
   getline(cin,s1);
   // Εμφανίζει το όνομα που πληκτρολογήθηκε
   cout<<s1<<endl<<"------------"<<endl;
   s2="Η γλώσσα ";
   s2.append("σε βάθος");      // Προσθέτει χαρακτήρες στο τέλος του s2
   cout<<s2<<endl;
   s2.insert(9,s);             // Παρεμβάλει τα περιεχόμενα του s στην 9η θέση του s2
   cout<<s2<<endl;
   s1=s2.substr(0,14);         // Καταχώριση των 14 πρώτων χαρακτήρων του s2 στο s1
   cout<<s1<<endl;
   s1.erase(12);               // Διαγραφή των χαρακτήρων του s1 από τη θέση 12 μέχρι τέλος
   cout<<s1<<endl;
   cout<<s1.capacity()<<endl; // Εμφανίζει τη χωρητικότητα του s1
   cout<<s1.size()<<endl;      // Εμφανίζει το πλήθος χαρακτήρων του s1
   // Μεταβάλει το μέγεθος του s1 σε 20 χαρακτήρες, γεμίζοντας τα κενά με παύλες '-'
   s1.resize(20,'-');
   cout<<s1<<endl;
   i=s2.find(s);               // Εντοπίζει τη θέση του s μέσα στο s2
   cout<<i<<endl;
   // Εντοπίζει τη θέση του πρώτου χαρακτήρα του s2 οποίος δεν υπάρχει στο s1
   i=s2.find_first_not_of(s1);
   cout << s2[i]<<endl;        // Εμφανίζει αυτόν τον χαρακτήρα
   // Εμφανίζει τα περιεχόμενα του s1 και του s2
   cout<<"s1="<<s1<<" "<<"s2="<<s2<<endl;
   s1.swap(s2);                // Αντιμεταθέτει τα περιεχόμενα των s1 και s2
   // Εμφανίζει τα περιεχόμενα του s1 και του s2
   cout<<"s1="<<s1<<" "<<"s2="<<s2<<endl;
   return 0;
}
```

Δημιουργία των αντικειμένων **s**, **s1** και **s2**, της κλάσης **string**. Στα αντικείμενα **s** και **s1** καταχωρίζονται αρχικές τιμές.

Στο αντικείμενο **s1** καταχωρίζεται η γραμμή που πληκτρολογείται, αντικαθιστώντας τα περιεχόμενά του.

Σύνθετες κλάσεις

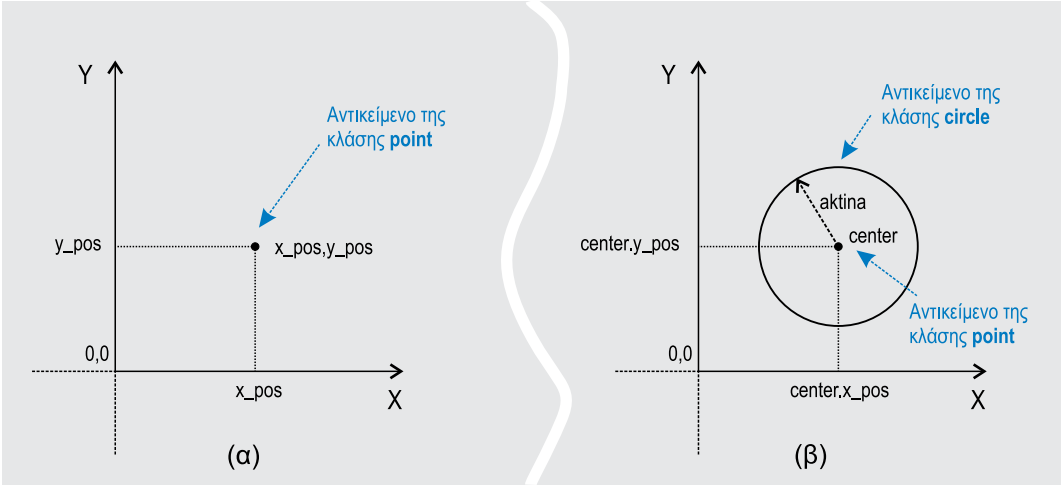
Όπως και στις δομές, έτσι και στις κλάσεις μια μεταβλητή-μέλος μπορεί να είναι ένα αντικείμενο ή πίνακας αντικειμένων μιας οποιασδήποτε άλλης κλάσης. Οι κλάσεις που περιέχουν τέτοιες μεταβλητές-μέλη λέγονται **σύνθετες κλάσεις** (ή *συνθέσεις* – compositions).

Έστω ότι έχουμε την ακόλουθη κλάση point η οποία διαχειρίζεται σημεία καρτεσιανών συντεταγμένων (Σχήμα 14.4α). Στις μεταβλητές μέλη x_pos και y_pos αποθηκεύονται αντίστοιχα οι τιμές των συντεταγμένων X και Y του σημείου.

```
class point
{
public:
   float x_pos;
   float y_pos;
};
```

Η κλάση circle που ακολουθεί διαθέτει μια μεταβλητή-μέλος, την center, η οποία ανήκει στη κλάση point και χρησιμοποιείται για να προσδιορίσει το κέντρο του κύκλου επάνω στο καρτεσιανό επίπεδο (σχήμα 14.4β).

```
class circle
{
public:
   float aktina;
   string xroma;
   point center;
   float emvado() {return 3.14*pow(aktina,2);};
};
```



Σχήμα 14.4 *Σημείο και κύκλος στο καρτεσιανό επίπεδο*

Κάθε αντικείμενο της κλάσης circle περιέχει ένα αντικείμενο της κλάσης point το οποίο προσδιορίζει το κέντρο του κύκλου (Σχήμα 14.5). Η πρόσβαση στα μέλη του «εσωτερικού» αντικειμένου center γίνεται όπως και στις σύνθετες δομές με χρήση του τελεστή τελεία (.) δύο φορές. Για παράδειγμα οι προτάσεις:

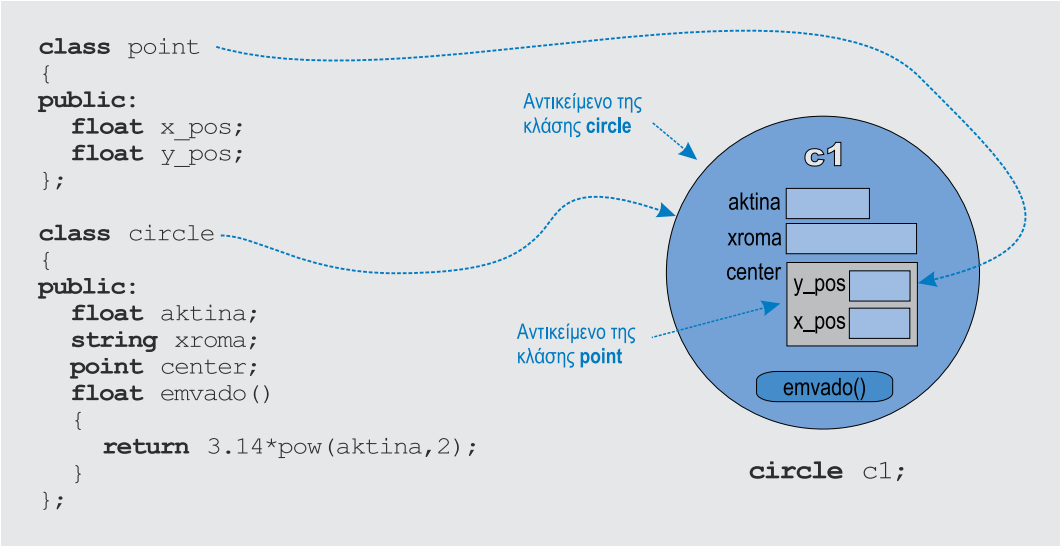
```
c1.center.x_pos=10;
c1.center.y_pos=20;
```

θα καταχωρίσουν ως συντεταγμένες X και Y του κέντρου του αντικειμένου-κύκλου c1 το 10 και το 20 αντίστοιχα.

Όταν δημιουργείται ένα αντικείμενο της κλάσης circle, δημιουργείται και ένα αντικείμενο της κλάσης point και μάλιστα πρώτο! Όταν καταστρέφεται ένα αντικείμενο της κλάσης circle, καταστρέφεται και το αντικείμενο της κλάσης point που περιέχει αλλά σε δεύτερο χρόνο!

Στο πρόγραμμα που ακολουθεί στην επόμενη σελίδα, οι κλάσεις point και circle έχουν διαφοροποιηθεί ώστε να περιλαμβάνουν μεθόδους δόμησης και αποδόμησης. Επίσης, στη κλάση point έχει προστεθεί η μέθοδος move() η οποία μετακινεί ένα σημείο σε μια νέα θέση επάνω στο καρτεσιανό επίπεδο και στην κλάση circle η μέθοδος info() η οποία εμφανίζει τα στοιχεία ενός αντικειμένου-κύκλου.

Μελετήστε με προσοχή το πρόγραμμα και τα αποτελέσματα που εμφανίζει ώστε να κατανοήσετε τη χρονική σειρά με την οποία εφαρμόζονται οι μέθοδοι δόμησης και αποδόμησης των δύο κλάσεων.



Σχήμα 14.5 *Σύνθετες κλάσεις - Αντικείμενα κλάσεων circle και point*

Συσχέτιση κλάσεων

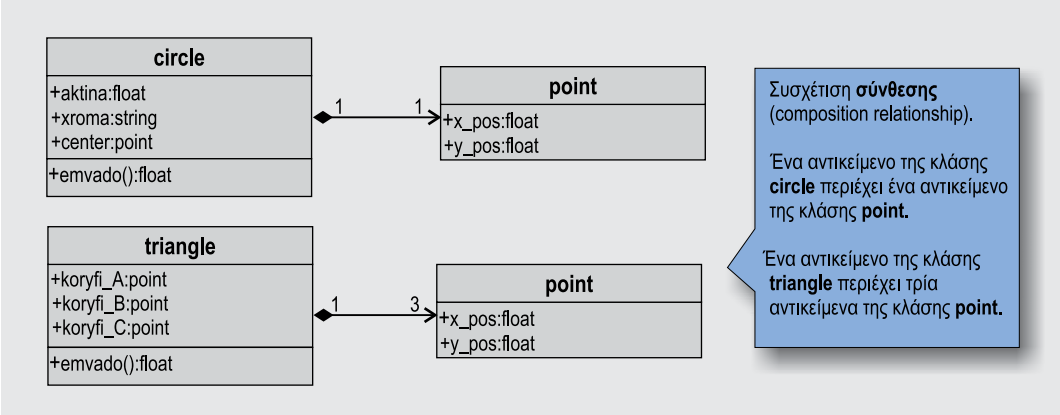
Στον πραγματικό κόσμο, αλλά και στον κόσμο του αντικειμενοστρεφούς προγραμματισμού, τα διάφορα αντικείμενα (άρα και οι κλάσεις) είναι δυνατόν να σχετίζονται με διάφορους τρόπους. Στο προηγούμενο παράδειγμα, οι κλάσεις circle και point σχετίζονται μεταξύ τους. Κάθε αντικείμενο circle εμπεριέχει ένα αντικείμενο point.

Ο τρόπος αυτός συσχέτισης λέγεται *συσχέτιση σύνθεσης* (composition relationship) όπου ένα αντικείμενο αποτελεί τμήμα ενός πιο σύνθετου αντικειμένου. Σε αυτόν τον τρόπο συσχέτισης τα δύο αντικείμενα συνδέονται με σχέση ζωής: Δηλαδή, όταν καταστραφεί ένας αντικείμενο της κλάσης circle δεν έχει νόημα να συνεχίσει να υπάρχει το μέλος-αντικείμενο center της κλάσης point που περιέχει.

> Υπάρχουν διάφοροι τρόποι συσχέτισης κλάσεων οι οποίοι θα αναλυθούν στο Κεφάλαιο 17 αφού θα έχουμε συναντήσει και την κληρονομικότητα η οποία αποτελεί ακόμα έναν τρόπο συσχέτισης.

Απεικόνιση συσχέτισης σύνθεσης με UML

Η συσχέτιση κλάσεων εμφανίζεται στα διαγράμματα κλάσεων UML ως μια γραμμή οι οποία συνδέει τις συσχετιζόμενες κλάσεις μεταξύ τους. Οι διάφοροι τρόποι συσχέτισης διακρίνονται από τα διαφορετικά σύμβολα τερματισμού αυτής της γραμμής. Στο διάγραμμα κλάσης UML που απεικονίζεται στο Σχήμα 14.6, η γραμμή σύνδεσης μεταξύ των κλάσεων circle και point υποδεικνύει τη συσχέτιση σύνθεσης των δύο κλάσεων. Αν υποθέσουμε ότι ένα τρίγωνο προσδιορίζεται από τις συντεταγμένες των τριών κορυφών του, στο ίδιο σχήμα εμφανίζεται και η συσχέτιση των κλάσεων triangle και point.



Σχήμα 14.6 *Συσχέτιση σύνθεσης (composition relationship) - UML απεικόνιση*

Η συσχέτιση σύνθεσης προσδιορίζεται από τον συμπαγή μαύρο ρόμβο από την πλευρά της περιβάλλουσας κλάσης, δηλαδή της κλάσης που περιέχει το αντικείμενο της άλλης κλάσης (εσωτερική κλάση). Το βέλος στο άλλο άκρο της γραμμής είναι προαιρετικό και υποδεικνύει ότι η σχέση είναι μονόδρομη, δηλαδή η κλάση circle γνωρίζει την ύπαρξη της κλάσης point και εξαρτάται από αυτήν, αλλά δεν ισχύει το αντίστροφο. Οι αριθμοί στα άκρα της γραμμής σύνδεσης είναι προαιρετικοί και υποδεικνύουν τη *πολλαπλότητα* της σχέσης, δηλαδή πόσα αντικείμενα από τη μία κλάση συσχετίζονται με πόσα αντικείμενα της άλλης κλάσης. Συνήθως η πολλαπλότητα μιας σχέσης σύνθεσης είναι 1 από τη πλευρά της περιβάλλουσας κλάσης (whole class) και 1 ή περισσότερα από τη πλευρά της εσωτερικής (part class). Στην περίπτωση της συσχέτισης circle-point η πολλαπλότητα είναι 1 προς 1, διότι κάθε αντικείμενο-κύκλος περιέχει ένα αντικείμενο της κλάσης point, ενώ στη συσχέτιση triangle-point είναι 1 προς 3 διότι κάθε αντικείμενο-τρίγωνο περιέχει τρία αντικείμενα της κλάσης point.

Λίστες αρχικοποίησης

Οι *λίστες αρχικοποίησης* (ή *λίστες καθορισμού αρχικών τιμών* – initialization lists) είναι ένας διαφορετικός τρόπος ανάθεσης αρχικών τιμών στις μεταβλητές-μέλη μιας κλάσης, από τις μεθόδους δόμησής της. Η ακόλουθη κλάση point διαθέτει δύο υπερφορτωμένες μεθόδους δόμησης: την προκαθορισμένη μέθοδο δόμησης (χωρίς παράμετρο) και μία μέθοδο δόμησης με δύο παραμέτρους. Και στις δύο μεθόδους γίνεται ανάθεση αρχικών τιμών στις μεταβλητές μέλη της κλάσης μέσα στο σώμα της μεθόδου, με χρήση του τελεστή ανάθεσης =. Επίσης, η κάθε μέθοδος εμφανίζει ένα αντίστοιχο μήνυμα στην οθόνη.

```
class point
{
public:
   float x_pos;
   float y_pos;
   point();
   point(float x, float y);
};
point::point()
{
   x_pos=0;
   y_pos=0;
   cout<<"Ενα σημείο δημιουργήθηκε στο 0,0"<<endl;
}
point::point(float x, float y)
{
   x_pos=x;
   y_pos=y;
   cout<<"Ενα σημείο δημιουργήθηκε στο "<<x_pos<<","<<y_pos<<endl;
}
```

Ασκήσεις Κεφαλαίου 14

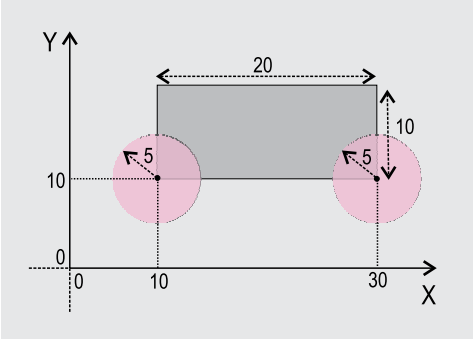
14.1 Ποια από τα επόμενα αληθεύουν; ★

- ❑ Όταν ένα αντικείμενο μεταβιβάζεται σε μια συνάρτηση με αναφορά, τότε η συνάρτηση μπορεί να τροποποιήσει τα δεδομένα του αντικειμένου το οποίο χρησιμοποιήθηκε ως όρισμα κατά τη κλήση της.
- ❑ Για να έχουμε πρόσβαση στα μέλη ενός αντικειμένου μέσω ενός δείκτη σε αυτό, χρησιμοποιούμε τον τελεστή τελείας (.).
- ❑ Κάθε κλάση πρέπει να έχει υποχρεωτικά μια μέθοδο δόμησης αντιγράφου.
- ❑ Η μέθοδος δόμησης αντιγράφου χρησιμοποιείται στην έμμεση δημιουργία αντικειμένων.
- ❑ Μια στατική μεταβλητή-μέλος καταλαμβάνει χώρο στη μνήμη μόνο μία φορά για ολόκληρη την κλάση.
- ❑ Μόνο οι στατικές μέθοδοι έχουν πρόσβαση στις στατικές μεταβλητές-μέλη της κλάσης.
- ❑ Οι φίλιες συναρτήσεις αποτελούν μεθόδους μιας κλάσης.
- ❑ Οι φίλιες συναρτήσεις μιας κλάσης έχουν πρόσβαση στα ιδιωτικά μέλη ενός αντικειμένου της κλάσης.

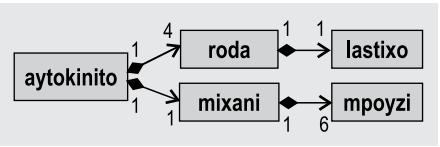
14.2 Τι θα εμφανίσει στην οθόνη το παρακάτω πρόγραμμα; ★★

```
#include <iostream>
using namespace std;
class triangle
{
   float basi;
   float ypsos;
public:
   static int pl;
   triangle(){pl++;}
   triangle(float b, float y){pl++;basi=b;ypsos=y;}
   float emvado(){return basi*ypsos/2;}
   void set(float b, float y){basi=b;ypsos=y;}
   void show(){cout<<"Βάση= "<<basi<<" Υψος="<<ypsos<<endl;}
};
int triangle::pl=0;
void do_it(triangle &tr)
{
   tr.set(10,8);
}
```

14.19 Θέλουμε να διαχειριζόμαστε αντικείμενα αραμπάδες. Κάθε αντικείμενο αποτελείται από δύο κύκλους ίδιας ακτίνας και ένα ορθογώνιο παραλληλόγραμμο με τις δύο κορυφές στα κέντρα τους, όπως εμφανίζεται στο διπλανό σχήμα. Δημιουργήστε μια κλάση arabas η οποία θα περιέχει ένα μέλος τύπου rectangle (για το παραλληλόγραμμο) και δύο μέλη τύπου circle (για τους κύκλους). Οι συντεταγμένες του κέντρου κάθε κύκλου θα πρέπει να προσδιορίζονται από ένα αντικείμενο point. Η θέση του αντικειμένου αραμπά θα προσδιορίζεται από τη θέση του κέντρου του αριστερού κύκλου. Φτιάξτε τις παραπάνω κλάσεις και δημιουργήστε ένα αντικείμενο arabas με τις διαστάσεις και τη θέση που φαίνεται στο σχήμα. Το αντικείμενο θα πρέπει να δημιουργηθεί με τη χρήση μιας μεθόδου δόμησης της κλάσης με πέντε παραμέτρους: Τη θέση του αραμπά (X,Y), το ύψος και το πλάτος του ορθογωνίου παραλληλογράμμου του, και την ακτίνα των κύκλων του. Να κατασκευαστεί επίσης το διάγραμμα κλάσεων UML. ★★★



14.20 Δημιουργήστε τις κατάλληλες κλάσεις ώστε να υλοποιείται το διάγραμμα κλάσεων UML του σχήματος. Επιλέξτε εσείς τα μέλη των κλάσεων με τρόπο ώστε να υλοποιούνται οι συσχετίσεις του διαγράμματος. ★★

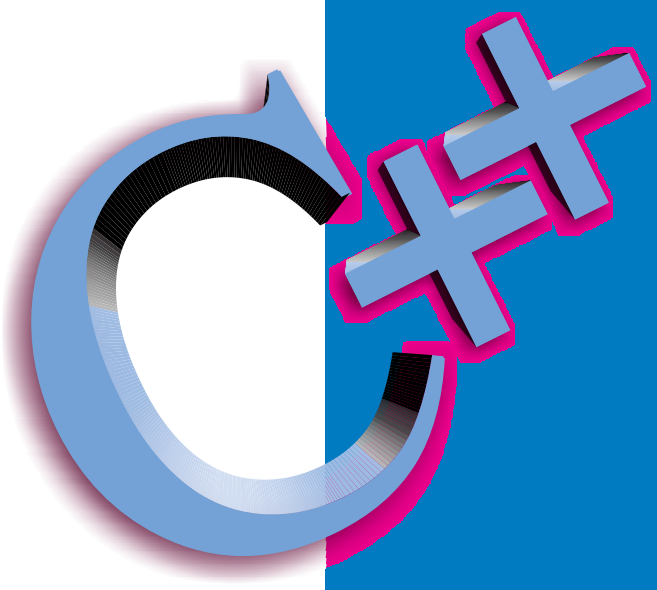


14.21 Θέλουμε να φτιάξουμε ένα παιχνίδι με τράπουλα. Γνωρίζουμε ότι κάθε τράπουλα διαθέτει 52 κάρτες. Να δημιουργήσετε μια κλάση με όνομα trapoula και μια κλάση με όνομα karta. Κάθε αντικείμενο της κλάσης karta θα διαθέτει μεταβλητές-μέλη για τη φιγούρα του (A,2,3,..10,J,Q,K) και το είδος του (Κούπα,Καρό,Μπαστούνι,Σπαθί). Κάθε νέο αντικείμενο της κλάσης trapoula αρχικά θα περιέχει 52 (13x4) κάρτες (αντικείμενα της κλάσης karta) όλων των συμβόλων (13) και όλων των ειδών (4). Αυτό θα υλοποιείται από τη μέθοδο δόμησης της κλάσης. Η κλάση trapoula να διαθέτει μέθοδο για το τυχαίο ανακάτεμα των καρτών και μια μέθοδο για την εμφάνιση του περιεχομένου της στην οθόνη. Φτιάξτε ένα ολοκληρωμένο πρόγραμμα, το οποίο θα δημιουργεί ένα αντικείμενο της κλάσης trapoula, θα ανακατεύει την τράπουλα, και θα εμφανίζει το περιεχόμενό της. Σχεδιάστε επίσης το διάγραμμα κλάσεων UML του προβλήματος. ★★★

Κεφάλαιο

15

Υπερφόρτωση τελεστών



Υπερφόρτωση τελεστών

Η υπερφόρτωση τελεστών (operator overloading) είναι ένα από τα πιο συναρπαστικά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού.

Οι τελεστές που υποστηρίζονται από μια γλώσσα προγραμματισμού εφαρμόζονται σε συγκεκριμένους βασικούς τύπους δεδομένων. Για παράδειγμα, ο τελεστής + χρησιμοποιείται για την πρόσθεση αριθμητικών τιμών και εφαρμόζεται σε δεδομένα τύπου int, float και char (ας μην ξεχνάμε ότι οι χαρακτήρες είναι αριθμοί).

Έστω ότι έχουμε δύο μεταβλητές τύπου int και δύο αντικείμενα της κλάσης rectangle:

```
int a=5, b=12;
rectangle rec1(10,2), rec2(3,5);
```

Μια παράσταση της μορφής a+b είναι απόλυτα αποδεκτή από τον μεταγλωττιστή και θα αποδώσει τιμή 17. Όμως η παράσταση rec1+rec2 δεν είναι αποδεκτή διότι ο τελεστής + δεν μπορεί να χρησιμοποιηθεί παρά μόνο με τους βασικούς τύπους αριθμητικών δεδομένων.

Τα ερωτήματα που τίθενται είναι δύο: Μπορούμε να χρησιμοποιήσουμε τους υπάρχοντες τελεστές σε νέους τύπους δεδομένων; Και αν ναι, πώς θα επενεργούν με το νέο τύπο δεδομένων;

> Η υπερφόρτωση τελεστών μας δίνει τη δυνατότητα να επανακαθορίσουμε τη χρήση ενός τελεστή για ένα νέο τύπο δεδομένων.

Δηλαδή, θα μπορούσαμε να επανακαθορίσουμε τον τελεστή + ώστε, όταν εφαρμόζεται σε αντικείμενα της κλάσης rectangle, να προσθέτει τις αντίστοιχες διαστάσεις των δύο αντικειμένων. Σε αυτή την περίπτωση, η παράσταση rec1+rec2 θα ήταν αποδεκτή, και θα είχε ως αποτέλεσμα ένα αντικείμενο της κλάσης rectangle με διαστάσεις 13 (10+3) και 7 (2+5).

> Όταν υπερφορτώνεται ένας τελεστής, αποκτά διαφορετική σημασία *μόνο* για την κλάση για την οποία υπερφορτώνεται. Διατηρεί την αρχική του σημασία για τους υπόλοιπους τύπους δεδομένων στους οποίους εφαρμόζεται.

Ο ίδιος τελεστής μπορεί να υπερφορτωθεί πολλές φορές για διαφορετικές κλάσεις. Για παράδειγμα, ο τελεστής + θα μπορούσε να έχει διαφορετικό νόημα για την κλάση rectangle, διαφορετικό για την κλάση string κ.ο.κ. Ο μεταγλωττιστής επιλέγει τη σωστή χρήση του τελεστή ανάλογα με την κλάση των αντικειμένων επάνω στα οποία εφαρμόζεται.

Πρέπει να έχουμε υπόψη μας τα παρακάτω σχετικά με την υπερφόρτωση τελεστών στη C++:

- Δεν μπορούν να υπερφορτωθούν όλοι οι τελεστές της C++.
- Δεν μπορούμε να δημιουργήσουμε νέους τελεστές. Πρέπει να χρησιμοποιήσουμε τους υπάρχοντες.
- Δεν μπορούμε να αλλάξουμε την προτεραιότητα των τελεστών.

Στον παρακάτω πίνακα αναφέρονται οι τελεστές της C++ οι οποίοι δεν μπορούν να υπερφορτωθούν.

Τελεστής	Σύμβολο
Ο τελεστής συνθήκης	`?:`
Ο τελεστής πρόσβασης στα μέλη δομών και κλάσεων	`.`
Ο τελεστής απόδοσης μεγέθους sizeof	`sizeof`
Ο τελεστής επίλυσης εμβέλειας	`::`
Ο ειδικός τελεστής αναφοράς σε μέλη κλάσης	`.*`

Υπερφόρτωση τελεστών με χρήση μεθόδων της κλάσης

Ο συνηθέστερος τρόπος υπερφόρτωσης τελεστών είναι με τη χρήση μεθόδων της κλάσης για την οποία υπερφορτώνεται ο τελεστής.

Για να υπερφορτώσουμε έναν τελεστή, πρέπει να χρησιμοποιήσουμε μια ειδική μέθοδο υπερφόρτωσης τελεστή στην οποία καθορίζεται η συμπεριφορά του τελεστή, σε σχέση πάντα με την κλάση για την οποία ορίζεται.

τύπος *κλάση*::operator<*τελεστής*>(*ορίσματα*)

{

Σώμα της μεθόδου, στο οποίο καθορίζεται
η συμπεριφορά του τελεστή σε αυτή κλάση.

}

Όπου:

τύπος Είναι ο τύπος δεδομένων που επιστρέφει η συνάρτηση. Συνήθως ο τύπος αυτός είναι ίδιος με τον τύπο της κλάσης για την οποία υπερφορτώνεται ο τελεστής.

κλάση Είναι η κλάση για την οποία υπερφορτώνεται ο τελεστής.

operator Δεσμευμένη λέξη της C++, προηγείται του συμβόλου του τελεστή.

<*τελεστής*> Το σύμβολο του τελεστή που υπερφορτώνεται, όπως +, > κ.λπ.

ορίσματα Κανένα, ένα, ή περισσότερα ορίσματα, ανάλογα με τον τελεστή που υπερφορτώνεται.

Παρακάτω φαίνεται ο ορισμός της κλάσης rectangle την οποία έχουμε ήδη χρησιμοποιήσει πολλές φορές με διάφορες παραλλαγές. Θα χρησιμοποιήσουμε την κλάση αυτή στα επόμενα παραδείγματα που έχουν σχέση με την υπερφόρτωση τελεστών.

```
class rectangle
{
   float plevra_a;
   float plevra_b;
public:
   string xroma;
   rectangle(float a, float b);
   rectangle();
   void set_ab(float a, float b);
   float emvado();
   void info();
};
void rectangle::set_ab(float a, float b)
{
   plevra_a = a;
   plevra_b = b;
}
float rectangle::emvado()
{
   return plevra_a * plevra_b;
}
void rectangle::info()
{
   cout  << plevra_a << "x" << plevra_b
         << " Χρώμα:" << xroma << endl;
}
rectangle::rectangle(float a, float b)
{
   plevra_a = a;
   plevra_b = b;
   xroma = "";
}
rectangle::rectangle()
{
   plevra_a = 0;
   plevra_b = 0;
   xroma = "";
}
```

Η μέθοδος **set_ab()** αναθέτει τιμές στις μεταβλητές-μέλη **plevra_a** και **plevra_b**.

Η μέθοδος **emvado()** επιστρέφει ως τιμή το εμβαδό του παραλληλογράμμου.

Η μέθοδος **info()** εμφανίζει τα στοιχεία ενός αντικειμένου της κλάσης.

Μέθοδος δόμησης με δύο παραμέτρους. Αναθέτει τις τιμές των παραμέτρων στις μεταβλητές-μέλη των διαστάσεων του παραλληλογράμμου.

Προκαθορισμένη μέθοδος δόμησης (χωρίς παραμέτρους). Θέτει μηδενικές τιμές στις μεταβλητές-μέλη των διαστάσεων του παραλληλογράμμου.

Οι τελεστές χωρίζονται στους διμελείς και τους μονομελείς. Οι διμελείς τελεστές είναι αυτοί που έχουν δύο μέλη, ένα αριστερό και ένα δεξί, ενώ οι μονομελείς έχουν μόνο ένα μέλος. Για παράδειγμα, ο τελεστής διαίρεσης (/) είναι ένας διμελής τελεστής. Στην παράσταση a/b, η μεταβλητή a αποτελεί το αριστερό μέλος του τελεστή, και η μεταβλητή

Υπερφόρτωση μονομελών τελεστών

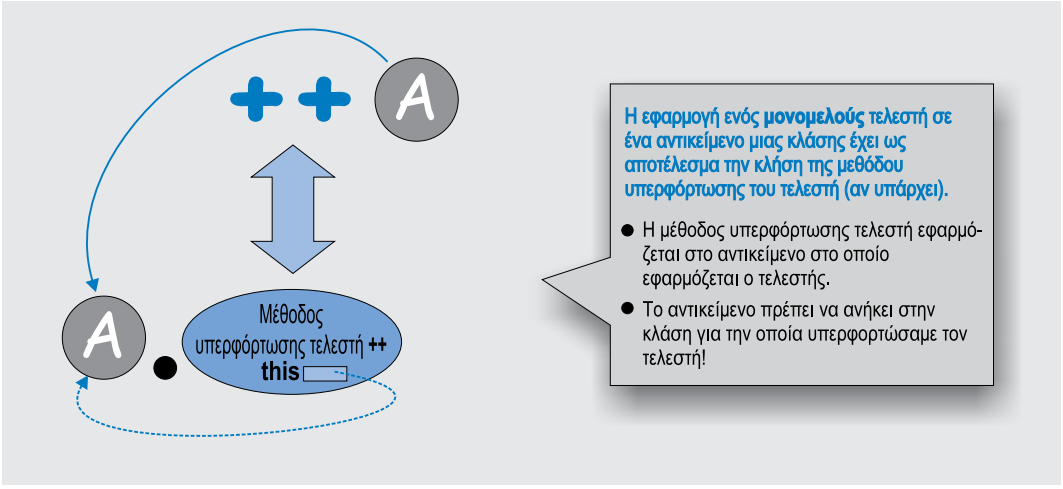
Οι μονομελείς τελεστές είναι αυτοί που έχουν ένα μόνο μέλος είτε στα αριστερά τους είτε στα δεξιά τους. Για παράδειγμα, ο τελεστής αύξησης (++) είναι ένας μονομελής τελεστής ο οποίος μπορεί να έχει μόνο ένα μέλος είτε στα δεξιά του (++a) είτε στα αριστερά του (a++).

Για έναν μονομελή τελεστή, η μέθοδος υπερφόρτωσής του καλείται από το μοναδικό μέλος στο οποίο εφαρμόζεται (Σχήμα 15.3). Η μέθοδος αυτή έχει άμεση πρόσβαση στα μέλη του αντικειμένου στο οποίο εφαρμόζεται. Η μέθοδος διαθέτει επίσης τον δείκτη this ο οποίος δείχνει στο ίδιο το αντικείμενο.

Η υπερφόρτωση των τελεστών ++ και -- έχει μια ιδιαιτερότητα, διότι οι τελεστές αυτοί μπορεί να προηγούνται (προθεματικοί) ή να έπονται (επιθηματικοί) του αντικειμένου που τους καλεί. Ας υποθέσουμε ότι θέλουμε να υπερφορτώσουμε τον τελεστή αύξησης ++ για την κλάση rectangle. Θέλουμε η εφαρμογή του τελεστή να έχει ως αποτέλεσμα την αύξηση των διαστάσεων του αντικειμένου κατά 1. Δηλαδή, οι παραστάσεις της μορφής rec1++ και ++rec1 θα πρέπει να αυξάνουν τις διαστάσεις του αντικειμένου rec1 κατά 1.

Αν θέλουμε επίσης να διατηρήσουμε και την προεπιλεγμένη λειτουργία του τελεστή, θα πρέπει η παράσταση ++rec1 να επιστρέφει ως τιμή το ίδιο αντικείμενο rec1 μετά από την αύξηση και η παράσταση rec1++ το ίδιο αντικείμενο rec1 πριν από την αύξηση.

Επομένως, θα πρέπει να έχουμε δύο εκδόσεις της μεθόδου υπερφόρτωσης του τελεστή ++, μία που θα εκτελείται όταν ο τελεστής είναι προθεματικός και μία όταν είναι επιθηματικός. Και οι δύο εκδόσεις θα επιστρέφουν τιμή τύπου rectangle.



Σχήμα 15.3 *Υπερφόρτωση μονομελών τελεστών*

Άρα, εδώ έχουμε μια περίπτωση υπερφόρτωσης της μεθόδου υπερφόρτωσης τελεστή. Για να μπορέσει ο μεταγλωττιστής να ξεχωρίσει την έκδοση της υπερφορτωμένης μεθόδου που θα καλέσει, ακολουθείται η παρακάτω τεχνική.

Στον ορισμό της υπερφορτωμένης μεθόδου του προθεματικού τελεστή ++, η μέθοδος δεν διαθέτει καμία παράμετρο:

```
rectangle rectangle::operator++()
{
   plevra_a++;
   plevra_b++;
   return *this;
}
```

Καλείται στη περίπτωση προθεματικού τελεστή ++.

Γίνεται αύξηση των διαστάσεων κατά μία μονάδα.

Επιστρέφει το ίδιο αντικείμενο με νέες διαστάσεις.

Στον ορισμό της υπερφορτωμένης μεθόδου του επιθηματικού τελεστή ++, υπάρχει μια παράμετρος τύπου int η οποία δεν χρησιμοποιείται και γι' αυτό δεν χρειάζεται να αναφερθεί το όνομά της:

```
rectangle rectangle::operator++(int)
{
   plevra_a++;
   plevra_b++;
   return *this;
}
```

Καλείται στη περίπτωση επιθηματικού τελεστή ++.

- Και στις δύο περιπτώσεις, ο δείκτης **this** δείχνει στο αντικείμενο που κάλεσε την μέθοδο υπερφόρτωσης τελεστή.
- Η ίδια τεχνική εφαρμόζεται και στην περίπτωση του μονομελή τελεστή μείωσης --.
- Με τον πρώτο τρόπο μπορούν να υπερφορτωθούν και οι υπόλοιποι μονομελείς τελεστές στη C++, αφού όλοι προηγούνται του αντικειμένου επάνω στο οποίο εφαρμόζονται.

Ο παρακάτω κώδικας δείχνει τη χρήση του υπερφορτωμένου τελεστή ++ σε αντικείμενα της κλάσης rectangle.

```
.....
rectangle rec1(10,2), rec2(5,7), rec3;
cout <<"rec1="; rec1.info();
rec1++;
cout <<"rec1="; rec1.info();
rec3 = rec2++;
cout <<"rec3="; rec3.info();
cout <<"rec2="; rec2.info();
.....
```

```
rec1=10x2 Χρώμα:
rec1=11x3 Χρώμα:
rec3=5x7 Χρώμα:
rec2=6x8 Χρώμα:
```

Στο αντικείμενο **rec3** καταχωρίζονται οι παλιές διαστάσεις του **rec2** οι οποίες όμως αυξάνονται κατά 1.

Στο προηγούμενο παράδειγμα, ο τελεστής ++ εκτέλεσε τη συνηθισμένη λειτουργία του, αυξάνοντας κατά 1 τις τιμές των μεταβλητών-μελών των αντικειμένων της κλάσης rectangle.

Παραδείγματα

Π15.1 Στο πρόγραμμα που ακολουθεί, υπερφορτώνεται ο τελεστής πρόσθεσης +, ώστε να επιστρέφει ένα αντικείμενο της κλάσης kafetiera το οποίο θα περιέχει το άθροισμα των αντίστοιχων υλικών των δύο μελών του τελεστή.

15_p15_1.cpp

```
#include <iostream>
using namespace std;
class kafetiera
{
   int kafes;
   int gala;
   int zaxari;
   int nero;
public:
   kafetiera(int k, int g, int z, int n);
   kafetiera(){};
   kafetiera operator+(kafetiera op2);
   void status();
};
kafetiera::kafetiera(int k, int g, int z, int n)
{
   kafes=k;
   gala=g;
   zaxari=z;
   nero=n;
}
void kafetiera::status()
{
   cout<<"==========================="<<endl;
   cout<<"Καφές:"<<kafes<<endl;
   cout<<"Γάλα:"<<gala<<endl;
   cout<<"Ζάχαρη:"<<zaxari<<endl;
   cout<<"Νερό:"<<nero<<endl;
   cout<<"==========================="<<endl;
}
kafetiera kafetiera::operator+(kafetiera op2)
{
   kafetiera temp;
   temp.kafes=kafes+op2.kafes;
   temp.gala=gala+op2.gala;
   temp.zaxari=zaxari+op2. zaxari;
   temp.nero=nero+op2.nero;
   return temp;
}
```

Μέθοδος υπερφόρτωσης του τελεστή +.

Η μέθοδος επιστρέφει ως τιμή το προσωρινό αντικείμενο-καφετιέρα **temp**, το οποίο περιέχει το άθροισμα των υλικών των δύο μελών του τελεστή.

```
int main()
{
   kafetiera kaf1(100,50,200,1000),kaf2(10,20,30,40),kaf3;
   kaf3=kaf1+kaf2;
   kaf3.status();
   return 0;
}
```

Δημιουργία τριών αντικειμένων της κλάσης. Τα αντικείμενα **kaf1** και **kaf2** αποκτούν ποσότητες υλικών σύμφωνα με τα ορίσματα που διαθέτουν.

Το αντικείμενο kaf3 θα αποκτήσει ως τιμές υλικών το άθροισμα των αντίστοιχων υλικών των αντικειμένων kaf1 και kaf2. Η κλήση της μεθόδου status() για το αντικείμενο kaf3 εμφανίζει τις ποσότητες των υλικών του.

```
Καφές=110
Γάλα=70
Ζάχαρη=230
Νερό= 1040
```

Π15.2 Το πρόγραμμα που ακολουθεί χρησιμοποιεί την κλάση ergasia για τη διαχείριση εργασιών φοιτητών, όπως και στο Παράδειγμα 14.3. Κάθε εργασία μπορεί να ανατεθεί σε μέχρι το πολύ πέντε φοιτητές. Η εργασία διαθέτει ημερομηνία παράδοσης του τύπου my_date που έχει δημιουργηθεί για τη διαχείριση ημερομηνιών. Και για τις δύο κλάσεις έχουν υπερφορτωθεί τελεστές με τρόπο που να κάνουν ευκολότερη τη διαχείριση των αντικειμένων τους. Μελετήστε τις μεθόδους και τις συναρτήσεις υπερφόρτωσης, τις επεξηγήσεις των πλαισίων και τα αποτελέσματα που εμφανίζει το πρόγραμμα.

15_p15_2.cpp

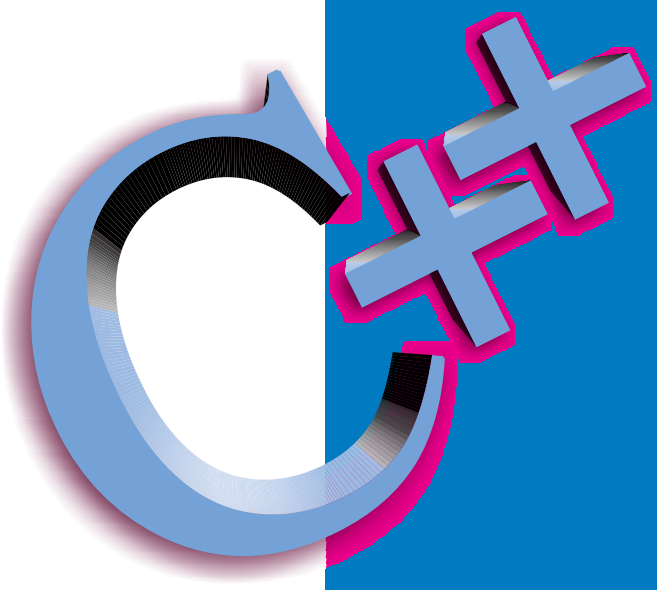
```
#include <iostream>
#include <string>
using namespace std;
class my_date
{
   int hmera;
   int minas;
   int etos;
public:
   void set_hmer(int h, int m, int e);
   void show();
   my_date operator++();
};
void my_date::set_hmer(int h, int m, int e)
{
   hmera=h;
   minas=m;
   etos=e;
}
void my_date::show()
{
   cout<<hmera<<"/"<<minas<<"/"<<etos<<endl;
}
```

Ορισμός της κλάσης **my_date**.

Η μέθοδος **set_hmer()** καταχωρίζει τις τιμές των παραμέτρων στις μεταβλητές-μέλη **hmera**, **minas** και **etos** αντίστοιχα.

Κεφάλαιο 16

Κληρονομικότητα



Κληρονομικότητα

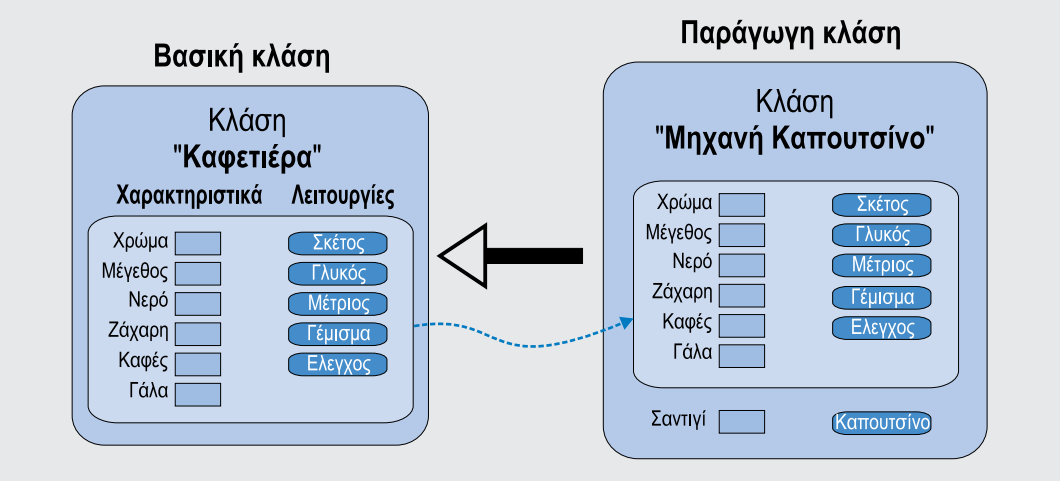
Η **κληρονομικότητα** (inheritance) αποτελεί ένα από τα πιο ισχυρά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού.

> Η κληρονομικότητα δίνει τη δυνατότητα να παραχθεί μια νέα κλάση από μια υπάρχουσα κλάση.

Αν θυμηθούμε ξανά το παράδειγμα της καφετιέρας, θα παρατηρήσουμε ότι η κλάση «Καφετιέρα» αναφέρεται γενικά σε μηχανές που φτιάχνουν καφέ και προσδιορίζει τα γενικά χαρακτηριστικά και τις λειτουργίες τους. Στην περίπτωση τώρα που θέλουμε να διαχειριστούμε καφετιέρες που φτιάχνουν και καπουτσίνο, θα αντιληφθούμε ότι η κλάση *Καφετιέρα* δεν μας καλύπτει διότι δεν προβλέπει *Σαντιγί* στα υλικά, και επίσης δεν διαθέτει την κατάλληλη λειτουργία για την κατασκευή καφέ αυτού του είδους. Η λύση είναι να δημιουργήσουμε μια νέα κλάση για τις «Μηχανές Καπουτσίνο», οι οποίες θα έχουν και όλα τα χαρακτηριστικά και τις λειτουργίες της γενικής κλάσης «Καφετιέρα».

Στη συγκεκριμένη περίπτωση, η κλάση *Μηχανή Καπουτσίνο* θα μπορούσε να παραχθεί από την κλάση *Καφετιέρα* αφού διαθέτει όλα τα χαρακτηριστικά και τις λειτουργίες της βασικής κλάσης. Στη νέα κλάση *Μηχανή Καπουτσίνο* θα προστεθεί το υλικό *Σαντιγί* και η λειτουργία *Καπουτσίνο* (Σχήμα 16.1).

Οι αντικειμενοστρεφείς γλώσσες προγραμματισμού παρέχουν τη δυνατότητα δημιουργίας μιας νέας κλάσης από μια υπάρχουσα. Η νέα κλάση λέγεται **παράγωγη κλάση** (derived class), και η κλάση από την οποία προέκυψε **βασική κλάση** (base class).



Σχήμα 16.1 *Κληρονομικότητα - Βασικές και παράγωγες κλάσεις*

Θα περιμέναμε το βέλος στο σχήμα να δείχνει στην αντίθετη κατεύθυνση, προς την παράγωγη κλάση. Ωστόσο, τόσο στη διεθνή βιβλιογραφία όσο και στα διαγράμματα κλάσεων UML, το βέλος σημαίνει «παράγεται από» και όχι «παράγει», οπότε εξηγείται και η κατεύθυνσή του.

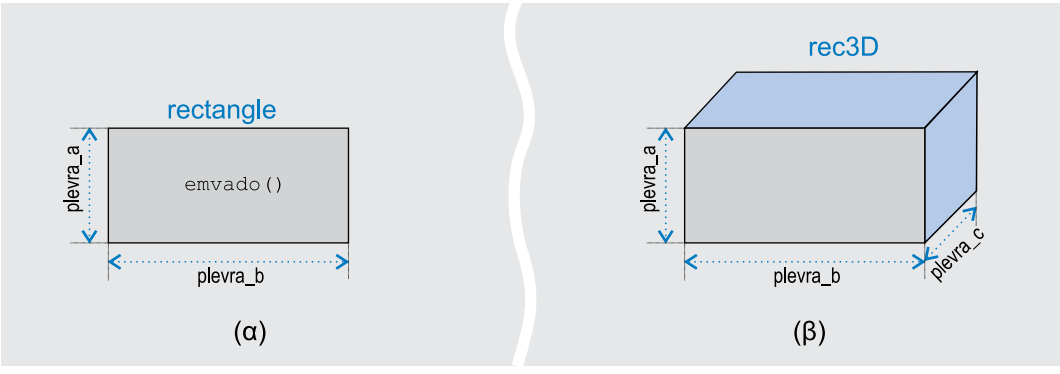
> Η παράγωγη κλάση κληρονομεί όλα τα χαρακτηριστικά της βασικής κλάσης. Η παράγωγη κλάση μπορεί να τροποποιήσει τα χαρακτηριστικά που κληρονόμησε, αλλά να προσθέσει και νέα. Η βασική κλάση παραμένει αναλλοίωτη.

Δημιουργία παράγωγης κλάσης

Για να αναλύσουμε τα χαρακτηριστικά της κληρονομικότητας, θα χρησιμοποιήσουμε την γνώριμη κλάση rectangle, η οποία χρησιμοποιείται για τη διαχείριση αντικειμένων ορθογωνίων παραλληλογράμμων (Σχήμα 16.2α).

```
class rectangle
{
   float plevra_a;
   float plevra_b;
public:
   float emvado(){return plevra_a * plevra_b;}
   void set_ab(float a, float b){plevra_a=a ; plevra_b=b;}
   void show() {cout<<plevra_a<<"x"<<plevra_b<<endl;}
};
```

Έστω ότι θέλουμε να διαχειριστούμε ορθογώνια παραλληλόγραμμα τριών διαστάσεων τα οποία θα έχουν και βάθος. Σε αυτή την περίπτωση, εκτός από τις διαστάσεις plevra_a και plevra_b, θα πρέπει να έχουμε και μια τρίτη διάσταση του αντικειμένου, την plevra_c (σχήμα 16.2β).



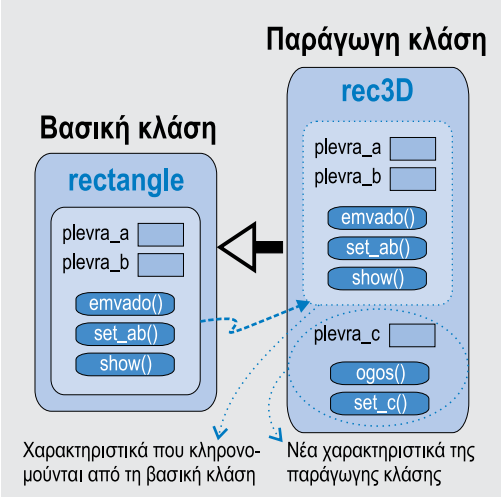
Σχήμα 16.2 *Κλάσεις rectangle και rec3D*

Ο προγραμματιστής λοιπόν θα πρέπει να φτιάξει μια νέα κλάση για τη διαχείριση των τριδιάστατων αυτών ορθογωνίων παραλληλογράμμων. Η κλάση rec3D, η οποία θα χρησιμοποιηθεί για τη διαχείριση αυτών των αντικειμένων έχει μικρές διαφορές από την κλάση rectangle. Θα πρέπει να προστεθεί μία μεταβλητή-μέλος plevra_c, καθώς και οι απαραίτητες μέθοδοι για τη διαχείριση των αντικειμένων τύπου rec3D.

Όπως βλέπουμε στον κώδικα παρακάτω, η C++ δίνει τη δυνατότητα στον προγραμματιστή να δημιουργήσει τη νέα κλάση rec3D από την ήδη υπάρχουσα rectangle κληρονομώντας ταυτόχρονα τα χαρακτηριστικά της:

```
class rec3D : public rectangle
{
   float plevra_c;
public:
   float ogos(){return emvado() * plevra_c;}
   void set_c(float c){plevra_c=c;}
};
```

Υπάρχουσα κλάση από την οποία θα δημιουργηθεί η νέα κλάση **rec3D**.



Η κλάση rec3D αποτελεί την παράγωγη κλάση και η rectangle τη βασική. Στη δήλωση της νέας κλάσης, μετά από το όνομά της ακολουθεί το **προσδιοριστικό πρόσβασης** (στη συγκεκριμένη περίπτωση το **public**) και το όνομα της βασικής κλάσης. Στη συγκεκριμένη περίπτωση, η νέα κλάση rec3D κληρονομεί όλα τα χαρακτηριστικά της rectangle. Στην παράγωγη κλάση rec3D προστίθεται η μεταβλητή μέλος plevra_c, καθώς και οι δύο μέθοδοι ogos() και set_c(), όπως φαίνεται στο σχήμα. Το προσδιοριστικό πρόσβασης **public** καθορίζει τον τρόπο με τον οποίο η παράγωγη κλάση κληρονομεί τα χαρακτηριστικά της βασικής κλάσης.

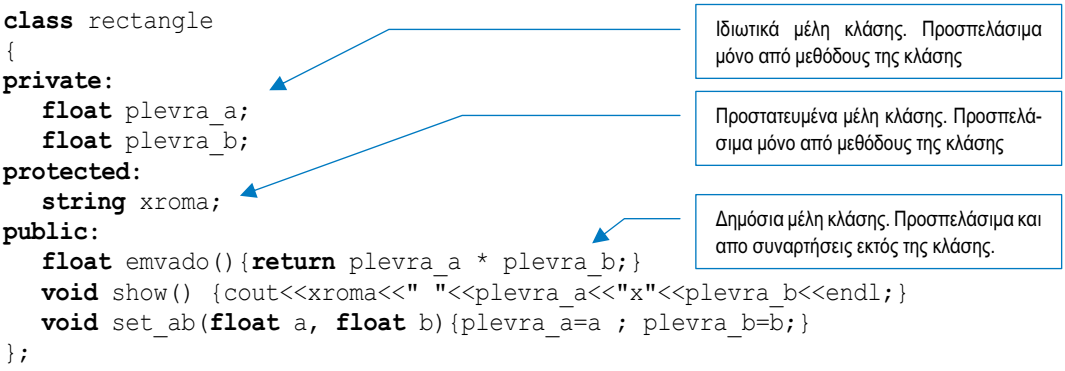
Η γενική μορφή της κληρονομικότητας είναι η παρακάτω:

class παράγωγη_κλάση : *προσδιοριστικό_πρόσβασης* βασική_κλάση
{
σώμα παράγωγης κλάσης
}

Το ***προσδιοριστικό_πρόσβασης*** μπορεί να είναι **public**, **private** ή **protected**. Ο τρόπος με τον οποίο το προσδιοριστικό πρόσβασης επηρεάζει την κληρονομικότητα θα αναλυθεί στη συνέχεια του κεφαλαίου.

Προστατευμένα μέλη κλάσης

Μέχρι τώρα γνωρίζαμε ότι τα μέλη μιας κλάσης μπορεί να είναι δημόσια ή ιδιωτικά χρησιμοποιώντας το προσδιοριστικό πρόσβασης **public** ή **private** αντίστοιχα. Υπάρχει όμως ακόμα μια κατηγορία μελών, τα προστατευμένα (protected), τα οποία ορίζονται με το προσδιοριστικό **protected**. Στην παρακάτω έκδοση της κλάσης rectangle, το μέλος xroma ορίζεται ως προστατευμένο.




```
class rectangle
{
private:
   float plevra_a;
   float plevra_b;
protected:
   string xroma;
public:
   float emvado(){return plevra_a * plevra_b;}
   void show() {cout<<xroma<<" "<<plevra_a<<"x"<<plevra_b<<endl;}
   void set_ab(float a, float b){plevra_a=a ; plevra_b=b;}
};
```

Όσον αφορά την ίδια την κλάση, τα προστατευμένα μέλη συμπεριφέρονται όπως ακριβώς και τα ιδιωτικά. Δηλαδή, σε αυτά έχουν πρόσβαση μόνο οι μέθοδοι της κλάσης και οι φίλιες συναρτήσεις της. Για παράδειγμα, αν δημιουργήσουμε ένα αντικείμενο rec1 της κλάσης rectangle:

```
rectangle rec1;
rec1.show();           // ✓ Σωστή πρόταση: η μέθοδος show() είναι δημόσια
rec1.plevra_a=20;      // ✗ Λάθος πρόταση: η μεταβλητή-μέλος plevra_a είναι ιδιωτική
rec1.xroma="Κοκκινο"; // ✗ Λάθος πρόταση: η μεταβλητή-μέλος xroma είναι προστατευμένη
```

τόσο τα ιδιωτικά μέλη όσο και τα προστατευμένα δεν είναι προσπελάσιμα από κώδικα εκτός της κλάσης.

> Κατά τη χρήση κληρονομικότητας, τα προστατευμένα μέλη μιας κλάσης μπορούν να προσπελάζονται και από τις παράγωγες κλάσεις. Στα διαγράμματα κλάσεων UML, τα προστατευμένα μέλη έχουν το πρόθεμα #.

Παράδειγμα κληρονομικότητας

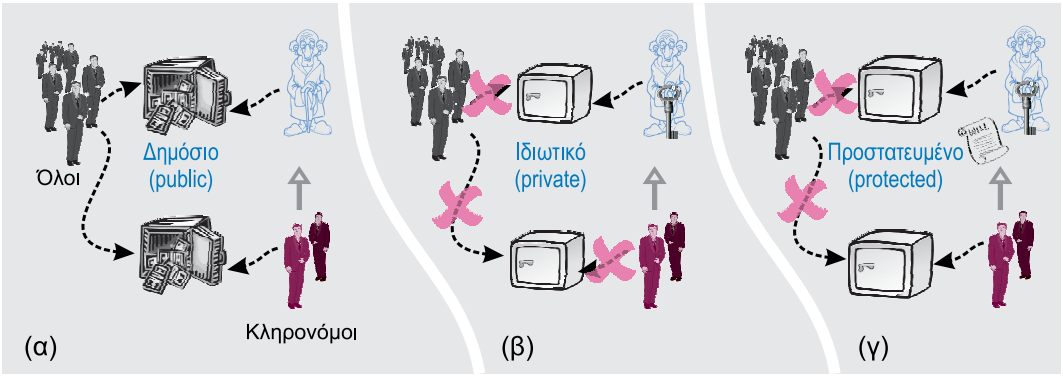
Για να μπορέσουμε να κατανοήσουμε σε βάθος τους μηχανισμούς της κληρονομικότητας, θα εξηγήσουμε αυτούς τους μηχανισμούς με παραδείγματα από την πραγματική ζωή. Εξάλλου, ο αντικειμενοστρεφής προγραμματισμός δεν είναι τίποτε άλλο παρά μια προσπάθεια μοντελοποίησης του πραγματικού κόσμου σε εφαρμογές προγραμματισμού. Ας υποθέσουμε λοιπόν ότι ο κύριος Γεροντόπουλος έχει ένα χρηματοκιβώτιο γεμάτο με

χρήματα. Ο κύριος Γεροντόπουλος είναι πολύ ανοιχτοχέρης και αφήνει το χρηματοκιβώτιο ανοιχτό (δημόσιο) ώστε, εκτός από τον ίδιο, ο καθένας να μπορεί να παίρνει χρήματα από αυτό (Σχήμα 16.3α). Επίσης, στη διαθήκη του επιθυμεί να παραμείνει η κατάσταση όπως είναι, δηλαδή το χρηματοκιβώτιο να είναι δημόσιο. Επομένως, οι κληρονόμοι του υποχρεούνται να διατηρήσουν ανοιχτό το χρηματοκιβώτιο που θα κληρονομήσουν και έτσι αυτό θα είναι προσπελάσιμο τόσο από αυτούς όσο και από όλους τους υπόλοιπους.

Ας υποθέσουμε τώρα ότι ο κύριος Γεροντόπουλος δεν είναι ανοιχτοχέρης αλλά τσιγκούνης, έχει κλειδωμένο (ιδιωτικό) το χρηματοκιβώτιο και μόνον εκείνος γνωρίζει πού είναι το κλειδί. Φυσικά, τώρα στο χρηματοκιβώτιο έχει πρόσβαση μόνον εκείνος και κανείς άλλος (Σχήμα 16.3β). Στη διαθήκη του δεν αφήνει καμία πληροφορία για το κλειδί, επομένως οι κληρονόμοι του, παρά το γεγονός ότι κληρονομούν το χρηματοκιβώτιο, δεν έχουν καμία πρόσβαση σε αυτό, ούτε φυσικά και κανείς άλλος. Ο παππούς πήρε το μυστικό μαζί του!

Στη τρίτη περίπτωση, ο κύριος Γεροντόπουλος έχει και πάλι κλειδωμένο το χρηματοκιβώτιο (προστατευμένο). Όσο ζει, στο χρηματοκιβώτιο έχει πρόσβαση μόνον εκείνος και κανείς άλλος (Σχήμα 16.3γ), όμως στη διαθήκη του γράφει που έχει κρυμμένο το κλειδί! Τώρα οι κληρονόμοι του έχουν πρόσβαση σε αυτό, αλλά μόνον αυτοί διότι αυτοί είναι οι μόνοι που γνωρίζουν πού είναι το κλειδί!

Ας φανταστούμε τώρα κάτι άλλο: Ότι βρισκόμαστε στην πρώτη περίπτωση που ο κύριος Γεροντόπουλος έχει το χρηματοκιβώτιο δημόσιο (το αφήνει ανοιχτό) αλλά στη διαθήκη του βάζει ως όρο οι κληρονόμοι του να το κρατάνε κλειδωμένο (ιδιωτικό). Διαπιστώνουμε επομένως ότι το πώς θα συμπεριφέρεται το κληροδότημα (αυτό που κληρονόμησαν οι κληρονόμοι) εξαρτάται από δύο παραμέτρους: Τι είναι αυτό που κληρονόμησαν (δημόσιο, ιδιωτικό ή προστατευμένο) και με ποιο τρόπο το κληρονόμησαν. Όπως θα αναλύσουμε στη συνέχεια, οι τρόποι κληρονόμησης είναι πάλι τρεις: Με δημόσια πρόσβαση, με ιδιωτική και με προστατευμένη!



Σχήμα 16.3 *Κληρονομικότητα μελών διαφορετικής πρόσβασης.*

Δημόσια πρόσβαση σε βασική κλάση

Η χρήση του προσδιοριστικού **public** κατά τη δημιουργία μιας παράγωγης κλάσης από μια βασική σημαίνει ότι τα μέλη της βασικής κλάσης κληρονομούνται από την παράγωγη κλάση σχεδόν όπως είναι. Δηλαδή, όλα τα δημόσια μέλη της βασικής κλάσης θα είναι και δημόσια μέλη της παράγωγης, τα ιδιωτικά μέλη της βασικής κλάσης κληρονομούνται ως ιδιωτικά, χωρίς όμως να είναι προσπελάσιμα από την παράγωγη κλάση, και τα προστατευμένα μέλη της βασικής κληρονομούνται ως προστατευμένα της παράγωγης.

> Σε μια παράγωγη κλάση, τα *κληρονομημένα* (inherited) χαρακτηριστικά είναι αυτά που κληρονομούνται από τη βασική κλάση, ενώ τα *εγγενή* χαρακτηριστικά είναι αυτά που ορίζονται μέσα στην ίδια την παράγωγη κλάση.

Το παρακάτω πρόγραμμα χρησιμοποιεί ως βασική την κλάση rectangle, από την οποία παράγεται η κλάση rec3D και δείχνει την απλή χρήση της κληρονομικότητας στη C++. Η βασική κλάση rectangle διαθέτει επιπλέον το προστατευμένο μέλος xroma και η νέα κλάση rec3D το επίσης προστατευμένο μέλος color.

16_rec3D1.cpp

```
#include <iostream>
#include <string>
using namespace std;
class rectangle
{
private:
   float plevra_a;
   float plevra_b;
protected:
   string xroma;
public:
   float emvado(){return plevra_a * plevra_b;}
   void show() {cout<<xroma<<" "<<plevra_a<<"x"<<plevra_b<<endl;}
   void set_ab(float a, float b, string x);
};
void rectangle::set_ab(float a, float b, string x)
{
      plevra_a=a;
      plevra_b=b;
      xroma=x;
}
class rec3D:public rectangle
{
   float plevra_c;
protected:
   string color;
```

Ιδιωτικά μέλη βασικής κλάσης.

Προστατευμένα μέλη βασικής κλάσης.

Δημόσια μέλη βασικής κλάσης.

Η κλάση **rec3D** παράγεται από τη βασική κλάση **rectangle** με δημόσια πρόσβαση.

Ιδιωτικά μέλη της παράγωγης κλάσης.

Προστατευμένα μέλη παράγωγης κλάσης.

```
public:
   float ogos();
   void set_c(float c, string xr);
};
float rec3D::ogos()
{
   return emvado()*plevra_c;
}
void rec3D::set_c(float c, string xr)
{
   plevra_c=c;
   color=xr;
}
int main()
{
   rectangle a;
   rec3D b;
   a.set_ab(10,20,"Κόκκινο");
   cout << a.emvado() << endl;
   a.show();
   b.set_ab(23,3,"Μαύρο");
   b.set_c(4,"Πράσινο");
   cout<<"Όγκος="<<b.ogos()<<endl;
   cout<<"Εμβαδό="<<b.emvado()<<endl;
   return 0;
}
```

Δημόσια μέλη της παράγωγης κλάσης.

Η μέθοδος **ogos()** χρησιμοποιεί για τον υπολογισμό του όγκου τη δημόσια μέθοδο **emvado()** της κλάσης **rectangle**, διότι δεν έχει πρόσβαση στα μέλη **plevra_a** και **plevra_b**.

Δημιουργία δύο αντικειμένων **a** και **b** των κλάσεων **rectangle** και **rec3D** αντίστοιχα.

Ανάθεση τιμών στο αντικείμενο **a**.

Κλήση των μεθόδων **emvado()** και **show()** για το αντικείμενο **a**.

Ανάθεση τιμών στο αντικείμενο **b**.

Κλήση των μεθόδων **ogos()** και **emvado()** για το αντικείμενο **b**.

```
200
Κόκκινο 10x20
Όγκος=276
Εμβαδο=69
```

Πριν προχωρήσουμε στην ανάλυση του κώδικα και των αποτελεσμάτων του παραπάνω προγράμματος, ας θυμηθούμε ότι η κλάση rec3D παράγεται από την rectangle με *δημόσια* (public) πρόσβαση. Αυτό σημαίνει ότι τα μέλη της βασικής κλάσης θα κληρονομηθούν από την παράγωγη σχεδόν όπως είναι! Δηλαδή, όλα τα δημόσια μέλη της κλάσης rectangle θα είναι και δημόσια μέλη της rec3D. Τα ιδιωτικά μέλη της κλάσης rectangle θα κληρονομηθούν ως ιδιωτικά, αλλά οι μέθοδοι της κλάσης rec3D *δεν* θα έχουν πρόσβαση σε αυτά. Επίσης τα προστατευμένα μέλη της κλάσης rectangle θα κληρονομηθούν ως προστατευμένα.

Στο Σχήμα 16.4α φαίνεται ο τρόπος παραγωγής της κλάσης rec3D από την κλάση rectangle με δημόσια πρόσβαση. Τα βέλη δείχνουν τις δυνατότητες πρόσβασης τόσο των μελών της βασικής όσο και των μελών της παράγωγης κλάσης.

Η διεπαφή των αντικειμένων της κλάσης rec3D με το προγραμματιστικό περιβάλλον γίνεται τόσο μέσω των δημόσιων μελών της κλάσης rec3D, όσο και μέσω των δημόσιων μελών της κλάσης rectangle, διότι μόνο αυτά είναι προσπελάσιμα από κώδικα εκτός της κλάσης (υποδεικνύεται στο Σχήμα 16.4α από τα βέλη).

Συνοψίζοντας…

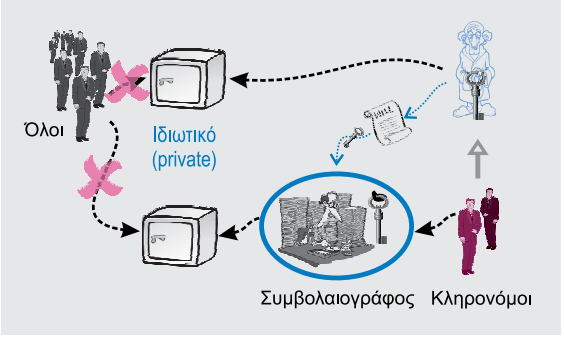
Στον παρακάτω πίνακα συνοψίζονται οι διαφορετικές δυνατότητες πρόσβασης στην κληρονομικότητα.

Προσδιοριστικό πρόσβασης κληρονομικότητας	Μέλη βασικής κλάσης	Κληρονομούνται από την παράγωγη κλάση ως ...
public	public	Δημόσια (public)
	private	Ιδιωτικά (private) χωρίς πρόσβαση
	protected	Προστατευμένα (protected)
private	public	Ιδιωτικά (private)
	private	Ιδιωτικά (private) χωρίς πρόσβαση
	protected	Ιδιωτικά (private)
protected	public	Προστατευμένα (protected)
	private	Ιδιωτικά (private) χωρίς πρόσβαση
	protected	Προστατευμένα (protected)

- Τα μέλη τα οποία έχουν κληρονομηθεί ως *δημόσια* είναι προσπελάσιμα από τα υπόλοιπα μέλη της παράγωγης κλάσης, καθώς και από κώδικα εκτός της κλάσης.
- Τα μέλη τα οποία έχουν κληρονομηθεί ως *ιδιωτικά* είναι προσπελάσιμα από τα υπόλοιπα μέλη της παράγωγης κλάσης, αλλά όχι από κώδικα εκτός της κλάσης.
- Τα μέλη τα οποία έχουν κληρονομηθεί ως *ιδιωτικά χωρίς πρόσβαση* δεν είναι προσπελάσιμα από τα υπόλοιπα μέλη της παράγωγης κλάσης, ούτε και από κώδικα εκτός της κλάσης.
- Τα μέλη τα οποία έχουν κληρονομηθεί ως *προστατευμένα* είναι προσπελάσιμα από τα υπόλοιπα μέλη της παράγωγης κλάσης, αλλά όχι από κώδικα εκτός της κλάσης.

Ας επιστρέψουμε πάλι στο παράδειγμα του κ. Γεροντόπουλου (Σχήμα 16.3)! Στις περιπτώσεις δημόσιας και προστατευμένης κληρονομικότητας οι κληρονόμοι είχαν πλήρη πρόσβαση στο χρηματοκιβώτιο, με αποτέλεσμα να έχουν τη δυνατότητα να κάνουν ανεξέλεγκτη ανάληψη χρημάτων.

Φανταστείτε τώρα το σενάριο του διπλανού σχήματος, όπου ο κ. Γεροντόπουλος αφήνει στη διαθήκη του το κλειδί σε έναν συμβολαιογράφο, με εντολή να μπορούν να παίρνουν οι κληρονόμοι από το χρηματοκιβώτιο το πολύ μέχρι 1000€ τον μήνα. Έτσι, αν και οι κληρονόμοι έχουν κληρονομήσει το χρηματοκιβώτιο, η πρόσβασή τους



σε αυτό γίνεται μόνο μέσω του εντεταλμένου συμβολαιογράφου! Αυτό το σενάριο στον αντικειμενοστρεφή προγραμματισμό υλοποιείται μέσω δημόσιων ή προστατευμένων μεθόδων της βασικής κλάσης.

Σε κάθε περίπτωση, σε ένα ιδιωτικό μέλος μιας κλάσης δεν έχει πρόσβαση *καμία* παράγωγη κλάση της. Για να μπορούν οι παράγωγες κλάσεις να έχουν πρόσβαση σε κάποιο ιδιωτικό μέλος της βασικής κλάσης, θα πρέπει στη βασική κλάση να υπάρχει μια δημόσια ή προστατευμένη μέθοδος μέσω της οποίας να προσπελάζεται το ιδιωτικό μέλος της.

Αλλαγή προσδιοριστικού πρόσβασης για συγκεκριμένα μέλη

Κάποιες φορές, ο προγραμματιστής ίσως θέλει να εξαιρέσει ένα ή περισσότερα μέλη μιας βασικής κλάσης από την επίδραση του προσδιοριστικού πρόσβασης με το οποίο παράχθηκε η παράγωγη κλάση. Για παράδειγμα, ενώ το προσδιοριστικό πρόσβασης **public** επιβάλλει όλα τα δημόσια μέλη της βασικής κλάσης να κληρονομηθούν ως δημόσια, ο προγραμματιστής πιθανώς να θέλει να εξαιρέσει κάποιο ή κάποια από αυτά και να τα κληρονομήσει ως ιδιωτικά. Στην περίπτωση της κλάσης rec3D, η μέθοδος emvado() που κληρονόμησε από τη rectangle επιστρέφει το εμβαδό μίας μόνο πλευράς ενός αντικειμένου rec3D! Η πληροφορία αυτή δεν είναι πολύ χρήσιμη και ίσως αρκετές φορές να μπερδεύει. Επομένως, είναι λογικό να μη θέλουμε να υπάρχει πρόσβαση στη μέθοδο emvado() από αντικείμενα της κλάσης rec3D.

Προσέξτε τον παρακάτω ορισμό της κλάσης rec3D :

```
class rec3D:public rectangle
{
private:
   float plevra_c;
   using rectangle::emvado;
public:
   float ogos(){return emvado() * plevra_c;}
   void set_c(float c){plevra_c=c;}
};
```

Προσδιοριστικό πρόσβασης κληρονομικότητας.

Ιδιωτικά μέλη της κλάσης **rec3D**

Η μέθοδος **emvado()** της κλάσης **rectancle** ορίζεται να κληρονομηθεί ως ιδιωτική!

Το προσδιοριστικό **using** χρησιμοποιείται, μέσα στον χώρο των ιδιωτικών δηλώσεων, για να δηλώσει ότι η μέθοδος emvado() της κλάσης rectangle θα κληρονομηθεί ως ιδιωτική (private), και όχι ως δημόσια (public) όπως καθορίζει το προσδιοριστικό πρόσβασης κληρονομικότητας της κλάσης rec3D. Η κλάση rec3D εξακολουθεί να έχει πρόσβαση στη μέθοδο emvado() και τη χρησιμοποιεί για τον υπολογισμό του όγκου, στη μέθοδο ogos().

Μπορούμε να αλλάξουμε το προσδιοριστικό πρόσβασης μόνο προς το αυστηρότερο. Δηλαδή δεν μπορούμε να κληρονομήσουμε ένα ιδιωτικό μέλος ως δημόσιο!

Δύο επίσης σημεία θα πρέπει να προσεχθούν:

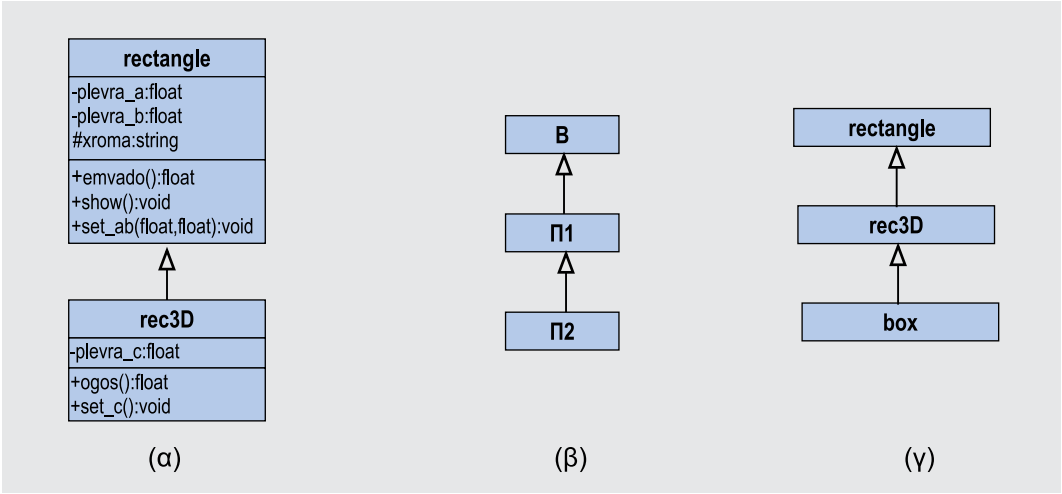
- Δεν ορίζεται η λίστα παραμέτρων της μεθόδου, παρά μόνο το όνομά της χωρίς τις παρενθέσεις.
- Πριν από το όνομα της μεθόδου πρέπει να προηγείται το όνομα της βασικής κλάσης ακολουθούμενο από τον τελεστή επίλυσης εμβέλειας (::).

Επίπεδα κληρονομικότητας

Μέχρι στιγμής, τα παραδείγματα κληρονομικότητας που συναντήσαμε αφορούσαν κληρονομικότητα ενός επιπέδου. Στα διαγράμματα κλάσεων UML η κληρονομικότητα συμβολίζεται με ένα βέλος από την παράγωγη προς τη βασική κλάση. Το Σχήμα 16.6α δείχνει τη σχέση των κλάσεων rectangle και rec3D που έχουμε ήδη συναντήσει. Το πρόθεμα # υποδεικνύει προστατευμένο μέλος της κλάσης, όπως το μέλος xroma.

Αρχίζουμε να συζητάμε για πολλά επίπεδα κληρονομικότητας (multi-level inheritance) από τη στιγμή που μια παράγωγη κλάση αποτελεί τη βασική κλάση για ένα επόμενο επίπεδο κληρονομικότητας. Αν θεωρήσουμε ως βασική κλάση την κλάση B, την Π1 μια κλάση που παράγεται από τη B και την Π2 μια κλάση η οποία παράγεται από την Π1, το διάγραμμα κλάσεων του Σχήματος 16.6β δείχνει τη σχέση των τριών αυτών κλάσεων.

Στην αλυσίδα της κληρονομικότητας, οι κλάσεις έχουν *ιεραρχία*. Κάθε βασική κλάση είναι ένα επίπεδο ιεραρχίας υψηλότερα από την παράγωγη κλάση. Έτσι, στο παραπάνω παράδειγμα η κλάση B είναι πρώτη σε ιεραρχία ακολουθούμενη με τη σειρά από τις κλάσεις Π1 και Π2.



Σχήμα 16.6 *Απεικόνιση της κληρονομικότητας σε διαγράμματα κλάσεων UML.*

Πού χρησιμεύει η κληρονομικότητα

Μια από τις βασικές ιδέες πίσω από τις αρχές του αντικειμενοστρεφούς προγραμματισμού είναι η επαναχρησιμοποίηση κώδικα. Δηλαδή, να μην ανακαλύπτουμε τον τροχό κάθε φορά που τον χρειαζόμαστε! Πολλές φορές διάφορες οντότητες ενός προγράμματος είναι παρόμοιες, με ελάχιστες διαφορές μεταξύ τους. Αν προσθέσουμε, αφαιρέσουμε ή αλλάξουμε κάτι από τη λειτουργικότητα μιας οντότητας, μπορεί να προκύψει η νέα οντότητα που χρειαζόμαστε. Στον διαδικαστικό προγραμματισμό αυτό γίνεται με τον παραδοσιακό τρόπο! Αντιγραφή του κώδικα και αλλαγή στα σημεία που χρειάζεται. Αυτό όμως έχει ως αποτέλεσμα την επανάληψη του ίδιου κώδικα πολλές φορές. Επίσης, μια αλλαγή στον σχεδιασμό της εφαρμογής επιβάλει αλλαγές σε πολλά διαφορετικά σημεία του προγράμματος.

Η κληρονομικότητα αντιμετωπίζει αυτά τα προβλήματα με ιδιαίτερα επιτυχή τρόπο. Μια οντότητα μπορεί να κληρονομεί τη λειτουργικότητα μιας άλλης. Στη νέα οντότητα μπορεί να προστεθούν νέα χαρακτηριστικά και λειτουργίες ακόμα και να κρυφτούν κάποια από τα υπάρχοντα αν δεν τα χρειαζόμαστε. Αυτό πρακτικά σημαίνει λιγότερο γράψιμο κώδικα αλλά και μεγαλύτερη ασφάλεια. Επίσης, αλλαγές ή προσθήκες στον κώδικα της βασικής οντότητας (βασική κλάση) έχει ως αποτέλεσμα την αυτόματη διαθεσιμότητά τους σε όλες τις παράγωγες οντότητες (αντικείμενα παράγωγων κλάσεων).

Η κλάση rectangle αποτελούσε την κλάση με την υψηλότερη ιεραρχία στη κληρονομικότητα πολλών επιπέδων του παραδείγματος της προηγούμενης ενότητας. Ούτε η rectangle ούτε καμία από τις παράγωγες κλάσεις rec3D και box διέθεταν δυνατότητες για τη διαχείριση του χρώματος των αντικειμένων τους. Η προσθήκη των ακόλουθων μελών στη κλάση rectangle

```
class rectangle
{
protected:
   .....
   string xroma;
public:
   void set_color(string x){xroma=x;}
   string get_color(){return xroma;}
   .....
};
```

Νέα μέλη της κλάσης **rectangle**

θα έχει ως αποτέλεσμα τη δυνατότητα διαχείρισης του χρώματος τόσο στα αντικείμενα αυτής της κλάσης όσο και στα αντικείμενα όλων των παράγωγων κλάσεων. Η νέα μεταβλητή-μέλος και οι νέες μέθοδοι κληρονομούνται από τις παράγωγες κλάσεις rec3D και box. Η διαχείριση του χρώματος γίνεται με προτάσεις της ακόλουθης μορφής:

```
r1.set_color("Πράσινο"); // Ορίζει το πράσινο χρώμα για το αντικείμενο r1 της κλάσης rec3D
cout<<b1.get_color();    // Εμφανίζει το χρώμα του αντικεμένου b1 της κλάσης box
```

Παραδείγματα

Π16.1 Παρακάτω ορίζεται η κλάση circle_based_shapes για τη διαχείριση αντικειμένων τα οποία έχουν ως βάση έναν κύκλο. Δηλαδή, οι κύκλοι, οι σφαίρες, οι κύλινδροι και οι κυκλικοί κώνοι.

16_p16_1.cpp

```
#include <iostream>
#include <cmath>
#define PI 3.141593
using namespace std;

class circle_based_shape
{
   float aktina;
public:
   float emvado(){return pow(get_r(),2)*PI;}
   float perimetros(){return 2*get_r()*PI;}
   void set_r(float r){aktina=r;}
   float get_r() {return aktina;}
};
```

Επιφάνεια κύκλου: $\pi \cdot r^2$
Περίμετρος κύκλου: $2 \cdot \pi \cdot r$

Θέλουμε να δημιουργήσουμε δύο νέες κλάσεις sphere και cylinder οι οποίες θα παράγονται από την κλάση circle_based_shape και θα διαχειρίζονται τρισδιάστατα αντικείμενα σχήματος σφαίρας και κυλίνδρου αντίστοιχα. Κάθε νέα κλάση θα πρέπει να διαθέτει μια μέθοδο emvado() η οποία θα υποσκελίζει την αντίστοιχη μέθοδο της circle_based_shape και θα υπολογίζει την επιφάνεια του σχήματος. Επίσης, και οι δύο κλάσεις θα διαθέτουν μια μέθοδο ogos() η οποία θα επιστρέφει τον όγκο του τριδιάστατου σχήματος.

```
class sphere:public circle_based_shape
{
public:
   float emvado(){return 4*pow(get_r(),2)*PI;}
   float ogos(){return 4/3.0*pow(get_r(),3)*PI;}
};

class cylinder:public circle_based_shape
{
   float ypsos;
public:
   void set_ypsos(float y){ypsos=y;}
   float get_ypsos() {return ypsos;}
   float emvado(){return 2*PI*ypsos+2*PI*pow(get_r(),2);}
   float ogos(){return ypsos*PI*pow(get_r(),2);}
};
```

Επιφάνεια σφαίρας: $\mathbf{4 \cdot \pi \cdot r^2}$
Όγκος σφαίρας: $\mathbf{4/3 \cdot \pi \cdot r^3}$

Ιδιωτική μεταβλητή-μέλος για το ύψος του κυλίνδρου.

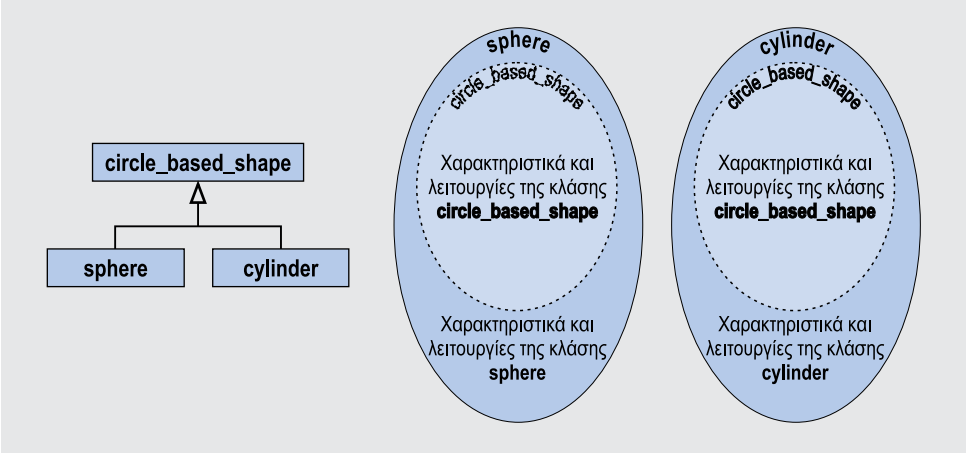
Επιφάνεια κυλίνδρου: $\mathbf{2 \cdot \pi \cdot h + 2 \cdot \pi \cdot r^2}$
Όγκος κυλίνδρου: $\mathbf{h \cdot \pi \cdot r^2}$

```
int main()
{
   sphere ball;
   cylinder c1;
   ball.set_r(10);
   c1.set_r(5);
   c1.set_ypsos(10);
   cout<<"Εμβαδό επιφάνειας σφαίρας:"<<ball.emvado()<<" m2"<<endl;
   cout<<"Όγκος σφαίρας:"<<ball.ogos()<<" m3"<<endl;
   cout<<"Εμβαδό επιφάνειας κυλίνδρου:"<<c1.emvado()<<" m2"<<endl;
   cout<<"Όγκος κυλίνδρου:"<<c1.ogos()<<" m"<<endl;
   cout<<"Εμβαδό βάσης κυλίνδρου:"
        <<c1.circle_based_shape::emvado()<<" m2"<<endl;
   return 0;
}
```

Ανατίθενται τιμές για την ακτίνα της σφαίρας (10) , της βάσης του κυλίνδρου (5) και του ύψους του (10). Η μέθοδος **set_r ()** έχει κληρονομηθεί, και για τις δύο παράγωγες κλάσεις, από τη βασική κλάση **circle_based_shape**.

```
Εμβαδό επιφάνειας σφαίρας:1256.64 m2
Όγκος σφαίρας:4188.79 m3
Εμβαδό επιφάνειας κυλίνδρου:219.912 m2
Όγκος κυλίνδρου:785.398 m
Εμβαδό βάσης κυλίνδρου:78.5398 m2
```

Παρατηρήστε την τελευταία πρόταση στην οποία καλείται η μέθοδος emvado() την οποία το αντικείμενο ball κληρονόμησε από τη βασική κλάση circle_based_shape. Η μέθοδος αυτή επιστρέφει το εμβαδό του κύκλου της βάσης του αντικειμένου κυλίνδρου c1 (στη συγκεκριμένη περίπτωση, το εμβαδό κύκλου με ακτίνα 5). Στο Σχήμα 16.8 που ακολουθεί, εμφανίζεται το διάγραμμα κλάσεων UML, καθώς και η δομή των αντικειμένων των δύο παράγωγων κλάσεων του παραδείγματος.

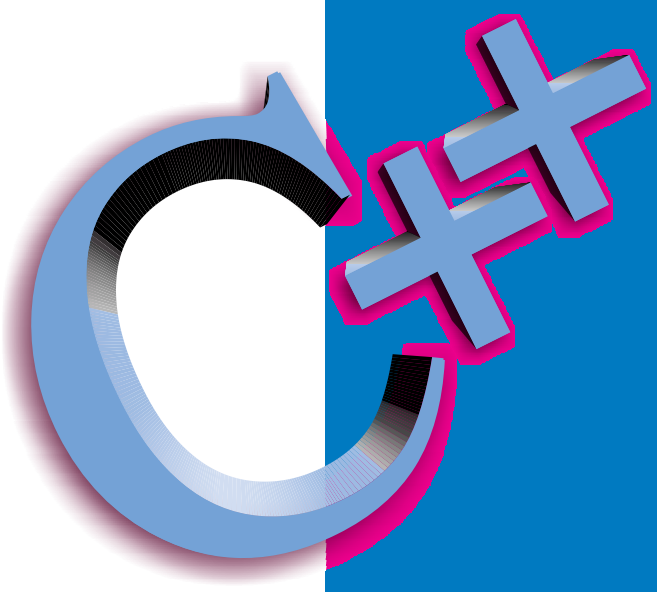


Σχήμα 16.8 *Το διάγραμμα κλάσεων UML του παραδείγματος - Δομή αντικειμένων*

Κεφάλαιο

17

Πολυμορφισμός και συσχέτιση κλάσεων



Πολυμορφισμός

Δείκτες σε παράγωγες κλάσεις

Γενικά ένας δείκτης δείχνει σε αντικείμενα συγκεκριμένου τύπου ανάλογα με την πρόταση δήλωσής του:

```
rectangle r1,*ptr;
triangle t1;
ptr=&r1;
```

Στο παραπάνω παράδειγμα, ο δείκτης ptr δείχνει σε αντικείμενα κλάσης rectangle και του ανατίθεται ως τιμή η διεύθυνση του αντικειμένου r1. Όμως η πρόταση

```
ptr=&t1;
```

δεν είναι έγκυρη, διότι προσπαθεί να καταχωρίσει στον δείκτη ptr τη διεύθυνση ενός αντικειμένου που δεν ανήκει στην κλάση rectangle (όπως έχει δηλωθεί ο δείκτης) αλλά στην κλάση triangle.

Εξαίρεση στον παραπάνω κανόνα αποτελούν οι δείκτες σε αντικείμενα βασικών και παράγωγων κλάσεων.

> Στη C++, ένας δείκτης σε μια βασική κλάση μπορεί επίσης να δείχνει σε αντικείμενα οποιασδήποτε κλάσης που παράγεται από αυτήν τη βασική κλάση.

Για παράδειγμα, αν έχουμε τη βασική κλάση B και την κλάση D που παράγεται από την B, τότε οποιοσδήποτε δείκτης της κλάσης B μπορεί επίσης να χρησιμοποιηθεί ώστε να δείχνει σε αντικείμενα της παράγωγης κλάσης D. Δείτε τις παρακάτω δηλώσεις αντικειμένων τόσο της βασικής όσο και της παράγωγης κλάσης:

```
B obj_b,*ptr;
D obj_d;
```

Οι παρακάτω αναθέσεις τιμών στη μεταβλητή δείκτη ptr είναι απολύτως αποδεκτές:

```
ptr=&obj_b;
ptr=&obj_d;
```

> Ένας δείκτης της βασικής κλάσης μπορεί να χρησιμοποιηθεί *μόνο* για την προσπέλαση των μελών ενός αντικειμένου της παράγωγης κλάσης που έχουν κληρονομηθεί από τη βασική κλάση.

Δηλαδή, με βάση το προηγούμενο παράδειγμα και την πρόταση ptr=&obj_d, μέσω του δείκτη ptr μπορούμε να προσπελάζουμε τα μέλη του αντικειμένου obj_d τα οποία κληρονόμησε από τη βασική κλάση B, αλλά *όχι* τα μέλη που ορίζονται στην ίδια τη κλάση

D. Κάτι που επίσης πρέπει να επισημανθεί είναι ότι, ενώ ένας δείκτης της βασικής κλάσης μπορεί να χρησιμοποιηθεί για να δείχνει σε αντικείμενα της παράγωγης κλάσης, δεν συμβαίνει και το αντίστροφο:

> Ένας δείκτης μιας παράγωγης κλάσης δεν μπορεί να δείχνει σε αντικείμενα της βασικής κλάσης.

Στο παρακάτω παράδειγμα γίνεται φανερή η διαφορά στη χρήση δεικτών σε βασικές και παράγωγες κλάσεις:

17_pointer_to_base_class.cpp

```
#include <iostream>
using namespace std;
class polygon
{
protected:
   int platos, ypsos;
public:
   void set(int a, int b) {platos=a; ypsos=b;}
   void show() {cout<<"Πολύγωνο"<<endl;}
};
class rectangle:public polygon
{
public:
   int emvado() {return (platos * ypsos);}
   void show() {cout<<"Παραλληλόγραμμο"<<endl;}
};
class triangle:public polygon
{
public:
   int emvado(){return (platos * ypsos / 2);}
   void show() {cout<<"Τρίγωνο"<<endl;}
};

int main()
{
   rectangle rect;
   triangle trgl;
   polygon *ptr1 = &rect;
   polygon *ptr2 = &trgl;
   ptr1->set(4,5);
   ptr2->set(4,5);
   cout << rect.emvado() << endl;
   cout << trgl.emvado() << endl;
   ptr1->show();
   return 0;
}
```

```
20
10
Πολύγωνο
```

Η κλάση **polygon** θα χρησιμοποιηθεί ως βασική κλάση.

Η κλάση **rectangle** είναι παράγωγη της κλάσης **polygon**.

Η κλάση **triangle** παράγεται επίσης από την κλάση **polygon**.

Οι δύο δείκτες **ptr1** και **ptr2** δηλώνονται ως δείκτες της βασικής κλάσης **polygon**, αλλά δείχνουν σε αντικείμενα των παράγωγων κλάσεων **rectangle** και **triangle** αντίστοιχα.

Μέσω των δεικτών **ptr1** και **ptr2**, έχουμε πρόσβαση στη μέθοδο **set()** του υπο-αντικειμένου της βασικής κλάσης που υπάρχει μέσα σε κάθε αντικείμενο της παράγωγης κλάσης.

Η κλήση της μεθόδου **show()** μέσω του δείκτη **ptr1** έχει ως αποτέλεσμα την κλήση της μεθόδου **show()** της βασικής κλάσης και όχι της παράγωγης **rectangle** στην οποία δείχνει ο δείκτης **ptr1**.

Γνήσιες εικονικές μέθοδοι

Στο προηγούμενο παράδειγμα, η βασική κλάση polygon δεν χρησιμοποιήθηκε για τη δημιουργία αντικειμένων. Δημιουργήσαμε αντικείμενα μόνο των παράγωγων κλάσεων rectangle και triangle. Η χρήση της βασικής κλάσης polygon είχε μόνο οργανωτικό και ιεραρχικό χαρακτήρα. Τα αντικείμενα των παράγωγων κλάσεων χρησιμοποιούσαν τη μέθοδο set() της βασικής κλάσης, αλλά κάθε ένα από αυτά είχε τη δική του έκδοση της εικονικής μεθόδου show().

> Στον αντικειμενοστρεφή σχεδιασμό, πολλές φορές δημιουργούμε βασικές κλάσεις μόνο και μόνο για τη δόμηση μιας ιεραρχίας παράγωγων κλάσεων, χωρίς να υπάρχει ανάγκη και νόημα να δημιουργούμε αντικείμενα της βασικής κλάσης.

Θα παρατηρήσουμε επίσης ότι η εικονική μέθοδος show() του παραδείγματος δεν έχει νόημα να περιέχει κώδικα αφού ποτέ δεν θα δημιουργήσουμε αντικείμενα της κλάσης polygon παρά μόνο των παράγωγων κλάσεων. Είναι πιθανόν επίσης να θέλουμε να αποτρέψουμε τη δημιουργία αντικειμένων της κλάσης polygon. Η λύση και στα δύο αυτά προβλήματα βρίσκεται στη χρήση των **γνήσιων εικονικών μεθόδων** (pure virtual methods). Μια γνήσια εικονική μέθοδος δηλώνεται με την παρακάτω γενική σύνταξη:

virtual τύπος *όνομα_μεθόδου*(*παράμετροι*, ...)=0;

Δηλαδή, δηλώνεται όπως μια κανονική εικονική μέθοδος, όμως δεν έχει σώμα. Η ανάθεση της τιμής 0 δεν έχει ουσιαστική σημασία, και απλώς είναι ο τρόπος με τον οποίο ο μεταγλωττιστής αντιλαμβάνεται ότι μια εικονική μέθοδος είναι γνήσια. Αν θέλαμε, για παράδειγμα, να δηλώσουμε ότι η εικονική μέθοδος show() είναι γνήσια, θα έπρεπε να χρησιμοποιήσουμε την παρακάτω πρόταση:

```
virtual void show()=0;
```

> Από τη στιγμή που μια γνήσια εικονική μέθοδος δεν έχει καθόλου κώδικα, θα πρέπει να ορίζεται μια έκδοσή της σε κάθε παράγωγη κλάση από την οποία θα θέλουμε να δημιουργούμε αντικείμενα. Δεν μπορεί να χρησιμοποιηθεί η έκδοση της βασικής κλάσης διότι απλούστατα δεν υπάρχει!

Ο παρακάτω κώδικας είναι μια παραλλαγή του προηγούμενου παραδείγματος με χρήση γνήσιων εικονικών μεθόδων. Στον ορισμό της βασικής κλάσης polygon, οι μέθοδοι show() και emvado() δηλώνονται ως γνήσιες εικονικές.

17_pure_virtual_methods.cpp

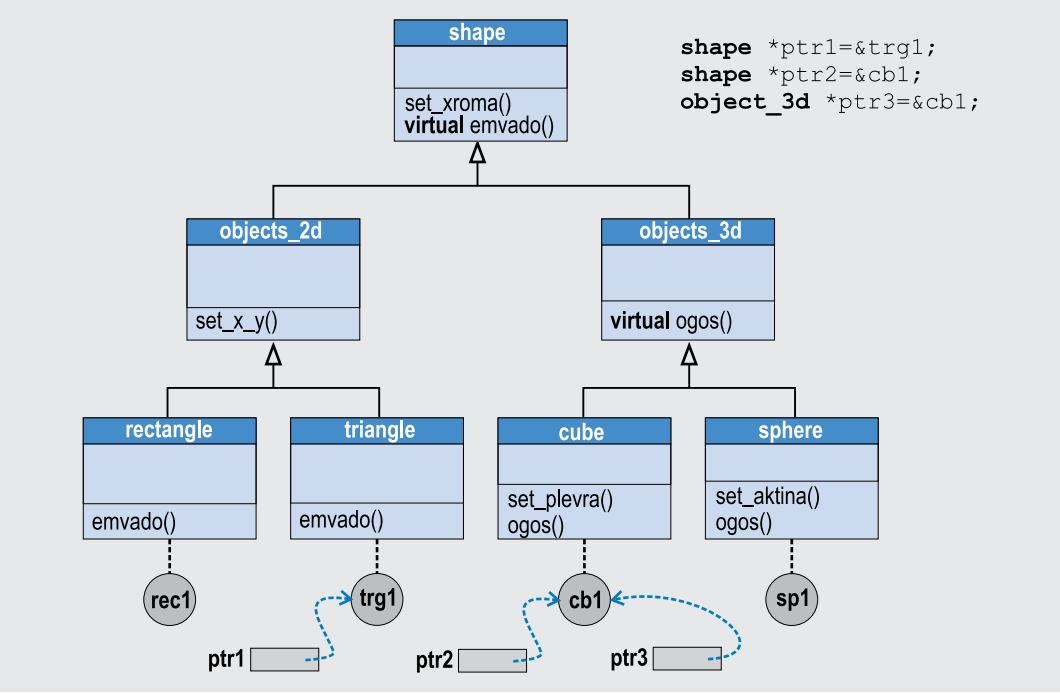
```
#include <iostream>
using namespace std;
class polygon
{
protected:
   float platos, ypsos;
```

ποίους ανατίθενται τιμές ώστε να δείχνουν σε κάποια από αυτά τα αντικείμενα. Το Σχήμα 17.1 δείχνει εποπτικά αυτή τη διαδικασία.

Μέσω των δεικτών στα αντικείμενα, μπορούμε να εφαρμόσουμε σε αυτά τις μεθόδους τόσο των παράγωγων όσο και των βασικών κλάσεων. Για παράδειγμα, το γεγονός ότι η μέθοδος emvado() έχει δηλωθεί ως εικονική, η πρόταση ptr1->emvado() εφαρμόζει τη μέθοδο emvado() της κλάσης triangle και όχι της κλάσης shape. Για τον ίδιο λόγο, η πρόταση ptr3->ogos() εφαρμόζει τη μέθοδο ogos() της κλάσης cube και όχι της κλάσης objects_3d. Αντίθετα, η πρόταση ptr2->ogos() δεν θα μεταγλωττιστεί, διότι η κλάση shape (σε αντικείμενα της οποίας δείχνει ο δείκτης ptr2) δεν διαθέτει μέθοδο με όνομα ogos(). Η ιεραρχία κλάσεων του Σχήματος 17.1, υλοποιείται στο παράδειγμα Π17.1 στο τέλος του τρέχοντος κεφαλαίου.

Αναπαράσταση εικονικών μεθόδων και αφηρημένων κλάσεων στη UML

Οι εικονικές μέθοδοι, οι γνήσιες εικονικές μέθοδοι, καθώς και οι αφηρημένες κλάσεις εμφανίζονται στα διαγράμματα κλάσεων UML με πλάγιους χαρακτήρες. Στον κώδικα που ακολουθεί, ορίζονται δύο βασικές κλάσεις oxima και mixani.

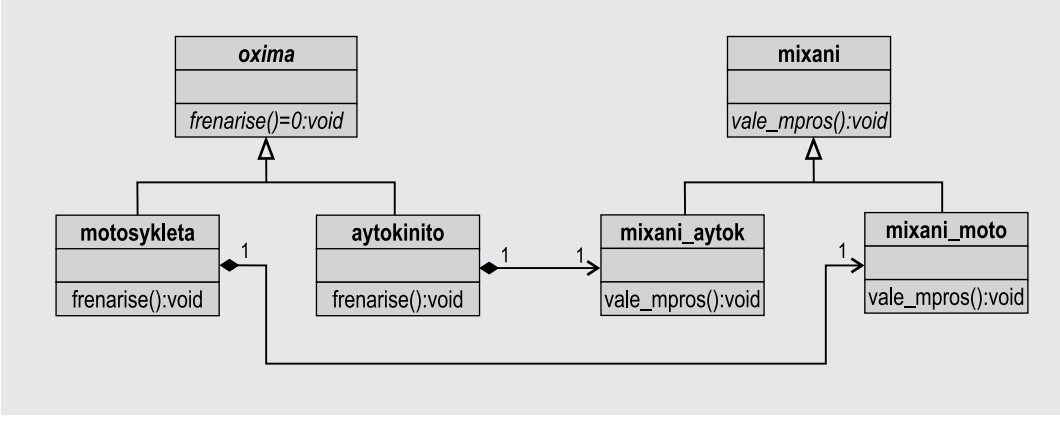


Σχήμα 17.1 *Δείκτες σε αντικείμενα παράγωγων κλάσεων*

Το διάγραμμα κλάσεων UML του Σχήματος 17.2 απεικονίζει τις παράγωγες κλάσεις, τη συσχέτιση των κλάσεων μεταξύ τους καθώς και τις εικονικές μεθόδους που διαθέτουν.

```
class oxima
{
public:
   .....
   virtual void frenarise()=0;
};

class mixani
{
public:
   .....
   virtual void vale_mpros()
   {cout<<"Η μηχανή πήρε
   μπρος"<<endl;}
};

class mixani_aytok:public mixani
{
public:
   .....
   void vale_mpros() {cout<<"Η μηχανή
   αυτοκινήτου πήρε μπρός"<<endl;}
};

class mixani_moto:public mixani
{
public:
   .....
   void vale_mpros() {cout<<"Η μηχανή
   μοτοσυκλέτας πήρε μπρος"<<endl;}
};

class motosykleta:public oxima
{
public:
   mixani_moto engine;
   void frenarise() {cout<<"Η
   μοτοσυκλέτα φρενάρισε"<<endl;}
};

class aytokinito:public oxima
{
public:
   mixani_aytok engine;
   void frenarise() {cout<<"Το
   αυτοκίνητο φρενάρισε"<<endl;}
};
```

Η κλάση oxima είναι μια αφηρημένη κλάση διότι διαθέτει μια γνήσια εικονική μέθοδο, τη frenarise(). Η κλάση mixani διαθέτει την εικονική μέθοδο vale_mpros(). Η αφηρημένη κλάση oxima και οι εικονικές μέθοδοι frenarise() και vale_mpros() εμφανίζονται στο διάγραμμα κλάσεων με πλάγια γραφή.



Σχήμα 17.2 *Απεικόνιση εικονικών μεθόδων και αφηρημένων κλάσεων στη UML*

Συσχέτιση κλάσεων

Αν και στο Κεφάλαιο 14 συναντήσαμε περιπτώσεις συσχέτισης κλάσεων, περιοριστήκαμε στη **συσχέτιση σύνθεσης** όπου μία κλάση διαθέτει ως μέλη αντικείμενα άλλων κλάσεων.

Στο παρόν κεφάλαιο γνωρίσαμε ακόμα ένα είδος συσχέτισης, τη **συσχέτιση κληρονομικότητας** (inheritance relationship). Το είδος αυτό περιγράφει τη σχέση μεταξύ παράγωγων και βασικών κλάσεων και αρκετές φορές το συναντάμε επίσης με τον όρο **συσχέτιση γενίκευσης** (generalization relationship).

Πριν όμως δούμε τα υπόλοιπα είδη συσχετίσεων, ας μελετήσουμε το ακόλουθο παράδειγμα από την πραγματική ζωή, ώστε να τις καταλάβουμε καλύτερα. Μελετήστε τις ακόλουθες προτάσεις:

- Ο φοιτητής είναι ένας άνθρωπος
- Κάθε φοιτητής διαθέτει μια φοιτητική ταυτότητα
- Μια ομάδα εργασίας αποτελείται από φοιτητές
- Υποθέστε ότι για την εισαγωγή φοιτητών σε μια σχολή, ένας καθηγητής μπορεί να εξετάσει όποιον φοιτητή θέλει αλλά και ο φοιτητής μπορεί να ζητήσει εξέταση από όποιον καθηγητή επιθυμεί.

Στις προτάσεις αυτές εντοπίζουμε τέσσερις κλάσεις: Φοιτητής, Άνθρωπος, Ταυτότητα, Ομάδα και Καθηγητής. Οι κλάσεις αυτές σχετίζονται μεταξύ τους με όλα τα δυνατά είδη συσχέτισης.

Οι κλάσεις **Φοιτητής** και **Άνθρωπος** σχετίζονται με **συσχέτιση κληρονομικότητας** (inheritance). Ένας φοιτητής είναι ένας άνθρωπος. Προσέξτε τη φράση **είναι ένας**. Αν η φράση αυτή χαρακτηρίζει μια κλάση, τότε έχουμε συσχέτιση κληρονομικότητας.

Οι κλάσεις **Φοιτητής** και **Ταυτότητα** σχετίζονται με **συσχέτιση σύνθεσης** (composition). Ένας φοιτητής έχει μία φοιτητική ταυτότητα. Στην περίπτωση αυτή έχουμε σχέση αλληλεξάρτησης μεταξύ των κλάσεων. Δηλαδή, δεν υπάρχει περίπτωση να υπάρχει φοιτητική ταυτότητα χωρίς φοιτητή και φοιτητής χωρίς φοιτητική ταυτότητα. Στην περίπτωση που καταστραφεί ένα αντικείμενο-φοιτητής δεν έχει νόημα ύπαρξης η φοιτητική του ταυτότητα. Επομένως, οι δύο κλάσεις έχουν κοινό χρόνο ζωής (lifetime). Εδώ προσέξτε τη λέξη **έχει**. Αν η λέξη αυτή χαρακτηρίζει μια κλάση, τότε έχουμε συσχέτιση σύνθεσης.

Μια ομάδα αποτελείται από φοιτητές. Όμως όταν καταστραφεί μια ομάδα δεν σημαίνει ότι παύουν να υφίστανται και οι φοιτητές που την αποτελούσαν. Επίσης, ακόμα και αν οι φοιτητές παύουν να υπάρχουν, η ομάδα δεν καταστρέφεται και μπορεί να δεχθεί άλλους φοιτητές. Επιπρόσθετα, ένας φοιτητής μπορεί να ανήκει σε πολλές ομάδες ταυτόχρονα. Επομένως, ο χρόνος ζωής των δύο κλάσεων *δεν* είναι κοινός. Οι κλάσεις **Ομάδα** και

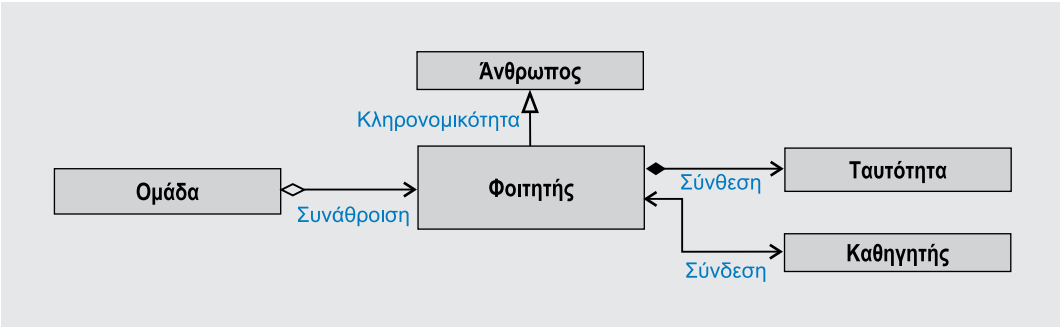
Φοιτητής σχετίζονται με ένα νέο είδος συσχέτισης, τη **συσχέτιση συνάθροισης** (aggregation). Εδώ προσέξτε τη φράση **αποτελείται από**. Αν η φράση αυτή χαρακτηρίζει μια κλάση, τότε έχουμε συσχέτιση συνάθροισης.

Οι κλάσεις **Φοιτητής** και **Καθηγητής** δεν φαίνεται να έχουν κάποια σχέση εξάρτησης. Όμως, στο παράδειγμά μας, συνεργάζονται με κάποιο τρόπο και ένας φοιτητής μπορεί να ζητήσει εξέταση από κάποιον καθηγητή ή και το αντίστροφο. Ο χρόνος ζωής των κλάσεων είναι ανεξάρτητος και η μόνη σχέση που έχουν είναι η δυνατότητα συνεργασίας. Οι δύο κλάσεις σχετίζονται με ένα νέο είδος συσχέτισης τη **συσχέτιση σύνδεσης** (association). Εδώ προσέξτε τη φράση **συνεργάζεται με** ή **συνδέεται με**. Αν οι φράσεις αυτές χαρακτηρίζουν μια κλάση, και η κλάση δεν περιλαμβάνεται σε κανένα άλλο είδος συσχέτισης, τότε έχουμε συσχέτιση σύνδεσης.

Συνοψίζοντας, έχουμε τις ακόλουθες περιπτώσεις συσχέτισης κλάσεων:

- Συσχέτιση **κληρονομικότητας/γενίκευσης** (inheritance/generalization relationship)
- Συσχέτιση **σύνθεσης** (composition relationship)
- Συσχέτιση **συνάθροισης** (aggregation relationship)
- Συσχέτιση **σύνδεσης** (association relationship)

Στο πρώτο μέρος του παρόντος κεφαλαίου καλύψαμε πλήρως τη συσχέτιση κληρονομικότητας και τον τρόπο με τον οποίο απεικονίζεται αυτή η σχέση στα διαγράμματα κλάσεων UML, και χρησιμοποιήσαμε αυτή τη συσχέτιση σε πρακτικά παραδείγματα κώδικα. Στη συνέχεια θα δούμε πώς υλοποιούνται οι υπόλοιποι τρόποι συσχέτισης και πώς αυτοί απεικονίζονται στα διαγράμματα κλάσεων UML. Σε κάθε περίπτωση, η συσχέτιση κλάσεων απεικονίζεται στα διαγράμματα κλάσεων με μια συνεχή γραμμή η οποία συνδέει τις δύο κλάσεις. Οι διάφοροι τρόποι συσχέτισης διακρίνονται από τα διαφορετικά σύμβολα τερματισμού αυτής της γραμμής. Το Σχήμα 17.4 δείχνει τις τέσσερις περιπτώσεις συσχέτισης κλάσεων του παραπάνω παραδείγματος.



Σχήμα 17.4 *Συσχέτιση κλάσεων – Περιπτώσεις συσχέτισης*

Ασκήσεις Κεφαλαίου 17

17.1 Τι θα εμφανίσει στην οθόνη το παρακάτω πρόγραμμα; ★★

```
#include <iostream>
using namespace std;
class sxima
{
   string color;
public:
   void set_col(string col) {color=col;}
   void show() {cout<<"Σχήμα χρώματος "<<color<<endl;}
};
class kyklos:public sxima
{
   float aktina;
public:
   void set_r(float r) {aktina=r;}
   void show() {cout<<"Κύκλος ακτίνας "<<aktina<<endl;}
};
class tetragono:public sxima
{
   float plevra;
public:
   void set_a(float a) {plevra=a;}
   void show() {cout<<"Τετράγωνο πλευράς "<<plevra<<endl;}
};
int main()
{
   kyklos k1;
   tetragono t1;
   sxima *ptr1 = &t1;
   t1.set_a(10);
   t1.set_col("Κόκκινο");
   k1.set_col("Μπλε");
   k1.set_r(5);
   ptr1->show();
   ptr1=&k1;
   ptr1->show();
   return 0;
}
```

17.2 Τι αλλαγές θα έπρεπε να γίνουν στον κώδικα της προηγούμενης άσκησης ώστε, όταν καλείται η μέθοδος show() μέσω του δείκτη ptr1, να καλείται η μέθοδος show() της κλάσης των αντικειμένων στα οποία δείχνει ο δείκτης και όχι η μέθοδος της βασικής κλάσης sxima; ★

17.12 Ένα αεροσκάφος έχει επιβάτες. Οι επιβάτες είναι άνθρωποι. Το αεροσκάφος έχει δύο τουρμπίνες. Οι τουρμπίνες είναι μηχανές. Το αεροσκάφος το οδηγεί ένας πιλότος. Ο πιλότος είναι και αυτός άνθρωπος. Το αεροσκάφος κάνει διάφορα δρομολόγια. ★★

Σημειώστε τις συσχετίσεις μεταξύ των ακόλουθων κλάσεων:

Αεροσκάφος - Μηχανή: _____

Μηχανή - Τουρμπίνα: _____

Αεροσκάφος - Τουρμπίνα: _____

Αεροσκάφος - Επιβάτης: _____

Επιβάτης - Άνθρωπος: _____

Πιλότος - Αεροσκάφος: _____

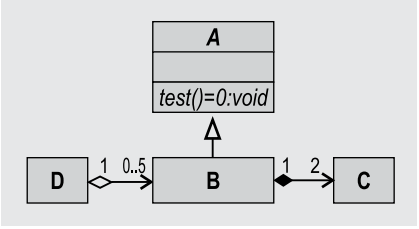
Πιλότος - Άνθρωπος: _____

Αεροσκάφος - Δρομολόγιο: _____

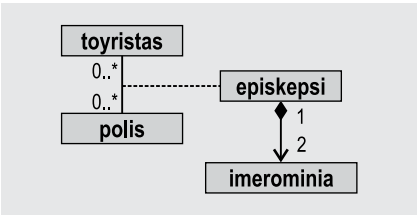
Πιλότος - Επιβάτης: _____

17.13 Ένα ποδήλατο είναι ένα όχημα. Το ποδήλατο διαθέτει κάποιο σετ ταχυτήτων. Το ποδήλατο το οδηγεί κάποιος άνθρωπος. Το ποδήλατο μπορεί να το επισκευάσει κάποιος τεχνικός. Να σχεδιαστεί το διάγραμμα κλάσεων UML στο οποίο να εμφανίζονται οι συσχετίσεις των κλάσεων Όχημα, Ποδήλατο, Σετ_ταχυτήτων, Άνθρωπος και Τεχνικός. ★★★

17.14 Γράψτε κώδικα στον οποίο να ορίζονται οι κλάσεις A, B, C και D, έτσι ώστε να υλοποιείται το διάγραμμα κλάσεων UML του διπλανού σχήματος. Αποφασίστε εσείς για τα μέλη των κλάσεων. Προσέξτε τη πλάγια γραφή! ★★★



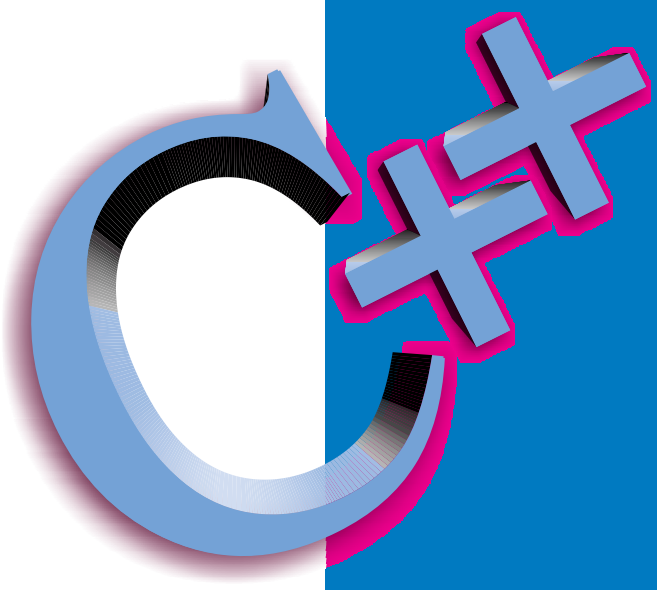
17.15 Ένας τουρίστας επισκέπτεται διάφορες πόλεις. Για κάθε επίσκεψη θέλουμε να καταχωρίζουμε την ημερομηνία άφιξης και την ημερομηνία αναχώρησης. Στο διπλανό διάγραμμα κλάσεων εμφανίζεται η επιθυμητή συσχέτιση κλάσεων καθώς και η πολλαπλότητα των σχέσεων. Γράψτε ένα ολοκληρωμένο πρόγραμμα το οποίο να υλοποιεί το διάγραμμα κλάσεων του σχήματος. Αποφασίστε εσείς για τα μέλη των κλάσεων. ★★★



Κεφάλαιο

18

Ρεύματα εισόδου και εξόδου της C++

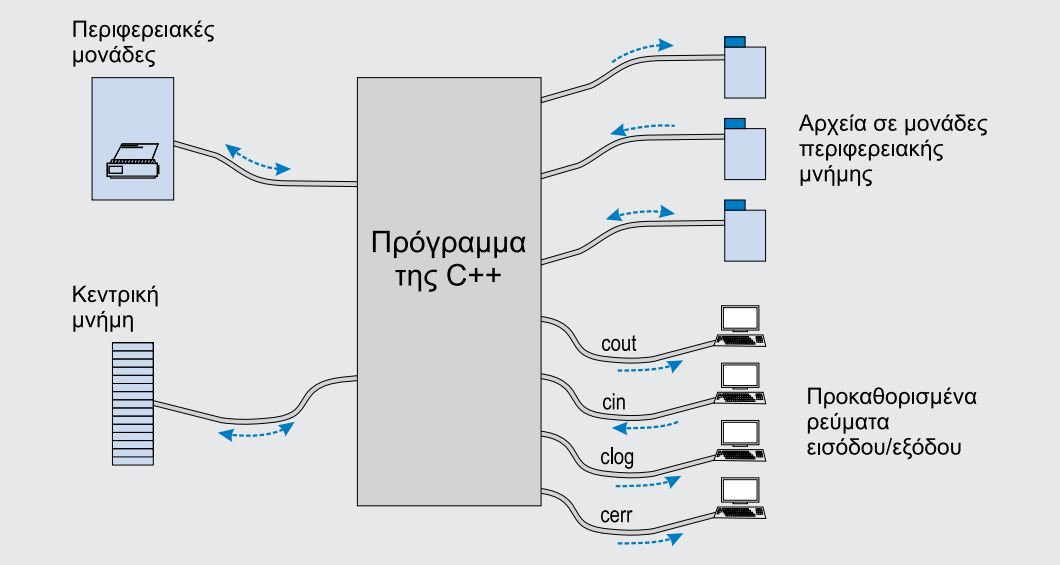


Ρεύματα εισόδου και εξόδου της C++

Στη C++, η είσοδος και η έξοδος επιτυγχάνεται μέσω των **ρευμάτων** (streams). Θεωρήστε ένα ρεύμα ως μια λογική διασύνδεση του προγράμματός σας με τις διάφορες συσκευές που απαρτίζουν έναν Η/Υ. Στη C++, τόσο η είσοδος όσο και η έξοδος υλοποιείται με ακολουθίες από byte. Ένα ρεύμα είναι στην πραγματικότητα ένα κανάλι μέσω του οποίου μπορούμε να εξάγουμε (διαβάζουμε) ή να εισάγουμε (στέλνουμε) ακολουθίες από byte, από ή σε περιφερειακές συσκευές.

Ένα **ρεύμα εισόδου** (input stream) δέχεται byte από μια περιφερειακή συσκευή, π.χ. από ένα αρχείο στον δίσκο, από το πληκτρολόγιο ή από άλλη συσκευή εισόδου. Ένα **ρεύμα εξόδου** (output stream) στέλνει μια σειρά από byte σε μια συσκευή εξόδου όπως στην οθόνη, σε ένα αρχείο, σε έναν εκτυπωτή κ.λπ. Υπάρχουν και τα **ρεύματα εισόδου/εξόδου** (i/o stream) τα οποία μπορούν και να εισάγουν και να εξάγουν πληροφορίες από και προς τη συσκευή με την οποία είναι συνδεδεμένα (π.χ. αρχείο στο δίσκο, μόντεμ κ.λπ). Μπορούμε να φανταστούμε τα ρεύματα εισόδου και εξόδου της C++ σαν «κανάλια» μέσω των οποίων το πρόγραμμά μας ανταλλάσει πληροφορίες με τις περιφερειακές συσκευές του συστήματός μας (Σχήμα 18.1).

Κάποια από αυτά τα «κανάλια» είναι μόνιμα συνδεδεμένα με συγκεκριμένες συσκευές, ενώ κάποια άλλα απαιτούν την εκτέλεση συγκεκριμένων ενεργειών του προγράμματός μας για να δημιουργηθούν. Τα «κανάλια» αυτά λέγονται προκαθορισμένα (ή καθιερωμένα) ρεύματα εισόδου/εξόδου και έχουν συγκεκριμένα ονόματα.



Σχήμα 18.1 *Ρεύματα εισόδου και εξόδου της C++*

Σε μια αντικειμενοστρεφή γλώσσα προγραμματισμού τα πάντα αντιμετωπίζονται ως αντικείμενα. Έτσι, ένα ρεύμα εισόδου/εξόδου (κανάλι) δεν είναι τίποτε άλλο παρά ένα αντικείμενο μιας συγκεκριμένης κλάσης ρευμάτων. Όπως θα δούμε και στη συνέχεια, υπάρχουν διαφορετικές κλάσεις ρευμάτων με μια αρκετά πολύπλοκη ιεραρχική δομή, και ανάλογα με το είδος του ρεύματος που επιθυμούμε δημιουργούμε ένα αντικείμενο της αντίστοιχης κλάσης.

Όπως αναφέρθηκε και προηγουμένως, κάποια ρεύματα είναι ήδη συνδεδεμένα με συγκεκριμένες περιφερειακές συσκευές χωρίς να απαιτείται καμία ενέργεια του προγραμματιστή. Αυτά λέγονται **προκαθορισμένα ρεύματα** εισόδου/εξόδου και αντιπροσωπεύονται από τα αντικείμενα cout, cin, cerr και clog.

Είμαστε ήδη εξοικειωμένοι με τα αντικείμενα cout και cin τα οποία έχουμε χρησιμοποιήσει για την έξοδο πληροφοριών στην οθόνη και την είσοδο από το πληκτρολόγιο αντίστοιχα. Τα αντικείμενα cout, cerr και clog είναι ρεύματα εξόδου και είναι συνδεδεμένα με την προκαθορισμένη έξοδο του συστήματός μας που είναι συνήθως η οθόνη. Το αντικείμενο cin είναι ένα ρεύμα εισόδου και είναι συνδεδεμένο με την προκαθορισμένη είσοδο του συστήματός μας που είναι συνήθως το πληκτρολόγιο[1]. Στο ρεύμα cerr στέλνονται τα μηνύματα σφαλμάτων του προγράμματός μας, ενώ στο ρεύμα clog τα μηνύματα καταγραφής.

> Πέρα από τα προκαθορισμένα ρεύματα, τα ρεύματα που θα δημιουργήσουμε στα προγράμματά μας μπορούν να συνδεθούν με οποιαδήποτε περιφερειακή συσκευή η οποία, για παράδειγμα, μπορεί να είναι ένα αρχείο στον δίσκο μας, ο εκτυπωτής μας, ένα μόντεμ, ακόμα και ένα τμήμα της κεντρικής μνήμης του συστήματός μας.

Η C++ διαθέτει δύο τρόπους διαχείρισης των ρευμάτων: Έναν **χαμηλού επιπέδου** (low-level) και έναν **υψηλού επιπέδου** (high level) . Ο πρώτος τρόπος χαμηλού επιπέδου αντιμετωπίζει ένα ρεύμα ως μια απλή σειρά από byte, και η διαχείρισή του περιορίζεται στην εισαγωγή ή την εξαγωγή συγκεκριμένου πλήθους byte από ή προς το συγκεκριμένο ρεύμα. Ο δεύτερος τρόπος υψηλού επιπέδου, ο οποίος καλείται «μορφοποιημένη είσοδος/έξοδος» (formatted I/O), αντιμετωπίζει τα byte ενός ρεύματος ως ομάδες οι οποίες μπορεί να αντιστοιχούν σε οντότητες όπως ακέραιοι αριθμοί, πραγματικοί αριθμοί, χαρακτήρες, συμβολοσειρές κ.λπ.

Τα ρεύματα χαμηλού επιπέδου αποκαλούνται **δυαδικά ρεύματα** (binary streams) , ενώ τα υψηλού επιπέδου **ρεύματα χαρακτήρων** (text streams) .

[1] Οι προκαθορισμένες συσκευές εξόδου και εισόδου του Η/Υ μπορούν να επαναπροσδιοριστούν από το λειτουργικό σύστημα και να μην είναι πλέον η οθόνη και το πληκτρολόγιο.

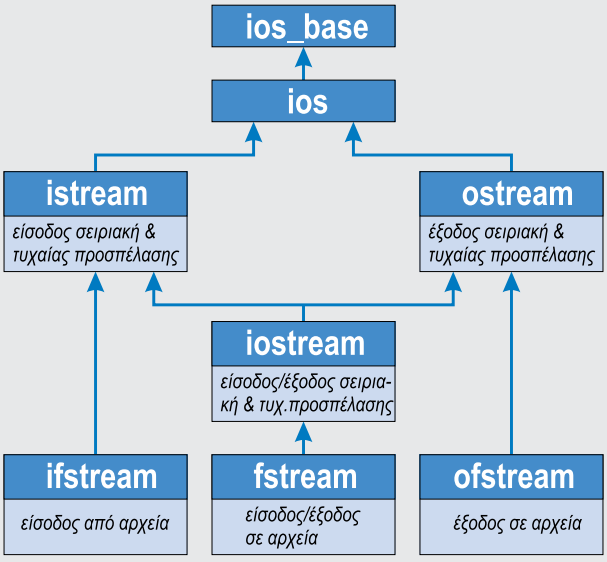
Στη C++, όλα τα ρεύματα αντιμετωπίζονται με τον ίδιο τρόπο, ανεξάρτητα από τη φυσική συσκευή του συστήματός μας με την οποία είναι συνδεδεμένα. Επομένως, με τον ίδιο τρόπο με τον οποίο διαβάζουμε χαρακτήρες από ένα αρχείο, διαβάζουμε και από το πληκτρολόγιο ή από μια σειριακή θύρα.

Στο Σχήμα 18.2 φαίνεται η ιεραρχία των κλάσεων που χρησιμοποιούνται για τη διαχείριση των ρευμάτων. Οι βασικές κλάσεις ios_base και ios καθορίζουν τον γενικό τρόπο χειρισμού των ρευμάτων. Από αυτές τις βασικές κλάσεις παράγονται όλες οι υπόλοιπες.

Οι κλάσεις ios_base και ios αποτελούν τις βασικές κλάσεις όλης της ιεραρχίας των κλάσεων ρευμάτων και περιγράφουν τα γενικά χαρακτηριστικά και λειτουργίες ενός ρεύματος. Στις δύο αυτές κλάσεις ορίζονται μέλη τα οποία είναι κοινά για όλα τα αντικείμενα ρευμάτων, ανεξάρτητα αν είναι ρεύματα εισόδου, εξόδου ή εισόδου/εξόδου.

Ένα ρεύμα εισόδου ανήκει στην κλάση istream, ένα ρεύμα εξόδου στην κλάση ostream, και ένα ρεύμα εισόδου/εξόδου στην κλάση iostream.

Ειδικά για τον χειρισμό αρχείων, χρησιμοποιούνται οι παράγωγες κλάσεις ρευμάτων ifstream, fstream και ofstream.



Σχήμα 18.2 *Η ιεραρχία των κλάσεων ρευμάτων στη C++*

Προκαθορισμένα αντικείμενα ρευμάτων

Όπως κάθε ρεύμα στη C++, τα προκαθορισμένα αντικείμενα ρευμάτων ανήκουν σε συγκεκριμένες κλάσεις ρευμάτων:

cout → ρεύμα εξόδου → αντικείμενο της κλάσης ostream
cin → ρεύμα εισόδου → αντικείμενο της κλάσης istream
cerr → ρεύμα εξόδου → αντικείμενο της κλάσης ostream
clog → ρεύμα εξόδου → αντικείμενο της κλάσης ostream

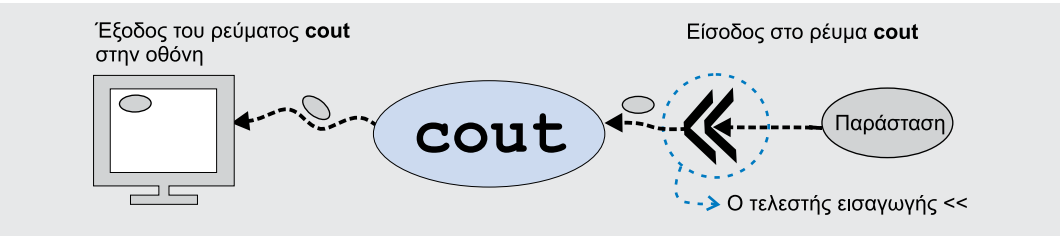
Η καθιερωμένη βιβλιοθήκη της C++ περιλαμβάνει το αρχείο κεφαλίδας iostream, στο οποίο δηλώνονται τα παραπάνω προκαθορισμένα αντικείμενα ρευμάτων.

Γενικά, τα προκαθορισμένα αντικείμενα ρευμάτων χρησιμοποιούν την οθόνη ως συσκευή εξόδου και το πληκτρολόγιο ως συσκευή εισόδου. Τα αντικείμενα cerr και clog συμπεριφέρονται με τον ίδιο ακριβώς τρόπο όπως το αντικείμενο cout, αφού και τα δύο είναι συνδεδεμένα με την προκαθορισμένη έξοδο η οποία, αν δεν οριστεί διαφορετικά από το λειτουργικό σύστημα, είναι η οθόνη.

Μέχρι αυτό το σημείο του βιβλίου έχουν χρησιμοποιηθεί τόσο το αντικείμενο cout όσο και το αντικείμενο cin, χωρίς όμως να έχουν προσδιοριστεί συγκεκριμένα ως ρεύματα. Από τώρα και στο εξής, τα δύο αυτά αντικείμενα θα τα αντιμετωπίζουμε ως ρεύματα, και θα περιγράψουμε αναλυτικά τις δυνατότητες που μας παρέχουν.

Οι τελεστές εισαγωγής << και εξαγωγής >>

Οι τελεστές εισαγωγής << και εξαγωγής >> χρησιμοποιούνται για την εισαγωγή και την εξαγωγή δεδομένων σε ρεύματα εξόδου και ρεύματα εισόδου, αντίστοιχα. Ο τελεστής εισαγωγής << έχει υπερφορτωθεί για την κλάση ostream ώστε να εισάγει πληροφορίες σε ένα ρεύμα εξόδου. Σε συνδυασμό με το αντικείμενο cout, εισάγει πληροφορίες στο συγκεκριμένο ρεύμα, οι οποίες εμφανίζονται στην προκαθορισμένη έξοδο, δηλαδή στην οθόνη.



Ο τελεστής εισαγωγής << εισάγει τις πληροφορίες που βρίσκονται μετά από αυτόν στο ρεύμα που βρίσκεται αριστερά του τελεστή.

Χειρισμός αρχείων

Στη C++, η ανάγνωση και η εγγραφή σε ένα αρχείο μιας μονάδας αποθήκευσης γίνεται μέσω αντικειμένων-ρευμάτων, ακριβώς με τον ίδιο τρόπο όπως σε οποιαδήποτε άλλη περιφερειακή συσκευή.

Η C++ διαθέτει τις παρακάτω κλάσεις ρευμάτων μέσω των οποίων πραγματοποιείται η είσοδος και η έξοδος σε αρχεία:

ofstream : για έξοδο (εγγραφή) σε αρχεία

ifstream : για είσοδο (ανάγνωση) από αρχεία

fstream : για ταυτόχρονη είσοδο και έξοδο από/σε αρχεία.

Οι κλάσεις αυτές παράγονται από τις κλάσεις istream και ostream (βλέπε σελ. 624), αντικείμενα των οποίων έχουμε ήδη χρησιμοποιήσει, όπως το cin και cout. Μπορούμε να χρησιμοποιήσουμε τα ρεύματα αρχείων με τον ίδιο τρόπο με τον οποίο χρησιμοποιήσαμε τα αντικείμενα cin και cout. Η μόνη διαφορά είναι ότι πρέπει πρώτα να συσχετίσουμε τα ρεύματα αυτά με πραγματικά αρχεία στις μονάδες αποθήκευσης του συστήματός μας.

Για παράδειγμα, η παρακάτω πρόταση:

```
ofstream myfile;
```

δημιουργεί το αντικείμενο myfile της κλάσης ofstream, το οποίο όμως θα πρέπει να συσχετίσουμε αμέσως μετά με ένα πραγματικό αρχείο του συστήματός μας.

Όμως, πριν προχωρήσουμε στον τρόπο με τον οποίο η C++ χειρίζεται τα αρχεία, πρέπει πρώτα να γνωρίσουμε τους διαφορετικούς τρόπους με τους οποίους μπορεί να γίνει η προσπέλαση των δεδομένων ενός αρχείου.

Σειριακή και τυχαία προσπέλαση

Ένα αρχείο δεν είναι τίποτε άλλο παρά μια ακολουθία από byte αποθηκευμένων στην περιφερειακή μνήμη (μονάδες αποθήκευσης) του Η/Υ (Σχήμα 18.5α). Η C++ διαθέτει δύο τρόπους προσπέλασης ενός αρχείου: Τη σειριακή και την τυχαία προσπέλαση.

Σειριακή προσπέλαση αρχείου σημαίνει ότι διαβάζουμε (ή γράφουμε) πληροφορίες στο αρχείο με τη σειρά, ξεκινώντας από την αρχή του αρχείου (ή και από το τέλος αν πραγματοποιούμε εγγραφή). Για παράδειγμα, για να διαβάσουμε το 100ό στοιχείο, πρέπει πρώτα να διαβάσουμε με τη σειρά τα προηγούμενα 99.

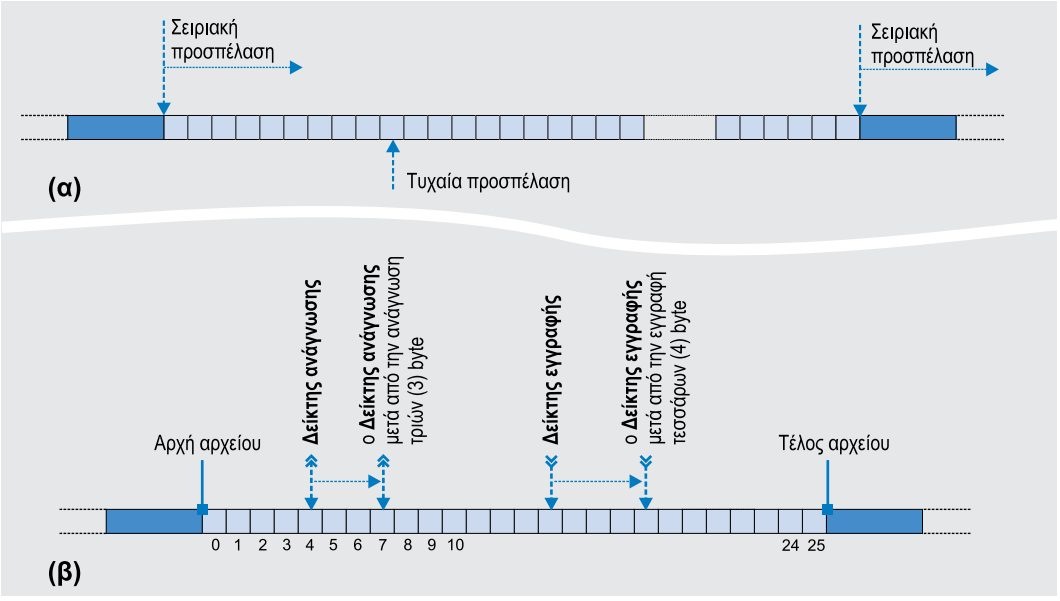
Στη σειριακή προσπέλαση μπορούμε:

- Να διαβάσουμε δεδομένα με τη σειρά ξεκινώντας από την αρχή του αρχείου.
- Να γράψουμε δεδομένα ξεκινώντας από την αρχή του αρχείου διαγράφοντας τα υπάρχοντα δεδομένα.
- Να γράψουμε δεδομένα ξεκινώντας από το τέλος του αρχείου και προσθέτοντας τα νέα δεδομένα στα υπάρχοντα.

Στην **τυχαία προσπέλαση** μπορούμε να μεταφερθούμε κατευθείαν σε ένα συγκεκριμένο σημείο του αρχείου και να διαβάσουμε ή να γράψουμε δεδομένα ξεκινώντας από αυτό το σημείο.

Για τη C++, ένα αρχείο είναι μια σειρά από byte όπου το πρώτο byte έχει αύξοντα αριθμό 0, το δεύτερο 1, κ.ο.κ.

Η αλήθεια είναι ότι στη C++ διατηρούνται δύο δείκτες για κάθε αρχείο: Ένας **δείκτης ανάγνωσης** και ένας **δείκτης εγγραφής** (Σχήμα 18.5β). Ο δείκτης ανάγνωσης δείχνει στο byte που πρόκειται να διαβαστεί στην επόμενη λειτουργία ανάγνωσης, ενώ ο δείκτης εγγραφής δείχνει στο byte το οποίο πρόκειται να εγγραφεί στην επόμενη διαδικασία εγγραφής. Μετά από την ανάγνωση ή την εγγραφή ενός πλήθους byte, οι αντίστοιχοι δείκτες θέσης μετακινούνται ισάριθμες θέσεις προς τα δεξιά. Με τους δείκτες ανάγνωσης και εγγραφής, θα ασχοληθούμε στη συνέχεια του παρόντος κεφαλαίου, όταν μιλήσουμε για την τυχαία προσπέλαση αρχείων.



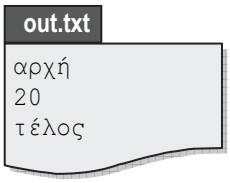
Σχήμα 18.5 *Σειριακή και τυχαία προσπέλαση*

Μορφοποιημένη είσοδος/έξοδος σε αρχεία κειμένου

Η μορφοποιημένη είσοδος/έξοδος σε αρχεία κειμένου (text files) γίνεται ακριβώς με τον ίδιο τρόπο όπως με τα προκαθορισμένα αντικείμενα ρευμάτων cin και cout, με χρήση των τελεστών εισαγωγής και εξαγωγής. Η διαφορά είναι ότι, αντί να χρησιμοποιήσουμε τα προκαθορισμένα αντικείμενα ρευμάτων, χρησιμοποιούμε ένα αντικείμενο ρεύματος ofstream, istream και fstream.

Για παράδειγμα, οι παρακάτω προτάσεις γράφουν τη λέξη «αρχή», τον αριθμό 20 και τη λέξη «τέλος» σε ξεχωριστές γραμμές σε ένα αρχείο εξόδου με όνομα **out.txt**.

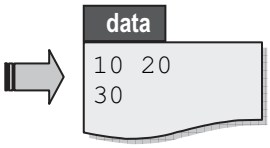
```
ofstream myoutfile;
myoutfile.open("out.txt");
myoutfile  << "αρχή" << endl
           << 20 << endl << "τέλος";
```



> Για να μπορούμε να χρησιμοποιήσουμε τις κλάσεις ρευμάτων σε αρχεία, θα πρέπει να συμπεριλάβουμε στο πρόγραμμά μας το αρχείο κεφαλίδας fstream στο οποίο δηλώνονται. Για παράδειγμα, #include <fstream>

Το παρακάτω πρόγραμμα ανοίγει ένα αρχείο εξόδου με όνομα **data**, στο οποίο γράφει τους αριθμούς 10, 20 και 30.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
int main()
{
   int i=10,a=20;
   ofstream myfile;
   myfile.open("data");
   myfile << i << " " << a << endl;
   myfile << 30;
   myfile.close();
   return 0;
}
```



Δημιουργία ενός αντικειμένου ρεύματος εξόδου με όνομα **myfile**.

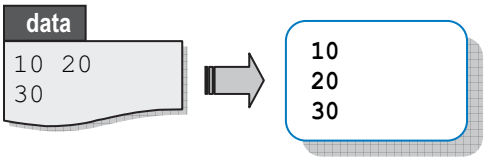
Άνοιγμα του αρχείου **data** και σύνδεσή του με το αντικείμενο **myfile**.

Εγγραφή στοιχείων στο αρχείο

Κλείσιμο του αρχείου.

Το παρακάτω πρόγραμμα ανοίγει το ίδιο αρχείο **data** για είσοδο δεδομένων, διαβάζει τους τρεις αριθμούς και τους εμφανίζει στην οθόνη.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
```



```
int main()
{
   int a,b,c;
   ifstream myfile;
   myfile.open("data");
   myfile >> a >> b >> c;
   myfile.close();
   cout << a << endl << b << endl << c;
   return 0;
}
```

Άνοιγμα του αρχείου **data** και σύνδεσή του με το αντικείμενο ρεύματος εισόδου **myfile**.

Ανάγνωση τριών αριθμών από το αρχείο και καταχώρισή τους στις μεταβλητές **a**, **b** και **c**.

Κλείσιμο του αρχείου.

Εμφάνιση των αριθμών στην οθόνη.

Στο παραπάνω παράδειγμα, για να γίνει σωστά η ανάγνωση του αρχείου, το αρχείο data θα πρέπει να περιέχει τουλάχιστον τρεις ακέραιους αριθμούς, χωρισμένους μεταξύ τους με έναν η περισσότερους χαρακτήρες λευκού διαστήματος.

> Η μορφοποιημένη ανάγνωση από ένα αρχείο προϋποθέτει ότι τα δεδομένα που περιέχει το αρχείο χωρίζονται μεταξύ τους με χαρακτήρες λευκού διαστήματος (δηλαδή, με κενά διαστήματα, χαρακτήρες tab, ή χαρακτήρες αλλαγής γραμμής.

Στην περίπτωση που το αρχείο δεν περιέχει ακέραιους αριθμούς, τότε η διαδικασία ανάγνωσης θα τερματιστεί μόλις εντοπιστεί ένα στοιχείο με ακατάλληλο τύπο. Ας υποθέσουμε ότι το αρχείο data περιέχει τα στοιχεία που φαίνονται στο διπλανό σχήμα. Αρχικά θα διαβαστεί σωστά ο αριθμός 10 και το επόμενο στοιχείο θα πρέπει να είναι επίσης τύπου int. Από τη στιγμή που το επόμενο στοιχείο είναι τύπου float, η διαδικασία της ανάγνωσης θα τερματιστεί με ένδειξη σφάλματος[2] στο ρεύμα εισόδου. Οι μεταβλητές b και c θα πάρουν απροσδιόριστες τιμές. Το ίδιο θα συμβεί αν το πρόγραμμά μας συναντήσει κάποιο αλφαριθμητικό στοιχείο κατά την ανάγνωση.

data
```
10
20.4
30
```

Η διαδικασία ανάγνωσης διακόπτεται όταν το στοιχείο που εξάγεται από το ρεύμα εισόδου, μέσω του τελεστή εξαγωγής (>>), δεν μπορεί να καταχωριστεί στη μεταβλητή που ακολουθεί.

data
```
10
λάθος
30
```

> Στην περίπτωση που το αρχείο δεν περιέχει τα στοιχεία που αναμένει ο τελεστής εξαγωγής (ανάλογα με τον τύπο της μεταβλητής που τον ακολουθεί), η διαδικασία της ανάγνωσης τερματίζεται με ένδειξη σφάλματος στο ρεύμα εισόδου.

Το πρόγραμμα που ακολουθεί ανοίγει το αρχείο data.txt για προσθήκη δεδομένων (append). Στο τέλος του αρχείου προστίθενται ο αριθμός 50 και το αλφαριθμητικό «Τέλος».

[2] Για τα σφάλματα ρευμάτων θα μιλήσουμε στη συνέχεια του κεφαλαίου.

Ανακεφαλαίωση

Ο τελεστής εισαγωγής << χρησιμοποιείται για τη μορφοποιημένη έξοδο σε αρχεία. Όπως και στην έξοδο στην οθόνη, αν παρεμβάλουμε στο ρεύμα εξόδου τους κατάλληλους χειριστές, μπορούμε να γράφουμε τα δεδομένα με τη μορφοποίηση που επιθυμούμε.

Το παρακάτω πρόγραμμα γράφει στο αρχείο data1.txt πραγματικούς αριθμούς, με ακρίβεια δύο δεκαδικών σε ένα πλάτος πεδίου επτά θέσεων.

18_write_decimals.cpp

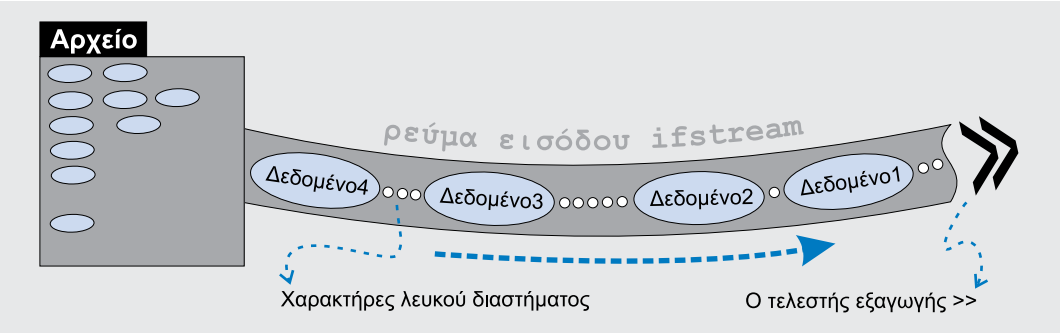
```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
int main()
{
   float i;
   ofstream myfile;
   myfile.open("data1.txt");
   for (i=10;i<=12;i=i+0.4)
      myfile<<setw(7)<<fixed<<setprecision(2)<<i<<endl;
   myfile.close();
   return 0;
}
```

data1.txt

```
10.00
10.40
10.80
11.20
11.60
```

Η μορφοποιημένη είσοδος από ένα αρχείο επιτυγχάνεται με τον τελεστή εξαγωγής >>. Φανταζόμαστε ένα τέτοιο ρεύμα εισόδου σαν ένα κανάλι το οποίο παρέχει δεδομένα τα οποία χωρίζονται μεταξύ τους με χαρακτήρες λευκού διαστήματος (white space characters), όπως φαίνεται στο Σχήμα 18.7.

Κάθε φορά που εφαρμόζεται ο τελεστής εξαγωγής, εξάγεται από το κανάλι το αμέσως επόμενο δεδομένο που προέρχεται από το συνδεδεμένο με το κανάλι αρχείο.



Σχήμα 18.7 *Μορφοποιημένη είσοδος από αρχείο*

Παράμετροι γραμμής εντολών

Μέχρι τώρα γνωρίζαμε ότι οι συναρτήσεις της C++ μπορούν να έχουν παραμέτρους μέσω των οποίων το πρόγραμμα που τις καλεί μπορεί να μεταβιβάσει κάποιες πληροφορίες.

Τι γίνεται όμως με τη συνάρτηση main();

Μπορεί να έχει παραμέτρους;

Πώς θα μεταβιβάσουμε τιμές σε αυτές τις παραμέτρους;

Η συνάρτηση main() μπορεί να έχει δύο (και μόνο δύο) συγκεκριμένες παραμέτρους, οι οποίες παίρνουν τιμές από τη γραμμή εντολών κατά την εκτέλεση του προγράμματος στο περιβάλλον του λειτουργικού συστήματος.

Οι παράμετροι αυτές είναι οι argc και argv[]:

int main(int *argc*, char **argv[]*)

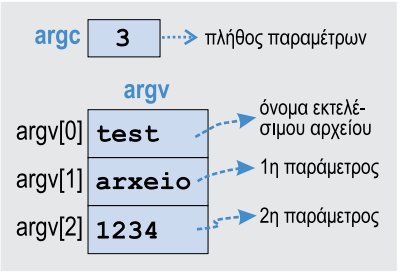
Η παράμετρος ***argc*** μεταβιβάζει στη main() το πλήθος των παραμέτρων με τις οποίες καλείται το πρόγραμμα. Η τιμή της argc είναι τουλάχιστον 1, αφού στο πλήθος των παραμέτρων υπολογίζεται και το όνομα του προγράμματος.

Η παράμετρος ***argv[]*** είναι ένας πίνακας που περιέχει δείκτες σε char. Η πρώτη θέση μνήμης του πίνακα argv[] δείχνει σε ένα σύνολο χαρακτήρων το οποίο περιέχει το όνομα του εκτελέσιμου αρχείου του προγράμματος (π.χ. test.exe). Οι υπόλοιποι δείκτες, στις θέσεις μνήμης του πίνακα argv[], δείχνουν σε σύνολα χαρακτήρων που περιέχουν τις παραμέτρους με τις οποίες έχει κληθεί το πρόγραμμα από τη γραμμή εντολών (*command line*).

Έστω ότι γράφουμε ένα πρόγραμμα σε ένα αρχείο test.cpp, το μεταγλωττίζουμε, και προκύπτει ένα εκτελέσιμο αρχείο test.exe. Αν εκτελέσουμε το test.exe με δύο παραμέτρους, για παράδειγμα:

```
test arxeio 1234
```

τότε τα περιεχόμενα των παραμέτρων argc και argv[] της main() διαμορφώνονται όπως φαίνονται στο σχήμα.



Ο παρακάτω κώδικας δείχνει με σαφήνεια τη χρήση των argc και argv[]. Εμφανίζει το πλήθος των παραμέτρων, το όνομα του εκτελέσιμου αρχείου, καθώς και τις υπόλοιπες παραμέτρους μία προς μία.

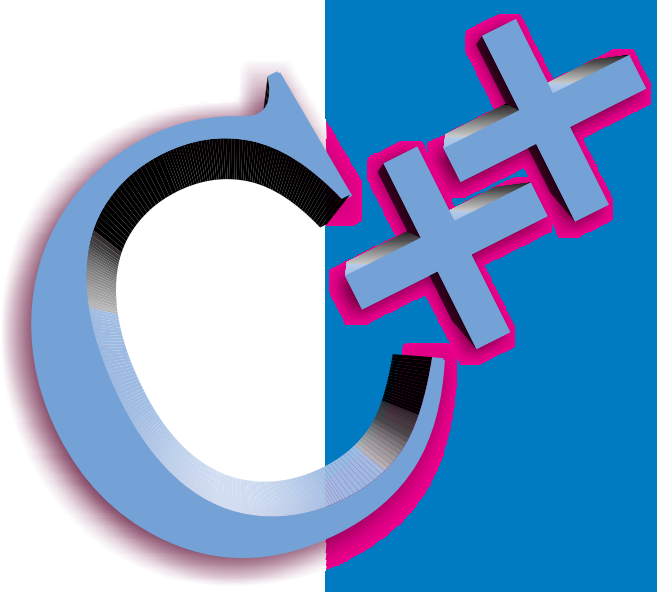
18_main_parameters.cpp

```
#include <iostream>
using namespace std;
int main(int argc,char *argv[])
```

Κεφάλαιο

19

Αναδρομή, αναζήτηση και ταξινόμηση



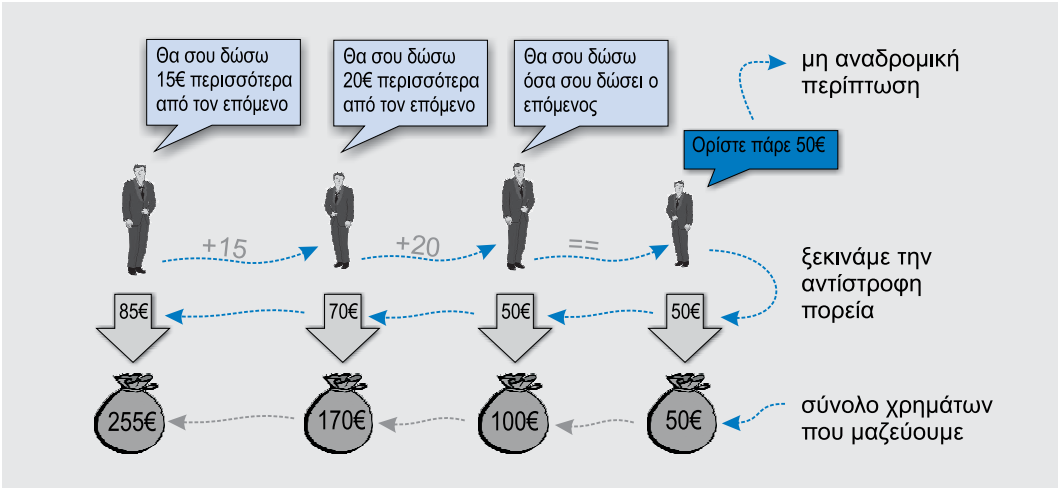
Αναδρομή

Αναδρομή είναι η διαδικασία κατά την οποία ορίζουμε μια λειτουργία σε σχέση με τον εαυτό της. Αν και είναι λίγο δυσνόητο, ας δούμε ένα απλό παράδειγμα:

Ας υποθέσουμε ότι κάνουμε έναν έρανο στον δρόμο. Αν ζητήσουμε χρήματα από τον πρώτο που θα συναντήσουμε μπορεί να μας απαντήσει ως εξής: «*Θα σου δώσω 15€ περισσότερα από όσα θα σου δώσει ο επόμενος*», πάμε στον επόμενο και αυτός απαντάει με τον ίδιο τρόπο: «*Θα σου δώσω 20€ περισσότερα από όσα θα σου δώσει ο επόμενος*» ή «*Θα σου δώσω όσα θα σου δώσει ο επόμενος*» κ.ο.κ. Έστω ότι αυτό συνεχίζεται και με άλλους πολίτες, μέχρι να βρούμε κάποιον ο οποίος δεν θα απαντήσει έτσι και θα μας δώσει τα χρήματα. Τότε αρχίζουμε την αντίστροφη πορεία και πάμε σε όλους όσους μας υποσχέθηκαν χρήματα, και τους ζητάμε να μας δώσουν τα χρήματα που τους αναλογούν.

Το Σχήμα 19.1 που ακολουθεί δείχνει ένα παράδειγμα αναδρομικής διαδικασίας, όπως αναφέρθηκε παραπάνω. Οι πρώτοι τρεις πολίτες που συναντάμε μας υπόσχονται χρήματα αναλογικά με το ποσό που θα μας δώσει ο επόμενος. Όμως, ο τέταρτος δεν ακολουθεί την ίδια τακτική και μας δίνει 50€. Σε εκείνο το σημείο ξεκαθαρίζουν όλα και πλέον μπορούμε να μαζέψουμε τα χρήματα που μας υποσχέθηκαν οι προηγούμενοι ακολουθώντας αντίστροφη πορεία.

> Κάθε αναδρομική διαδικασία πρέπει να έχει μία *μη αναδρομική περίπτωση*, διαφορετικά θα συνεχίζεται επ' άπειρο.



Σχήμα 19.1 *Παράδειγμα αναδρομικής διαδικασίας*

Στη C++, μια συνάρτηση μπορεί να καλεί τον εαυτό της. Αν στο σώμα μιας συνάρτησης μια πρόταση καλεί την ίδια τη συνάρτηση, τότε η συνάρτηση λέγεται **αναδρομική** (recursive).

Ας δούμε την επόμενη συνάρτηση, η οποία επιστρέφει ως τιμή το παραγοντικό ενός αριθμού n (1*2*3 .....* n). Για παράδειγμα, το παραγοντικό του 3 είναι 6 (1*2*3) και του 8 είναι 40320 (1*2*3*4*5*6*7*8).

```
int par(int n)
{
   int i,p;
   p=1;
   for (i=1;i<=n;i++) p=p*i;
   return p;
}
```

Η προηγούμενη συνάρτηση υπολογίζει (με τον κλασικό τρόπο) και επιστρέφει ως τιμή το παραγοντικό ενός αριθμού **n**. Το παραγοντικό (n!) ενός αριθμού n είναι το γινόμενο όλων των ακέραιων αριθμών από το 1 μέχρι το n. Επίσης, το παραγοντικό (n!) ενός αριθμού n ορίζεται ως το γινόμενο του n και του παραγοντικού του προηγούμενου αριθμού δηλαδή n! = n * (n-1)!, δηλαδή το παραγοντικό του 6 είναι 6 * 5!. Έτσι ο υπολογισμός του παραγοντικού μπορεί να γίνει με αναδρομική διαδικασία.

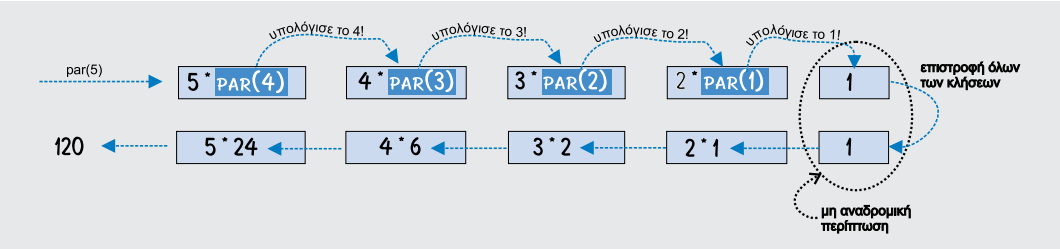
```
int par(int n)
{
   int p;
   if (n==1) return 1;
   p=n*par(n-1);
   return p;
}
```

Έλεγχος για μη αναδρομική περίπτωση

Για παράδειγμα, αν η παραπάνω συνάρτηση κληθεί με παράμετρο 5

```
a=par(5);
```

θα ακολουθήσει τη διαδικασία του επόμενου σχήματος σύμφωνα με την οποία η συνάρτηση θα επιστρέψει την τιμή της, που τελικά θα αποθηκευτεί στη μεταβλητή a:



Οι αναδρομικές κλήσεις της συνάρτησης par() θα σταματήσουν όταν ζητήσει το παραγοντικό του 1, το οποίο αποτελεί τη μη αναδρομική περίπτωση.

Αναζήτηση και ταξινόμηση

Η αναζήτηση και η ταξινόμηση αποτελούν τα πλέον «προκλητικά» θέματα στη διαχείριση δεδομένων. Η αναζήτηση αφορά τον εντοπισμό μιας συγκεκριμένης τιμής σε ένα σύνολο δεδομένων, ενώ η ταξινόμηση αφορά τη διάταξη ενός συνόλου δεδομένων σε αύξουσα ή φθίνουσα σειρά.

Ένα σύνολο δεδομένων σε έναν Η/Υ είναι αποθηκευμένο με έναν συγκεκριμένο τρόπο και καλείται *δομή δεδομένων*[1]. Η πιο απλή δομή δεδομένων είναι ένας πίνακας.

Η δομή δεδομένων μπορεί να είναι εσωτερική (αποθηκευμένη στην κεντρική μνήμη του Η/Υ) – όπως είναι οι πίνακες, οι συνδεδεμένες λίστες, τα δυαδικά δένδρα κ.λπ, ή εξωτερική, οπότε αναφερόμαστε σε δεδομένα αποθηκευμένα σε αρχεία.

Είτε πρόκειται για εσωτερικές είτε για εξωτερικές δομές, οι αλγόριθμοι που χρησιμοποιούνται για την αναζήτηση και την ταξινόμηση είναι ίδιοι. Αυτό που διαφέρει είναι ο τρόπος υλοποίησής τους για κάθε περίπτωση.

Βασικός στόχος κάθε αλγόριθμου αναζήτησης και ταξινόμησης είναι η ταχύτητα. Οι διάφοροι αλγόριθμοι που έχουν σχεδιαστεί διαφοροποιούνται μόνο ως προς την απόδοσή τους. Συνήθως οι πιο απλοί στη σύλληψη αλλά και στην υλοποίηση είναι οι λιγότερο αποδοτικοί. Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά οι πιο «κλασικές» μέθοδοι αναζήτησης και ταξινόμησης. Σε όλες τις περιπτώσεις, τα δεδομένα είναι αποθηκευμένα σε πίνακες. Όμως οι ίδιες μέθοδοι με μικρές παραλλαγές μπορούν να χρησιμοποιηθούν και σε δεδομένα αποθηκευμένα σε αρχεία περιφερειακών μονάδων αποθήκευσης.

Μια δομή δεδομένων μπορεί να αποτελείται και από αντικείμενα προσαρμοσμένων κλάσεων.

> Αν η δομή μας αποτελείται από αντικείμενα, τότε η αναζήτηση για κάποιο αντικείμενο βασίζεται σε μια συγκεκριμένη τιμή, ενώ η ταξινόμηση αφορά τη διάταξη των αντικειμένων με βάση κάποιο επιμέρους στοιχείου του.

Σειριακή αναζήτηση

Η *σειριακή αναζήτηση* αποτελεί τον απλούστερο τρόπο αναζήτησης τιμών μέσα σε μια δομή δεδομένων. Η λογική του αλγόριθμου της σειριακής αναζήτησης είναι ότι ελέγχει ένα-ένα τα δεδομένα με τη σειρά (από την αρχή προς το τέλος) μέχρι να εντοπίσει (ή να μην εντοπίσει) την τιμή που αναζητούμε.

Ο αλγόριθμος χρησιμοποιεί μια βοηθητική μεταβλητή (π.χ. τη found), στην οποία δίνει μια αρχική τιμή false. Μόλις εντοπίσει το δεδομένο που αναζητάμε, τότε καταχωρίζει

[1] Στις δομές δεδομένων θα αναφερθούμε διεξοδικά στο Κεφάλαιο 21.

Αντικειμενοστρεφής προσέγγιση

Ο τρόπος με τον οποίο διατυπώθηκαν οι συναρτήσεις αναζήτησης και ταξινόμησης ήταν περισσότερο διαδικαστικός παρά αντικειμενοστρεφής. Όμως, όταν θέλουμε τα προγράμματά μας να είναι αντικειμενοστρεφή, δεν αρκεί να χρησιμοποιούμε μια αντικειμενοστρεφή γλώσσα προγραμματισμού, αλλά πρέπει και ο σχεδιασμός τους να αποκτήσει μια αντικειμενοστρεφή προσέγγιση.

Αν θεωρήσουμε μια κλάση η οποία περιέχει ως μέλος της έναν πίνακα ακεραίων 100 θέσεων, τότε οι συναρτήσεις αναζήτησης και ταξινόμησης θα πρέπει να σχεδιαστούν ως μέθοδοι αυτής της κλάσης. Κάθε αντικείμενο αυτής της κλάσης θα έχει τη δυνατότητα αποθήκευσης 100 ακέραιων αριθμών. Η ταξινόμηση και η αναζήτηση θα υλοποιούνται με εφαρμογή της κατάλληλης μεθόδου στο συγκεκριμένο αντικείμενο.

Το παρακάτω πρόγραμμα δείχνει αυτή τη προσέγγιση. Η κλάση arithmoi διαθέτει ως μέλος τον πίνακα ακεραίων ar[] 100 θέσεων, αλλά και μεθόδους για τη διαχείρισή του: Η fill_random() συμπληρώνει τον πίνακα με τυχαίους αριθμούς, η display() τον εμφανίζει στην οθόνη, η sort() τον ταξινομεί και η find() εντοπίζει κάποιο στοιχείο του.

19_oop.cpp

```
#include <iostream>
#include <iomanip>
#include <cstdlib>

using namespace std;

class arithmoi
{
public:
   int ar[100];
   void fill_random();
   void display();
   void sort();
   bool find(int n);
};
void arithmoi::fill_random()
{
   int i;
   for (i=0;i<100;i++) ar[i]=rand();
}
void arithmoi::display()
{
   int i;
   for (i=0;i<100;i++) cout<<ar[i]<<endl;
}
void arithmoi::sort()
{
   int i,k,temp;
   bool found;
```

Η κλάση **arithmoi** διαθέτει ως μέλος έναν πίνακα ακεραίων 100 θέσεων.

Μέθοδοι της κλάσης **arithmoi** οι οποίες διαχειρίζονται τον πίνακα ακεραίων.

Η μέθοδος **fill_random()** συμπληρώνει τον πίνακα-μέλος με τυχαίους αριθμούς.

Η μέθοδος **display()** εμφανίζει τα περιεχόμενα του πίνακα στην οθόνη.

Η μέθοδος **sort()** ταξινομεί τον πίνακα χρησιμοποιώντας τον αλγόριθμο φυσαλίδας (bubble sort).

```
   for (i=1;i<100;i++)
   {
      found=false;
      for (k=99;k>=i;k--)
      {
         if (ar[k]<ar[k-1])
         {
            temp=ar[k];
            ar[k]=ar[k-1];
            ar[k-1]=temp;
            found=true;
         }
      }
      if (!found) break;
   }
}
bool arithmoi::find(int num)
{
   int i;
   bool found=false;
   for (i=0;i<100;i++)
   {
      if (ar[i]==num)
      {
         found=true;
         break;
      }
   }
   return found;
}
int main()
{
   arithmoi lotaria;
   int num;
   lotaria.fill_random();   // γεμίζει τον πίνακα του αντικειμένου lotaria με τυχαίους αριθμούς
   lotaria.sort();          // ταξινομεί τον πίνακα του αντικειμένου lotaria
   lotaria.display();       // εμφανίζει τον ταξινομημένο πλέον πίνακα
   cout<<"Δώσε αριθμό:";
   cin>>num;
   if (lotaria.find(num))
      cout<<"Βρέθηκε"<<endl;
   else
      cout<<"Δεν βρέθηκε"<<endl;
   return 0;
}
```

Η μέθοδος **find()** χρησιμοποιεί τον αλγόριθμο της σειριακής αναζήτησης για να εντοπίσει την τιμή της παραμέτρου της μέσα τον πίνακα.

Η μέθοδος **find()** στην περίπτωση που εντοπίσει την τιμή επιστρέφει **true** διαφορετικά **false**.

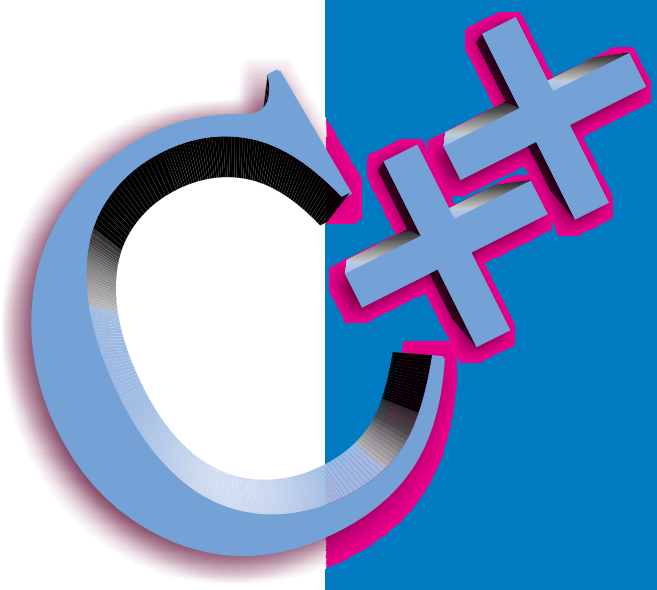
Δημιουργία του αντικειμένου **lotaria** της κλάσης **arithmoi**. Το αντικείμενο περιέχει έναν πίνακα ακεραίων 100 θέσεων.

Αναζητά μέσα στον πίνακα του αντικειμένου **lotaria** τον αριθμό που δόθηκε από το πληκτρολόγιο.

Κεφάλαιο

20

Δυναμική διαχείριση μνήμης



Δυναμική διαχείριση μνήμης

Όπως έχουμε δει μέχρι τώρα, η C++ διαχειρίζεται τη μνήμη είτε με τη χρήση απλών μεταβλητών είτε με τη χρήση αντικειμένων. Τα αντικείμενα, όπως γνωρίζουμε, αποθηκεύουν τα δεδομένα μέσα στις μεταβλητές-μέλη τους. Επίσης, η C++ μας δίνει τη δυνατότητα να δημιουργήσουμε πίνακες τόσο των βασικών τύπων δεδομένων όσο και αντικειμένων.

> Μπορούμε να θεωρήσουμε ότι οι βασικοί τύποι δεδομένων αποτελούν προκαθορισμένες κλάσεις, και οι μεταβλητές τους είναι αντικείμενα αυτών των κλάσεων.

Έτσι η δήλωση

```
int a,b;
```

δημιουργεί δύο «αντικείμενα» a και b της «κλάσης» int.

> Από τώρα και στο εξής, για τους σκοπούς αυτού του κεφαλαίου θα χρησιμοποιούμε τον όρο *αντικείμενο* είτε για μεταβλητές βασικών τύπων είτε για αντικείμενα κλάσεων.

Τα αντικείμενα και οι πίνακες αντικειμένων μπορεί να είναι τριών κατηγοριών, ανάλογα με τον τρόπο και τη θέση που δηλώνονται:

- Καθολικά
- Στατικά
- Τοπικά

Τα καθολικά και τα στατικά αντικείμενα καταλαμβάνουν συγκεκριμένο χώρο στη μνήμη και διατηρούν τα περιεχόμενά τους σε όλη τη διάρκεια εκτέλεσης του προγράμματος.

> Η κατανομή της μνήμης των καθολικών και των στατικών αντικειμένων λέγεται **στατική κατανομή** και οι θέσεις μνήμης δεσμεύονται κατά τη στιγμή της μεταγλώττισης.

Τα τοπικά αντικείμενα (λέγονται επίσης *αυτόματα* αντικείμενα) δεσμεύουν χώρο στη μνήμη και διατηρούν το περιεχόμενό τους όσο διαρκεί η εμβέλειά τους. Για παράδειγμα, όταν ένα τοπικό αντικείμενο δηλωθεί μέσα σε μια συνάρτηση, δεσμεύει χώρο μνήμης ανάλογο με την κλάση του. Όταν η συνάρτηση επιστρέφει, ο χώρος που δέσμευσε το αντικείμενο αποδεσμεύεται και τα περιεχόμενά του χάνονται.

Το ίδιο ακριβώς γίνεται και όταν δηλωθεί ένα αντικείμενο μέσα σε μια σύνθετη πρόταση. Μόλις τελειώσει η σύνθετη πρόταση, το αντικείμενο καταστρέφεται και αποδεσμεύεται ο χώρος μνήμης που καταλάμβανε.

Η κατανομή της μνήμης των τοπικών αντικειμένων λέγεται **αυτόματη κατανομή** και οι θέσεις μνήμης δεσμεύονται προσωρινά (κατά τη δήλωσή τους) και αποδεσμεύονται μόλις λήξει η εμβέλεια τους.

Έτσι, οι προτάσεις

```
char a,b;
int num[100];
class rectangle
{
public:
   float plevra_a;
   float plevra_b;
   .....
}rec1;
```

δεσμεύουν δύο byte για τις μεταβλητές a και b, 400 byte (4 x 100) για τον πίνακα num και 8 byte για το αντικείμενο rec1 (τέσσερα για κάθε μεταβλητή-μέλος του). Οι θέσεις αυτές δεσμεύονται τη στιγμή της δήλωσης και αποδεσμεύονται μόλις λήξει η εμβέλεια των αντικειμένων.

Πρέπει να έχουμε υπόψη ότι δεν είναι δυνατόν να μεταβάλλουμε το μέγεθος ενός πίνακα. Ο πίνακας έχει πάντα το μέγεθος που ορίζουμε με την πρόταση δήλωσής του.

Αν όμως θέλουμε, για παράδειγμα, έναν πίνακα στον οποίο θα καταχωρίζουμε τους αριθμούς των αυτοκινήτων που περνάνε μπροστά από ένα σημείο παρατήρησης, το ερώτημα είναι πόσο μεγάλος πρέπει να είναι αυτός ο πίνακας. Αν τον ορίσουμε μικρό (π.χ. με 100 θέσεις) είναι πιθανό να μην επαρκεί επειδή μπορεί να περάσουν περισσότερα αυτοκίνητα. Αν τον ορίσουμε αρκετά μεγάλο, προβλέποντας (αν αυτό είναι δυνατόν) τη μέγιστη κίνηση που θα μπορούσαμε να έχουμε (π.χ. με 1000 θέσεις), το πιθανότερο είναι να σπαταλήσουμε άσκοπα μνήμη που δε θα χρησιμοποιήσουμε ποτέ.

Τι γίνεται λοιπόν σε αυτές τις περιπτώσεις;

Η λύση βρίσκεται στον μηχανισμό δυναμικής κατανομής μνήμης που διαθέτει η C++.

Η C++ διαθέτει έναν μηχανισμό **δυναμικής κατανομής** μνήμης μέσω του οποίου μπορούμε να δεσμεύουμε περισσότερη μνήμη αν την έχουμε ανάγκη και να αποδεσμεύουμε μνήμη που δεν χρειαζόμαστε.

Δυναμική κατανομή μνήμης

Στο διπλανό σχήμα απεικονίζεται ο τρόπος με τον οποίο διαχειρίζεται η C++ τη μνήμη. Ένα τμήμα μνήμης χρησιμοποιείται για την αποθήκευση του κώδικα του προγράμματος και ένα γειτονικό τμήμα καταλαμβάνουν οι καθολικές και στατικές μεταβλητές.

Η **στοίβα** (stack) είναι ένας χώρος μνήμης στον οποίο αποθηκεύονται οι τοπικές μεταβλητές και αυξάνεται ή μειώνεται ανάλογα με τις ανάγκες του προγράμματος.

Ο υπόλοιπος αδιάθετος χώρος μνήμης μεταξύ της στοίβας και του χώρου των γενικών μεταβλητών λέγεται **σωρός** (heap).

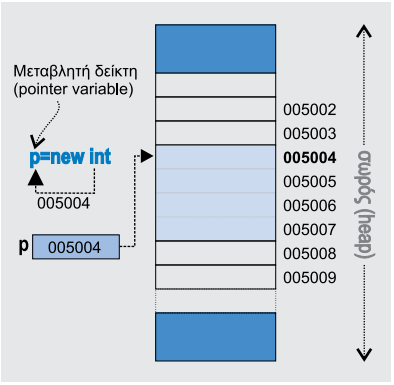
Στη δυναμική κατανομή μνήμης, κατά τον χρόνο εκτέλεσης (runtime) του προγράμματος μπορούν να διατεθούν τμήματα μνήμης (memory blocks) από τον σωρό, να χρησιμοποιηθούν, και να αποδεσμευτούν όταν δεν χρειάζονται.

Στη C++, η δυναμική κατανομή μνήμης επιτυγχάνεται με τους τελεστές new και delete. Ο μηχανισμός και οι τελεστές που χρησιμοποιούνται στη δυναμική κατανομή μνήμης κάνουν εκτενή χρήση των δεικτών.

Ο τελεστής new

Ο τελεστής new δεσμεύει από τον σωρό ένα τμήμα (block) μνήμης με μέγεθος (σε byte) όσο το μέγεθος του τύπου των αντικειμένων στα οποία αναφέρεται.

Ο τελεστής new επιστρέφει έναν δείκτη σε αντικείμενα αυτού του τύπου. Στην ουσία, επιστρέφει ως τιμή τη διεύθυνση της πρώτης θέσης μνήμης του τμήματος που δέσμευσε.



Για παράδειγμα, ο κώδικας που ακολουθεί

```
int *p;
p=new int;
*p=44;
```

δεσμεύει χώρο στον σωρό για ένα αντικείμενο τύπου int (4 byte). Η παράσταση new int επιστρέφει ως τιμή έναν δείκτη τύπου int με τιμή τη διεύθυνση της πρώτης θέσης μνήμης που δέσμευσε για αυτό το αντικείμενο. Η διαδικασία αυτή φαίνεται εποπτικά στο παραπάνω σχήμα. Η πρόσβαση στο νέο αντικείμενο γίνεται πλέον μέσω του δείκτη p. Για παράδειγμα, η πρόταση *p=44 καταχωρίζει στη θέση που δεσμεύτηκε το 44.

Μέθοδοι δόμησης αντιγράφου

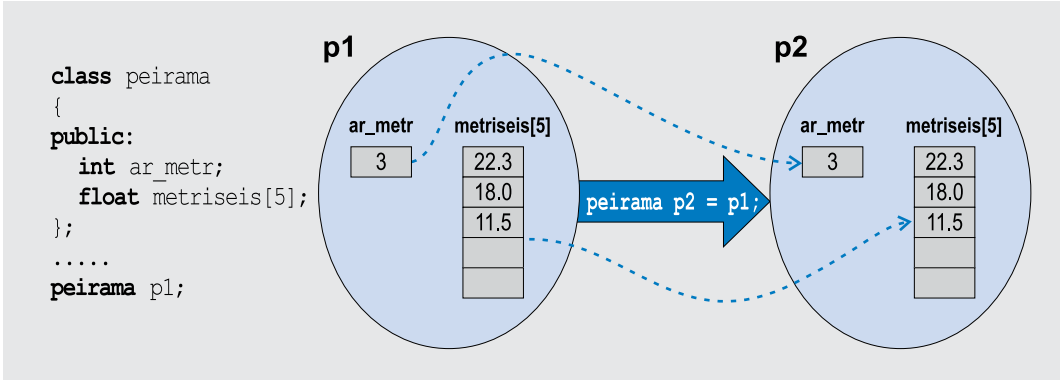
Μια μέθοδος δόμησης αντιγράφου (copy constructor) είναι μια ειδική μέθοδος δόμησης της C++, η οποία χρησιμοποιείται για να δημιουργήσει ένα αντίγραφο ενός υπάρχοντος αντικειμένου. Μια μέθοδος δόμησης αντιγράφου καλείται στις παρακάτω περιπτώσεις:

- Όταν σε μια πρόταση δήλωσης δημιουργείται ένα νέο αντικείμενο με αρχική τιμή ένα άλλο αντικείμενο του ίδιου τύπου.
- Όταν ένα αντικείμενο μεταβιβάζεται με τιμή ως παράμετρος σε μια συνάρτηση.
- Όταν μια συνάρτηση επιστρέφει ως τιμή ένα αντικείμενο.

Αν δεν έχει οριστεί μέθοδος δόμησης αντιγράφου για μια κλάση, ο μεταγλωττιστής δημιουργεί μία μόνος του, η οποία εξασφαλίζει τη λεγόμενη «ρηχή» αντιγραφή (shallow copy) – με αυτήν αντιγράφονται τα μέλη του ενός αντικειμένου στο άλλο, bit προς bit.

Αν η κλάση δεν περιέχει δείκτες με δυναμική κατανομή μνήμης, τότε η προκαθορισμένη μέθοδος δόμησης αντιγράφου που δημιουργεί ο μεταγλωττιστής είναι επαρκής και δεν χρειάζεται να μας απασχολεί ο ορισμός μιας εξειδικευμένης μεθόδου δόμησης αντιγράφου. Όταν όμως η κλάση χρησιμοποιεί δυναμική κατανομή μνήμης, τότε πρέπει οπωσδήποτε να ορίσουμε μια ξεχωριστή μέθοδο δόμησης αντιγράφου.

Ας υποθέσουμε ότι έχουμε την απλή κλάση peirama του Σχήματος 20.4, η οποία διαθέτει δύο μέλη: μια μεταβλητή τύπου int στην οποία καταχωρίζεται το πλήθος των μετρήσεων που πήραμε και έναν πίνακα τύπου float πέντε θέσεων, στον οποίο καταχωρίζονται οι τιμές μέχρι πέντε μετρήσεων. Στην περίπτωση που δημιουργηθεί ένα αντίγραφο ενός αντικειμένου της κλάσης peirama, με οποιονδήποτε από τους παραπάνω τρεις τρόπους που αναφέρθηκαν, στο αντίγραφο αντιγράφονται bit προς bit όλα τα δεδομένα του αρχικού αντικειμένου. Το αντικείμενο p2 είναι ένα ακριβές αντίγραφο του p1 και αυτό δεν δημιουργεί κανένα πρόβλημα.



Σχήμα 20.4 *Δημιουργία αντιγράφου αντικειμένου σε κλάση με στατική κατανομή μνήμης*

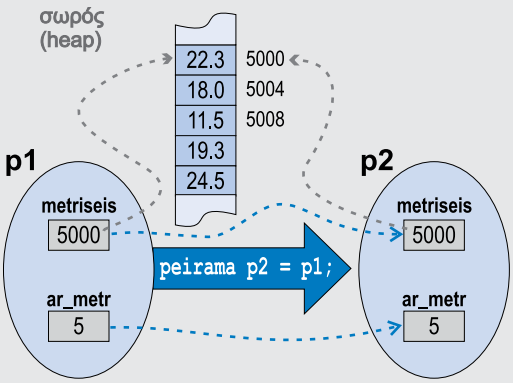
Ας υποθέσουμε τώρα ότι η κλάση peirama χρησιμοποιεί δυναμική κατανομή μνήμης για τη καταχώριση των μετρήσεων (Σχήμα 20.5). Σε αυτή την περίπτωση, στα αντικείμενα της κλάσης καταχωρίζεται μόνο ο δείκτης προς το τμήμα μνήμης που δεσμεύτηκε. Αν η αντιγραφή γίνει με χρήση της προκαθορισμένης μεθόδου δόμησης αντιγράφου της κλάσης peirama, το αντικείμενο p2, όπως είναι φυσικό, θα είναι ένα ακριβές αντίγραφο του p1. Επομένως, όπως φαίνεται στο Σχήμα 20.5, ο δείκτης metriseis θα περιέχει την ίδια τιμή και θα δείχνει στην ίδια περιοχή μνήμης που είχε δεσμεύσει το αντικείμενο p1. Αυτό δεν είναι καθόλου καλό διότι οι αλλαγές στις μετρήσεις του αντικειμένου p2, μέσω του δείκτη metriseis, θα επηρεάζουν και τις μετρήσεις του αντικειμένου p1! Παρατηρήστε τον παρακάτω κώδικα:

```
peirama p1(5);
p1.metriseis[2]=100;
cout<<"3η μέτρηση p1:"<<p1.metriseis[2]<<endl;
peirama p2=p1;
p2.metriseis[2]=200;
cout<<"3η μέτρηση p1:"<<p1.metriseis[2]<<endl;
```

```
3η μέτρηση p1:100
3η μέτρηση p1:200
```

Στη πρόταση p2.matriseis[2]=100; καταχωρίζεται το 200 ως τιμή της τρίτης μέτρησης του πειράματος p2. Όμως, επειδή όπως αναφέρθηκε παραπάνω, ο δείκτης metriseis του αντικειμένου p2 στην πραγματικότητα δείχνει στον χώρο που δέσμευσε το αντικείμενο p1, η καταχώριση αυτή επηρεάζει την τρίτη μέτρηση του πειράματος p1! Με τον τρόπο αυτό δεν δεσμεύτηκε ξεχωριστός χώρος για τις μετρήσεις του αντικειμένου-πειράματος p2. Τη λύση στο πρόβλημα θα δώσει η δημιουργία μιας εξειδικευμένης μεθόδου δόμησης αντιγράφου για την κλάση peirama.

```
class peirama
{
public:
   int ar_metr;
   float *metriseis;
   peirama(int ar);
};
peirama::peirama(int ar)
{
   ar_metr=ar;
   metriseis=new float[ar];
}
.....
peirama p1(5);
```

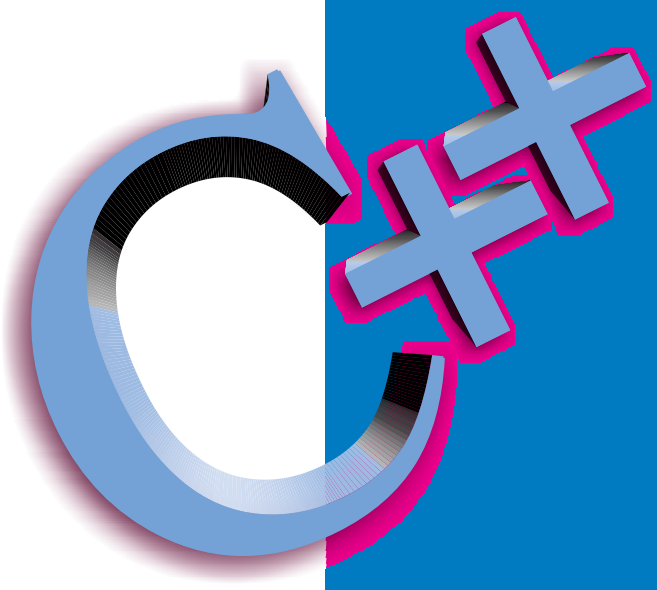


Σχήμα 20.5 *Δημιουργία αντιγράφου αντικειμένου, σε κλάση με δυναμική κατανομή μνήμης*

Κεφάλαιο

21

Δυναμικές δομές δεδομένων



Δυναμικές δομές δεδομένων

Μέχρι στιγμής, η μοναδική δομή δεδομένων που έχουμε γνωρίσει είναι ο πίνακας (array). Ως δομές καταχώρισης δεδομένων, οι πίνακες έχουν ένα μεγάλο μειονέκτημα: το μέγεθός τους πρέπει να καθορίζετε στην πρόταση δήλωσής τους, και δεν έχουμε τη δυνατότητα να το αλλάξουμε (αυξήσουμε ή μειώσουμε) κατά τη διάρκεια εκτέλεσης του προγράμματος.

Το αποτέλεσμα είναι ότι πρέπει να χρησιμοποιούμε μεγάλα μεγέθη πινάκων ώστε να είμαστε σίγουροι πως θα μπορούμε να αποθηκεύσουμε το μεγαλύτερο πιθανό πλήθος δεδομένων. Όμως η λύση αυτή οδηγεί σε υπερκατανάλωση μνήμης, η οποία τις περισσότερες φορές δεν είναι απαραίτητη. Επίσης, αν θέλουμε να διατηρούμε συνεχώς ταξινομημένα τα δεδομένα του πίνακα, η διαδικασία επαναταξινόμησης (ή παρεμβολής) μετά από την προσθήκη κάθε νέου στοιχείου είναι αρκετά χρονοβόρα, ιδιαίτερα σε πίνακες μεγάλου μεγέθους.

> Η λύση σε αυτά τα προβλήματα είναι η χρήση **δυναμικών δομών**, οι οποίες δεσμεύουν μνήμη όταν τη χρειάζονται και την αποδεσμεύουν όταν δεν τους είναι πια απαραίτητη.

Η χρήση δυναμικών δομών, εκτός από το ότι λύνει το πρόβλημα της κατανομής μνήμης, μας δίνει τη δυνατότητα να διατηρούμε τα δεδομένα ταξινομημένα συνεχώς, ώστε να μην είναι πια απαραίτητες οι χρονοβόρες διαδικασίες αναζήτησης και ταξινόμησης.

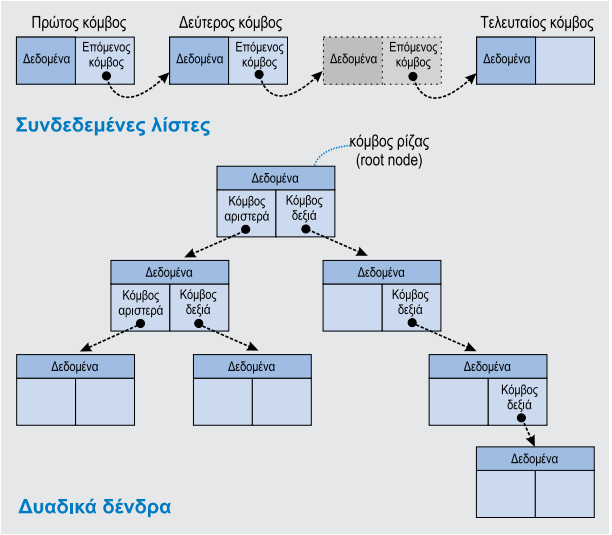
Το κεφάλαιο αυτό δεν φιλοδοξεί να καλύψει όλες τις δυναμικές δομές δεδομένων, οι οποίες από μόνες τους θα απαιτούσαν ένα ολόκληρο βιβλίο για να αναπτυχθούν, αλλά να κάνει μια μικρή εισαγωγή και να εξηγήσει τη δυναμική κατανομή μνήμης με πιο πρακτικό τρόπο.

Οι δυναμικές δομές δεδομένων που θα συζητηθούν στο παρόν κεφάλαιο είναι δύο:

- Οι συνδεδεμένες λίστες (linked lists)
- Τα δυαδικά δένδρα αναζήτησης (binary search trees)

Το δομικό στοιχείο των δυναμικών δομών δεδομένων είναι ο **κόμβος** (node). Μια δομή δεδομένων αποτελείται από πολλούς συνδεδεμένους κόμβους. Κάθε κόμβος είναι ένα σύνολο πληροφοριών που περιέχει δεδομένα, αλλά και πληροφορίες για τον επόμενο και τον προηγούμενο κόμβο. Η διάταξη των κόμβων διαφέρει ανάλογα με το είδος της δομής δεδομένων (συνδεδεμένη λίστα ή δυαδικό δένδρο). Στις δυναμικές δομές δεδομένων κάθε κόμβος δεσμεύει δυναμικά τον δικό του χώρο μνήμης. Επομένως μια δυναμική δομή δεδομένων καταλαμβάνει χώρο ο οποίος βρίσκεται όχι σε συνεχόμενα αλλά σε διάσπαρτα τμήματα στη μνήμη του Η/Υ.

Στις συνδεδεμένες λίστες, κάθε κόμβος έχει έναν προηγούμενο και έναν επόμενο κόμβο (εκτός φυσικά από τον πρώτο και τον τελευταίο). Επομένως, αν διαβάσουμε τα στοιχεία ενός κόμβου μιας λίστας, θα γνωρίζουμε ποιος είναι ο επόμενος! Η δομή των δυαδικών δένδρων διαφέρει. Κάθε κόμβος του δένδρου μπορεί να συνδεθεί με έναν κόμβο προς τα αριστερά του και έναν κόμβο προς τα δεξιά του. Ο πρώτος κόμβος του δένδρου λέγεται κόμβος «ρίζας». Αν διαβάσουμε τα στοιχεία ενός κόμβου ενός δυαδικού δένδρου γνωρίζουμε ποιος είναι ο επόμενος κόμβος προς τα αριστερά του και προς τα δεξιά του!



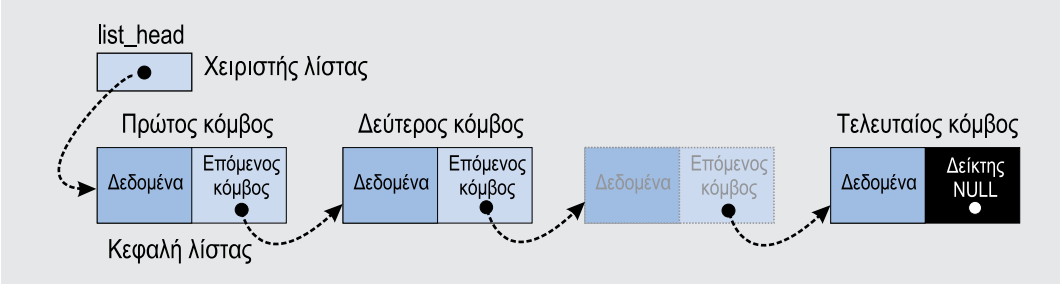
Συνδεδεμένες λίστες

Απλά συνδεδεμένη λίστα

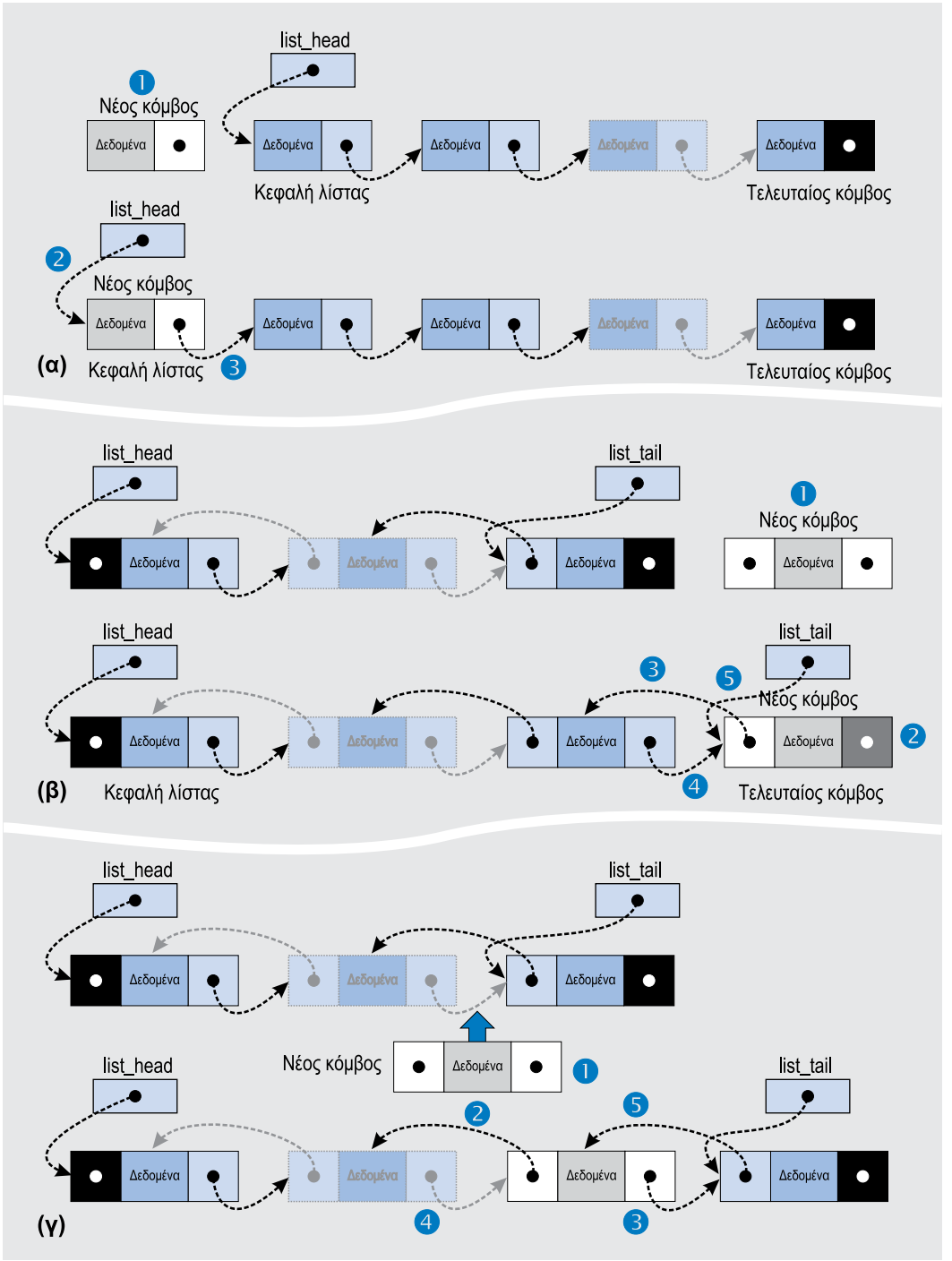
Η απλά συνδεδεμένη λίστα (singly linked list) αποτελείται από στοιχεία (κόμβους – nodes) τα οποία περιέχουν δεδομένα. Κάθε στοιχείο της λίστας περιέχει και έναν δείκτη ο οποίος δείχνει στο επόμενο στοιχείο της λίστας (Σχήμα 21.1). Επομένως, κάθε στοιχείο της λίστας αποτελείται από τα εξής:

- Το τμήμα δεδομένων όπου καταχωρίζονται τα δεδομένα του στοιχείου
- Τον δείκτη επόμενου κόμβου, που δείχνει στο επόμενο στοιχείο (κόμβο) της λίστας

Ο τελευταίος κόμβος περιέχει έναν δείκτη NULL ο οποίος σηματοδοτεί ότι ο συγκεκριμένος κόμβος είναι το τέλος (η «ουρά») της συνδεδεμένης λίστας.



Σχήμα 21.1 *Απλά συνδεδεμένη λίστα*



Σχήμα 21.3 *Προσθήκη νέου κόμβου σε συνδεδεμένη λίστα*

Αντικειμενοστρεφής σχεδιασμός στοίβας

Η προηγούμενη υλοποίηση στοίβας δεν μπορεί να θεωρηθεί αντικειμενοστρεφής, παρά το γεγονός ότι χρησιμοποιεί κλάσεις και αντικείμενα.

Ο παρακάτω κώδικας, παρά το γεγονός ότι φαινομενικά είναι παρόμοιος με τον προηγούμενο, χρησιμοποιεί μια εντελώς διαφορετική προσέγγιση για την υλοποίηση μιας στοίβας. Θεωρεί μια στοίβα ως ένα αντικείμενο το οποίο με τη σειρά του αποτελείται από επιμέρους αντικείμενα-κόμβους.

Για τον σκοπό αυτό ορίζεται μια νέα κλάση mystack, μέσα στην οποία ορίζεται η κλάση node. Η κλάση mystack περιέχει τη μεταβλητή-μέλος list_head η οποία είναι ο δείκτης προς την κορυφή της στοίβας. Η κλάση περιέχει επίσης τις μεθόδους push(), pop(), gettop(), isempty() και display() για τη διαχείριση της στοίβας.

21_stack_object.cpp

```
#include <iostream>
#include <string>
using namespace std;

class mystack
{
   class node
   {
   public:
      string data;
      node *next;
   };
   node *list_head;
public:
   void pop();
   bool push(string l);
   bool isempty();
   void gettop();
   void display();
   mystack();
   ~mystack();
};

mystack::mystack()
{
   list_head=NULL;
}

mystack::~mystack()
{
   node *temp;
   while (list_head!=NULL)
   {
      temp = list_head->next;
      delete list_head;
```

Η κλάση **mystack** ορίζει αντικείμενα στοίβας. Περιέχει τον χειριστή **list_head** (δείχνει στην κορυφή της στοίβας) καθώς και τις μεθόδους διαχείρισής της.

Η κλαση **node** ορίζεται μέσα στην κλάση **mystack** και είναι προσπελάσιμη μόνο από τα μέλη αυτής της κλάσης. Η κλάση **node** προσδιορίζει τη μορφή του κάθε κόμβου της στοίβας.

Δηλώσεις μεθόδων για τη διαχείριση των αντικειμένων στοίβας.

Δηλώσεις μεθόδων δόμησης και αποδόμησης της στοίβας.

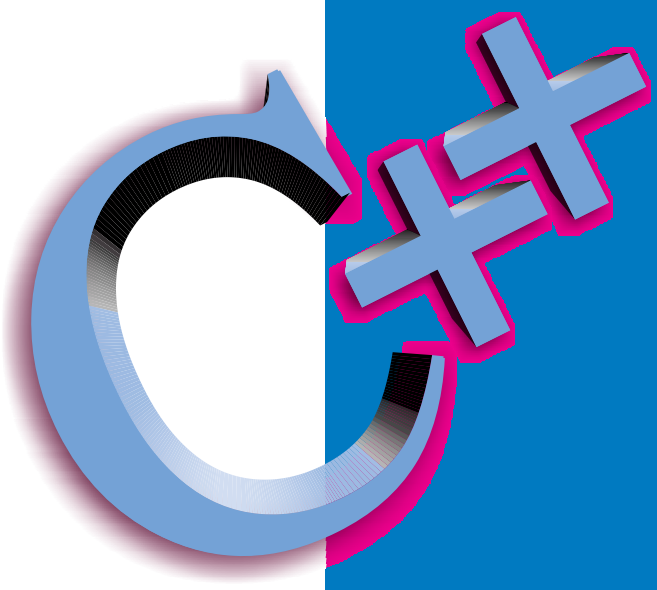
Στη μέθοδο δόμησης της κλάσης αποδίδεται αρχική τιμή NULL στον χειριστή της λίστας.

Η μέθοδος αποδόμησης της κλάσης αποδεσμεύει όλο τον χώρο που έχει δεσμεύσει δυναμικά η στοίβα.

Κεφάλαιο

22

Πρότυπα, εξαιρέσεις, και χώροι ονομάτων



Πρότυπα συναρτήσεων και κλάσεων

Τα πρότυπα (templates) συναρτήσεων και κλάσεων μας δίνουν τη δυνατότητα να σχεδιάζουμε συναρτήσεις και κλάσεις γενικού τύπου.

Τα πρότυπα συναρτήσεων και κλάσεων (ή *συναρτήσεις-πρότυπα* και *κλάσεις-πρότυπα*) χρησιμοποιούνται όταν έχουμε πολλά αντίγραφα κώδικα με την ίδια λογική, για διαφορετικούς όμως τύπους δεδομένων. Αν ένα σύνολο συναρτήσεων ή κλάσεων έχει την ίδια λειτουργικότητα για διαφορετικούς τύπους δεδομένων, τότε είναι οι καλύτερες υποψήφιες για να γραφούν ως πρότυπα.

Με απλά λόγια, τα πρότυπα μας δίνουν τη δυνατότητα να γράφουμε κώδικα σε μια γενική μορφή (generic programming) και να τον χρησιμοποιούμε για διαφορετικούς τύπους δεδομένων, χωρίς να χρειάζεται να τον ξαναγράψουμε!

> Τα πρότυπα συναρτήσεων και κλάσεων στη C++ εκπληρώνουν το όνειρο κάθε προγραμματιστή: τη δημιουργία επαναχρησιμοποιήσιμου κώδικα.

Πρότυπα συναρτήσεων

Γνωρίζουμε ότι, όταν ορίζουμε μια συνάρτηση, πρέπει να δηλώσουμε τον τύπο της καθώς και τον τύπο των παραμέτρων της. Αν θελήσουμε να εκτελούμε την ίδια ακριβώς λειτουργία, αλλά για διαφορετικούς τύπους δεδομένων, θα πρέπει να γράψουμε μια άλλη συνάρτηση ή να υπερφορτώσουμε την αρχική. Σε κάθε περίπτωση, όμως, θα χρειαστεί να γράψουμε νέο κώδικα.

Τα πρότυπα συναρτήσεων μπορούν να χειρίζονται διαφορετικούς τύπους δεδομένων χωρίς να χρειάζεται να γράψουμε ξεχωριστό κώδικα για κάθε έναν από αυτούς. Για ίδιες λειτουργίες σε διαφορετικούς τύπους δεδομένων, ο προγραμματιστής δεν χρειάζεται να γράψει διαφορετικές εκδόσεις της συνάρτησης: αρκεί να δημιουργήσει ένα πρότυπο συνάρτησης.

Ένα πρότυπο συνάρτησης ορίζεται με την παρακάτω σύνταξη:

template <class *Ttype*> τύπος *όνομα_συνάρτησης* (*λίστα παραμέτρων*)

{

    προτάσεις συνάρτησης;

}

Το προσδιοριστικό **template** προηγείται του τύπου και του ονόματος ενός προτύπου συνάρτησης. Το προσδιοριστικό ακολουθείται από τη δεσμευμένη λέξη **class** και το όνομα της παραμέτρου προτύπου ***Ttype***.

Η παράμετρος προτύπου (template parameter) ***Ttype*** είναι μια ειδική κατηγορία παραμέτρου η οποία χρησιμοποιείται για να μεταβιβάσει έναν τύπο δεδομένων στο πρότυπο συνάρτησης.

Όπως στις κανονικές συναρτήσεις οι τυπικές παράμετροι χρησιμοποιούνται για να μεταβιβάσουν τιμές στη συνάρτηση, έτσι και στα πρότυπα συναρτήσεων οι παράμετροι χρησιμοποιούνται για να μεταβιβάσουν τύπους δεδομένων.

> Τα πρότυπα συναρτήσεων χρησιμοποιούν τις παραμέτρους τους σαν να ήταν κανονικοί τύποι δεδομένων.

Ας υποθέσουμε ότι θέλουμε να φτιάξουμε μια συνάρτηση mymax() η οποία θα δέχεται δύο ορίσματα του ίδιου τύπου και θα επιστρέφει ως τιμή το μεγαλύτερο από τα δύο. Έτσι, ανάλογα με τον τύπο των ορισμάτων θα έπρεπε να φτιάξουμε διαφορετικές εκδόσεις της ίδιας συνάρτησης, π.χ. μία για ακέραιους τύπους (int) και μία για τύπους κινητής υποδιαστολής (float):

```
int mymax(int a, int b)
{
   return (a>b?a:b);
}

float mymax(float a, float b)
{
   return (a>b?a:b);
}
```

Αντί της **if** χρησιμοποιείται ο τελεστής συνθήκης ?: ο οποίος περιγράφηκε στο Κεφάλαιο 5. Η παράσταση επιστρέφει τη μεγαλύτερη τιμή μεταξύ των μεταβλητών **a** και **b**.

Αν θέλαμε να επεκτείνουμε την παραπάνω λειτουργία και για άλλους τύπους δεδομένων, θα πρέπει να υπερφορτώσουμε την παραπάνω συνάρτηση ισάριθμες φορές επαναλαμβάνοντας τον ίδιο κώδικα και αλλάζοντας μόνο τους τύπους δεδομένων. Αν όμως δημιουργήσουμε ένα πρότυπο συνάρτησης με την ίδια ακριβώς λειτουργικότητα, μπορούμε να το χρησιμοποιήσουμε για διαφορετικούς τύπους δεδομένων χωρίς να χρειάζεται να ξαναγράψουμε τη συνάρτηση.

Ο παρακάτω κώδικας ορίζει ένα πρότυπο συνάρτησης με όνομα mymax():

```
template <class T> T mymax(T a, T b)
{
   return (a>b?a:b);
}
```

Η δεσμευμένη λέξη template ορίζει τη δημιουργία ενός προτύπου συνάρτησης, ενώ η δεσμευμένη λέξη class προσδιορίζει την παράμετρο T ως παράμετρο προτύπου στην οποία θα μεταβιβαστεί ένας τύπος δεδομένων.

```
int main()
{
   float f1=23.4,f2=34.45;
   int i1=6,i2=28;
   displaymax(i1,i2);
   displaymax(i1,f2);
   return 0;
}
```

```
28
34.45
```

Στην πρώτη κλήση της displaymax() και στις δύο παραμέτρους προτύπου T1 και T2 θα μεταβιβαστεί ο τύπος int. Στη δεύτερη κλήση της, στην παράμετρο T1 θα μεταβιβαστεί ο τύπος int και στην T2 ο τύπος float. Ο μεταγλωττιστής θα δημιουργήσει κάθε φορά την ανάλογη έκδοση της συνάρτησης.

Επομένως...

Τα πρότυπα συναρτήσεων μπορούν να χρησιμοποιηθούν όταν χρειαζόμαστε ίδια λειτουργικότητα για διαφορετικούς τύπους δεδομένων.

Παρά το γεγονός ότι αποτελούν μια πολύ χρήσιμη πρακτική, πρέπει να σχεδιάζονται με ιδιαίτερη προσοχή και να ελέγχονται εξονυχιστικά πριν να τα χρησιμοποιούμε.

Ένα καλογραμμένο πρότυπο συνάρτησης μπορεί να μας γλιτώσει από αρκετό χρόνο και κώδικα, αλλά ένα προχειρογραμμένο μπορεί να δημιουργήσει προβλήματα που είναι αρκετά δύσκολο να εντοπιστούν.

Πρότυπα κλάσεων

Τα πρότυπα κλάσεων δίνουν τη δυνατότητα να σχεδιάζουμε κλάσεις οι οποίες έχουν μέλη ο τύπος των οποίων προσδιορίζεται από παραμέτρους προτύπου. Για παράδειγμα:

```
template <class T>
class zevgos
{
   T a;
   T b;
public:
   void set(T x, T y) {a=x; b=y;}
   T add() {return a+b;}
};
```

Η παράμετρος **T** μπορεί να χρησιμοποιηθεί για τη δήλωση μελών της κλάσης-προτύπου.

Δήλωση δύο ιδιωτικών μελών της κλάσης που θα είναι του τύπου **T**. Κατά τη δήλωση αντικειμένων της κλάσης **zevgos** καθορίζεται ο τύπος της παραμέτρου προτύπου **T**.

Η μέθοδος **set()** έχει δύο παραμέτρους τύπου **T**, και η μέθοδος **add()** επιστρέφει τιμή τύπου **T**.

Η κλάση zevgos χρησιμοποιείται για να αποθηκεύσει δύο στοιχεία οποιουδήποτε τύπου. Χρησιμοποιεί τη μέθοδο set() για την καταχώρισή τους, και διαθέτει τη μέθοδο add() η οποία επιστρέφει το άθροισμά τους.

Αν θέλουμε να δηλώσουμε ένα αντικείμενο της κλάσης zevgos στο οποίο θα καταχωρίσουμε δύο ακέραιους αριθμούς, θα πρέπει να χρησιμοποιήσουμε την παρακάτω σύνταξη:

```
zevgos<int> z1;
```

Στην περίπτωση που θέλουμε το αντικείμενο να αποθηκεύει ένα ζεύγος αριθμών διπλής ακρίβειας, η σύνταξη τροποποιείται ως εξής:

```
zevgos<double> z2;
```

Στην πρώτη περίπτωση, στην παράμετρο T του πρότυπου κλάσης zevgos, μεταβιβάζεται ο τύπος int, ενώ στη δεύτερη ο τύπος double.

Ο γενικός τρόπος σύνταξης ενός πρότυπου κλάσης είναι ο παρακάτω:

template <class *Ttype1,* class *Ttype2, ...*>

class *όνομα_κλάσης*

{

μέλη κλάσης;

}

Οι παράμετροι ***Ttype1***, ***Ttype2***, ... είναι παράμετροι στις οποίες μεταβιβάζονται τύποι δεδομένων. Αυτοί οι τύποι δεδομένων ορίζονται κατά τη δήλωση των αντικειμένων της κλάσης. Τα αντικείμενα θα δημιουργηθούν με βάση την εκάστοτε έκδοση της κλάσης.

Η σύνταξη της δήλωσης αντικειμένων μιας κλάσης-προτύπου είναι η ακόλουθη:

όνομα_κλάσης <*τύπος1, τυπος2, ...*> *αντικείμενα;*

όπου ***τύπος1***, ***τύπος2***, ... είναι τύποι δεδομένων οι οποίοι μεταβιβάζονται στις αντίστοιχες παραμέτρους ***Ttype1***, ***Ttype2***, ... του προτύπου της κλάσης.

Το παρακάτω πρόγραμμα δείχνει τη χρήση της κλάσης-προτύπου zevgos:

22_class_template.cpp

```
#include <iostream>
using namespace std;
template <class T>

class zevgos
{
   T a;
   T b;
public:
   void set(T x, T y) {a=x; b=y;}
   T add() {return a+b;}
};
```

Ορισμός του πρότυπου κλάσης **zevgos**.

```
int main()
{
   zevgos<int> z1;
   zevgos<double> z2;
   z1.set(4,6);
   z2.set(34.5612,12.00233);
   cout<<z1.add()<<endl;
   cout<<z2.add()<<endl;
   return 0;
}
```

Δημιουργία του αντικειμένου **z1** βασισμένο στην έκδοση της κλάσης **zevgos** για ακέραιους αριθμούς.

Δημιουργία του αντικειμένου **z2** βασισμένο στην έκδοση της κλάσης **zevgos** για αριθμούς διπλής ακρίβειας.

Καταχώριση τιμών στα μέλη των δύο αντικειμένων.

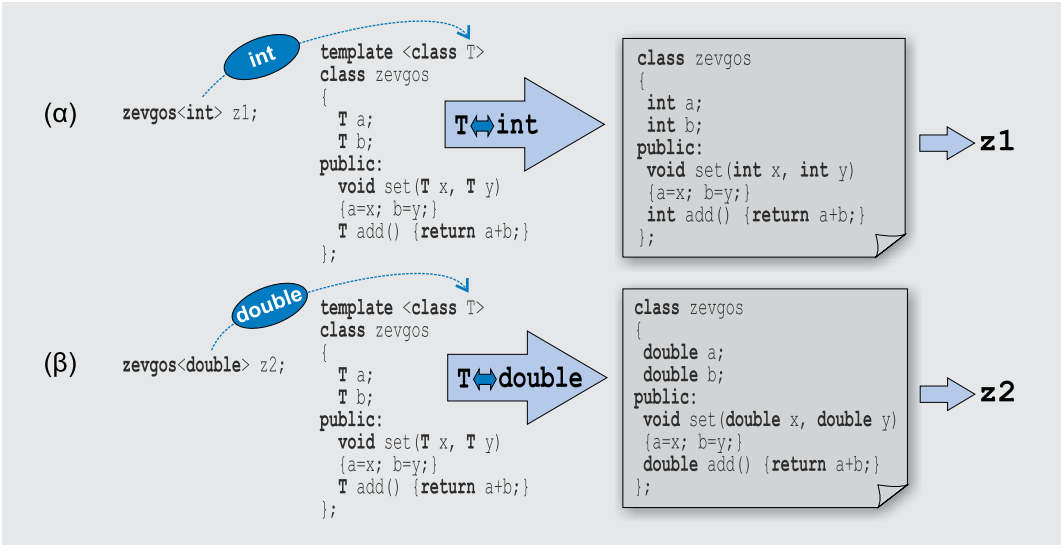
Εμφάνιση του αθροίσματος του ζεύγους τιμών.

Ιδιαίτερη σημασία πρέπει να δώσουμε στις προτάσεις:

```
zevgos<int> z1;
zevgos<double> z2;
```

με τις οποίες δημιουργούνται δύο αντικείμενα z1 και z2 της κλάσης zevgos. Όμως η μαγεία της πολυμορφικότητας έχει ως αποτέλεσμα τα δύο αυτά αντικείμενα να βασίζονται σε δύο διαφορετικές «εκδόσεις» της κλάσης zevgos.

Στην πρώτη περίπτωση, ο τύπος int μεταβιβάζεται στην παράμετρο T του πρότυπου κλάσης zevgos και έχει ως αποτέλεσμα το αντικείμενο z1 να δημιουργείται με βάση την «έκδοση» της κλάσης για ακέραιους αριθμούς. Στη δεύτερη περίπτωση, στην παράμετρο T μεταβιβάζεται ο τύπος double με αποτέλεσμα το αντικείμενο z1 να δημιουργείται με βάση την «έκδοση» της κλάσης για αριθμούς διπλής ακρίβειας. Το Σχήμα 22.2 δείχνει την παραπάνω διαδικασία.



Σχήμα 22.2 *Πρότυπα κλάσεων*

Εξαιρέσεις

Οι εξαιρέσεις (exceptions) είναι σφάλματα τα οποία μπορούν να παρουσιαστούν κατά τον χρόνο εκτέλεσης (run-time) ενός προγράμματος. Τα σφάλματα αυτά αφορούν καταστάσεις τις οποίες δεν μπορεί να χειριστεί το πρόγραμμά μας και συνήθως ανήκουν στις παρακάτω κατηγορίες:

- Διαίρεση με το μηδέν.
- Προβλήματα στο άνοιγμα, στην εγγραφή, ή στην ανάγνωση αρχείων.
- Εξάντληση της ελεύθερης μνήμης του συστήματος.
- Πρόσβαση εκτός των ορίων ενός πίνακα.

Οι εξαιρέσεις χρειάζονται άμεση αντιμετώπιση ώστε το πρόγραμμα να τις αντιμετωπίσει με επιτυχία, ή απλώς να ενημερώσει τον χρήστη για το συγκεκριμένο πρόβλημα.

Η C++ διαθέτει ένα ενσωματωμένο υποσύστημα χειρισμού εξαιρέσεων, το οποίο επιτρέπει τον χειρισμό των εξαιρέσεων με έναν αυτοματοποιημένο, δομημένο και ελεγχόμενο τρόπο. Αυτό το χαρακτηριστικό της γλώσσας επιτρέπει τα διάφορα τμήματα του προγράμματος να επικοινωνούν με ένα ενιαίο σύστημα διαχείρισης σφαλμάτων, το οποίο είναι ανεξάρτητο (και πολλές φορές ανεπτυγμένο ξεχωριστά) από το κυρίως πρόγραμμα.

Πώς αντιμετωπίζονταν οι εξαιρέσεις μέχρι τώρα

Πριν από την εμφάνιση ενός ενσωματωμένου συστήματος εξαιρέσεων, αναπτύχθηκαν διάφοροι τρόποι από τους προγραμματιστές για την αντιμετώπιση των σφαλμάτων.

Στον πλέον δημοφιλή από αυτούς, κάθε συνάρτηση που ήταν επιρρεπής σε κάποιο σφάλμα, επέστρεφε μια τιμή (error code) η οποία υποδήλωνε την επιτυχή της εκτέλεση ή το σφάλμα που προέκυψε. Το πρόγραμμα που καλούσε τη συνάρτηση έλεγχε την τιμή που επέστρεψε η συνάρτηση και ανάλογα χειριζόταν την κατάσταση σφάλματος (αν υπήρχε).

Τα προβλήματα που προέκυπταν με αυτόν το τρόπο διαχείρισης σφαλμάτων ήταν πολλά. Καταρχήν, ο κώδικας που καλούσε τη συνάρτηση έπρεπε να συμπληρωθεί με δομές ελέγχου για τον έλεγχο της τιμής που επέστρεφε η συνάρτηση, καθώς και με κώδικα αντιμετώπισης των σφαλμάτων. Επίσης, η συνάρτηση δεν μπορούσε να επιστρέψει τίποτε άλλο παρά μόνο έναν κωδικό σφάλματος.

Το πιο σημαντικό πρόβλημα όμως είναι ότι η διαχείριση των εξαιρέσεων ήταν διάσπαρτη σε όλο το πρόγραμμα και κάθε τμήμα του προγράμματος χειριζόταν τις δικές του εξαιρέσεις.

Η φιλοσοφία του χειρισμού εξαιρέσεων

Η φιλοσοφία του χειρισμού εξαιρέσεων είναι απλή. Η βασική ιδέα είναι ότι κάθε φορά που παρουσιάζεται μια εξαίρεση ενεργοποιείται μια σημαία σφάλματος. Ακολούθως, ένα σύστημα παρακολουθεί συνεχώς για την ενεργοποίηση αυτής της σημαίας σφάλματος και, μόλις την εντοπίσει, καλεί τον κώδικα χειρισμού του σφάλματος.

Η ενεργοποίηση της σημαίας σφάλματος καλείται *δημιουργία*, *έγερση* ή *κατάθεση* (throw) εξαίρεσης. Όταν κατατίθεται μια εξαίρεση, το σύστημα χειρισμού εξαιρέσεων τη συλλαμβάνει (catch) και εκτελεί τον κώδικα τον οποίο ο προγραμματιστής έχει καθορίσει για τη διαχείριση του συγκεκριμένου σφάλματος.

Για να μπορέσει να «συλληφθεί» μια εξαίρεση, ο κώδικας που δημιούργησε την εξαίρεση πρέπει να βρίσκεται μέσα σε ένα *τμήμα δοκιμής* (try-block).

Ένα από τα πιο δυνατά χαρακτηριστικά του χειρισμού εξαιρέσεων είναι ότι μια εξαίρεση μπορεί να κατατεθεί πέρα από την εμβέλεια μιας συνάρτησης. Αυτό πρακτικά σημαίνει ότι αν κάποια συνάρτηση, η οποία βρίσκεται «βαθιά» μέσα στον κώδικά μας, καταθέσει κάποια εξαίρεση, αυτή μπορεί να συλληφθεί από το τμήμα δοκιμής το οποίο βρίσκεται σε ανώτερο επίπεδο από τη συνάρτηση που προκάλεσε το σφάλμα.

Αυτό επιτρέπει τον προγραμματιστή να τοποθετεί τον κώδικα χειρισμού των εξαιρέσεων σε ένα σημείο, όπως στη συνάρτηση main() του προγράμματος.

Ο χειρισμός εξαιρέσεων της C++

Οι δομές χειρισμού εξαιρέσεων είναι από τα τελευταία χαρακτηριστικά που προστέθηκαν στη C++ και σχετίζονται με την ορολογία που αναφέρθηκε σε προηγούμενες παραγράφους.

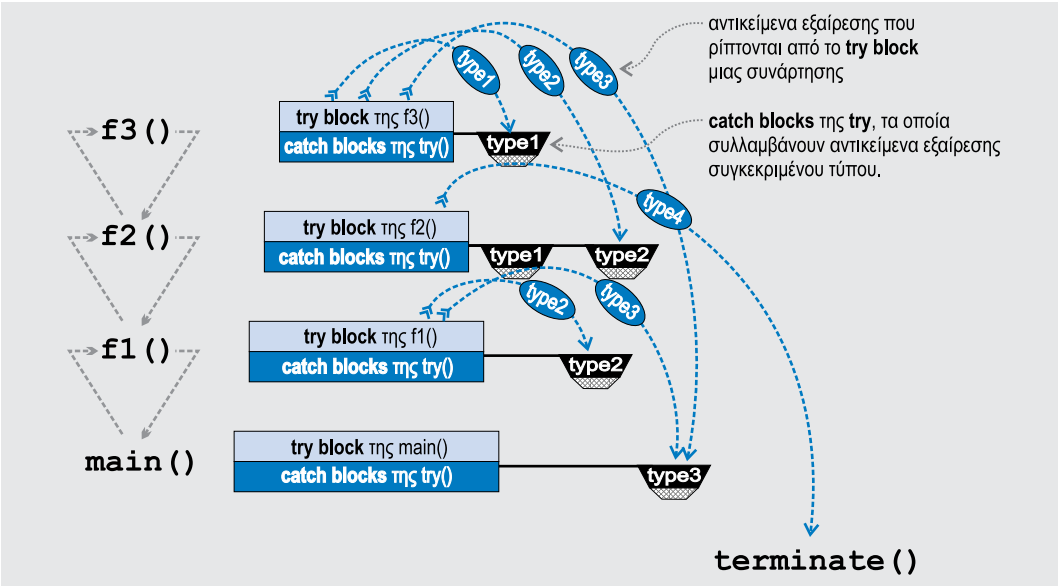
Το τμήμα δοκιμής για το οποίο θέλουμε να συλλαμβάνουμε εξαιρέσεις ξεκινάει με την εντολή **try** και περικλείεται σε αγκύλες. Μέσα από το τμήμα δοκιμής μπορούμε να καταθέτουμε εξαιρέσεις χρησιμοποιώντας την εντολή **throw**. Η εντολή **throw**, όπως θα δούμε αργότερα, ακολουθείται από ένα αντικείμενο το οποίο προσδιορίζει το είδος του σφάλματος. Αμέσως μετά από το τμήμα δοκιμής ακολουθεί το *τμήμα σύλληψης* (catch-block). Το τμήμα σύλληψης περιλαμβάνει τον κώδικα χειρισμού των σφαλμάτων. Όπως θα δούμε αργότερα, μπορεί να υπάρχουν πολλά τμήματα σύλληψης τα οποία συλλαμβάνουν διαφορετικά είδη εξαιρέσεων. Κάθε τμήμα σύλληψης ξεκινάει από την εντολή **catch**.

Το παρακάτω ενδεικτικό τμήμα κώδικα δείχνει τη βασική φιλοσοφία του χειρισμού εξαιρέσεων.

```
try
{
   ...
   throw εξαίρεση1
   throw εξαίρεση2
}
catch(type1 ε1)
{
   ...
}
catch(type2 ε2)
{
   ...
}
```

Η σύνθετη πρόταση της **try** περιέχει το τμήμα δοκιμής (try-block). Το τμήμα δοκιμής, μέσω της εντολής **throw**, μπορεί να καταθέτει εξαιρέσεις διαφορετικών τύπων.

Ένα ή περισσότερα τμήματα σύλληψης (catch blocks) ακολουθούν το τμήμα δοκιμής. Κάθε ένα τμήμα σύλληψης συλλαμβάνει αντικείμενα εξαίρεσης συγκεκριμένου τύπου.



Σχήμα 22.3 *Εξαιρέσεις που συμβαίνουν ... στα βαθιά*

Από τις εξαιρέσεις type1 , type2 και type3 που δημιουργούνται στη συνάρτηση f3(), μόνο η type1 συλλαμβάνεται από τα τμήματα σύλληψης της f3(). Οι υπόλοιπες προβιβάζονται στον κώδικα της συνάρτησης f2() από την οποία κλήθηκε η f3(). Τα τμήματα σύλληψης της f2() συλλαμβάνουν την εξαίρεση type2 της f1(), ενώ η εξαίρεση type3 της ίδιας συνάρτησης συλλαμβάνεται τελικά από τα τμήματα σύλληψης της main().

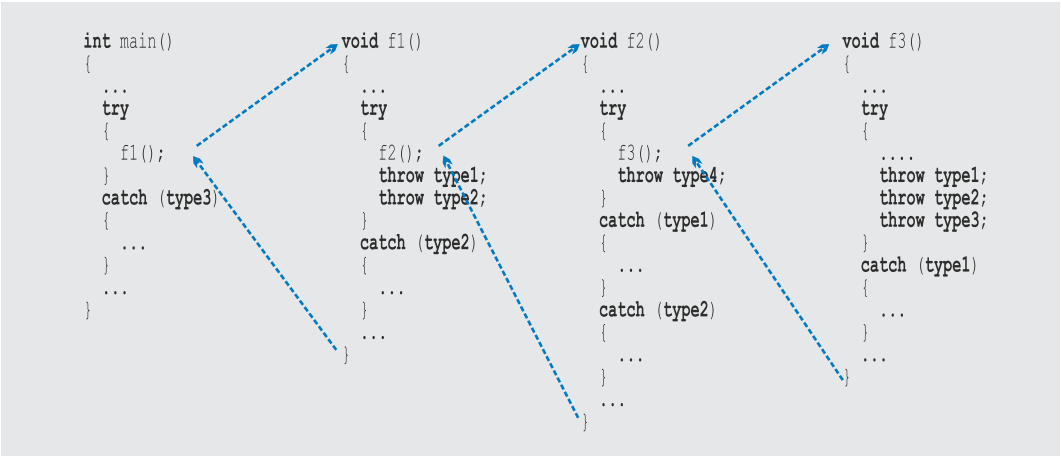
Η εξαίρεση type4, που δημιουργείται στη συνάρτηση f2(), δεν συλλαμβάνεται από κανένα τμήμα σύλληψης και τελικά καλεί τη συνάρτηση συστήματος terminate(), η οποία οδηγεί σε αντικανονικό τερματισμό του προγράμματος.

Με την ίδια λογική, οι εξαιρέσεις type2 και type3 της f1() συλλαμβάνονται από τα τμήματα σύλληψης της συνάρτησης f1() και της main(), αντίστοιχα.

Προϋπόθεση για να λειτουργήσει το σύστημα διαχείρισης εξαιρέσεων όπως ακριβώς περιγράφηκε είναι ότι κάθε συνάρτηση πρέπει να καλείται μέσα από ένα τμήμα δοκιμής (try-block). Το Σχήμα 22.4 δείχνει την περίπτωση διαδοχικής κλήσης τριών συναρτήσεων. Η κλήση κάθε συνάρτησης γίνεται μέσα από ένα τμήμα δοκιμής της συνάρτησης που την καλεί.

Κλάσεις εξαιρέσεων

Γνωρίζουμε ότι ένα τμήμα σύλληψης (catch-block) συλλαμβάνει μόνο αντικείμενα εξαίρεσης συγκεκριμένου τύπου. Αν λοιπόν περιοριστούμε στους βασικούς τύπους δεδομένων, η ποικιλία των εξαιρέσεων που μπορούμε να δημιουργήσουμε, αλλά και να



Σχήμα 22.4 *Παράδειγμα εξαιρέσεων που συμβαίνουν σε διαδοχικά καλούμενες συναρτήσεις*

συλλάβουμε, είναι πολύ περιορισμένη. Για αυτό τον λόγο δημιουργούμε προσαρμοσμένες κλάσεις τις οποίες χρησιμοποιούμε μόνο για τον χειρισμό εξαιρέσεων. Οι κλάσεις αυτές λέγονται *κλάσεις εξαιρέσεων*. Οι κλάσεις εξαιρέσεων συνήθως δεν περιέχουν κανένα μέλος. Το μόνο που χρησιμοποιούμε συνήθως από μια τέτοια κλάση είναι το όνομα της:

```
class exception
{
};
```

Το σώμα της κλάσης είναι κενό.

Η δημιουργία μιας εξαίρεσης, με χρήση της παραπάνω κλάσης, γίνεται με την πρόταση που ακολουθεί:

```
throw exception(); // Δημιουργεί ένα αντικείμενο εξαίρεσης της κλάσης exception
```

Η εντολή throw δεν μπορεί να ακολουθείται από τύπο δεδομένων, αλλά από αντικείμενο. Στη συγκεκριμένη περίπτωση, η παράσταση exception() χρησιμοποιεί την προκαθορισμένη συνάρτηση δόμησης της κλάσης exception και επιστρέφει ένα αντικείμενο αυτής της κλάσης.

Μια πρόταση της μορφής throw exception; θα ήταν λάθος!

Το παρακάτω τμήμα σύλληψης συλλαμβάνει αντικείμενα εξαίρεσης της κλάσης exception.

```
catch(exception) { προτάσεις ...}
```

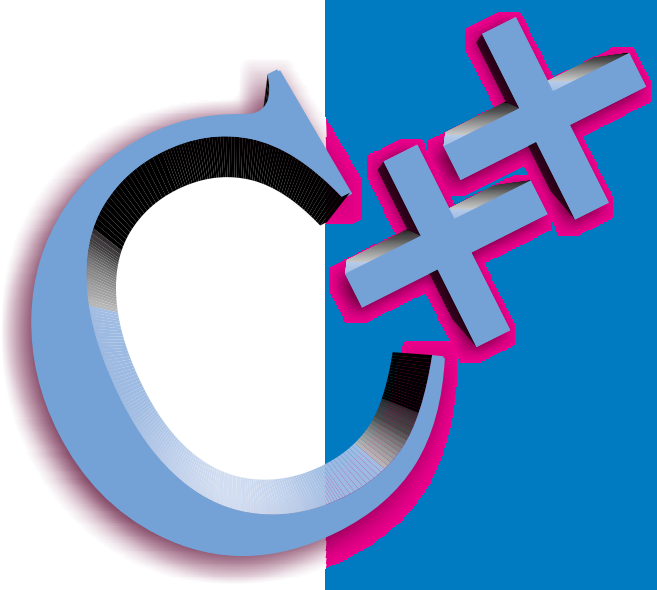
Συνήθως, μια κλάση εξαίρεσης δηλώνεται μέσα σε μια άλλη κλάση για την οποία θέλουμε να διαχειριστούμε περιπτώσεις εξαιρέσεων.

Το παρακάτω πρόγραμμα παρουσιάζει μια τέτοια περίπτωση:

Κεφάλαιο

23

Ο μεταγλωττιστής GCC



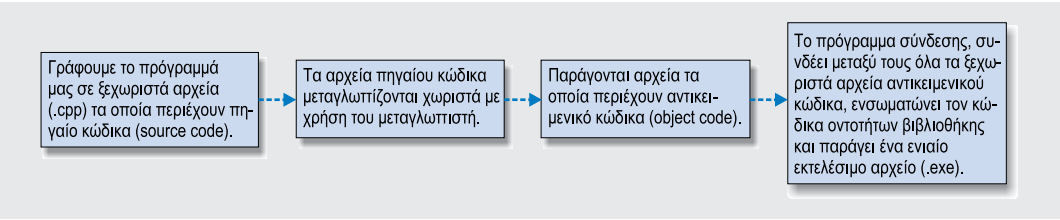
Ο μεταγλωττιστής GCC

Ο μεταγλωττιστής GCC είναι ο πιο δημοφιλής μεταγλωττιστής τόσο για τη γλώσσα C++ όσο και για τη C. Το ολοκληρωμένο περιβάλλον **Code::Blocks**, το οποίο περιέχεται στο συνοδευτικό CD και χρησιμοποιήθηκε για τα παραδείγματα και τις ασκήσεις του βιβλίου, χρησιμοποιεί τον μεταγλωττιστή GCC. Τον ίδιο μεταγλωττιστή χρησιμοποιούν και άλλα ολοκληρωμένα περιβάλλοντα όπως το παλαιό αλλά ευέλικτο DEV/C++. Ο μεταγλωττιστής GCC είναι ένα πρόγραμμα «δύο σε ένα»: μπορεί να χρησιμοποιηθεί ως μεταγλωττιστής αλλά και ως πρόγραμμα σύνδεσης (linker)! Επίσης, μπορεί να χρησιμοποιηθεί και εκτός κάποιου ολοκληρωμένου περιβάλλοντος, από τη γραμμή εντολών του λειτουργικού συστήματος.

Προγράμματα με πολλά πηγαία αρχεία

Μέχρι τώρα, όλα τα προγράμματα με τα οποία ασχοληθήκαμε αποτελούνταν από ένα μόνο πηγαίο αρχείο. Όμως, ένα πρόγραμμα της C++ μπορεί να χωριστεί σε περισσότερα πηγαία αρχεία, κάθε ένα από τα οποία μεταγλωττίζεται ξεχωριστά. Η τεχνική αυτή βοηθάει στη καλύτερη δομή του προγράμματος, και επίσης κάνει πιο εύκολη την αποσφαλμάτωση καθώς και τη συντήρησή του. Για παράδειγμα, η αλλαγή στον κώδικα μιας συνάρτησης θα οδηγήσει στη μεταγλώττιση μόνο του αρχείου που περιέχει τη συγκεκριμένη συνάρτηση, και όχι ολόκληρου του προγράμματος.

Η παραγωγή ενός εκτελέσιμου προγράμματος με τη C++ περνάει από δύο στάδια: Κατά το στάδιο της **μεταγλώττισης**, ο μεταγλωττιστής μεταφράζει τον πηγαίο κώδικα σε αντικειμενικό κώδικα (object code), δηλαδή σε κώδικα γλώσσας μηχανής. Κατά το δεύτερο στάδιο, το στάδιο της **σύνδεσης**, το πρόγραμμα σύνδεσης (linker ή συνδέτης) συνενώνει όλα τα αρχεία αντικειμενικού κώδικα, μαζί με κώδικα που προέρχεται από τυχόν βιβλιοθήκες που χρησιμοποιούνται, σε ένα ενιαίο εκτελέσιμο αρχείο. Το σχήμα που ακολουθεί δείχνει τα διαφορετικά αυτά στάδια:



Μεταγλώττιση και σύνδεση μεμονωμένων αρχείων

Για να μπορέσουμε να κατανοήσουμε πλήρως τα διαφορετικά στάδια της μεταγλώττισης και της σύνδεσης, θα πρέπει να εγκαταλείψουμε το εύχρηστο γραφικό περιβάλλον του Code::Blocks και να ανοίξουμε ένα παράθυρο γραμμής εντολών (command line). Η χρήση του μεταγλωττιστή από τη γραμμή εντολών θα μας δώσει τη δυνατότητα να χρη-

σιμοποιήσουμε λειτουργίες που είναι αυτοματοποιημένες στο ολοκληρωμένο περιβάλλον μεταγλώττισης. Ο μεταγλωττιστής GCC συνοδεύεται από διάφορα εκτελέσιμα αρχεία τα οποία, για την εγκατάσταση του Code::Blocks, βρίσκονται στον φάκελο **CodeBlocks\MinGW\bin**. Για να μπορούμε να τα εκτελούμε από οποιονδήποτε φάκελο, θα πρέπει να συμπεριλάβουμε στη μεταβλητή **path** του περιβάλλοντός μας τον συγκεκριμένο φάκελο. Η παρακάτω εντολή της γραμμής εντολών θα κάνει αυτή τη δουλειά:

```
c:\>path %path%;C:\Program Files\CodeBlocks\MinGW\bin
```

Τα εκτελέσιμα αρχεία που συνοδεύουν τον μεταγλωττιστή GCC χρησιμοποιούνται για τη μεταγλώττιση πηγαίων αρχείων, τόσο της C++ όσο και της C, για τη σύνδεσή τους, καθώς και για τη δημιουργία αρχείων βιβλιοθήκης.

Για τη μεταγλώττιση και σύνδεση αρχείων της C++ χρησιμοποιείται το εκτελέσιμο αρχείο **g++.exe**. Η χρήση του **g++.exe** για τη μεταγλώττιση ενός πηγαίου αρχείου της C++, για τη σύνδεση και για την παραγωγή του εκτελέσιμου αρχείου, είναι η ακόλουθη:

```
g++ -Wall πηγαίο_αρχείο.cpp -o εκτελεσιμο_αρχείο
```

Το πηγαίο αρχείο της C++

Το όνομα του εκτελέσιμου αρχείου που θα παραχθεί.

Η επιλογή **–Wall** ενεργοποιεί την εμφάνιση όλων των προειδοποιητικών μηνυμάτων (warnings) του μεταγλωττιστή.

Ας υποθέσουμε ότι μέσα στον φάκελο 23_separate (τον οποίο έχουμε αντιγράψει από το συνοδευτικό CD στον H/Y μας) έχουμε το ακόλουθο αρχείο πηγαίου κώδικα:

23_separate\hello.cpp

```
#include <iostream>
using namespace std;

int main()
{
   cout<<"Hello world!"<<endl;
   return 0;
}
```

Η ακόλουθη χρήση του g++ θα έχει ως αποτέλεσμα την μεταγλώττιση του πηγαίου αρχείου hello.cpp, τη σύνδεσή του με τα αρχεία βιβλιοθήκης της γλώσσας και την παραγωγή του τελικού εκτελέσιμου αρχείου hello.exe:

```
   \23_separate>g++ -Wall hello.cpp -o hello
```

Για να εκτελέσουμε τώρα το εκτελέσιμο αρχείο, χρησιμοποιούμε τη γραμμή εντολών γράφοντας απλά το όνομά του και πατώντας <Enter>:

```
   \23_separate>hello
Hello World!
```

Μεταγλώττιση και σύνδεση ξεχωριστών αρχείων

Το Σχήμα 23.1 δείχνει τη διαδικασία μεταγλώττισης ενός προγράμματος το οποίο έχει γραφεί σε τρία ξεχωριστά αρχεία πηγαίου κώδικα.

Ας υποθέσουμε ότι έχουμε χωρίσει το πρόγραμμά μας σε τρία διαφορετικά αρχεία πηγαίου κώδικα. Το πρώτο αρχείο **into.c** περιέχει τη συνάρτηση book(). Η συνάρτηση εμφανίζει στην οθόνη το κείμενο «*Η Γλώσσα C++ σε βάθος*».

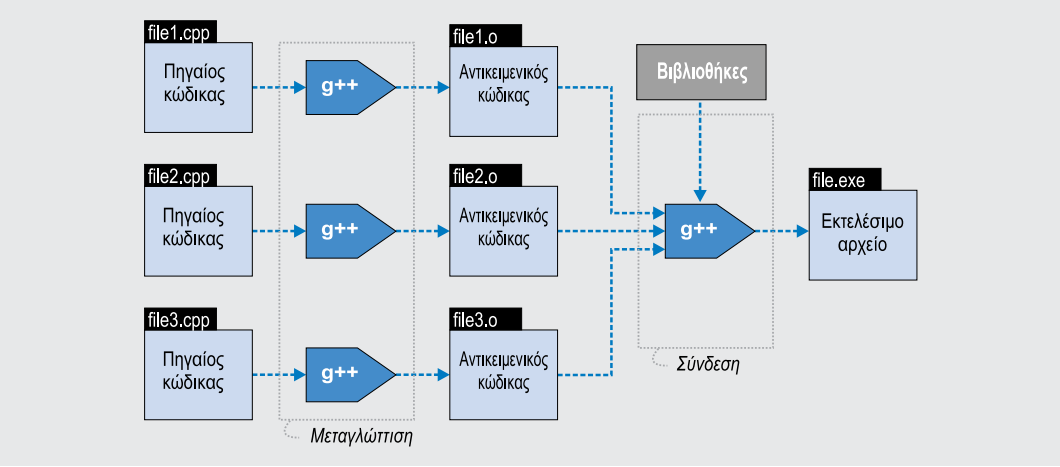
23_separate\intro.cpp

```
// Το αρχείο πηγαίου κώδικα intro.cpp
#include <iostream>
using namespace std;

void book()
{
   cout<<"Η γλώσσα C++ σε βάθος."<<endl;
}
```

Το δεύτερο αρχείο **bye.c** περιέχει τη συνάρτηση thanks(). Η συνάρτηση δέχεται ως παράμετρο μια συμβολοσειρά και εμφανίζει στην οθόνη το κείμενο «*Ευχαριστώ ##!*», όπου το ## είναι η συμβολοσειρά.

23_separate\bye.cpp

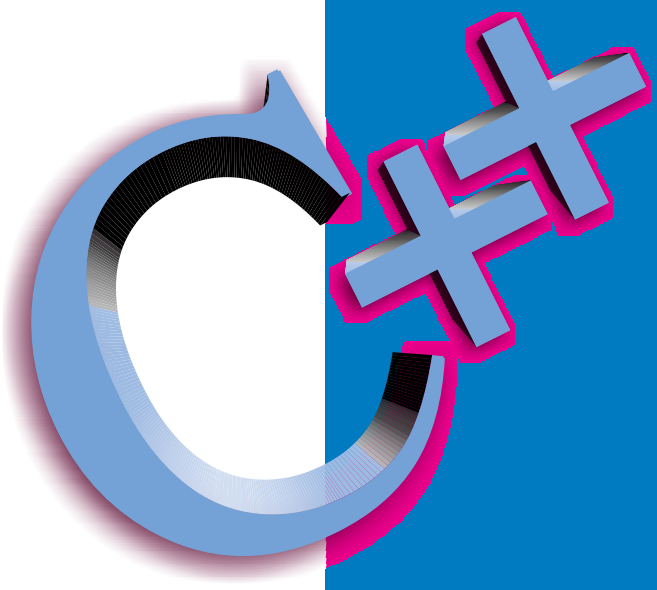
```
// Το αρχείο πηγαίου κώδικα bye.cpp
#include <iostream>
#include <string>

using namespace std;

void thanks(string frasi)
{
   cout<<"Ευχαριστώ "<<frasi<<endl;
}
```



Σχήμα 23.1 *Μεταγλώττιση και σύνδεση ξεχωριστών αρχείων με χρήση του GCC*

Κεφάλαιο 24

Μια ματιά στη καθιερωμένη βιβλιοθήκη προτύπων



Μια ματιά στην καθιερωμένη βιβλιοθήκη προτύπων

Τι είναι η καθιερωμένη βιβλιοθήκη προτύπων STL;

Η καθιερωμένη βιβλιοθήκη προτύπων (Standard Template Library - STL) είναι μια βιβλιοθήκη προγραμμάτων. Η STL είναι πλέον βασικό τμήμα της τυπικής βιβλιοθήκης της C++. Παρέχει στους προγραμματιστές ένα πλήρες πακέτο έτοιμων κλάσεων και αλγορίθμων που μπορούν να χρησιμοποιηθούν σε ένα ευρύ φάσμα εφαρμογών.

Η STL επιτυγχάνει τον σκοπό της με τη χρήση προτύπων κλάσεων και συναρτήσεων. Αυτή η προσέγγιση της δίνει τη δυνατότητα του πολυμορφισμού, ώστε να μπορεί να χρησιμοποιηθεί τόσο με βασικούς όσο και με οριζόμενους από τον χρήστη τύπους δεδομένων. Η STL αποτέλεσε την πρώτη βιβλιοθήκη που υποστήριξε τη φιλοσοφία του γενικευμένου προγραμματισμού (generic programming). Περιέχει πλήθος έτοιμων κλάσεων και αλγορίθμων, των οποίων η αναφορά μόνο θα απαιτούσε ένα ολόκληρο βιβλίο. Το κεφάλαιο αυτό δίνει μια πρώτη γεύση από τις δυνατότητες αυτού του εκπληκτικού εργαλείου, ωστόσο επικεντρώνεται περισσότερο στη φιλοσοφία του και λιγότερο στην πλήρη κάλυψη των δυνατοτήτων του.

Η STL διαθέτει την κλάση **string**, την οποία ήδη έχουμε χρησιμοποιήσει, και η οποία δίνει μια διαφορετική δυναμική στη διαχείριση των συμβολοσειρών. Η STL περιέχει επίσης ένα πλήθος οντοτήτων, τρεις από τις οποίες έχουν ιδιαίτερο ενδιαφέρον: οι **αποδέκτες**, οι **επαναλήπτες**, και οι **αλγόριθμοι**.

Στο κεφάλαιο αυτό, εξηγείται η φιλοσοφία και η χρήση των αποδεκτών, των επαναληπτών, και των αλγορίθμων μέσα από παραδείγματα.

Αποδέκτες, αλγόριθμοι και επαναλήπτες

Ένα μεγάλο τμήμα της καθιερωμένης βιβλιοθήκης προτύπων STL αφορά την διαχείριση οντοτήτων που ονομάζονται **αποδέκτες** (containers). Για τον χειρισμό αυτών των οντοτήτων χρησιμοποιούνται οι **επαναλήπτες** (iterators) και οι **αλγόριθμοι** (algorithms).

> Οι αποδέκτες (containers) είναι αντικείμενα τα οποία περιέχουν άλλα αντικείμενα! Στην ουσία, οι αποδέκτες είναι πρότυπα κλάσεων και επιτρέπουν στον προγραμματιστή να καθορίσει εκείνος τα αντικείμενα τα οποία θα περιέχουν.

Οι αποδέκτες είναι δομές δεδομένων (όπως λίστες, ουρές, κ.λπ.) στους οποίους η μορφή των κόμβων ορίζεται δυναμικά από τον προγραμματιστή.

Στους αποδέκτες αποθηκεύονται **ομοειδή αντικείμενα** τα οποία μπορούμε να τα διαχειριστούμε με έναν ενιαίο τρόπο ανεξάρτητα από τον τύπο τους. Με τον ίδιο τρόπο γίνεται η αποθήκευση και η διαχείριση των βασικών τύπων δεδομένων αλλά και των αντικειμέ-

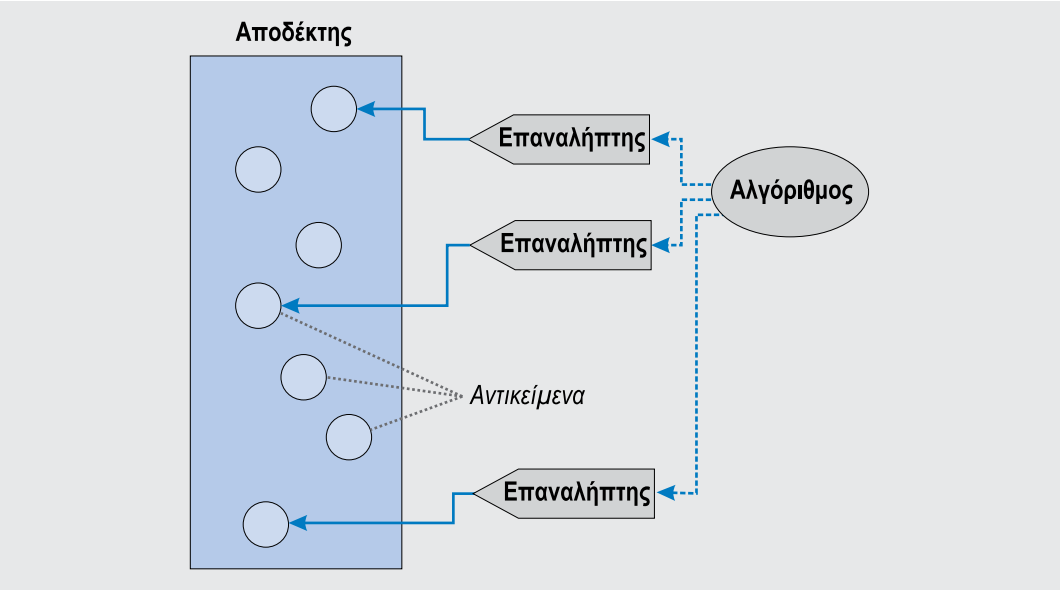
νων προσαρμοσμένων κλάσεων. Στη συνέχεια θα δούμε ότι υπάρχουν διάφορες κατηγορίες αποδεκτών.

Οι επαναλήπτες (iterators) χρησιμοποιούνται για την πρόσβαση στα επιμέρους αντικείμενα των αποδεκτών και η χρήση τους είναι παρόμοια με αυτή των δεικτών. Στην ουσία, ένας επαναλήπτης δείχνει σε ένα από τα αντικείμενα που περιέχει ένας αποδέκτης.

Υπάρχουν διαφορετικοί τύποι επαναληπτών και ανάλογα με το είδος του αποδέκτη χρησιμοποιείται ο κατάλληλος τύπος επαναλήπτη. Φυσικά, το ζητούμενο είναι η επεξεργασία των δεδομένων που είναι καταχωρισμένα στα αντικείμενα ενός αποδέκτη.

Η επεξεργασία των δεδομένων που είναι καταχωρισμένα στους αποδέκτες γίνεται μέσω των αλγορίθμων. Στην πραγματικότητα, οι αλγόριθμοι (algorithms) είναι πρότυπα συναρτήσεων που χρησιμοποιούνται για να εκτελέσουν κάποια ενέργεια στα αντικείμενα ενός αποδέκτη.

Οι αλγόριθμοι της STL είναι ανεξάρτητες συναρτήσεις και όχι μέλη κάποιων κλάσεων. Αυτό μας δίνει τη δυνατότητα να τους χρησιμοποιούμε όχι μόνο με αποδέκτες, αλλά ακόμη και με τους κλασικούς πίνακες της C++. Το Σχήμα 24.1 που ακολουθεί δείχνει τη σχέση μεταξύ αποδεκτών, επαναληπτών και αλγορίθμων.



Σχήμα 24.1 *Σχέση αποδεκτών, επαναληπτών, και αλγορίθμων*

Συνειρμικοί αποδέκτες

Οι **συνειρμικοί αποδέκτες** (associative containers) είναι διατεταγμένες συλλογές αντικειμένων σύμφωνα με κάποιο κλειδί, το οποίο μπορεί να είναι οποιουδήποτε τύπου.

Τα αντικείμενα των συνειρμικών αποδεκτών διατηρούνται σε μία προκαθορισμένη διάταξη η οποία εξαρτάται από την τιμή του κλειδιού τους. Σε αντίθεση με τους ακολουθιακούς, στους συνειρμικούς αποδέκτες δεν μπορούμε να παρεμβάλουμε ένα αντικείμενο σε μια συγκεκριμένη θέση. Η θέση του καθορίζεται αυτόματα μόνο από την τιμή του κλειδιού του. Οι συνειρμικοί αποδέκτες υλοποιούνται με δυαδικά δένδρα. Η STL διαθέτει τέσσερις κλάσεις συνειρμικών αποδεκτών: set, multiset, map και multimap.

set

Οι αποδέκτες set περιέχουν μόνο αντικείμενα-κλειδιά χωρίς να επιτρέπουν διπλότυπες τιμές (unique keys). Η πληροφορία που καταχωρίζεται σε ένα τέτοιο αντικείμενο αποτελεί ταυτόχρονα και το κλειδί με το οποίο καθορίζεται η θέση του στη διάταξη (Σχήμα 24.3α).

multiset

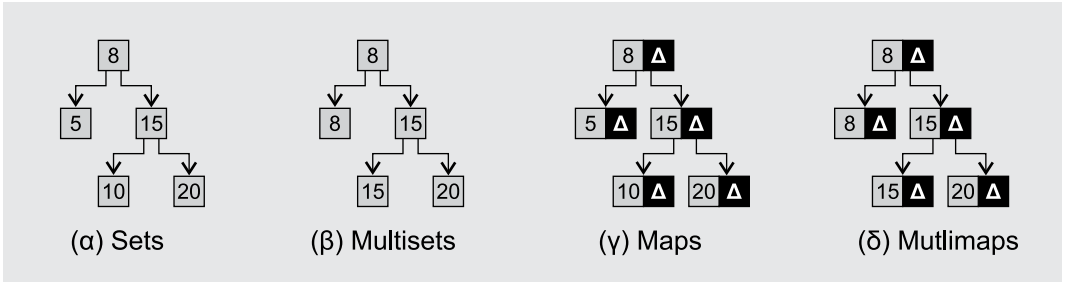
Οι αποδέκτες multiset είναι όπως ακριβώς και οι αποδέκτες set, με τη διαφορά ότι επιτρέπουν τις διπλότυπες τιμές (Σχήμα 24.3β). Δηλαδή, δύο αντικείμενα της διάταξης μπορούν να έχουν την ίδια τιμή κλειδιού.

map

Οι αποδέκτες map είναι παρόμοιοι με τους αποδέκτες set, με τη διαφορά ότι κάθε αντικείμενό τους, πέρα από το κλειδί, περιέχει και δεδομένα (Σχήμα 24.3γ). Η διάταξη των αντικειμένων καθορίζεται από την τιμή του κλειδιού τους. Οι αποδέκτες map δεν επιτρέπουν τις διπλότυπες τιμές.

multimap

Οι αποδέκτες multimap είναι όπως ακριβώς και οι αποδέκτες map, με τη διαφορά ότι επιτρέπουν τις διπλότυπες τιμές (Σχήμα 24.3δ).



Σχήμα 24.3 *Sets, Multisets, Maps και Multimaps.*

Σε κάθε συνειρμικό αποδέκτη, ισχύουν τα εξής:

- Το κόστος της προσθήκης, της διαγραφής και της αναζήτησης ενός αντικειμένου είναι λογαριθμικό. Δηλαδή, είναι ανάλογο με τον λογάριθμο του πλήθους των αντικειμένων που περιέχει.
- Επιτρέπεται η σειριακή προσπέλαση όλων των αντικειμένων που περιέχει.

Προσαρμογείς αποδεκτών

Οι **προσαρμογείς αποδεκτών** (container adapters) περιορίζουν ή τροποποιούν τις δυνατότητες άλλων αποδεκτών, παρέχοντάς τους έτσι μια διαφορετική διασύνδεση (interface). Προσαρμογείς (adaptors) υπάρχουν και για άλλες οντότητες της C++. Γενικά, ένας προσαρμογέας παρέχει σε ένα αντικείμενο μια διαφορετική διασύνδεση ώστε να μπορέσει να χρησιμοποιηθεί από άλλα αντικείμενα.

Η STL διαθέτει τρεις προκαθορισμένες κλάσεις προσαρμογέων αποδεκτών: stack, queue και priority_queue. Και οι τρεις αυτές κλάσεις είναι διαφορετικές εκδόσεις διασύνδεσης της κλάσης deque των ακολουθιακών αποδεκτών.

stack

Η κλάση stack υλοποιεί μια κλασική δομή στοίβας LIFO (Last Input First Output), όπως αυτή περιγράφηκε στο Κεφάλαιο 21. Το χαρακτηριστικό παράδειγμα μιας δομής LIFO είναι η στοίβα με τα πιάτα. Το τελευταίο που βάζουμε στην κορυφή της στοίβας είναι το πρώτο που θα πάρουμε αν θελήσουμε να το χρησιμοποιήσουμε.

queue

Η κλάση queue υποστηρίζει μια δομή ουράς FIFO (First Input First Output). Ένα κλασικό παράδειγμα μιας δομής ουράς FIFO είναι η σειρά αναμονής στο ταμείο μιας τράπεζας. Οι νέοι πελάτες που προσέρχονται στη τράπεζα προστίθενται στο τέλος της σειράς, ενώ ο ταμίας εξυπηρετεί τον πελάτη από την αρχή της σειράς.

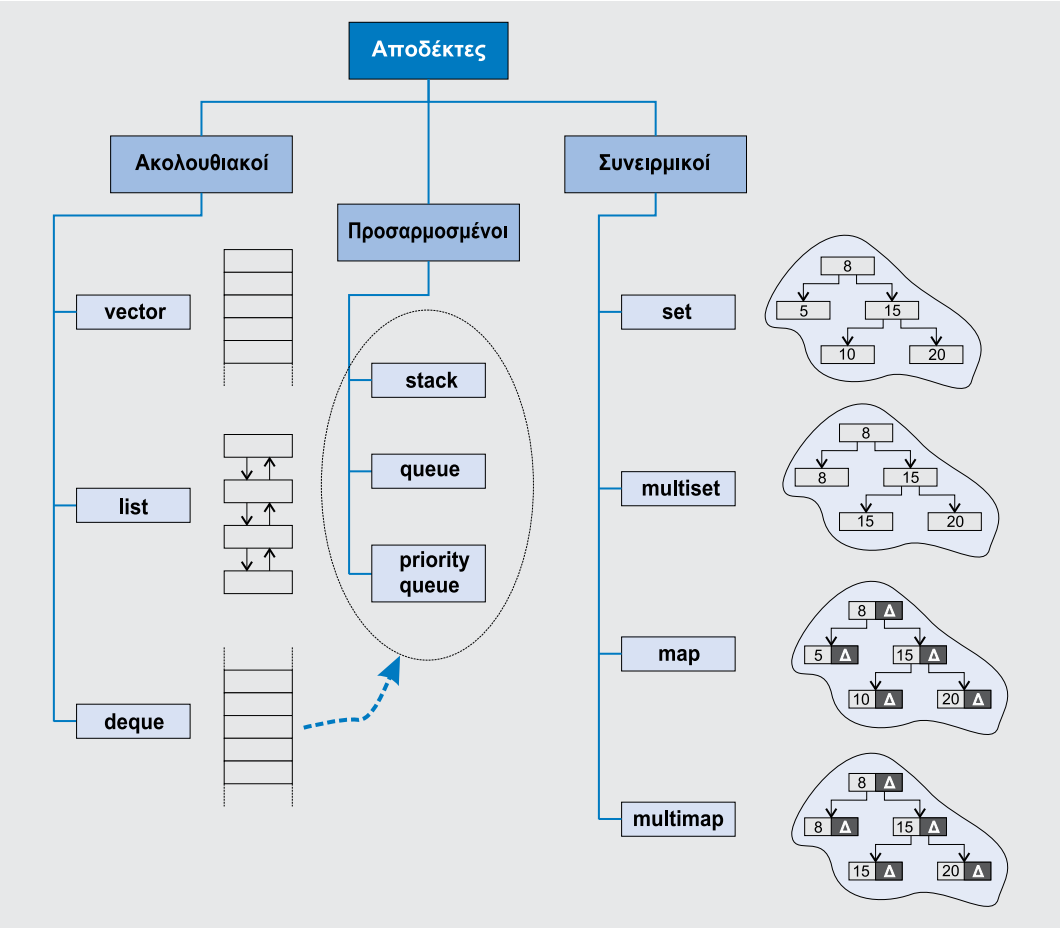
priority_queue

Η κλάση priority_queue είναι μια δομή ουράς όπως η queue, με τη διαφορά ότι διατηρεί τα αντικείμενά της ταξινομημένα ως προς κάποιο στοιχείο τους. Ένα καλό παράδειγμα θα ήταν αν στην ουρά αναμονής της τράπεζας οι πελάτες έμπαιναν στην ουρά κατά ύψος: Πρώτος ο πιο ψηλός και στο τέλος ο κοντύτερος. Στην περίπτωση αυτή, θα ήταν λίγο ρατσιστικό, αλλά οι πιο ψηλοί θα είχαν προτεραιότητα στην εξυπηρέτηση.

☞ Φανταστείτε έναν προσαρμογέα (adaptor) σαν τον αντάπτορα που χρησιμοποιούμε όταν πάμε ταξίδι σε μια άλλη χώρα με διαφορετικό τύπο πριζών ρεύματος. Μας δίνει τη δυνατότητα να χρησιμοποιούμε τις ηλεκτρικές συσκευές μας παρά το γεγονός ότι οι υπάρχουσες πρίζες δεν είναι συμβατές.

☞ Οι αποδέκτες vector και deque χρησιμοποιούν συνεχόμενες θέσεις μνήμης. Η μεταβολή του μεγέθους αυτών των αποδεκτών πολλές φορές επιβάλλει την επανατοποθέτησή τους (reallocation) σε διαφορετικό σημείο της μνήμης, λειτουργία η οποία εκτελείται αυτόματα.

Το Σχήμα 24.4 που ακολουθεί απεικονίζει τις διαφορετικές κατηγορίες αποδεκτών.

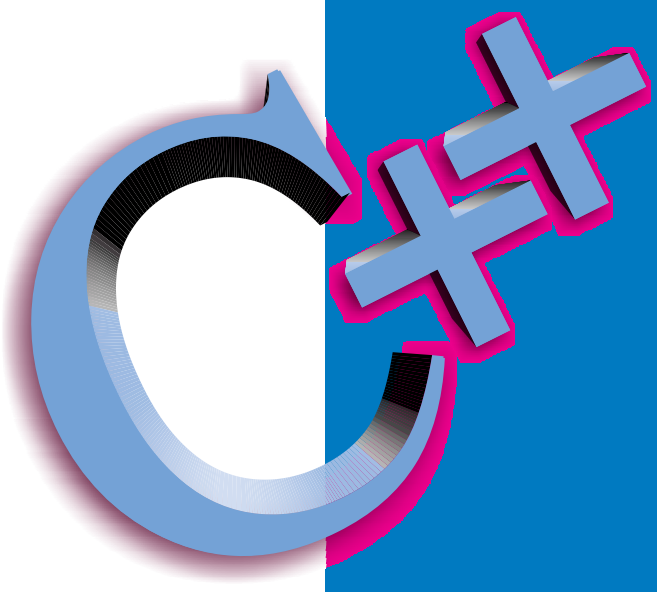


Σχήμα 24.4 *Κατηγορίες αποδεκτών*

Παράρτημα

A

Επιλεγμένες συναρτήσεις βιβλιοθήκης της C++



Επιλεγμένες συναρτήσεις βιβλιοθήκης της C++

Η καθιερωμένη βιβλιοθήκη της C++ περιλαμβάνει κάθε είδους συναρτήσεις οι οποίες είναι ομαδοποιημένες κατά κατηγορίες και δηλώνονται σε διαφορετικά αρχεία κεφαλίδων (header files). Οι περισσότερες από αυτές τις συναρτήσεις είναι «κληρονομιά» από τη βιβλιοθήκη της C. Στο παράρτημα αυτό θα αναφερθούμε σε λίγες συναρτήσεις οι οποίες χρησιμοποιούνται πιο συχνά σε προγράμματα γενικής χρήσης. Θεωρώ πιο χρήσιμο να αναφέρουμε τις συναρτήσεις ανάλογα με το αρχείο κεφαλίδας στο οποίο δηλώνονται.

Για να χρησιμοποιήσουμε οποιαδήποτε από αυτές τις συναρτήσεις, θα πρέπει να συμπεριλάβουμε στον κώδικα του προγράμματος το αρχείο κεφαλίδας στο οποίο δηλώνεται με την οδηγία **#include**.

Σε αρκετούς μεταγλωττιστές, εκτός από το αρχείο κεφαλίδας το οποίο καθορίζει το πρότυπο ANSI, αρκετές συναρτήσεις βιβλιοθήκης δηλώνονται και στο αρχείο κεφαλίδας iostream. Σε αυτή την περίπτωση, η συμπερίληψη του αρχείου iostream στον κώδικά μας επαρκεί για τη χρήση των περισσότερων συναρτήσεων βιβλιοθήκης.

Αρχείο κεφαλίδας <cctype>

int islower(int c);

Η συνάρτηση **islower**() επιστρέφει μη μηδενική τιμή αν η παράμετρος c είναι πεζός χαρακτήρας, διαφορετικά επιστρέφει τιμή 0.

int isupper(int c);

Η συνάρτηση **isupper()** επιστρέφει μη μηδενική τιμή αν η παράμετρος c είναι κεφαλαίος χαρακτήρας, διαφορετικά επιστρέφει τιμή 0.

int isprint(int c);

Η συνάρτηση **isprint()** επιστρέφει μη μηδενική τιμή αν η παράμετρος c είναι εκτυπώσιμος χαρακτήρας (συμπεριλαμβανομένου και του διαστήματος), διαφορετικά επιστρέφει τιμή 0.

int tolower(int c);

Η συνάρτηση **tolower()** επιστρέφει τον αντίστοιχο πεζό χαρακτήρα της παραμέτρου c. Δεν υποστηρίζει ελληνικούς χαρακτήρες.

int toupper(int c);

Η συνάρτηση **toupper()** επιστρέφει τον αντίστοιχο κεφαλαίο χαρακτήρα της παραμέτρου c. Δεν υποστηρίζει ελληνικούς χαρακτήρες.

Αρχείο κεφαλίδας <cstdlib>

void exit(int status);

Η συνάρτηση **exit()** έχει αποτέλεσμα τον άμεσο τερματισμό του προγράμματος. Η παράμετρος status προσδιορίζει τον κωδικό εξόδου του προγράμματος.

int system(char *s);

Η συνάρτηση **system()** εκτελεί στο περιβάλλον του λειτουργικού συστήματος την εντολή η οποία βρίσκεται καταχωρισμένη στον πίνακα χαρακτήρων που προσδιορίζεται από τον δείκτη s.

Παράδειγμα

Το επόμενο πρόγραμμα εμφανίζει τα περιεχόμενα του κεντρικού φακέλου του δίσκου C: και περιμένει να πατηθεί κάποιο πλήκτρο για να τερματιστεί.

```
#include <iostream>
using namespace std;
int main()
{
   system("dir c:\");
   system("pause");
   return 0;
}
```

int rand();

Επιστρέφει έναν ψευδοτυχαίο θετικό ακέραιο αριθμό.

Παράδειγμα

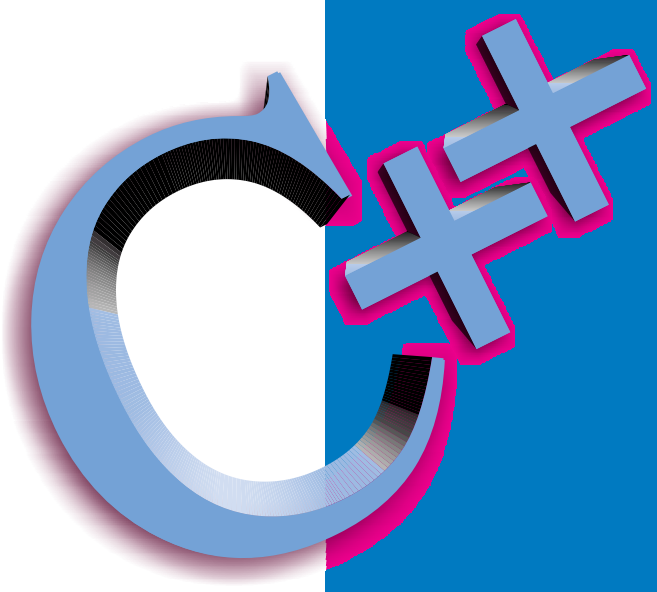
Το επόμενο πρόγραμμα συμπληρώνει τον πίνακα a με 100 τυχαίους αριθμούς. Στη συνέχεια υπολογίζει και εμφανίζει το άθροισμα των θέσεων μνήμης του πίνακα.

```
#include <iostream>
using namespace std;
int main()
{
   int i,a[100],sum=0;
   for (i=0;i<100;i++)   a[i]=rand();
   for (i=0;i<100;i++)   sum=sum+a[i];
   cout<<"Αθροισμα στοιχείων="<<sum<<endl;
   return 0;
}
```

Παράρτημα

B

Το ολοκληρωμένο περιβάλλον του Code::Blocks



Το ολοκληρωμένο περιβάλλον του Code::Blocks

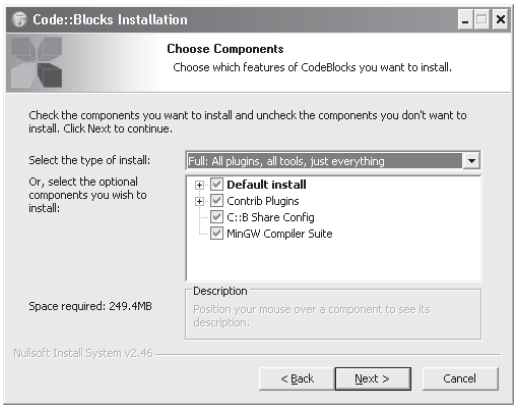
Το **Code::Blocks** είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης για τις γλώσσες C και C++. Χρησιμοποιεί τον μεταγλωττιστή GCC και παράγει τριανταδυάμπιτες εφαρμογές (Win32) τόσο για τη γραμμή εντολών (console applications) όσο και για το περιβάλλον γραφικών των Windows (GUI applications). Το περιβάλλον Code::Blocks διατίθεται δωρεάν υπό τους όρους της γενικής άδειας χρήσης της GNU (GPL v3.0).

Στο συνοδευτικό CD θα βρείτε το πρόγραμμα εγκατάστασης του ολοκληρωμένου περιβάλλοντος Code::Blocks. Επίσης, συνδέσμους για τη λήψη του προγράμματος θα βρείτε στην ιστοσελίδα της εφαρμογής http://www.codeblogs.org, καθώς και στον δικτυακό τόπο αυτού του βιβλίου **http://cpp.bytes.gr**.

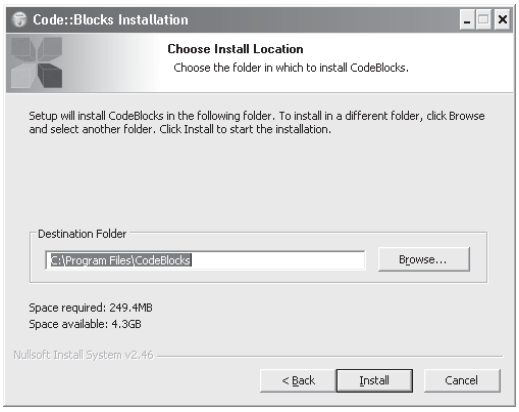
Όλα τα προγράμματα που υπάρχουν σε αυτό το βιβλίο έχουν δοκιμαστεί στο περιβάλλον του Code::Blocks.

Εγκατάσταση του Code::Blocks

Για να εγκαταστήσετε το ολοκληρωμένο περιβάλλον του Code::Blocks σε περιβάλλον Windows, εκτελέστε το αρχείο CodeBlocks_setup.exe από το συνοδευτικό CD. Στην περίπτωση που θέλετε την πιο πρόσφατη έκδοση του περιβάλλοντος, ή κάποια άλλη έκδοσή του (π.χ. για Linux) επισκεφτείτε την ιστοσελίδα http://www.codeblogs.org, και κατεβάστε από εκεί το πρόγραμμα εγκατάστασης που επιθυμείτε.



- Επιλέξτε ως τύπο εγκατάστασης **Full**.
- Αποδεχθείτε όλα τα προτεινόμενα επιλεγμένα τμήματα της εφαρμογής.



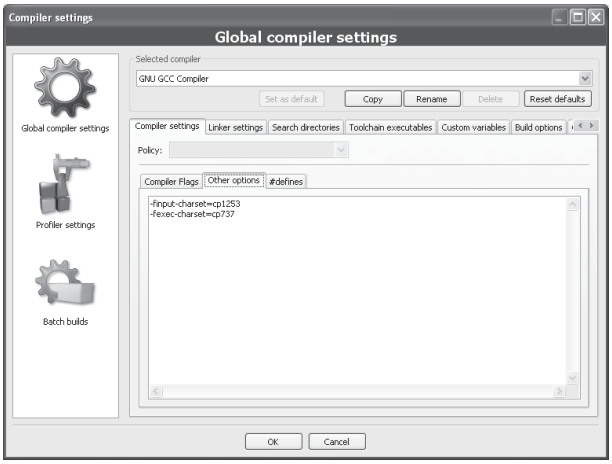
- Αποδεχθείτε τον προτεινόμενο φάκελο εγκατάστασης της εφαρμογής.
- Πατήστε στο κουμπί **Install** για την εκκίνηση της εγκατάστασης.

Οι βασικές λειτουργίες του Code::Blocks

Στη συνέχεια αναφέρουμε τις βασικές λειτουργίες του περιβάλλοντος ώστε να μπορείτε να δοκιμάζετε τα παραδείγματα του βιβλίου ή τα προγράμματα που δημιουργείτε μόνοι σας.

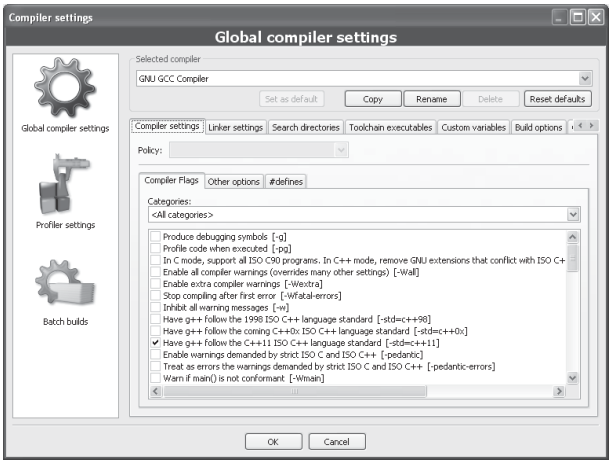
Παραμετροποίηση του περιβάλλοντος για χρήση ελληνικών χαρακτήρων

Από την επιλογή του μενού: **Settings** – **Compiler**, ανοίξτε την καρτέλα **Other options** και πληκτρολογήστε τις ακόλουθες γραμμές:
-finput-charset=cp1253
-fexec-charset=cp737

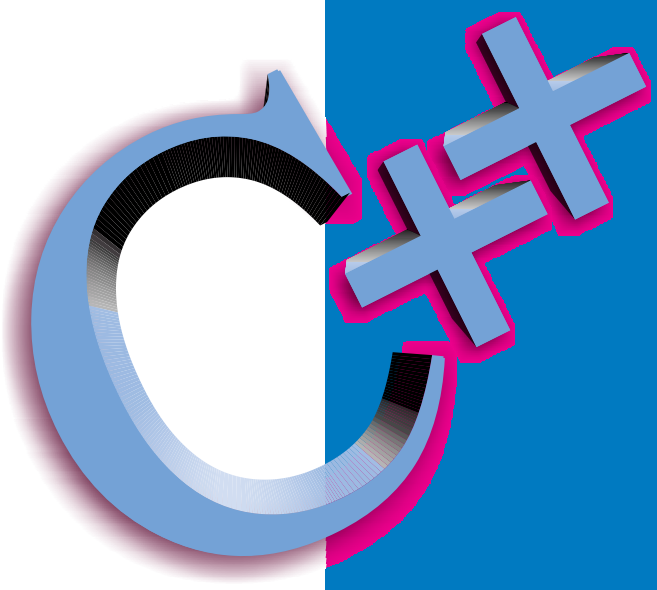


Παραμετροποίηση του περιβάλλοντος για χρήση του πρότυπου C++11

Από το μενού: **Settings** – **Compiler**, ανοίξτε την καρτέλα **Compiler Flags** και ενεργοποιήστε την επιλογή:
Have g++ flollow the C++11



Αντιστοίχιση ελληνικών και αγγλικών όρων



Αντιστοίχιση ελληνικών και αγγλικών όρων

Αναφορά	Reference
Αγκύλη	Bracket
Αθροιστής	Accumulator
Ακολουθιακοί αποδέκτες	Sequence containers
Αλφαριθμητικό	Character string
Αναγνωριστικό	Identifier
Αναδρομή	Recursion
Αναζήτηση	Searching
Αναφορική μεταβλητή	Reference variable
Ανικειμενοστρεφής	Object oriented
Ανικειμενοστρεφής Προγραμματισμός	Object oriented programming (OOP)
Ανικειμενοστρεφής σχεδιασμός	Object oriented design
Αντικείμενο	Object
Αντιστοιχία	Map
Ανώνυμοι χώροι ονομάτων	Anonymous namespaces
Απαριθμήσεις	Enumerations
Απλά συνδεδεμένη λίστα	Simply linked list
Αποδέκτης	Container
Αποσφαλμάτωση	Debugging
Αρχείο	File
Αρχείο κειμένου	Text file
Αρχικές τιμές	Initial values
Ασάφεια	Ambiguity
Άτυπη παράμετρος	Non-type parameter
Αφαιρετικότητα	Abstraction
Αφηρημένη κλάση	Abstract class
Βασική κλάση	Base class, Superclass
Βιβλιοθήκη	Library
Βρόχος	Loop
Γνήσια εικονική μέθοδος	Pure virtual method

Η γλώσσα C++ σε βάθος

2η αναθεωρημένη έκδοση

Το βιβλίο προσεγγίζει όλα τα χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού και της γλώσσας C++ με έναν ιδιαίτερα εποπτικό και επεξηγηματικό τρόπο. Δίνει τη δυνατότητα στον αναγνώστη να μπει εύκολα και γρήγορα στη φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού και να κατανοήσει πλήρως τα στοιχεία της γλώσσας. Δεν προϋποθέτει γνώσεις προγραμματισμού και απευθύνεται τόσο στον αρχάριο σπουδαστή όσο και στον έμπειρο προγραμματιστή ο οποίος θέλει να γνωρίσει τα ιδιαίτερα χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού.

Εξετάζονται αναλυτικά:

- Τύποι δεδομένων, παραστάσεις, εντολές ελέγχου και επανάληψης
- Συναρτήσεις
- Δείκτες και πίνακες
- Κλάσεις και αντικείμενα
- Υπερφόρτωση τελεστών και συναρτήσεων
- Κληρονομικότητα, παράγωγες και βασικές κλάσεις, πολυμορφισμός
- Συσχέτιση κλάσεων, διαγράμματα κλάσεων της UML
- Ρεύματα εισόδου/εξόδου, χειρισμός αρχείων
- Αναδρομή, αλγόριθμοι αναζήτησης και ταξινόμησης
- Δυναμική διαχείριση μνήμης
- Δυναμικές δομές δεδομένων – συνδεδεμένες λίστες, δυαδικά δένδρα αναζήτησης και υλοποίησή τους στη γλώσσα C++
- Πρότυπα κλάσεων και συναρτήσεων
- Εξαιρέσεις και διαχείριση σφαλμάτων, χώροι ονομάτων
- Ο μεταγλωττιστής GCC
- Χρήση της καθιερωμένης βιβλιοθήκης προτύπων (STL) της C++. Αποδέκτες, αλγόριθμοι και επαναλήπτες

Κάθε κεφάλαιο περιλαμβάνει σύντομη ανασκόπηση, λυμένα χαρακτηριστικά παραδείγματα, ασκήσεις διαφορετικών βαθμών δυσκολίας, καθώς και τα συχνότερα λάθη που γίνονται. Η δομή αυτή καθιστά το βιβλίο ιδανικό για εκπαιδευτική χρήση.

Το συνοδευτικό **CD** περιέχει το ολοκληρωμένο περιβάλλον ανάπτυξης **Code::Blocks**, τον κώδικα των προγραμμάτων τού βιβλίου, τις απαντήσεις όλων των ασκήσεων, καθώς και τον κώδικα των λύσεών τους. Επίσης, αποκλειστικά για αυτό το βιβλίο, έχει δημιουργηθεί ένας δικτυακός τόπος στη διεύθυνση **http://cpp.bytes.gr** με χρήσιμο εκπαιδευτικό υλικό.

Δυνατότητα για **on-line** μαθήματα ...

e-lessons

στο **http://cpp.bytes.gr/elessons**

Ο **Ν. Χατζηγιαννάκης** είναι διπλωματούχος του Τμήματος Ηλεκτρολόγων Μηχανικών Η/Υ της Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών και κάτοχος Master στην επιστήμη των υπολογιστών από το Πανεπιστήμιο Bath της Αγγλίας.

Από το 1987 δραστηριοποιείται στην εκπαίδευση Η/Υ, έχοντας στο ενεργητικό του χιλιάδες ώρες διδασκαλίας σε θέματα προγραμματισμού και χειρισμού Η/Υ.

Παράλληλα, ως ειδικός επιστήμων σε θέματα ανάλυσης και σχεδίασης συστημάτων Η/Υ της Γ.Γ Αιγαίου και Νησιωτικής Πολιτικής, έχει αναλάβει τον σχεδιασμό και την ανάπτυξη πρωτοποριακών πληροφοριακών συστημάτων για τον Δημόσιο τομέα. Στα πλαίσια αυτά, είναι ο επιστημονικός υπεύθυνος έργων στα προγράμματα "Κοινωνία της Πληροφορίας" και "Ψηφιακή Σύγκλιση" που αφορούν την ηλεκτρονική διακυβέρνηση, την εξυπηρέτηση του πολίτη και τον εκσυγχρονισμό της Δημόσιας Διοίκησης.

Το 2009 ξεκίνησε να διδάσκει διαδικαστικό και αντικειμενοστρεφή προγραμματισμό, με τις γλώσσες C και C++, στο τμήμα Πολιτισμικής Τεχνολογίας και Επικοινωνίας του Πανεπιστημίου Αιγαίου.

Ο Νίκος Χατζηγιαννάκης είναι επίσης ο συγγραφέας του βιβλίου **"Η γλώσσα C σε βάθος"** το οποίο αναφέρεται στον διαδικαστικό προγραμματισμό με τη γλώσσα C και κυκλοφορεί από τις Εκδόσεις Κλειδάριθμος.

Επικοινωνία: nikos@bytes.gr

ISBN 978-960-461-620-6

9 789604 616206

εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ

Δομοκού 4, Σταθμός Λαρίσης, 10440 ΑΘΗΝΑ, Τηλ. 210-5237635

info@klidarithmos.gr **www.klidarithmos.gr**

www.facebook.com/klidarithmos.gr