ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΕΠΙΣΤΗΜΗΣ ΥΛΙΚΩΝ

ΥΠΟΛΟΓΙΣΤΕΣ ΙΙ
Σημειώσεις εργαστηρίου

Ελευθέριος Λοιδωρίκης
Δημήτριος Γ. Παπαγεωργίου

Ιστοσελίδα μαθήματος
pc164.materials.uoi.gr/dpapageo/courses/comp2

ΙΩΑΝΝΙΝΑ 2012

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Κατά τη διάρκεια της δεκαετίας του '60 ενώ οι υπολογιστές ήταν ακόμα στα αρχικά στάδια ανάπτυξης, εμφανίστηκαν πολλές νέες γλώσσες προγραμματισμού. Μεταξύ αυτών η Algol που αναπτύχθηκε ως εναλλακτική της γλώσσας Fortran, αλλά διατηρώντας ορισμένα χαρακτηριστικά του δομημένου προγραμματισμού, που θα ενέπνεαν αργότερα τις περισσότερες διαδικαστικές γλώσσες όπως τη CPL και τους διαδόχους της. Όμως η Algol ήταν μια σχετικά αφηρημένη γλώσσα και αυτό την καταστούσε μη πρακτική για τις περισσότερες εφαρμογές.

Το 1963 εμφανίστηκε η CPL (Combined Programming Language – συνδυασμένη γλώσσα προγραμματισμού) με την ιδέα να είναι πιο συγκεκριμένη από την Algol ή την Fortran σε πρακτικά προβλήματα. Εντούτοις αυτό το χαρακτηριστικό την έκανε μια μεγάλη γλώσσα και επομένως δύσκολη στην εκμάθηση και υλοποίηση.

Το 1967, ο Martin Richards ανέπτυξε την BCPL (Basic Combined Programming Language – βασική συνδυασμένη γλώσσα προγραμματισμού), που αποτελούσε μια απλοποίηση της CPL κρατώντας όμως τα περισσότερα σημαντικά χαρακτηριστικά γνωρίσματα που η προσέφερε η γλώσσα. Η BCPL ήταν επίσης μια μεγάλη γλώσσα.

Το 1970 ο Ken Thompson που ασχολήθηκε με την ανάπτυξη του UNIX στα εργαστήρια Bell, δημιούργησε τη γλώσσα B. Ήταν μια μεταφορά της γλώσσας BCPL για ένα συγκεκριμένο υπολογιστή (DEC pdp–7) και λειτουργικό σύστημα (Unix), και προσαρμόστηκε στις ιδιαίτερες ανάγκες αυτού του συστήματος. Το τελικό αποτέλεσμα ήταν μια ακόμα μεγαλύτερη απλοποίηση της CPL, αν και εξαρτώμενη από το συγκεκριμένο σύστημα. Η υλοποίηση αυτή είχε διάφορους περιορισμούς όπως για παράδειγμα δεν μπορούσε να φτιάξει εκτελέσιμο κώδικα και κατά συνέπεια ήταν ανεπαρκής για την ανάπτυξη ενός λειτουργικού συστήματος. Το 1971 ο Dennis Ritchie από τα εργαστήρια Bell άρχισε την ανάπτυξη του μεταγλωττιστή B που, μεταξύ άλλων, ήταν σε θέση να παράγει άμεσα τον εκτελέσιμο κώδικα. Αυτή η νέα γλώσσα B, η οποία στη συνέχεια ονομάστηκε C, εισήγαγε νέες έννοιες όπως τους τύπους δεδομένων.

Το 1973 ο Dennis Ritchie είχε αναπτύξει το βασικό κομμάτι της γλώσσας C. Οι τύποι δεδομένων και ο χειρισμός τους, οι βελτιωμένοι πίνακες και οι δείκτες μαζί με τη δυνατότητα να είναι μεταφέρσιμη μεταξύ διαφορετικών υπολογιστικών συστημάτων συνέβαλλαν στη γρήγορη διάδοση της γλώσσας C. Η γλώσσα C καθιερώθηκε με το βιβλίο "Η γλώσσα προγραμματισμού C" από τους Brian Kernighan και Dennis Ritchie, γνωστό και ως άσπρο βιβλίο. Αυτό χρησίμευσε ως de facto πρότυπο μέχρι τη

δημοσίευση των επίσημων προτύπων Ansi (επιτροπή Ansi X3J11) το 1989.

Το 1980 ο Bjarne Stroustrup από τα εργαστήρια Bell άρχισε την ανάπτυξη της γλώσσας C++ η οποία θα λάμβανε τυπικά αυτό το όνομα στο τέλος του 1983, όταν επρόκειτο να δημοσιευθεί το πρώτο εγχειρίδιό της. Τον Οκτώβριο του 1985, εμφανίστηκε η πρώτη εμπορική έκδοση της γλώσσας όπως επίσης και η πρώτη έκδοση του βιβλίου "Η γλώσσα προγραμματισμού C++" από τον Bjarne Stroustrup.

Κατά τη διάρκεια της δεκαετίας του '80 η C++ γλώσσα εξελισσόταν έως ότου έγινε μια γλώσσα με τη δική της προσωπικότητα, όμως με πολύ λίγες απώλειες συμβατότητας με την ήδη υπάρχουσα γλώσσα C, και χωρίς να αλλοιωθούν τα σημαντικότερα χαρακτηριστικά της.

Από το 1990 και μετά η επιτροπή Ansi X3J16 άρχισε την ανάπτυξη συγκεκριμένων προτύπων για την γλώσσα C++. Στην περίοδο που παρήλθε μέχρι τη δημοσίευση των προτύπων το 1998, η χρήση της C++ επεκτάθηκε και είναι σήμερα η γλώσσα που προτιμάται για την ανάπτυξη εφαρμογών σε όλες τις πλατφόρμες.

Κεφάλαιο 2

Πως γράφεται ένα πρόγραμμα C++

2.1 Η μορφή του προγράμματος

Οι εντολές ενός προγράμματος C++ γράφονται με "ελεύθερο" τρόπο αφού το συντακτικό της γλώσσας δεν επιβάλλει συγκεκριμένο τρόπο γραφής. Πχ. είναι συντακτικά αποδεκτό να γράφονται δύο ή παραπάνω εντολές σε μια γραμμή.

Η γενική μορφή ενός προγράμματος C++ φαίνεται παρακάτω:

επικεφαλίδες βιβλιοθηκών

```
int main ( )
{
    δηλώσεις
       ⋮
    εκτελέσιμες εντολές
       ⋮
    return 0;
}
```

Η σημασία των επιμέρους τμημάτων εξηγείται παρακάτω:

επικεφαλίδες βιβλιοθηκών

Εδώ μπαίνουν δηλώσεις που σχετίζονται με τη χρήση συναρτήσεων από έτοιμες βιβλιοθήκες της γλώσσας. Οι συνηθέστερες τέτοιες δηλώσεις είναι:

`#include <iostream>`	Για να χρησιμοποιήσουμε τις συναρτήσεις εισόδου–εξόδου.
`#include <cmath>`	Για να χρησιμοποιήσουμε τις μαθηματικές συναρτήσεις.
`using namespace std;`	Για να χρησιμοποιήσουμε τον προκαθορισμένο *χώρο ονομάτων* της γλώσσας. Οι χώροι ονομάτων είναι συλλογές από μεταβλητές, συναρτήσεις και άλλες οντότητες που χρησιμοποιούνται στον κώδικα.

`int main ( )`

Αυτή είναι η επικεφαλίδα του προγράμματος. Όλες οι εντολές του προγράμματος βρίσκονται ανάμεσα στο ζεύγος από άγκιστρα `{ }`.

δηλώσεις

Εδώ τοποθετούνται διάφορες δηλώσεις, όπως για παράδειγμα δηλώνεται ο τύπος κάθε μεταβλητής. Κάθε δήλωση τερματίζεται με ερωτηματικό ;

εκτελέσιμες εντολές

Εδώ γράφονται οι εκτελέσιμες εντολές του προγράμματος. Κάθε εντολή τερματίζεται με ερωτηματικό ; Μια εντολή μπορεί να γραφεί σε παραπάνω από μια γραμμές, αλλά επίσης μπορεί δύο ή περισσότερες εντολές να γραφούν στην ίδια γραμμή.

`return 0`

Κάθε πρόγραμμα C++ αφού εκτελεστεί επιστρέφει στο λειτουργικό σύστημα μια ακέραια τιμή η οποία δείχνει αν η εκτέλεση του προγράμματος ήταν επιτυχής. Το σύμβολο `int` στην επικεφαλίδα του προγράμματος δηλώνει ότι αυτός ο επιστρεφόμενος αριθμός είναι ακέραιος. Κατά σύμβαση η τιμή 0 δείχνει ότι το πρόγραμμα εκτελέστηκε επιτυχώς. Με την εντολή `return 0` τερματίζεται η λειτουργία του προγράμματος και επιστρέφεται η τιμή 0 στο λειτουργικό σύστημα. Η εντολή `return` μπορεί να παραλειφθεί οπότε το πρόγραμμα τερματίζεται όταν φτάσει στο τελικό άγκιστρο `}`.

Παράδειγμα:

Το παρακάτω πρόγραμμα θα εμφανίσει στην οθόνη τη λέξη `Hello`.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`// Αυτό το πρόγραμμα εμφανίζει`	Οι γραμμές αυτές αποτελούν σχόλια που δεν
`// στην οθόνη τη λέξη Hello`	απευθύνονται στον μεταφραστή C++ αλλά σε όποιον διαβάσει το πρόγραμμα.
`int main ( )`	Η εκτέλεση του προγράμματος ξεκινά από αυτό το σημείο.
`{`	
`cout << "Hello \n";`	Με την εντολή `cout` εμφανίζεται στην οθόνη ένα μήνυμα. Το μήνυμα πρέπει να βρίσκεται εντός διπλών αποστρόφων. Το σύμβολο `\n` δηλώνει την αλλαγή γραμμής.
`return 0;`	Τερματισμός του προγράμματος.
`}`	

2.2 Ορισμένες παρατηρήσεις

1. Το κυρίως πρόγραμμα έχει ένα προκαθορισμένο όνομα (`main`) που δεν μπορεί να αλλάξει.
2. Οι εντολές του προγράμματος τερματίζονται με ερωτηματικό `;`
3. Κεφαλαίοι και μικροί χαρακτήρες θεωρούνται διαφορετικοί. Έτσι πχ το `main` **δεν** μπορεί να γραφεί ως `MAIN` ούτε το `cout` μπορεί να γραφεί ως `COUT`.
4. Για να είναι εύκολη η ανάγνωση του κώδικα έχει καθιερωθεί ένας συγκεκριμένος τρόπος με τον οποίο γράφονται τα προγράμματα:
   - Γράφουμε μία εντολή ανά γραμμή.
   - Κάθε τμήμα εντολών που περικλείεται από άγκιστρα `{ }` γράφεται πιο δεξιά (αφήνουμε μερικά κενά ή πατάμε TAB).
5. Θα μπορούσαμε όμως να γράψουμε μια εντολή σε δύο γραμμές, πχ.
   ```
   cout <<
   "HELLO \n";
   ```
   ή ακόμη να γράψουμε περισσότερες εντολές σε μια γραμμή (δεν συνιστάται), πχ.
   ```
   int main ( ) { cout << "HELLO \n"; return 0; }
   ```

2.3 Σχόλια

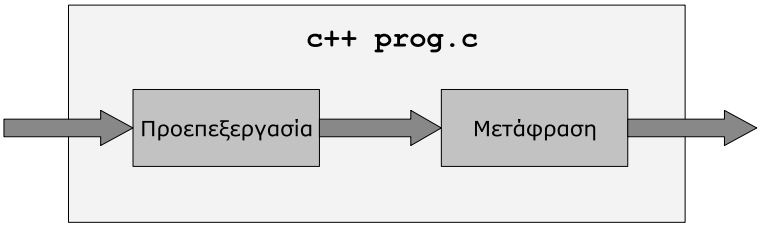
Σχόλια μπορούν να τοποθετηθούν σε οποιοδήποτε σημείο του προγράμματος. Στη C++ υπάρχουν δύο τρόποι για να βάλουμε σχόλια σε ένα πρόγραμμα.

- *Σχόλια πολλαπλών γραμμών.*
  Οτιδήποτε περικλείεται μεταξύ των συμβόλων `/*` και `*/` θεωρείται σχόλιο. Πχ.
  ```
  /* Το πρόγραμμα αυτό
     εμφανίζει στην οθόνη
     τη λέξη HELLO        */
  int main ( )       /* Η επικεφαλίδα του προγράμματος */
  ```
- *Σχόλια μιας γραμμής.*
  Ξεκινάνε με τους χαρακτήρες `//` και εκτείνονται μέχρι το τέλος της γραμμής. Μπορούν να τοποθετηθούν σε μια γραμμή μόνα τους ή μετά το τέλος μιας εντολής. Πχ.
  ```
  cout << "Hello \n";    // Εμφανίζεται στην οθόνη η λέξη Hello.
  ```

2.4 Συγγραφή και μετάφραση του προγράμματος

Για να γράψουμε το πρόγραμμα χρησιμοποιούμε ένα κειμενογράφο (text editor) και αποθηκεύουμε το αρχείο δίνοντας ένα όνομα με την κατάληξη `.c` (Πχ. `prog.c`, `lab1.c` κοκ). Για να μεταφράσουμε το πρόγραμμα και να δημιουργήσουμε το εκτελέσιμο χρησιμοποιούμε την εντολή:

Σχήμα 2.1: Η διαδικασία παραγωγής του εκτελέσιμου.



Σχήμα 2.2: Η διαδικασία της προεπεξεργασίας και μετάφρασης.

```
c++ prog.c
```

Η διαδικασία απεικονίζεται στο σχ. 2.1.

Οι γραμμές του προγράμματος που ξεκινάνε με το σύμβολο # ονομάζονται *οδηγίες προεπεξεργαστή*. Ο προεπεξεργαστής είναι ένα ανεξάρτητο πρόγραμμα που διαβάζει τον πηγαίο κώδικα και ενεργεί κατάλληλα όπου βρει σχετικές οδηγίες. Η γραμμή

```
#include <iostream>
```

αποτελεί οδηγία για τον προεπεξεργαστή που του λέει να συμπεριλάβει στο σημείο εκείνο τα περιεχόμενα του αρχείου `iostream`. Το αρχείο `iostream` βρίσκεται ήδη εγκατεστημένο στον υπολογιστή και περιέχει διάφορες δηλώσεις σχετικές με τις εντολές εισόδου–εξόδου. Αργότερα θα δούμε και άλλες οδηγίες για τον προεπεξεργαστή όπως την οδηγία `#define`.

Έτσι στην πραγματικότητα η εντολή `c++ prog.c` πραγματοποιεί δύο λειτουργίες: αρχικά ενεργοποιεί τον προεπεξεργαστή και στη συνέχεια το προεπεξεργασμένο πρόγραμμα παραδίδεται στο μεταφραστή για συντακτικό έλεγχο και παραγωγή του εκτελέσιμου. Η διαδικασία απεικονίζεται στο σχ. 2.2.

Κεφάλαιο 3

Τύποι δεδομένων, μεταβλητές και είσοδος–έξοδος

3.1 Τύποι δεδομένων

Η γλώσσα C++ μπορεί να χειριστεί τους παρακάτω βασικούς τύπους δεδομένων τους οποίους θα εξετάσουμε αναλυτικά:

- Ακέραιοι αριθμοί (`int`).
- Πραγματικοί αριθμοί απλής ακρίβειας (`float`).
- Πραγματικοί αριθμοί διπλής ακρίβειας (`double`).
- Χαρακτήρες (`char`).

Επιπλέον υπάρχει δυνατότητα να οριστούν από το χρήστη νέοι παράγωγοι τύποι δεδομένων βασισμένοι στους ανωτέρω βασικούς τύπους.

3.1.1 Ακέραιοι αριθμοί

Πρόκειται για αριθμούς που έχουν μόνο ακέραιο μέρος και όχι δεκαδικά ψηφία. Για το λόγο αυτό η αναπαράστασή τους στον HY είναι πάντα ακριβής. Η αποθήκευσή τους απαιτεί 4 bytes μνήμης, κατά συνέπεια μπορεί να λάβουν τιμές στην περιοχή -2 147 483 648 έως 2 147 483 647.

Μεταξύ δύο ακεραίων αριθμών μπορούν να γίνουν οι πράξεις πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση και υπόλοιπο διαίρεσης χρησιμοποιώντας τους τελεστές `+` `-` `*` `/` και `%` αντίστοιχα. Σημειώστε ότι στη γλώσσα C++ δεν υπάρχει τελεστής για ύψωση σε δύναμη.

Όλες οι πράξεις μεταξύ ακεραίων δίνουν ακέραιο αποτέλεσμα. Ειδικότερα, **το αποτέλεσμα της διαίρεσης μεταξύ δύο ακέραιων αριθμών είναι πάντα ακέραιος**, αφού αν προκύψουν δεκαδικά ψηφία αυτά αποκόπτονται. Προσοχή: δεν γίνεται στρογγύλευση του αποτελέσματος, απλώς αποκοπή

των δεκαδικών ψηφίων. Αυτή η συγκεκριμένη λειτουργία ονομάζεται *ακέραια διαίρεση*. Ορισμένα παραδείγματα φαίνονται στον παρακάτω πίνακα:

Πράξη	*Αποτέλεσμα*	
`3+5`	8	
`8-4`	4	
`2*6`	12	
`9/3`	3	
`7/2`	3	(Ακέραια διαίρεση)
`3/4`	0	(Ακέραια διαίρεση)
`4%2`	0	
`8%3`	2	

3.1.2 Πραγματικοί αριθμοί διπλής ακρίβειας

Ονομάζονται και αριθμοί κινητής υποδιαστολής. Πρόκειται για αριθμούς οι οποίοι έχουν ακέραιο μέρος και δεκαδικά ψηφία. Για την αποθήκευσή τους απαιτούνται 8 bytes (64 bits) μνήμης. Στη C++ οι αριθμοί αυτοί ονομάζονται `double`. Παραδείγματα πραγματικών αριθμών διπλής ακρίβειας:

```
1.27   -761.841   5.0
```

Σημειώστε ότι ακόμη και αν το δεκαδικό μέρος είναι μηδέν (όπως στον 5.0), ο αριθμός εξακολουθεί να θεωρείται πραγματικός και όχι ακέραιος. Επιπλέον ένας πραγματικός αριθμός μπορεί να γραφεί στην λεγόμενη "επιστημονική αναπαράσταση", δηλαδή χρησιμοποιώντας δυνάμεις του 10. Πχ. ο αριθμός 325.476 μπορεί να γραφεί ως 3.25476e2 που σημαίνει 3.25476 x 10^2. Η επιστημονική αναπαράσταση εξυπηρετεί για την αναπαράσταση πολύ μικρών ή πολύ μεγάλων αριθμών. Ορισμένα παραδείγματα αριθμών στην επιστημονική αναπαράσταση φαίνονται παρακάτω:

Επιστημονική αναπαράσταση	*Σημαίνει*
`1.727e5`	172700.0
`-32.476e-2`	-0.32476
`8.45e0`	8.45

Στην πραγματικότητα ένας πραγματικός αριθμός διπλής ακρίβειας αποθηκεύεται στη μνήμη στην επιστημονική αναπαράσταση (όμως χρησιμοποιώντας δυνάμεις του 2 και όχι του 10). Συγκεκριμένα απαιτούνται 1 bit για το πρόσημο του αριθμού, 11 bits για τον εκθέτη και 52 bits τα σημαντικά ψηφία. Για το λόγο αυτό το πλήθος των σημαντικών ψηφίων που μπορούν να αποθηκευτούν είναι πεπερασμένο (περίπου 15). Για παράδειγμα με την πράξη 2.0/3.0 η οποία στην πραγματικότητα δίνει τον αριθμό 0.6666..., το αποτέλεσμα που θα πάρουμε στον υπολογιστή είναι 0.666666666666667. Παρατηρείστε ότι το τελευταίο σημαντικό ψηφίο στρογγυλεύεται. Για τον ίδιο λόγο η περιοχή τιμών που μπορεί να πάρει ένας πραγματικός αριθμός διπλής ακρίβειας είναι περίπου από -10^{308} έως -10^{-308} και από 10^{-308} έως 10^{308}. Όταν από μια πράξη προκύψει αριθμός μεγαλύτερος (κατ' απόλυτη τιμή) από 10^{308} αυτό αποτελεί λογικό λάθος (υπερπλήρωση–overflow) και η λειτουργία του προγράμματός μας σταματά. Παρατηρήστε ότι υπάρχει μια περιοχή μεταξύ -10^{-308} και 10^{-308} όπου οι πραγματικοί

αριθμοί δεν μπορούν να πάρουν τιμές πλην του αριθμού 0. Όλοι οι υπόλοιποι αριθμοί σε αυτή την περιοχή δεν μπορούν να αναπαρασταθούν στον υπολογιστή και αν από μια πράξη προκύψει πραγματικός αριθμός σε αυτή την περιοχή η κατάσταση ονομάζεται υποπλήρωση–underflow.

Οι πράξεις που μπορεί να γίνουν μεταξύ δύο πραγματικών αριθμών διπλής ακρίβειας είναι πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση χρησιμοποιώντας τους τελεστές `+ - * και /` αντίστοιχα. Οποιαδήποτε πράξη μεταξύ δύο πραγματικών αριθμών διπλής ακρίβειας δίνει ως αποτέλεσμα και πάλι ένα πραγματικό αριθμό διπλής ακρίβειας.

3.1.3 Πραγματικοί αριθμοί απλής ακρίβειας

Πρόκειται για πραγματικούς αριθμούς που καταλαμβάνουν 4 bytes (32 bits) στη μνήμη (το μισό χώρο από τους αριθμούς διπλής ακρίβειας). Από τα 32 bits ενός πραγματικού αριθμού απλής ακρίβειας 1 bit καταλαμβάνει το πρόσημο του αριθμού, 8 bits ο εκθέτης (μαζί με το πρόσημό του) και 23 bits τα σημαντικά ψηφία. Σαν συνέπεια η περιοχή τιμών που μπορεί να πάρει ένας πραγματικός αριθμός είναι περίπου από 10^{-38} έως 10^{38} και για τους αρνητικούς από -10^{38} έως -10^{-38}. Επιπλέον η ακρίβειά τους είναι περίπου 7 σημαντικά ψηφία. Στη C++ οι πραγματικοί αριθμοί απλής ακρίβειας ονομάζονται `float` και γράφονται με το επίθεμα `f`. Ορισμένα παραδείγματα φαίνονται στον παρακάτω πίνακα:

Αν γράψουμε	*Σημαίνει*	*Ο τύπος του αριθμού είναι*
`2.95e3`	2950.0	`double`
`-652.12e-4`	-0.065212	`double`
`951.43`	951.43	`double`
`85.91f`	85.91	`float`
`3.15e2f`	315.0	`float`

Οι πράξεις που μπορεί να γίνουν μεταξύ δύο πραγματικών αριθμών απλής ακρίβειας είναι πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση χρησιμοποιώντας τους τελεστές `+ - * και /` αντίστοιχα. Οποιαδήποτε πράξη μεταξύ δύο πραγματικών αριθμών απλής ακρίβειας δίνει ως αποτέλεσμα και πάλι ένα πραγματικό αριθμό απλής ακρίβειας.

3.1.4 Χαρακτήρες

Στη C++ οι χαρακτήρες ονομάζονται `char` και καταλαμβάνουν 1 byte μνήμης. Κατά συνέπεια υπάρχουν 256 διαφορετικοί χαρακτήρες. Οι χαρακτήρες γράφονται μέσα σε απλές αποστρόφους. Πχ.

```
'a'   '8'   'Z'
```

Κάθε χαρακτήρας αντιστοιχεί σε ένα ακέραιο αριθμό στην περιοχή $0 \ldots 255$. Οι χαρακτήρες $a \ldots z$ βρίσκονται στη σειρά, δηλαδή αντιστοιχούν σε διαδοχικούς ακέραιους αριθμούς. Το ίδιο συμβαίνει και για τους $A \ldots Z$ και $0 \ldots 9$.

Ορισμένοι ειδικοί χαρακτήρες γράφονται χρησιμοποιώντας το σύμβολο `\` και ονομάζονται *χαρακτήρες διαφυγής*. Πχ. `'\n'   '\0'`
Ο παρακάτω πίνακας συνοψίζει τους συχνότερα χρησιμοποιούμενους χαρακτήρες διαφυγής.

Χαρακτήρας διαφυγής	*Σημαίνει*
`\n`	Αλλαγή γραμμής (new line)
`\b`	Μία θέση πίσω (backspace)
`\f`	Αλλαγή σελίδας (form feed)
`\r`	Αρχή γραμμής (carriage return)
`\t`	Στηλοθέτης (tab)
`\\`	Ο χαρακτήρας \
`\'`	Απλή απόστροφος
`\"`	Διπλή απόστροφος
`\0`	Χαρακτήρας που αντιστοιχεί στον ακέραιο 0

3.2 Αριθμητικές παραστάσεις

Χρησιμοποιώντας τις πέντε απλές πράξεις (`+ - * / %`) μεταξύ αριθμών, μπορούμε να κατασκευάσουμε πιο σύνθετες παραστάσεις. Πχ `2*3 + 7*(14 - 6%4/2)`

Για να υπολογιστεί το αποτέλεσμα μιας τέτοιας σύνθετης παράστασης χρησιμοποιούμε τους παρακάτω κανόνες:

1. Η σειρά των πράξεων καθορίζεται από την *προτεραιότητα* των αντίστοιχων τελεστών σύμφωνα με τον παρακάτω πίνακα:

Πράξη	*Προτεραιότητα*
`* / %`	Υψηλή
`+ -`	Χαμηλή

2. Μεταξύ τελεστών με την ίδια προτεραιότητα η σειρά των πράξεων καθορίζεται από την *προσεταιριστικότητα* του τελεστή. Για όλους τους τελεστές του ανωτέρω πίνακα η προσεταιριστικότητα είναι από αριστερά προς τα δεξιά. Πχ. στην παράσταση 2*6/3 όπου ο πολλαπλασιασμός και η διαίρεση έχουν ίδια προτεραιότητα πρώτα θα γίνει η πράξη 2*6 και κατόπιν το αποτέλεσμα θα διαιρεθεί με το 3. Σημειώστε όμως ότι υπάρχουν τελεστές (που περιγράφονται στα επόμενα κεφάλαια), όπου η προσεταιριστικότητά τους είναι από δεξιά προς τα αριστερά.

3. Ζεύγη παρενθέσεων μπορούν να χρησιμοποιηθούν για να αλλάξουν τη σειρά των πράξεων. Πχ. στην παράσταση (3+5)*4 παρότι ο πολλαπλασιασμός έχει μεγαλύτερη προτεραιότητα έναντι της πρόσθεσης, θα γίνει πρώτα η πράξη που βρίσκεται εντός των παρενθέσεων και κατόπιν το αποτέλεσμα θα πολλαπλασιαστεί με 4.

4. Όπως ήδη έχει αναφερθεί πράξεις μεταξύ ακεραίων δίνουν πάντα ακέραιο αποτέλεσμα, πράξεις μεταξύ πραγματικών δίνουν πραγματικό αποτέλεσμα, ενώ πράξεις μεταξύ πραγματικών αριθμών διπλής ακρίβειας δίνουν πάντα ως αποτέλεσμα αριθμό διπλής ακρίβειας. Στην περίπτωση που σε μια πράξη υπάρχουν δύο διαφορετικοί τύποι δεδομένων τότε πριν γίνει η πράξη ο κατώτερος τύπος προάγεται στον ανώτερο, ενώ το αποτέλεσμα είναι πάντα του ανώτερου τύπου. Ο τύπος `double` θεωρείται ανώτερος του `float`, ο οποίος με τη σειρά του θεωρείται ανώτερος του `int`.

Παράδειγμα:
Για να υπολογιστεί η παράσταση 4+7.2 ο ακέραιος αριθμός 4 προάγεται στον πραγματικό διπλής ακρίβειας 4.0 ενώ το αποτέλεσμα (11.2) είναι πραγματικός αριθμός διπλής ακρίβειας. Επίσης για να υπολογιστεί η παράσταση 8/3.2f ο ακέραιος αριθμός 8 προάγεται στον πραγματικό αριθμό απλής ακρίβειας 8.0f ενώ το αποτέλεσμα (2.5) είναι πραγματικός αριθμός απλής ακρίβειας.

Παράδειγμα:
Θεωρήστε την παράσταση: `2*3 + 7*(14 - 6%4/2)`. Η σειρά των πράξεων έχει ως εξής:

	Πράξη	*Μερικό αποτέλεσμα*
	Αρχική παράσταση	`2*3 + 7*(14 - 6%4/2)`
1	Πρώτα θα γίνουν οι πράξεις εντός των παρενθέσεων. Συγκεκριμένα το υπόλοιπο διαίρεσης και η διαίρεση έχουν τη μεγαλύτερη προτεραιότητα. Η προσεταιριστικότητα αυτών των δύο πράξεων είναι από αριστερά προς τα δεξιά, κατά συνέπεια πρώτα θα γίνει το υπόλοιπο διαίρεσης.	`2*3 + 7*(14 - 2/2)`
2	Κατόπιν θα γίνει η διαίρεση 2/2	`2*3 + 7*(14 - 1)`
3	Ακολουθεί η αφαίρεση 14-1, οπότε ολοκληρώνονται οι πράξεις εντός των παρενθέσεων.	`2*3 + 7*13`
4	Από τις πράξεις που απομένουν ο πολλαπλασιασμός έχει μεγαλύτερη προτεραιότητα. Επειδή υπάρχουν δύο πολλαπλασιασμοί, θα γίνει πρώτα αυτός που είναι πιο αριστερά, δηλαδή 2*3	`6 + 7*13`
5	Κατόπιν γίνεται ο επόμενος πολλαπλασιασμός 7*13	`6 + 91`
6	Τέλος γίνεται η πρόσθεση δίνοντας το τελικό αποτέλεσμα	97

3.3 Μαθηματικές συναρτήσεις

Η C++ διαθέτει ένα σημαντικό αριθμό από μαθηματικές συναρτήσεις που είναι διαθέσιμες σε όποιο πρόγραμμα τις χρειαστεί. Όλες οι συναρτήσεις πλην της **pow** δέχονται ως όρισμα αριθμούς τύπου `float` ή `double` και επιστρέφουν αποτέλεσμα του αντίστοιχου τύπου. Για να χρησιμοποιήσουμε οποιαδήποτε μαθηματική συνάρτηση πρέπει να προσθέσουμε στην αρχή του προγράμματος:

```
#include <cmath>
```

Στον παρακάτω πίνακα παρουσιάζονται οι συνηθέστερα χρησιμοποιούμενες μαθηματικές συναρτήσεις.

Συνάρτηση	*Τι επιστρέφει*
`sqrt(x)`	Τετραγωνική ρίζα, $\sqrt{x}$
`exp(x)`	Εκθετικό, e^x
`abs(x)`	Απόλυτη τιμή, $\|x\|$
`ln(x)`	Φυσικός λογάριθμος, $\ln x$
`log(x)`	Δεκαδικός λογάριθμος, $\log_{10} x$
`sin(x)`	Ημίτονο, $\sin x$
`cos(x)`	Συνημίτονο, $\cos x$
`tan(x)`	Εφαπτομένη, $\tan x$
`acos(x)`	Τόξο συνημιτόνου, $\arccos x$
`asin(x)`	Τόξο ημιτόνου, $\arcsin x$
`atan(x)`	Τόξο εφαπτομένης, $\arctan x$
`pow(a,b)`	Ύψωση σε δύναμη a^b.

Ορισμένες παρατηρήσεις για τις συναρτήσεις και τον τρόπο που χρησιμοποιούνται:

1. Οι ανωτέρω συναρτήσεις πλην της `pow` δεν δέχονται ως όρισμα ακέραιο αριθμό. Έτσι για να βρούμε την τετραγωνική ρίζα του 2 πρέπει να γράψουμε `sqrt(2.)` και όχι `sqrt(2)` που αποτελεί συντακτικό λάθος.

2. Οι τριγωνομετρικές συναρτήσεις `sin`, `cos` και `tan` δέχονται τα ορίσματά τους σε ακτίνια και όχι μοίρες. Έτσι αν γράψουμε `sin(45.)` θα υπολογιστεί το ημίτονο των 45 ακτινίων και όχι των 45 μοιρών.

3. Επίσης και οι αντίστροφες τριγωνομετρικές συναρτήσεις `asin`, `acos` και `atan` επιστρέφουν το αποτέλεσμά τους σε ακτίνια και όχι μοίρες.

4. Τα ορίσματα κάθε συνάρτησης πρέπει να είναι εντός του πεδίου ορισμού της συνάρτησης. Για παράδειγμα δεν μπορούμε να ζητήσουμε την τετραγωνική ρίζα αρνητικού αριθμού, πράγμα που αποτελεί λογικό σφάλμα. Σε μια τέτοια περίπτωση το πρόγραμμα θα σταματήσει στο σημείο αυτό εμφανίζοντας ένα μήνυμα λάθους.

5. Ειδικά για τη συνάρτηση `pow(a,b)` ο παρακάτω πίνακας συνοψίζει τους επιτρεπτούς τύπους ορισμάτων και τα αντίστοιχα αποτελέσματα.

Τύπος βάσης (a)	*Τύπος εκθέτη (b)*	*Τύπος αποτελέσματος pow(a,b)*
`float`	`float`	`float`
`double`	`double`	`double`
`float`	`int`	`float`
`double`	`int`	`double`

   Προσοχή: Η βάση (a) δεν μπορεί να είναι ακέραιος αριθμός. Πχ. `pow(4,2)` είναι **λάθος**.

6. Εάν χρειάζεται να υπολογίσουμε τη βάση των φυσικών λογαρίθμων e (≈ 2.71828) γράφουμε `exp(1.0)`.

7. Για να υπολογίσουμε τον αριθμό π (≈ 3.14159) γράφουμε `acos(-1.0)`.

3.4 Μεταβλητές

Μεταβλητή είναι ένα συμβολικό όνομα που δίνεται σε κάποια θέση μνήμης του υπολογιστή για να αποθηκευτεί ένας αριθμός (ή άλλος τύπος δεδομένων). Κάθε μεταβλητή έχει όνομα και τύπο, που δηλώνονται υποχρεωτικά στην αρχή του προγράμματος. Τα ονόματα των μεταβλητών επιλέγονται από τον προγραμματιστή, και πρέπει να ακολουθούν τους παρακάτω κανόνες.

1. Επιτρέπονται μόνο λατινικοί χαρακτήρες, αριθμοί και το σύμβολο _
2. Ο πρώτος χαρακτήρας πρέπει να είναι γράμμα.
3. Κεφαλαία και μικρά θεωρούνται διαφορετικά.
4. Δεν υπάρχει μέγιστο μήκος στο όνομα, όμως πρακτικά κάθε μεταφραστής επιβάλλει ένα μέγιστο μήκος (πχ 32 χαρακτήρες).

Οι ίδιοι κανόνες ισχύουν και για άλλα ονόματα, πχ. ονόματα συναρτήσεων.

Παράδειγμα:
Σε ένα πρόγραμμα απαιτούνται οι ακέραιες μεταβλητές `n`, `kmin` και `pti`, οι πραγματικές μεταβλητές `t` και `xmin` και οι μεταβλητές διπλής ακρίβειας `f` και `radius`. Το αντίστοιχο απόσπασμα προγράμματος θα ήταν ως εξής:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int n, kmin, pti;`	Δηλώνονται οι ακέραιες μεταβλητές.
`float t, xmin;`	Δηλώνονται οι πραγματικές μεταβλητές.
`double f, radius;`	Δηλώνονται οι μεταβλητές διπλής ακρίβειας.
⋮	

Χρησιμοποιώντας μεταβλητές μπορούμε να κάνουμε αριθμητικές πράξεις και να κατασκευάσουμε σύνθετες παραστάσεις ακριβώς όπως περιγράφηκε στις προηγούμενες παραγράφους.

3.5 Ανάθεση τιμής

3.5.1 Εντολή ανάθεσης τιμής

Για να δώσουμε τιμή σε μια μεταβλητή χρησιμοποιούμε την εντολή ανάθεσης τιμής. Η εντολή αυτή έχει τη μορφή:

$$\mu\varepsilon\tau\alpha\beta\lambda\eta\tau\acute{\eta} = \alpha\rho\iota\theta\mu\eta\tau\iota\kappa\acute{\eta}\ \pi\alpha\rho\acute{\alpha}\sigma\tau\alpha\sigma\eta\ ;$$

Στο αριστερό μέρος του = βρίσκεται πάντα το όνομα μιας μεταβλητής (που θα πρέπει ήδη να έχουμε δηλώσει στην αρχή του προγράμματος). Στο δεξί μέρος της εντολής βρίσκεται πάντα μια αριθμητική παράσταση.

Η εντολή λειτουργεί ως εξής: Πρώτα υπολογίζεται η τιμή της αριθμητικής παράστασης στο δεξί μέρος και κατόπιν το αποτέλεσμα που θα προκύψει ανατίθεται ως τιμή της μεταβλητής που βρίσκεται στο αριστερό μέρος της εντολής.

Παράδειγμα:

```
sum = a+b;
```

Εδώ πρώτα θα υπολογιστεί το άθροισμα `a+b` και κατόπιν το αποτέλεσμα θα ανατεθεί ως τιμή της μεταβλητής `sum`.

Άλλα παραδείγματα:

```
k = 1;
d = b*b-4*a*c;
erm = (xm-xs)/2;
m = m+1;
```

Ειδικότερα στο τελευταίο παράδειγμα υπολογίζεται πρώτα η παράσταση `m+1` και κατόπιν το αποτέλεσμα ανατίθεται και πάλι ως τιμή της μεταβλητής `m` αντικαθιστώντας την προηγούμενη τιμή. Έτσι η εντολή αυτή έχει ως αποτέλεσμα να αυξηθεί κατά 1 η τιμή της μεταβλητής `m`.

3.5.2 Ανάθεση τιμής κατά τη δήλωση

Μπορούμε να δώσουμε αρχική τιμή σε μια μεταβλητή μέσω της δήλωσής της.

Παραδείγματα:

```
int k=0;
double x=1.45, y=0.0;
```

Σημειώστε ότι η τιμή που ανατίθεται είναι μια αρχική τιμή όταν το πρόγραμμα ξεκινάει να εκτελείται και μπορεί να τροποποιηθεί στη συνέχεια.

3.6 Άλλοι αριθμητικοί τελεστές

3.6.1 Τελεστές αύξησης και μείωσης

Για την ειδική περίπτωση της πράξης μεταξύ ακεραίων:

```
k = k+1 ;
```

η C++ έχει ένα ειδικό τελεστή. Μπορούμε να γράψουμε:

```
++k ;
```
(προθεματική μορφή)

ή

```
k++ ;
```
(επιθεματική μορφή)

Στην προθεματική μορφή πρώτα αυξάνεται η τιμή της μεταβλητής και κατόπιν η τιμή χρησιμοποιείται. Στην επιθεματική μορφή πρώτα χρησιμοποιείται η τιμή της μεταβλητής και κατόπιν η τιμή της μεταβλητής αυξάνεται.

Παράδειγμα:

```
k = 5;
a = ++k;
```

Εδώ χρησιμοποιείται η προθεματική μορφή του τελεστή αύξησης, οπότε πρώτα αυξάνεται η τιμή της μεταβλητής `k` και γίνεται 6. Κατόπιν η τιμή αυτή ανατίθεται στη μεταβλητή `a` που γίνεται και αυτή 6.

Παράδειγμα:

```
k = 5;
a = k++;
```

Χρησιμοποιείται η επιθεματική μορφή του τελεστή αύξησης, οπότε πρώτα ανατίθεται στη μεταβλητή `a` η τιμή της `k` (δηλαδή 5) και κατόπιν η τιμή της `k` αυξάνεται και γίνεται 6.

Αντίστοιχα με τον τελεστή αύξησης `++` υπάρχει ο τελεστής μείωσης `--` σε προθεματική και επιθεματική μορφή. Οι τελεστές `++` και `--` χρησιμοποιούνται μόνο σε ακέραιους αριθμούς. Εντός μιας αριθμητικής παράστασης οι τελεστές `++` και `--` έχουν υψηλότερη προτεραιότητα από τους τελεστές `+ - * / %` ενώ η προσεταιριστικότητα τους είναι από δεξιά προς τα αριστερά.

Παράδειγμα:

Ποια είναι η τιμή των ακέραιων μεταβλητών `n`, `j` μετά από τις εντολές:

```
n = 10;
j = ++n % 4;
```

Επειδή χρησιμοποιείται η προθεματική μορφή του τελεστή `++` η τιμή της μεταβλητής `n` θα γίνει 11 και κατόπιν θα υπολογιστεί το υπόλοιπο της διαίρεσης του 11 από το 4. Έτσι τελικά η μεταβλητή `j` λαμβάνει την τιμή 3.

Παράδειγμα:

Ποια είναι η τιμή των ακέραιων μεταβλητών `m`, `k` μετά από τις εντολές:

```
m = 4;
k = 2 + m++;
```

Χρησιμοποιείται η επιθεματική μορφή του τελεστή `++`, οπότε χρησιμοποιείται η τιμή που ει η μεταβλητή `m` (δηλαδή 4) για να γίνει το άθροισμα 2+4 και το αποτέλεσμα ανατίθεται στη μεταβλητή `k`. Κατόπιν αυξάνεται η τιμή της μεταβλητής `m` και γίνεται 5.

3.6.2 Τελεστές ανάθεσης τιμής

Η εντολή ανάθεσης τιμής

```
k = k+m ;
```

μπορεί να γραφεί στη C++ ως:

```
k += m ;
```

Όμοια υπάρχουν και τελεστές για τις υπόλοιπες πράξεις όπως φαίνεται στον παρακάτω πίνακα.

Τελεστής	*Σημαίνει*	
`k -= m`	`k = k-m`	
`k *= m`	`k = k*m`	
`k /= m`	`k = k/m`	
`k %= m`	`k = k%m`	(Μόνο για ακέραιους)

Όλοι οι ανωτέρω τελεστές μπορούν να χρησιμοποιηθούν για πραγματικούς ή ακέραιους, πλην του τελεστή `%=` που είναι μόνο για ακεραίους. Η προτεραιότητά τους είναι πιο χαμηλή από αυτή των αριθμητικών πράξεων, ενώ η προσεταιριστικότητά τους είναι από δεξιά προς τα αριστερά.

3.6.3 Τελεστής αλλαγής τύπου

Μπορούμε να αλλάξουμε τον τύπο μιας αριθμητικής παράστασης με τον τελεστή αλλαγής τύπου.

$$\textit{μεταβλητή} = \texttt{(}\ \textit{τύπος}\ \texttt{)}\ \textit{παράσταση}\ \texttt{;}$$

όπου ο *τύπος* μπορεί να είναι `int`, `float`, `double`, `char` κλπ. Όταν αλλάζουμε μια παράσταση σε κατώτερο τύπο (πχ. από `double` σε `float`) χάνουμε σε ακρίβεια. Πιο συγκεκριμένα όταν μετατρέπουμε μια πραγματική παράσταση σε ακέραια με τον τελεστή `(int)` τα δεκαδικά ψηφία αποκόπτονται.

Η αλλαγή τύπου έχει υψηλότερη προτεραιότητα από τις πράξεις. Πχ η μετατροπή
```
(int)x+0.5
```
είναι διαφορετική από την
```
(int)(x+0.5)
```

3.7 Απλές εντολές εισόδου–εξόδου

3.7.1 Εντολή cout

Όπως είδαμε για να εμφανιστεί ένα μήνυμα στην οθόνη χρησιμοποιούμε την εντολή `cout` ως εξής:

$$\texttt{cout << "}\textit{μήνυμα}\texttt{" ;}$$

Με την ανωτέρω εντολή εμφανίζεται στην οθόνη ότι βρίσκεται εντός των διπλών αποστρόφων. Πχ.

```
cout << "Hello\n";
```

Ο χαρακτήρας διαφυγής `\n` στο τέλος του μηνύματος είναι απαραίτητος όταν θέλουμε να εμφανίσουμε μια νέα γραμμή στην οθόνη.

Παράδειγμα:
Οι εντολές:
```
cout << "Hello\n";
out << "World\n";
```
θα εμφανίσουν στην οθόνη:

```
Hello
World
```
ενώ με τις εντολές:
```
cout << "Hello ";
cout << "World\n";
```
θα εμφανιστεί στην οθόνη:
```
Hello World
```

Με την εντολή `cout` μπορούμε να εμφανίσουμε στην οθόνη και την τιμή μιας μεταβλητής. Στην περίπτωση αυτή η εντολή `cout` έχει τη μορφή:

`cout <<` *μεταβλητή* `;`

Προσέξτε ότι η μεταβλητή δεν τοποθετείται εντός αποστρόφων. Πολλαπλές μεταβλητές και μηνύματα μπορούν να εμφανιστούν στην οθόνη από μία μόνο εντολή `cout`. Πχ.

```
cout << "Ο όγκος είναι " << v << "ενώ το εμβαδόν είναι " << s << '\n';
```

Ότι βρίσκεται εντός αποστρόφων θεωρείται μήνυμα και εμφανίζεται στην οθόνη ως έχει (συμπεριλαμβανομένων και τυχόν κενών), ενώ τα υπόλοιπα θεωρούνται μεταβλητές (ή γενικότερα αριθμητικές παραστάσεις) και εμφανίζεται η αντίστοιχη τιμή. Ο χαρακτήρας `'\n'` στο τέλος της εντολής, χρησιμεύει για την αλλαγή γραμμής και στη θέση του θα μπορούσαμε να χρησιμοποιήσουμε το προκαθορισμένο συμβολικό όνομα `endl`. Πχ.

```
cout << "Ο όγκος είναι " << v << "ενώ το εμβαδόν είναι " << s << endl;
```

Το συμβολικό όνομα `endl` δεν μπορεί όμως να χρησιμοποιηθεί εντός αποστρόφων. Πχ. το παρακάτω είναι λάθος:

```
cout << "Hello World endl";
```
ΛΑΘΟΣ.

Επίσης σημειώστε ότι το ίδιο αποτέλεσμα με την ανωτέρω εντολή `cout` θα μπορούσαμε να πετύχουμε με πολλαπλές εντολές `cout` ως εξής;

```
cout << "Ο όγκος είναι " << v ;
cout << "ενώ το εμβαδόν είναι " << s ;
cout << endl ;
```

3.7.2 Εντολή cin

Με την εντολή `cin` μπορούμε να εισάγουμε από το πληκτρολόγιο την τιμή μιας μεταβλητής κατά την εκτέλεση του προγράμματος. Η εντολή συντάσσεται ως εξής:

`cin >>` *μ1* `>>` *μ2* `>>` *μ3* `...;`

όπου *μ1*, *μ2*, *μ3*, ... είναι μεταβλητές.

Παράδειγμα:
```
cin >> x >> y;
```

Η εντολή λειτουργεί ως εξής: Μόλις το πρόγραμμα φτάσει στην εντολή `cin` σταματά και περιμένει από το χρήστη να πληκτρολογήσει δύο αριθμούς. Ότι πληκτρολογήσει ο χρήστης ανατίθεται ως τιμή στις αντίστοιχες μεταβλητές (`x`,`y`) και το πρόγραμμα συνεχίζει. Είναι σύνηθες η εντολή `cin` να χρησιμοποιείται μαζί με μια εντολή `cout` που εμφανίζει στην οθόνη κάποιο μήνυμα σχετικά με το τι πρέπει να πληκτρολογήσει ο χρήστης.

Παράδειγμα:

```
cout << "Εισάγετε τις συντεταγμένες X και Y \n";
cin >> x >> y;
```

Εάν ο χρήστης πληκτρολογήσει λιγότερα νούμερα και πατήσει ENTER, το πρόγραμμα εξακολουθεί να περιμένει τα υπόλοιπα. Εάν ο χρήστης εισάγει περισσότερα νούμερα από όσα απαιτούνται τότε το πρόγραμμα διαβάζει αυτά που χρειάζεται και τα υπόλοιπα αγνοούνται.

3.7.3 Μορφοποίηση της εξόδου

Θα παρατηρήσατε ότι όταν εμφανίζεται ένας πραγματικός αριθμός με την εντολή `cout` το πλήθος των δεκαδικών ψηφίων που θα δούμε αλλά και το εύρος των θέσεων που καταλαμβάνει στην οθόνη αποφασίζεται αυτόματα από τον υπολογιστή. Για αν μορφοποιήσουμε κατά βούληση την εμφάνιση των αποτελεσμάτων μας στην οθόνη χρησιμοποιούμε τους *χειριστές* στην εντολή `cout`.

Πχ για να εμφανίσουμε ακέραια μεταβλητή `k` έτσι ώστε ώστε να καταλαμβάνει 5 θέσεις στην οθόνη:

```
cout << setw(5) << k << endl;
```

Οι βασικότεροι χειριστές φαίνονται στον παρακάτω πίνακα.

Χειριστής	*Τι κάνει*
`setw(n)`	Η επόμενη έξοδος θα καταλαμβάνει `n` θέσεις στην οθόνη.
`setprecision(n)`	Όλες οι επόμενες έξοδοι θα γίνουν με `n` δεκαδικά ψηφία.
`fixed`	Όλες οι επόμενες έξοδοι θα γίνουν με την κλασσική αναπαράσταση.
`scientific`	Όλες οι επόμενες έξοδοι θα γίνουν με την επιστημονική αναπαράσταση.

Για να χρησιμοποιήσουμε τους ανωτέρω χειριστές μορφοποίησης πρέπει να προσθέσουμε στην αρχή του προγράμματος:

```
#include <iomanip>
```

Παράδειγμα:
Για να εμφανίσουμε μια πραγματική μεταβλητή `x` με την κλασσική αναπαράσταση σε εύρος 10 θέσεων στην οθόνη με 7 δεκαδικά ψηφία, γράφουμε:

```
cout << fixed << setprecision(7) << setw(10) << x << endl;
```

3.8 Παραδείγματα

3.8.1 Επιφάνεια και όγκος κυλίνδρου

Θεωρείστε ένα κύλινδρο με ακτίνα βάσης r και ύψος h. Κατασκευάστε πρόγραμμα που θα υπολογίζει τη συνολική επιφάνεια και τον όγκο του κυλίνδρου που δίνονται αντίστοιχα από:

$$S = 2\pi r^2 + 2\pi rh \qquad V = \pi r^2 h$$

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double r, h;`	Δήλωση πραγματικών μεταβλητών διπλής ακρίβειας. `r` είναι η ακτίνα βάσης και `h` είναι το ύψος.
`double pi;`	Δήλωση πραγματικών μεταβλητών διπλής ακρίβειας. `pi` είναι ο αριθμός π.
`double s, v;`	Δήλωση πραγματικών μεταβλητών διπλής ακρίβειας. `s` είναι η επιφάνεια και `v` είναι ο όγκος.
`cout << "Εισάγετε ακτίνα και ύψος ";`	Εμφανίζεται προτρεπτικό μήνυμα.
`cin >> r >> h;`	Εισάγονται από το πληκτρολόγιο οι τιμές των `r` και `h`.
`pi = acos(-1.0);`	Υπολογίζεται η τιμή του π.
`s = 2*pi*r*r + 2*pi*r*h;`	Υπολογίζεται το εμβαδόν.
`v = pi*r*r*h;`	Υπολογίζεται ο όγκος.
`cout << "Το εμβαδόν είναι "` `<< s << endl;`	Εμφανίζονται στην οθόνη τα αποτελέσματα.
`cout << "Ο όγκος είναι "` `<< v << endl;`	
`}`	

3.8.2 Μετατροπή δευτερολέπτων

Κατασκευάστε πρόγραμμα που θα μετατρέπει ένα δεδομένο αριθμό δευτερολέπτων σε ώρες, λεπτά και δευτερόλεπτα.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int sec;`	Ο αρχικός αριθμός δευτερολέπτων.
`int hour;`	Οι ώρες μετά τη μετατροπή.
`int ypolmin;`	Τα λεπτά μετά τη μετατροπή.
`int ypolsec;`	Τα δευτερόλεπτα μετά τη μετατροπή.
`int min;`	Χρησιμοποιείται κατά τη διάρκεια της μετατροπής.
`cout << "Εισάγετε τα δευτερόλεπτα ";`	Εμφανίζεται προτρεπτικό μήνυμα.
`cin >> sec;`	Εισάγεται η τιμή της μεταβλητής sec από το πληκτρολόγιο.
`min = sec / 60;`	Υπολογίζονται τα λεπτά (ακέραια διαίρεση).
`ypolsec = sec % 60;`	Υπολογίζονται τα δευτερόλεπτα που περισσεύουν από την ανωτέρω διαίρεση.
`hour = min / 60 ;`	Υπολογίζονται οι ώρες (ακέραια διαίρεση).
`ypolmin = min % 60 ;`	Υπολογίζονται τα λεπτά που περισσεύουν από την ανωτέρω διαίρεση.
`cout << sec << " δευτερόλεπτα είναι\n";`	Εμφανίζονται στην οθόνη
`cout << hour << " ώρες \n";`	τα αποτελέσματα.
`cout << ypolmin << " λεπτά \n";`	
`cout << ypolsec << " δευτερόλεπτα \n;`	
`}`	

Κεφάλαιο 4

Εντολές ελέγχου

4.1 Εντολή if

4.1.1 Πως συντάσσεται

Η εντολή `if` επιτρέπει να γίνονται συγκρίσεις μεταξύ μεταβλητών και ανάλογα με το αποτέλεσμα να εκτελείται ή όχι κάποιο τμήμα προγράμματος. Η εντολή συντάσσεται ως εξής:

`if (` *σύγκριση* `)`
    εντολή `;`

Η *σύγκριση* είναι μια σύγκριση μεταξύ δύο μεταβλητών ή γενικότερα μεταξύ δύο παραστάσεων, ενώ *εντολή* είναι οποιαδήποτε εκτελέσιμη εντολή. Οι δυνατές συγκρίσεις που μπορεί να γίνουν είναι οι παρακάτω:

Σύγκριση	*Σημαίνει*
`a == b`	$a = b$
`a != b`	$a \neq b$
`a < b`	$a < b$
`a <= b`	$a \leq b$
`a > b`	$a > b$
`a >= b`	$a \geq b$

Το αποτέλεσμα κάθε σύγκρισης είναι είτε αληθές (ισχύει) είτε ψευδές (δεν ισχύει). Στην πράξη για το αληθές χρησιμοποιείται ο ακέραιος αριθμός 1, ενώ για το ψευδές ο ακέραιος αριθμός 0.

Η ανωτέρω εντολή `if` λειτουργεί ως εξής: Αρχικά πραγματοποιείται η *σύγκριση*. Αν το αποτέλεσμά της είναι αληθές τότε η *εντολή* που βρίσκεται αμέσως μετά εκτελείται. Αν το αποτέλεσμα της σύγκρισης είναι ψευδές, η *εντολή* δεν εκτελείται (παρακάμπτεται).

Σε περίπτωση που θέλουμε να εκτελεστούν παραπάνω από μια εντολές όταν η `if` είναι αληθής, χρησιμοποιούμε άγκιστρα:

```
if ( σύγκριση ) {
     εντολή ;
     εντολή ;
       ⋮
}
```

Στην περίπτωση που θέλουμε να εκτελέσουμε ένα τμήμα κώδικα όταν η *σύγκριση* είναι αληθής και ένα άλλο όταν είναι ψευδής, τότε η εντολή `if` έχει την μορφή:

```
if ( σύγκριση )
    εντολή ;
else
    εντολή ;
```

Με αυτή τη σύνταξη όταν η *σύγκριση* είναι αληθής εκτελείται η εντολή που βρίσκεται μετά το `if`, ενώ όταν η *σύγκριση* είναι ψευδής εκτελείται η εντολή που βρίσκεται μετά το `else`. Και εδώ μπορούν να χρησιμοποιηθούν άγκιστρα όταν θέλουμε να εκτελούνται πολλές εντολές:

```
if ( σύγκριση ) {
     εντολή ;
     εντολή ;
       ⋮
} else {
     εντολή ;
     εντολή ;
       ⋮
}
```

Ορισμένες παρατηρήσεις για την εντολή `if`.

1. Εντός μιας εντολής `if` μπορεί να υπάρχει οποιαδήποτε άλλη εντολή συμπεριλαμβανομένων και άλλων εντολών `if`.

2. Σε μια εντολή `if` εκτελούνται είτε οι εντολές που βρίσκονται μετά το `if` ή οι εντολές που βρίσκονται μετά το `else`.

3. Το τμήμα `else` δεν είναι υποχρεωτικό να υπάρχει.

4. Οι εντολές που βρίσκονται εντός της εντολής `if` γράφονται λίγο πιο δεξιά, αφήνοντας μερικά κενά ή πατώντας το πλήκτρο TAB. Αυτό δεν απαιτείται από το συντακτικό της γλώσσας, αλλά είναι χρήσιμο ώστε ο κώδικας να είναι ευανάγνωστος.

5. Η εντολή `if` από το σημείο που αρχίζει μέχρι το σημείο που τελειώνει (συμπεριλαμβανομένων και όλων των εντολών που περιέχει) **θεωρείται συντακτικά ως μία εντολή**. Αυτό έχει σημασία σε σημεία όπου το συντακτικό της γλώσσας απαιτεί μία εντολή.

4.1.2 Παραδείγματα

4.1.2.1 Εύρεση του μεγαλύτερου από δύο αριθμούς

Δεδομένων δύο αριθμών a, b να βρεθεί ποιος είναι ο μεγαλύτερος.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int a, b;`	Οι δύο αριθμοί που θα ελέγξουμε.
`int max;`	Ο μεγαλύτερος από τους δύο.
`cout << "Εισάγετε δύο ακέραιους ";`	Εμφανίζεται στην οθόνη προτρεπτικό μήνυμα.
`cin >> a >> b;`	Εισαγωγή των αριθμών από το πληκτρολόγιο.
`if ( a > b )`	Ελέγχουμε αν ο a είναι μεγαλύτερος από τον b.
`max = a;`	Αν η ανωτέρω σύγκριση είναι αληθής, θέτουμε ως μεγαλύτερο τον a.
`else`	Εδώ έρχεται το πρόγραμμα αν η σύγκριση είναι ψευδής.
`max = b;`	Θέτουμε ως μεγαλύτερο τον b.
`cout << "Ο μεγαλύτερος είναι ο "`	Εμφάνιση στην οθόνη του μεγαλύτερου.
`<< max << endl ;`	
`}`	

Στο ανωτέρω παράδειγμα χρησιμοποιήσαμε ακέραιους αριθμούς, όμως η ίδια τεχνική εφαρμόζεται και με άλλους τύπους δεδομένων.

4.1.2.2 Εύρεση του μεγαλύτερου από τρεις αριθμούς

Δεδομένων τριών αριθμών a, b και c να βρεθεί ποιος είναι ο μεγαλύτερος.

Εδώ εφαρμόζουμε την τεχνική του προηγούμενου παραδείγματος δύο φορές. Συγκεκριμένα βρίσκουμε το μεγαλύτερο από τους a και b και τον ονομάζουμε `max2`. Κατόπιν βρίσκουμε το μεγαλύτερο από τους `max2` και c.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int a, b, c;`	Οι τρεις αριθμοί που θα ελέγξουμε.
`int max2;`	Ο μεγαλύτερος από τους a και b.
`int max;`	Ο μεγαλύτερος από τους τρεις αριθμούς.
`cout << "Δώστε τρεις ακέραιους ";`	Εμφανίζεται στην οθόνη προτρεπτικό μήνυμα.
`cin >> a >> b >> c;`	Εισαγωγή των αριθμών από το πληκτρολόγιο.
`if ( a > b )`	Με αυτή την εντολή `if` βρίσκουμε το
`max2 = a;`	μεγαλύτερο από τους a και b και τον
`else`	αποθηκεύουμε στη μεταβλητή `max2`.
`max2 = b;`	
`if ( c > max2 )`	Εδώ βρίσκουμε το μεγαλύτερο από τους `max2`
`max = c;`	και c και τον αποθηκεύουμε στη μεταβλητή `max`.
`else`	
`max = max2;`	
`cout << "Ο μεγαλύτερος είναι ο "`	Εμφάνιση στην οθόνη του μεγαλύτερου.
`<< max << endl;`	
`}`	

4.1.2.3 Διαχωρισμός άρτιων–περιττών

Το παρακάτω πρόγραμμα αποφασίζει αν ένας ακέραιος αριθμός που εισάγεται από το πληκτρολόγιο είναι άρτιος ή περιττός.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int k;`	Ο αριθμός που θα ελέγξουμε.
`cout << "Εισάγετε έναν ακέραιο ";`	Εμφανίζεται προτρεπτικό μήνυμα.
`cin >> k;`	Εισαγωγή του αριθμού από το πληκτρολόγιο.
`if ( k%2 == 0 )`	Ελέγχουμε αν το υπόλοιπο της διαίρεσης
`cout << "Είναι άρτιος\n";`	του k με το δύο είναι μηδέν.
`else`	
`cout << "Είναι περιττός\n";`	
`}`	

4.1.2.4 Εξίσωση δευτέρου βαθμού

Το παρακάτω πρόγραμμα βρίσκει τις πραγματικές λύσεις της εξίσωσης δευτέρου βαθμού

$$ax^2 + bx + c$$

όταν δίνονται οι συντελεστές a, b, c. Σημειώστε ότι το πρόγραμμα λαμβάνει υπόψη και την περίπτωση $a = 0$, δηλαδή όταν πρόκειται για εξίσωση πρώτου βαθμού, και βρίσκει την αντίστοιχη λύση.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double a, b, c;`	Οι συντελεστές της εξίσωσης.
`double d;`	Η διακρίνουσα.
`double x1, x2;`	Οι δύο λύσεις.
`cout << "Εισάγετε τα a,b,c ";`	Εμφάνιση προτρεπτικού μηνύματος.
`cin >> a >> b >> c;`	Εισαγωγή των συντελεστών από το πληκτρολόγιο.
`if ( a != 0 ) {`	Ελέγχεται αν ο συντελεστής a είναι διάφορος του μηδενός.
`d = b*b-4*a*c;`	Υπολογίζεται η διακρίνουσα.
`if ( d > 0 ) {`	Ελέγχεται αν η διακρίνουσα είναι θετική.
`x1 = (-b+sqrt(d))/(2*a);`	Υπολογίζεται η πρώτη λύση.
`x2 = (-b-sqrt(d))/(2*a);`	Υπολογίζεται η δεύτερη λύση.
`cout << "Δύο λύσεις:\n";`	Εμφανίζονται στην οθόνη οι δύο λύσεις.
`cout << x1 << endl;`	
`cout << x2 << endl;`	
`} else if ( d == 0 ) {`	Εδώ έρχεται το πρόγραμμα όταν η διακρίνουσα είναι μηδέν.
`x1 = -b/(2*a);`	Υπολογίζεται η μία μοναδική λύση.
`cout << "Μία λύση:\n";`	Εμφανίζεται η λύση στην οθόνη.
`cout << x1 << endl;`	
`} else`	Εδώ φτάνει το πρόγραμμα όταν η
`cout << "Καμία λύση\n";`	διακρίνουσα είναι αρνητική.
`} else`	Εδώ έρχεται το πρόγραμμα όταν ο συντελεστής a είναι μηδενικός (δηλαδή πρόκειται για εξίσωση πρώτου βαθμού).
`if ( b != 0 ) {`	Ελέγχεται αν το b είναι μη μηδενικό.
`x1 = -c/b;`	Υπολογίζεται η μοναδική λύση.
`cout << "Μία λύση:\n";`	Εμφανίζεται η λύση στην οθόνη.
`cout << x1 << endl;`	

```
    } else                                  Εδώ έρχεται το πρόγραμμα όταν b = 0.
        cout << "Καμία λύση\n";
}
```

4.1.3 Σύνθετες λογικές παραστάσεις

Δύο ή περισσότερες συγκρίσεις μπορούν να συνδυαστούν έτσι ώστε να κατασκευαστεί μια πιο σύνθετη λογική παράσταση με αποτέλεσμα που μπορεί να είναι αληθές ή ψευδές. Για το σκοπό αυτό χρησιμοποιούνται οι *λογικοί τελεστές* `&&` (και–and), `||` (ή–or) και `!` (όχι–not). Οι δύο πρώτοι χρησιμοποιούνται με δύο λογικές παραστάσεις ως:

Λ1 `&&` *Λ2*

Λ1 `||` *Λ2*

ενώ ο τελεστής `!` με μία λογική παράσταση ως:

`!` *Λ*

Παραδείγματα:

```
x>0 && y>0
a!=0 || b!=0
! k==4
a==0 && b==0 || x==1 && y==2
```

Το αποτέλεσμα των τελεστών `&&` και `||` καθορίζεται από τον παρακάτω πίνακα (όπου *Λ1* και *Λ2* είναι λογικές παραστάσεις).

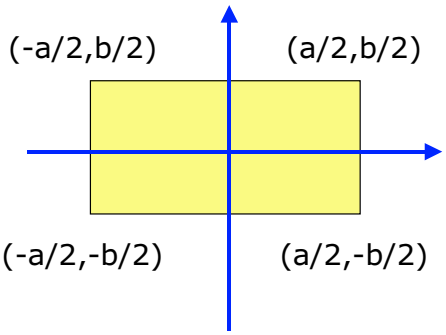
Λ1	*Λ2*	*Λ1* `&&` *Λ2*	*Λ1* `||` *Λ2*
Αληθής	Αληθής	Αληθής	Αληθής
Αληθής	Ψευδής	Ψευδής	Αληθής
Ψευδής	Αληθής	Ψευδής	Αληθής
Ψευδής	Ψευδής	Ψευδής	Ψευδής

Το αποτέλεσμα του τελεστή `!` καθορίζεται από τον παρακάτω πίνακα (όπου *Λ* είναι μια λογική παράσταση).

Λ	`!` *Λ*
Αληθής	Ψευδής
Ψευδής	Αληθής

Σε μια σύνθετη λογική έκφραση η σειρά των πράξεων καθορίζεται από την προτεραιότητα και την προσεταιριστικότητα του κάθε τελεστή:

Λογικός τελεστής	*Προτεραιότητα*	*Προσεταιριστικότητα*
`!`	Υψηλή	Από δεξιά προς αριστερά
`&&`		Από αριστερά προς δεξιά
`||`	Χαμηλή	Από αριστερά προς δεξιά



Σχήμα 4.1: Το παραλληλόγραμμο και οι γωνίες του.

4.1.4 Παραδείγματα

4.1.4.1 Σημεία εντός παραλληλογράμμου

Ένα παραλληλόγραμμο με κέντρο την αρχή των αξόνων έχει πλευρές a και b όπως φαίνεται στο σχ. 4.1. Βρείτε αν ένα σημείο με συντεταγμένες x, y βρίσκεται εντός του παραλληλογράμμου

Στο σχ. 4.1 φαίνονται και οι συντεταγμένες των τεσσάρων γωνιών του παραλληλογράμμου. Παρατηρείστε ότι για να είναι η συντεταγμένη x του σημείου εντός του παραλληλογράμμου θα πρέπει να βρίσκεται εντός του διαστήματος $[-a/2, a/2]$. Ομοίως η συντεταγμένη y θα πρέπει να βρίσκεται εντός του διαστήματος $[-b/2, b/2]$. Τα σημεία που ανήκουν στην περιφέρεια του παραλληλογράμμου θεωρήστε ότι βρίσκονται "εντός".

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double a, b;`	Τα μήκη των πλευρών.
`double x, y;`	Οι συντεταγμένες του σημείου.
`double a2, b2;`	Το μισό της κάθε πλευράς.
`cout << "Δώστε τα μήκη a,b ";`	Εμφάνιση του προτρεπτικού μηνύματος.
`cin >> a >> b;`	Εισάγονται τα μήκη των πλευρών από το πληκτρολόγιο.
`cout << "Δώστε τις συντεταγμένες x,y ";`	Εμφάνιση του προτρεπτικού μηνύματος.
`cin >> x >> y;`	Εισάγονται οι συντεταγμένες του σημείου από το πληκτρολόγιο.
`a2 = a/2;`	Υπολογίζεται το μισό της πλευράς a.
`b2 = b/2;`	Υπολογίζεται το μισό της πλευράς b.
`if (x>=-a2 && x<=a2 && y>=-b2 && y<=b2)`	Ελέγχεται αν το σημείο είναι εντός του
`cout << "Το σημείο είναι εντός\n";`	παραλληλογράμμου.
`else`	Εδώ φτάνει το πρόγραμμα εάν το σημείο

```
        cout << "Το σημείο είναι εκτός\n";     είναι εκτός του παραλληλογράμμου.
}
```

4.1.4.2 Κατηγοριοποίηση χαρακτήρων

Κατασκευάστε πρόγραμμα που θα δέχεται ως είσοδο ένα χαρακτήρα και θα βρίσκει αν είναι κεφαλαίος, μικρός, αριθμητικό ψηφίο ή κάποιο άλλο σύμβολο.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`    char c;`	Ο χαρακτήρας που θα ελέγξουμε.
`    cout << "Εισάγετε ένα χαρακτήρα ";`	Εισαγωγή του χαρακτήρα από το
`    cin >> c;`	πληκτρολόγιο.
`    if (c>='A' && c<='Z')`	Αν είναι κεφαλαίος.
`        cout << "Κεφαλαίος\n";`	
`    else`	Δεν είναι κεφαλαίος.
`        if (c>='a' && c<='z')`	Αν είναι μικρός.
`            cout << "Μικρός\n";`	
`        else`	Δεν είναι μικρός.
`            if (c>='0' && c<='9')`	Αν είναι ψηφίο.
`                cout << "Ψηφίο\n";`	
`            else`	Δεν είναι ψηφίο.
`                cout << "Αλλο σύμβολο\n";`	Είναι άλλο σύμβολο.
`}`	

4.1.4.3 Δίσεκτο έτος

Ένα έτος θεωρείται δίσεκτο όταν διαιρείται ακριβώς με το 4. Όμως από αυτά, τα έτη που διαιρούνται ακριβώς με το 100 δεν είναι δίσεκτα, εκτός αν διαιρούνται με το 400. Το παρακάτω πρόγραμμα δέχεται ως είσοδο το έτος και αποφασίζει αν είναι δίσεκτο ή όχι.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int year;`	Το έτος.
`cout << "Εισάγετε το έτος\n";`	Εισαγωγή του έτους από το πληκτρολόγιο.
`cin >> year;`	
`if ( year%4 == 0 )`	Αν διαιρείται με το 4.
`if ( year%100 == 0 )`	Αν διαιρείται με το 100.
`if ( year%400 == 0 )`	Αν διαιρείται με το 400.
`cout << "Είναι δίσεκτο\n";`	
`else`	Δεν διαιρείται με το 400.
`cout << "Δεν είναι δίσεκτο\n";`	
`else`	Δεν διαιρείται με το 100.
`cout << "Είναι δίσεκτο\n";`	
`else`	Δεν διαιρείται με το 4.
`cout << "Δεν είναι δίσεκτο\n";`	
`}`	

Εναλλακτικά, επειδή όταν το έτος διαιρείται με το 400 σημαίνει ότι διαιρείται και με το 4 και με το 100, οι ανωτέρω εντολές `if` θα μπορούσαν να αντικατασταθούν από:

```
if ( year%4==0 && year%100!=0 || year%400==0 )
    cout << "Είναι δίσεκτο\n";
else
    cout << "Δεν είναι δίσεκτο\n";
```

4.2 Εντολή switch

4.2.1 Πως συντάσσεται

Η εντολή `switch` ελέγχει αν η τιμή μιας ακέραιας μεταβλητής ισούται με κάποια σταθερά (από ένα πλήθος σταθερών) και ανάλογα εκτελεί συγκεκριμένες εντολές. Η εντολή συντάσσεται ως εξής:

```
switch ( ακέραια παράσταση ) {
    case σταθερά1:
        εντολή ;
        εντολή ;
            ⋮
        break ;
    case σταθερά2:
        εντολή ;
        εντολή ;
            ⋮
        break ;
    ⋮
    default :
        εντολή ;
        εντολή ;
            ⋮
}
```

Η εντολή `switch` αποτελείται από μια σειρά από *εντολές* και *σημεία εισόδου* (όπου υπάρχει `case` ή `default`). Η εκτέλεση της εντολής `switch` ξεκινάει από ένα σημείο εισόδου, που για να προσδιοριστεί ελέγχεται η *ακέραια παράσταση* με τις αντίστοιχες *σταθερές* στις `case`. Κατόπιν εκτελούνται όλες οι εντολές από το σημείο εισόδου μέχρι το τελικό άγκιστρο, ή μέχρι να βρεθεί `break`. Προσοχή: αν δεν υπάρχει `break` θα εκτελεστούν όλες οι εντολές μέχρι το τέλος της `switch`. Εάν δεν ταιριάζει καμία σταθερά στις `case`, τότε το σημείο εισόδου είναι το `default`. Σημειώστε ότι το τμήμα `default` μπορεί να μην γραφεί. Σε αυτή την περίπτωση εάν δεν ταιριάζει καμία σταθερά με την *ακέραια παράσταση*, δεν εκτελείται καμία εντολή μέσα από την `switch`.

4.2.2 Παραδείγματα

4.2.2.1 Όγκοι αντικειμένων

Το παρακάτω πρόγραμμα υπολογίζει τον όγκο ενός από τρία δυνατά αντικείμενα.

1. Ορθογώνιο παραλληλεπίπεδο με πλευρές a, b, c. $V = abc$
2. Σφαίρα με ακτίνα r. $V = \frac{4}{3}\pi r^3$
3. Κύλινδρος με ακτίνα βάσης r και ύψος h. $V = \pi r^2 h$

Η επιλογή γίνεται με βάση ένα ακέραιο αριθμό που εισάγει ο χρήστης. Μετά την επιλογή του σχήματος ζητούνται οι απαραίτητες διαστάσεις για τον υπολογισμό του όγκου (διαφορετικές για κάθε αντικείμενο).

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int n;`	
`double a, b, c;`	Οι πλευρές του παραλληλεπιπέδου.
`double pi;`	Ο αριθμός π.
`double r, h;`	Ακτίνα και ύψος.
`double v;`	Ο όγκος.
`cout << "Επιλέξτε αντικείμενο: \n";`	Εμφανίζεται προτρεπτικό μήνυμα.
`cout << "1. Ορθογ. παραλληλεπίπεδο\n";`	
`cout << "2. Σφαίρα\n";`	
`cout << "3. Κύλινδρος\n";`	
`cin >> n;`	Εισάγεται ο αριθμός `n` από το πληκτρολόγιο.
`switch (n) {`	
`case 1:`	Εάν η μεταβλητή `n` είναι 1.
`cout << "Εισάγετε τις πλευρές\n";`	Εισαγωγή από το πληκτρολόγιο των πλευρών του παραλληλεπιπέδου.
`cin >> a >> b >> c;`	
`v = a*b*c;`	Υπολογισμός του όγκου του παραλληλεπιπέδου.
`cout << "Ο όγκος είναι "`	
`<< v << endl;`	
`break;`	Έξοδος από την εντολή `switch`.
`case 2:`	Εάν η μεταβλητή `n` είναι 2.
`cout << "Εισάγετε την ακτίνα\n";`	Εισαγωγή από το πληκτρολόγιο της ακτίνας της σφαίρας.
`cin >> r;`	
`pi = acos(-1.0);`	Ο αριθμός π.
`v = 4*pi*r*r*r/3;`	Υπολογισμός του όγκου της σφαίρας.
`cout << "Ο όγκος είναι "`	
`<< v << endl;`	
`break;`	Έξοδος από την εντολή `switch`.
`case 3:`	Εάν η μεταβλητή `n` είναι 3.
`cout <<"Εισάγετε ακτίνα, ύψος\n";`	Εισαγωγή της ακτίνας βάσης και του ύψους του κυλίνδρου.
`cin >> r >> h;`	
`pi = acos(-1.0);`	Ο αριθμός π.
`v = pi*r*r*h;`	Υπολογισμός του όγκου του κυλίνδρου.

```
         cout << "Ο όγκος είναι "
              << v << endl;
         break;                                Έξοδος από την εντολή switch.
      default:                                 Εάν η μεταβλητή n δεν ικανοποιεί καμία
         cout << "Λανθασμένη επιλογή\n";       από τις ανωτέρω περιπτώσεις.
   }
}
```

Κεφάλαιο 5

Εντολές επανάληψης

Στη C++ υπάρχουν 3 διαφορετικές εντολές επανάληψης:

- `while`
- `for`
- `do while`

5.1 Εντολή while

Με τη εντολή `while` έχουμε τη δυνατότητα να επαναλάβουμε την εκτέλεση ενός τμήματος προγράμματος όσες φορές θέλουμε. Η εντολή συντάσσεται ως εξής:

```
while ( συνθήκη )
   εντολή ;
```

όπου *συνθήκη* είναι οποιαδήποτε απλή ή σύνθετη λογική παράσταση και *εντολή* είναι μια οποιαδήποτε εντολή της γλώσσας.

Η εντολή λειτουργεί ως εξής: Μόλις το πρόγραμμα φτάσει στην εντολή `while` ελέγχεται η *συνθήκη*. Εάν είναι αληθής τότε η *εντολή* εκτελείται, το πρόγραμμα επιστρέφει στην εντολή `while` και η διαδικασία επαναλαμβάνεται και πάλι. Εάν η συνθήκη βρεθεί ψευδής τότε οι επαναλήψεις σταματούν. Σημειώστε ότι επειδή ο έλεγχος της συνθήκης γίνεται στην αρχή της `while` μπορεί να υπάρξει περίπτωση να μην εκτελεστεί καμία επανάληψη. Η εντολή `while` μαζί με όλες τις εντολές που περιλαμβάνει θεωρείται συντακτικά ως μια εντολή. Σε περίπτωση που απαιτείται η επανάληψη παραπάνω από μιας εντολής, τότε χρησιμοποιούμε άγκιστρα:

```
while ( συνθήκη ) {
   εντολή ;
   εντολή ;
      ⋮
}
```

Παράδειγμα:

```
n = 1;
while ( n <= 10 ) {
    cout << n << endl;
    ++n;
}
```

Το παράδειγμα εμφανίζει στην οθόνη τους ακεραίους από $1 \ldots 10$. Αρχικά η μεταβλητή `n` λαμβάνει την τιμή 1. Η συνθήκη `n <= 10` στην εντολή `while` εξασφαλίζει ότι θα γίνονται επαναλήψεις όσο η `n` είναι μικρότερη ή ίση με 10. Σε κάθε επανάληψη εμφανίζεται στην οθόνη η τιμή της `n` και στη συνέχεια αυξάνεται κατά 1. Μόλις ολοκληρωθεί η δέκατη επανάληψη η τιμή της `n` θα είναι 11, οπότε η συνθήκη θα βρεθεί ψευδής και η λειτουργία της `while` θα σταματήσει.

Στο ανωτέρω παράδειγμα που είναι τυπικό για την εντολή `while` παρατηρείστε ότι υπάρχουν τρία σημαντικά σημεία:

1. Η *αρχική τιμή* `n = 1;`
2. Η *συνθήκη* `n <= 10`
3. Η *αύξηση* της μεταβλητής `++n;`

5.2 Εντολή for

Στις περισσότερες περιπτώσεις επαναλήψεων εκτός από τη συνθήκη που καθορίζει αν θα εκτελεστεί η επανάληψη ή όχι, χρησιμοποιούμε κάποια μεταβλητή (συνήθως ακέραια) της οποίας η τιμή αυξάνεται (ή μειώνεται) σε κάθε επανάληψη. Επιπλέον πριν ξεκινήσουν οι επαναλήψεις η μεταβλητή αυτή λαμβάνει κάποια αρχική τιμή. Η εντολή `for` περιλαμβάνει όλα τα ανωτέρω σε μια συμπαγή σύνταξη ως εξής:

```
for ( αρχική_τιμή ; συνθήκη ; αύξηση )
    εντολή ;
```

Η εντολή `for` είναι ισοδύναμη με το παρακάτω τμήμα προγράμματος που χρησιμοποιεί την εντολή `while`:

```
αρχική_τιμή ;
while ( συνθήκη ) {
    εντολή ;
    αύξηση ;
}
```

Η εντολή λειτουργεί ως εξής: Αρχικά ανατίθεται η *αρχική_τιμή*. Εάν η *συνθήκη* είναι αληθής εκτελείται η *εντολή*, εκτελείται η *αύξηση* και το πρόγραμμα επιστρέφει στην `for` για να ελέγξει και πάλι τη *συνθήκη*. Εάν η *συνθήκη* είναι ψευδής τερματίζεται η λειτουργία της εντολής `for`.

Σε περίπτωση που απαιτείται η επανάληψη παραπάνω από μιας εντολής, τότε χρησιμοποιούμε άγκιστρα:

```
for ( αρχική_τιμή ; συνθήκη ; αύξηση ) {
    εντολή ;
    εντολή ;
      ⋮
    εντολή ;
}
```

Παραδείγματα:

`for ( k=1; k<=7; k=k+2 )`	Θα γίνουν 4 επαναλήψεις με τη μεταβλητή `k` να λαμβάνει τιμές 1, 3, 5, 7.
`for ( k=10; k<17; k+=3 )`	Θα γίνουν 3 επαναλήψεις με τη μεταβλητή `k` να λαμβάνει τιμές 10, 13, 16
`for ( m=1; m<=0; m+=25 )`	Δεν θα γίνει καμία επανάληψη.
`for ( n=-8; n>=-10; n=n+2 )`	Θα γίνουν άπειρες επαναλήψεις με τη μεταβλητή `n` να λαμβάνει τιμές -8, -6, -4, ...

Ορισμένες παρατηρήσεις για την εντολή `for`:

- Ως *αρχική_τιμή* μπορούμε να βάλουμε οποιαδήποτε εντολή ανάθεσης τιμής.
- Ως *συνθήκη* μπορούμε να χρησιμοποιήσουμε οποιαδήποτε απλή ή σύνθετη λογική παράσταση.
- Ως αύξηση βάζουμε οποιαδήποτε εντολή που τροποποιεί (αυξάνει ή μειώνει) την τιμή της μεταβλητής.
- Όπως και η `while`, η εντολή `for` ενδέχεται να μην εκτελέσει καμία επανάληψη.
- Συντακτικά η `for` (μαζί με όλες τις εντολές που περιλαμβάνει) θεωρείται ως μία εντολή.
- Οποιοδήποτε από τα *αρχική_τιμή*, *συνθήκη*, *αύξηση* μπορεί να παραληφθεί.

  Παραδείγματα:

  ```
  for ( ; k<10 ; ++k )
  for ( k=1 ; ; )
  for ( ; ; )
  ```

  Εάν παραληφθεί η *συνθήκη* τότε η εντολή `for` εκτελεί επαναλήψεις σαν να υπήρχε μια συνθήκη που είναι διαρκώς αληθής.

5.3 Εντολή do–while

Στους δύο προηγούμενες εντολές (`while` και `for`) ο έλεγχος για να αποφασιστεί αν θα γίνει μια επανάληψη πραγματοποιείται στην αρχή της εντολής. Με την εντολή `do while` επαναλαμβάνεται ένα τμήμα προγράμματος, αλλά ο έλεγχος γίνεται στο τέλος της εντολής. Αυτό έχει σαν αποτέλεσμα να εκτελείται τουλάχιστον μία επανάληψη, σε αντίθεση με τις δύο άλλες εντολές που μπορεί να μην εκτελέσουν καμία επανάληψη.

Η εντολή `do while` συντάσσεται ως εξής:

```
do
    εντολή ;
while ( συνθήκη ) ;
```

Σε περίπτωση που απαιτείται η επανάληψη παραπάνω από μιας εντολής, τότε χρησιμοποιούμε άγκιστρα:

```
do {
    εντολή ;
    εντολή ;
      ⋮
    εντολή ;
} while ( συνθήκη ) ;
```

Συντακτικά η `do while` (μαζί με την εντολή που περιλαμβάνει) θεωρείται ως μία εντολή.

Παράδειγμα:
Μια συνηθισμένη εφαρμογή της εντολής `do while` είναι σε περίπτωση εισαγωγής λανθασμένων δεδομένων να ζητείται ξανά η εισαγωγή τους. Το παρακάτω τμήμα προγράμματος ζητά ένα θετικό ακέραιο. Αν ο χρήστης εισάγει λάθος αριθμό (αρνητικό ή μηδέν) εμφανίζει ένα μήνυμα λάθους και ξαναζητά τον αριθμό.

```
do {
    cout << "Εισάγετε ένα θετικό ακέραιο " ;
    cin >> n ;
    if ( n <= 0 )
        cout << "Λάθος. Ο αριθμός πρέπει να είναι θετικός. \n\n";
} while ( n <=0 );
```

Παρατηρείστε ότι στις ανωτέρω εντολές `if` και `do while` γράφουμε την ίδια συνθήκη.

5.4 Εντολή break

Αν για οποιοδήποτε λόγο χρειάζεται να "βγούμε" από μια εντολή επανάληψης, χρησιμοποιούμε την εντολή `break`. Συνήθως η `break` χρησιμοποιείται σε συνδυασμό με κάποια εντολή `if`.

Παράδειγμα:

```
for ( k=1 ; ; ++k ) {
    ⋮
    if ( ··· ) break;
    ⋮
}
```

5.5 Παραδείγματα

5.5.1 Αθροίσματα και γινόμενα με σταθερούς όρους

Για να υπολογίσουμε αθροίσματα χρησιμοποιούμε μια μεταβλητή που ονομάζεται *αθροιστής* και στην οποία σταδιακά συσσωρεύουμε όλους τους όρους του αθροίσματος. Στις μεταβλητές που χρησιμοποιούνται ως αθροιστές απαιτείται να αναθέσουμε μια αρχική τιμή (μηδέν, ή κάποια άλλη ανάλογα με το πρόβλημα). Η συσσώρευση στον αθροιστή γίνεται με εντολές του τύπου:

`s = s +` *αριθμητική παράσταση* `;`

Στην εντολή αυτή πρώτα θα γίνουν οι πράξεις στα δεξιά του συμβόλου =, δηλαδή θα προστεθεί η *αριθμητική παράσταση* στον αθροιστή `s`. Κατόπιν το αποτέλεσμα τίθεται ως νέα τιμή του `s`.

Παράδειγμα:
Γράψτε τμήμα προγράμματος που για δεδομένο N θα υπολογίζει το κάτωθι άθροισμα.

$$1 + 2 + 3 + \cdots + N$$

Στο συγκεκριμένο παράδειγμα το αποτέλεσμα της άθροισης είναι αναλυτικά γνωστό και ίσο με $N(N+1)/2$.

Κώδικας C++	*Σχολιασμός*
`int n, k, s;`	Δήλωση ακέραιων μεταβλητών. `n` είναι το πλήθος των όρων του αθροίσματος, `s` είναι ο αθροιστής και η `k` χρησιμοποιείται στην εντολή `for`.
⋮	
`s = 0;`	Ο αθροιστής λαμβάνει αρχική τιμή.
`for ( k=1; k<=n; ++k )`	Επανάληψη `n` φορές.
`  s += k;`	Στο άθροισμα προστίθεται η τιμή της μεταβλητής `k`.

Παράδειγμα:
Γράψτε τμήμα προγράμματος που για δεδομένο N θα υπολογίζει το κάτωθι άθροισμα.

$$1^2 + 2^2 + 3^2 + \cdots + N^2$$

Το αποτέλεσμα της άθροισης είναι αναλυτικά γνωστό και ίσο με $N(N+1)(2N+1)/6$.

Κώδικας C++	*Σχολιασμός*
`int n, k, s;`	Δήλωση ακέραιων μεταβλητών. `n` είναι το πλήθος των όρων του αθροίσματος, `s` είναι ο αθροιστής και η `k` χρησιμοποιείται στην εντολή `for`.
⋮	
`s = 0;`	Ο αθροιστής λαμβάνει αρχική τιμή.
`for ( k=1; k<=n; ++k )`	Επανάληψη `n` φορές.
`  s += k*k;`	Στο άθροισμα προστίθεται το k^2.

Παράδειγμα:
Γράψτε τμήμα προγράμματος που για δεδομένο N θα υπολογίζει το κάτωθι γινόμενο γνωστό και ως N–παραγοντικό (συμβολίζεται N!).

$$N! = 1 \cdot 2 \cdot 3 \cdots N$$

Κώδικας C++	*Σχολιασμός*
`int n, k, p;`	Δήλωση ακέραιων μεταβλητών. `n` είναι το πλήθος των όρων του γινομένου, `p` είναι το γινόμενο και η `k` χρησιμοποιείται στην εντολή `for`.
⋮	
`p = 1;`	Το γινόμενο λαμβάνει αρχική τιμή.
`for ( k=2; k<=n; ++k )`	Επανάληψη για όλους τους όρους του γινομένου πλην του αρχικού.
`p *= k;`	Το γινόμενο πολλαπλασιάζεται με την τιμή της μεταβλητής `k`.

5.5.2 Σειρές με εναλλασσόμενο πρόσημο

Γράψτε τμήμα προγράμματος που για δεδομένο N θα αθροίζει την κάτωθι σειρά:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots \frac{1}{N}$$

Η σειρά αυτή αποτελεί μια προσέγγιση του $\pi/4$. Η ιδιαιτερότητα που έχει σε σχέση με τα προηγούμενα παραδείγματα είναι ότι τα πρόσημα των όρων εναλλάσσονται. Για το λόγο αυτό ορίζουμε μια μεταβλητή προσήμου (`p`). Η μεταβλητή αυτή λαμβάνει αρχικά την τιμή 1 επειδή το πρόσημο του πρώτου όρου είναι θετικό. Κατόπιν μόλις ο πρώτος όρος προστεθεί στο άθροισμα η μεταβλητή `p` αλλάζει πρόσημο (`p = -p`) και γίνεται -1. Έτσι στη δεύτερη επανάληψη ο όρος 1/3 αφαιρείται από το άθροισμα κοκ. Σημειώστε ότι η σειρά συγκλίνει πολύ αργά και για να πετύχουμε σωστά 4 σημαντικά ψηφία απαιτείται περίπου N=10000.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int k, n;`	Δήλωση ακέραιων μεταβλητών. `n` είναι ο παρονομαστής του τελευταίου όρου του αθροίσματος και η `k` χρησιμοποιείται στην εντολή `for`.
`double s, p;`	Δήλωση μεταβλητών διπλής ακρίβειας. `p` είναι το πρόσημο και `s` ο αθροιστής.
`cout << "Εισάγετε το n ";`	
`cin >> n;`	Εισαγωγή του `n` από το πληκτρολόγιο.
`s = 0.0;`	Αρχική τιμή του αθροιστή.
`p = 1.0;`	Αρχική τιμή του προσήμου.
`for ( k=1; k<=n ; k+=2 ) {`	Επανάληψη για όλους τους όρους. Προσέξτε ότι το βήμα είναι 2.
`s += p/k;`	Ο κάθε όρος προστίθεται στο άθροισμα πολλαπλασιασμένος με το πρόσημο.
`p = -p;`	Αλλαγή προσήμου για τον επόμενο όρο του αθροίσματος.
`}`	
`cout << s << endl;`	Εμφάνιση του αποτελέσματος στην οθόνη.
`}`	

Σημειώστε ότι στο παράδειγμα αυτό η μεταβλητή προσήμου `p` δηλώθηκε ως `double` και όχι ακέραια. Αυτό γίνεται έτσι ώστε όταν γίνει η διαίρεση `p/k` να προκύψει πραγματικό και όχι ακέραιο αποτέλεσμα.

5.5.3 Αναδρομικές σχέσεις

5.5.3.1 Με ένα προηγούμενο όρο

Θα γράψουμε πρόγραμμα που θα υπολογίζει τον N–στό όρο της ακολουθίας:

$$x_k = \frac{3x_{k-1} + 4}{2x_{k-1} + 3}$$

με πρώτο όρο $x_1 = 1$. Η ακολουθία αυτή είναι γνωστό ότι συγκλίνει γρήγορα στην τιμή $\sqrt{2}$.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int k, n;`	`n` είναι το πλήθος των όρων που θέλουμε ενώ η `k` χρησιμοποιείται στην εντολή `for`.
`double x;`	Στη μεταβλητή `x` αποθηκεύεται ο κάθε όρος της ακολουθίας.
`cout << "Πόσους όρους? ";`	Εισαγωγή του πλήθους των όρων από το πληκτρολόγιο.
`cin >> n;`	
`x = 1.0;`	Ο πρώτος όρος της ακολουθίας.
`for ( k=2; k<=n; ++k )`	Επανάληψη για τους υπόλοιπους όρους (από τον δεύτερο και μετά).
`x = (3*x+4)/(2*x+3);`	Υπολογισμός του επόμενου όρου. Το αποτέλεσμα αποθηκεύεται και πάλι στη μεταβλητή `x`.
`cout << x << endl;`	Εμφάνιση του αποτελέσματος στην οθόνη.
`}`	

5.5.3.2 Με δύο προηγούμενους όρους

Το παρακάτω πρόγραμμα εμφανίζει στην οθόνη την ακολουθία αριθμών Fibonacci που δίνονται από τη σχέση:

$$F_n = F_{n-1} + F_{n-2}$$

με αρχικές τιμές $F_0 = 0$ και $F_1 = 1$. Παρατηρείστε ότι κάθε όρος της ακολουθίας εξαρτάται από τους δύο αμέσως προηγούμενους όρους.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`int fa;`	Ο προ–προηγούμενος όρος της ακολουθίας.
`int fb;`	Ο προηγούμενος όρος της ακολουθίας.
`int fc;`	Ο τρέχων όρος της ακολουθίας.
`int n, i;`	`n` είναι ο δείκτης του όρου που θέλουμε και η `i` χρησιμοποιείται στην εντολή `for`.
`cout << "Τελευταίος όρος ";`	
`cin >> n;`	Εισαγωγή του `n` από το πληκτρολόγιο.
`fa = 0;`	Η τιμή του πρώτου όρου.
`fb = 1;`	Η τιμή του δεύτερου όρου.
`for ( i=2; i<=n; ++i ) {`	Επανάληψη για όλους τους όρους της ακολουθίας μέχρι το N–στό.
`fc = fa+fb;`	Υπολογισμός του όρου υπ' αριθμόν `i` (τρέχων όρος)
`fa = fb;`	Ο προ–προηγούμενος παίρνει την τιμή του προηγούμενου.
`fb = fc;`	Ο προηγούμενος παίρνει την τιμή του τρέχοντα όρου.
`}`	
`cout << fc << endl;`	Εμφάνιση στην οθόνη του N–στου όρου.
`}`	

5.5.4 Σειρές Taylor

5.5.4.1 Εκθετικό

Θα γράψουμε πρόγραμμα το οποίο θα αθροίζει τη σειρά:

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^N}{N!}$$

Το ανωτέρω άθροισμα αποτελεί μια προσέγγιση του e^x.

Παρατηρήστε ότι σε κάθε όρο του αθροίσματος $x^k/k!$ υπάρχει στον παρονομαστή το $k!$. Συνεπώς χρειαζόμαστε δύο εντολές `for`. Η μια θα διατρέχει όλους τους όρους του αθροίσματος και η δεύτερη που βρίσκεται εντός της πρώτης θα υπολογίζει το αντίστοιχο παραγοντικό για κάθε όρο.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double x, s, p;`	`s` είναι ο αθροιστής και η `p` χρησιμοποιείται στον υπολογισμό του παραγοντικού.
`int n, i, k;`	
`cout << "Εισάγετε τα x, n \n";`	Εισαγωγή των `x` και `n` από το πληκτρολόγιο.
`cin >> x >> n;`	
`s = 1.0;`	Αρχική τιμή του αθροιστή.
`for ( k=1; k<=n; ++k ) {`	Επανάληψη για όλους τους όρους του αθροίσματος.
`p = 1.0;`	Αρχική τιμή για το παραγοντικό.
`for ( i=2; i<=k; ++i )`	Επανάληψη για τον υπολογισμό του n!
`p *= i;`	
`s += pow(x,k)/p;`	Προστίθεται στο άθροισμα ο όρος $x^k/k!$
`}`	
`cout << s << endl;`	Εμφάνιση του αποτελέσματος στην οθόνη.
`}`	

Ας δούμε πόσες αριθμητικές πράξεις θα κάνει το παραπάνω πρόγραμμα για μια δεδομένη τιμή του N. Παρατηρήστε ότι το εξωτερικό `for` σε κάθε επανάληψη υπολογίζει το k! και επιπλέον κάνει 3 πράξεις στην εντολή `s += pow(x,k)/p` (θεωρήστε την ύψωση σε δύναμη ως μία πράξη). Το εσωτερικό `for` (που υπολογίζει το παραγοντικό) για μια δεδομένη τιμή του k κάνει k-1 πολλαπλασιασμούς. Ο παρακάτω πίνακας συνοψίζει

k	*Πράξεις στο εσωτερικό* `for`	*Πράξεις στο εξωτερικό* `for`
1	0	3
2	1	3
3	2	3
⋮	⋮	3
N	N-1	3

Το άθροισμα της δεύτερης στήλης $(0+1+2+\cdots+N-1)$ είναι $N(N-1)/2$, ενώ το άθροισμα της τρίτης στήλης είναι 3N. Συνολικά λοιπόν χρειάζονται $N(N-1)/2+3N = (N^2+5N)/2$ αριθμητικές πράξεις εκ των οποίων N είναι υψώσεις σε δύναμη.

Ένας άλλος τρόπος για να αθροίσουμε τη σειρά βασίζεται στην παρατήρηση ότι κάθε όρος της σειράς μπορεί να γραφεί σαν συνάρτηση του προηγούμενου, δηλαδή:

$$\frac{x^k}{k!} = \frac{x^{k-1}}{(k-1)!}\,\frac{x}{k}$$

Για παράδειγμα ο τέταρτος όρος μπορεί να γραφεί σαν συνάρτηση του τρίτου ως:

$$\frac{x^4}{4!} = \frac{x^3}{3!}\,\frac{x}{4}$$

Έτσι μπορούμε να κατασκευάσουμε τον κάθε όρο $x^k/k!$ από τον προηγούμενό του, χωρίς να χρειάζεται να υπολογίζουμε το $k!$ κάθε φορά. Πρόκειται δηλαδή για μια ακολουθία με ένα προηγούμενο όρο, μόνο που πρέπει να υπολογιστεί το άθροισμα όλων των όρων και όχι η επιμέρους τιμή κάποιου όρου όπως στην παράγραφο 5.5.3.1.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double x, s, t;`	`s` είναι ο αθροιστής και `t` ο κάθε όρος του αθροίσματος.
`int n, k;`	`n` είναι το πλήθος των όρων και η `k` χρησιμοποιείται στην εντολή `for`.
`cout << "Εισάγετε τα x, n \n";`	
`cin >> x >> n;`	Εισαγωγή των `x` και `n` από το πληκτρολόγιο.
`s = 1.0;`	Αρχική τιμή του αθροιστή.
`t = 1.0;`	Η μεταβλητή `t` λαμβάνει την τιμή του πρώτου όρου του αθροίσματος.
`for ( k=1; k<=n; ++k ) {`	Επανάληψη για όλους τους όρους του αθροίσματος.

```
        t *= x/k;                          Η μεταβλητή t παίρνει την τιμή του επόμενου
                                           όρου.
        s += t;                            Προσθήκη στο άθροισμα.
    }
    cout << s << endl;                     Εμφάνιση του αποτελέσματος στην οθόνη.
}
```

Παρατηρήστε ότι απαιτείται μόνο μια εντολή **for**, εντός της οποίας γίνονται 3 πράξεις σε κάθε επανάληψη (καμία εκ των οποίων δεν είναι ύψωση σε δύναμη), οπότε το συνολικό πλήθος πράξεων είναι 3N.

Πόσοι όροι όμως χρειάζονται για προσεγγίσουμε το e^x με 6 δεκαδικά ψηφία; Αντί να προκαθορίσουμε το N θα μπορούσαμε να συγκρίνουμε εντός της **for** το άθροισμα με την ακριβή τιμή **exp(x)** και να σταματήσουμε όταν η διαφορά γίνει μικρότερη από 10^{-6}. Εναλλακτικά, για να μην χρησιμοποιήσουμε τη συνάρτηση **exp(x)**, μπορούμε να σταματήσουμε όταν ο προστιθέμενος όρος **t** είναι τόσο μικρός που δεν μπορεί να επηρεάσει τα προηγούμενα ψηφία. Έτσι αντί να προσδιορίζουμε από την αρχή το πλήθος των όρων της σειράς σταματάμε την άθροιση όταν ο προστιθέμενος όρος $x^n/n!$ είναι μικρότερος κατ' απόλυτη τιμή από μια μικρή προκαθορισμένη τιμή e.

Κώδικας C++ | *Σχολιασμός*

```
#include <iostream>
using namespace std;
int main ( )
{
    double x, e, s, t;                 s είναι ο αθροιστής και t ο κάθε όρος του αθροί-
                                       σματος.
    int k;                             Το πλήθος των όρων που θα αθροιστούν.

    cout << "Εισάγετε τα x, e \n";
    cin >> x >> e;                     Εισαγωγή των x και e από το πληκτρολόγιο.

    s = 1.0;                           Αρχική τιμή του αθροίσματος.
    t = 1.0;                           Η μεταβλητή t λαμβάνει την τιμή του πρώτου
                                       όρου του αθροίσματος.
    k = 1;                             Αρχική τιμή για το πλήθος των όρων.
    while ( abs(t) > e ) {             Σύγκριση του τρέχοντος όρου με τη σταθερά e.
        t *= x/k;                      Η μεταβλητή t παίρνει την τιμή του επόμενου
                                       όρου.
        s += t;                        Προσθήκη στο άθροισμα.
        ++k;
    }
    cout << "Αθροισμα: " << s << endl; Εμφάνιση του αποτελέσματος στην οθόνη.
    cout << "Πλήθος όρων: "<<k<< endl;
}
```

5.5.4.2 Ημίτονο

Θα γράψουμε πρόγραμμα το οποίο θα αθροίζει τη σειρά Taylor του ημιτόνου:

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \frac{x^N}{N!}$$

Θα εφαρμόσουμε την τεχνική του προηγούμενου παραδείγματος. Παρατηρούμε ότι ο όρος $x^k/k!$ μπορεί να γραφεί σαν συνάρτηση του προηγούμενου όρου:

$$\frac{x^k}{k!} = -\frac{x^2}{(k-1)k}\ \frac{x^{k-2}}{(k-2)!}$$

Για παράδειγμα ο όρος $x^5/5!$ μπορεί να γραφεί με βάση τον προηγούμενο ως:

$$\frac{x^5}{5!} = -\frac{x^2}{4 \cdot 5}\ \frac{x^3}{3!}$$

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double x, s, t;`	`s` είναι ο αθροιστής και `t` ο κάθε όρος του αθροίσματος.
`int n, k;`	`n` είναι το πλήθος των όρων και η `k` χρησιμοποιείται στην εντολή `for`.
`cout << "Εισάγετε τα x, n \n";`	
`cin >> x >> n;`	Εισαγωγή των `x` και `n` από το πληκτρολόγιο.
`s = x;`	Αρχική τιμή του αθροιστή.
`t = x;`	Η μεταβλητή `t` λαμβάνει την τιμή του πρώτου όρου του αθροίσματος.
`for ( k=3; k<=n; k+=2 ) {`	Επανάληψη για όλους τους όρους του αθροίσματος.
`t *= -x*x/(k*(k-1));`	Η μεταβλητή `t` παίρνει την τιμή του επόμενου όρου.
`s += t;`	Προσθήκη στο άθροισμα.
`}`	
`cout << s << endl;`	Εμφάνιση του αποτελέσματος στην οθόνη.
`}`	

5.5.4.3 Διωνυμικές σειρές

Θα υπολογίσουμε τις παρακάτω σειρές, αθροίζοντας μέχρι τον όρο τάξης N όταν μας δίνεται ο πραγματικός αριθμός x και ο ακέραιος N.

$$\frac{1}{\sqrt{1+x}} = 1 - \frac{1}{2}x + \frac{1\cdot 3}{2\cdot 4}x^2 - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}x^3 + \cdots \qquad -1 < x \leq 1$$

$$\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{2\cdot 4}x^2 + \frac{1\cdot 3}{2\cdot 4\cdot 6}x^3 - \cdots \qquad -1 < x \leq 1$$

$$\frac{1}{\sqrt[3]{1+x}} = 1 - \frac{1}{3}x + \frac{1\cdot 4}{3\cdot 6}x^2 - \frac{1\cdot 4\cdot 7}{3\cdot 6\cdot 9}x^3 + \cdots \qquad -1 < x \leq 1$$

$$\sqrt[3]{1+x} = 1 + \frac{1}{3}x - \frac{2}{3\cdot 6}x^2 + \frac{2\cdot 5}{3\cdot 6\cdot 9}x^3 - \cdots \qquad -1 < x \leq 1$$

Αν συμβολίσουμε με t_k τον όρο που περιέχει το x^k παρατηρούμε ότι αυτός μπορεί να γραφεί σαν συνάρτηση του προηγούμενου όρου t_{k-1}. Έτσι για τις τέσσερις σειρές:

$$t_k = -\frac{2k-1}{2k}x\ t_{k-1}$$

$$t_k = -\frac{2k-3}{2k}x\ t_{k-1}$$

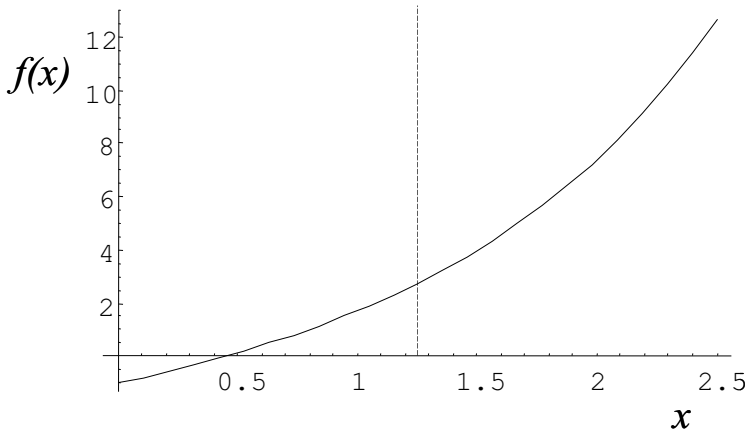
$$t_k = -\frac{3k-2}{3k}x\ t_{k-1}$$

$$t_k = -\frac{3k-4}{3k}x\ t_{k-1}$$

Κώδικας C++	*Σχολιασμός*

```
#include <iostream>
using namespace std;
int main ( )
{
    int n, k;
    double x, s, t;

    cout << "Εισάγετε τα x, n ";          Εισαγωγή των x και n από το πληκτρολόγιο.
    cin >> x >> n;

    s = 1.0;                              Άθροιση τη σειράς 1/√(1+x)
    t = 1.0;
    for (k=1; k<=n; ++k) {
        t = -t*x*(2*k-1)/(2*k);
        s += t;
    }
```

```
    cout << s << endl;

    s = 1+x/2;                          Άθροιση τη σειράς √(1+x)
    t = x/2;
    for (k=2; k<=n; ++k) {
        t = -t*x*(2*k-3)/(2*k);
        s += t;
    }
    cout << s << endl;

    s = 1.0;                            Άθροιση τη σειράς 1/∛(1+x)
    t = 1.0;
    for (k=1; k<=n; ++k) {
        t = -t*x*(3*k-2)/(3*k);
        s += t;
    }
    cout << s << endl;

    s = 1+x/3;                          Άθροιση τη σειράς ∛(1+x)
    t = x/3;
    for (k=2; k<=n; ++k) {
        t = -t*x*(3*k-4)/(3*k);
        s += t;
    }
    cout << s << endl;
}
```



Σχήμα 5.1: Η συνάρτηση $f(x) = x + e^x - 2$ στο διάστημα $[0, 2.5]$. Με τη διακεκομμένη γραμμή φαίνεται το μέσον του διαστήματος.

5.5.5 Εύρεση ριζών με τη μέθοδο της διχοτόμησης

Θα βρούμε τις λύσης της εξίσωσης $f(x) = 0$ με

$$f(x) = x + e^x - 2$$

χρησιμοποιώντας τη *μέθοδο της διχοτόμησης*. Η ανωτέρω συνάρτηση φαίνεται στο σχήμα 5.1. Αρχικά δίνεται ένα διάστημα $[a, b]$ εντός του οποίου βρίσκεται η λύση. Παρατηρείστε ότι για να υπάρχει λύση εντός του $[a, b]$ πρέπει οι τιμές της $f(x)$ στα άκρα να είναι ετερόσημες, δηλαδή $f(a)f(b) \leq 0$. Στη συνέχεια υπολογίζουμε το μέσον του διαστήματος $x_m = (a + b)/2$. Από τα δύο διαστήματα που προκύπτουν $[a, x_m], [x_m, b]$ κρατάμε εκείνο στο οποίο υπάρχει λύση, δηλαδή εκείνο στο οποίο οι τιμές της συνάρτησης στα άκρα του είναι ετερόσημες. Για παράδειγμα στο σχήμα 5.1, το αρχικό διάστημα είναι το $[0, 2.5]$, οπότε το μέσον του είναι το 1.25. Μετά τη διχοτόμηση προκύπτουν τα διαστήματα $[0, 1.25]$ και $[1.25, 2.5]$. Μόνο στο πρώτο όμως οι τιμές της συνάρτησης είναι ετερόσημες στα άκρα του, οπότε κρατάμε αυτό το διάστημα. Η διαδικασία επαναλαμβάνεται και πάλι στο νέο διάστημα μέχρι το μήκος του διαστήματος να μειωθεί κάτω από ένα προκαθορισμένο όριο ε.

Ως τελική λύση x_s λαμβάνουμε το μέσον αυτού του τελικού διαστήματος. Το σφάλμα της λύσης δεν υπερβαίνει το μισό του διαστήματος, δηλαδή

$$x_s = \frac{b+a}{2} \pm \frac{b-a}{2}$$

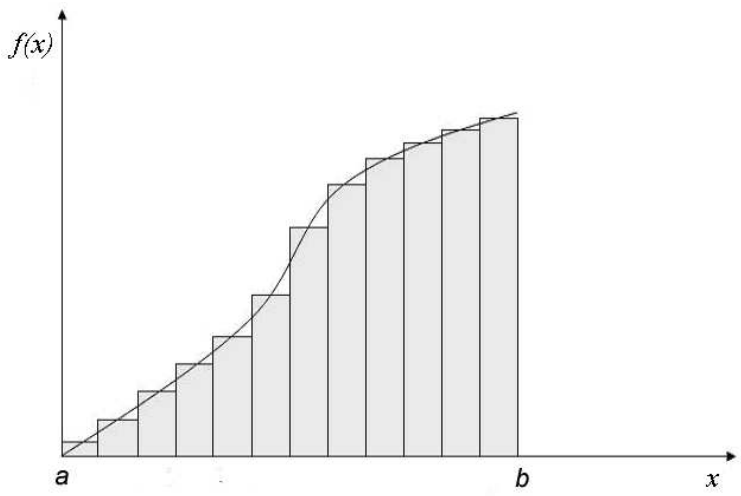
Δεδομένου ότι το μήκος του τελικού διαστήματος θα είναι μικρότερο από ε, ο αριθμός αυτός καθορίζει και την ακρίβεια της τελικής λύσης. Η μέθοδος υλοποιείται από τον παρακάτω κώδικα:

Κώδικας C++	*Σχολιασμός*

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

```
{
    double a, b;                          Τα όρια του διαστήματος αναζήτησης.
    double fa, fb;                        Οι τιμές της συνάρτησης στα όρια.
    double xm;                            Το μέσο του διαστήματος.
    double fm;                            Η τιμή της συνάρτησης στο μέσον.
    double eps;                           Ακρίβεια (μήκος τελικού) διαστήματος.
    int n;                                Μετρητής επαναλήψεων.
    double xs;                            Τελική λύση.
    double fs;                            Τιμή της συνάρτησης στο σημείο xs.

    cout << "Αρχικό διάστημα ? ";
    cin >> a >> b;                        Εισάγονται τα άκρα του διαστήματος.

    fa = a+exp(a)-2;                      Υπολογισμός της συνάρτησης στο σημείο
                                          a.
    fb = b+exp(b)-2;                      Υπολογισμός της συνάρτησης στο σημείο
                                          b.
    if ( fa*fb > 0 ) {                    Ελέγχεται αν η συνάρτηση είναι
        cout << "Δεν περιέχεται λύση της "  ετερόσημη στα άκρα.
             << "εξίσωσης σε αυτό "
             << "το διάστημα.\n";
        return 1;
    }

    cout << "Πόση ακρίβεια ? ";
    cin >> eps;                           Εισάγεται η ακρίβεια της λύσης.
    if ( eps <= 0 ) {                     Ελέγχεται αν δόθηκε σωστός αριθμός.
        cout << "Πρέπει να δώσετε "
             << "θετικό αριθμό.\n";
        return 1;                         Τερματισμός του προγράμματος.
    }

    n = 0;
    while ( b-a > eps ) {
        xm = (b+a)/2;
        fm = xm+exp(xm)-2;                Υπολογισμός της συνάρτησης στο σημείο
                                          xm.
        if ( fa*fm <= 0 ) {               Ελέγχουμε αν στο διάστημα [a, x_m] οι τιμές
                                          της συνάρτησης είναι ετερόσημες.
            b = xm;                       Πρέπει να κρατήσουμε το αριστερό ήμισυ
            fb = fm;                      του αρχικού διαστήματος, οπότε θέτουμε
                                          ως δεξί άκρο το μέσον.
        } else {
```

```
            a = xm;                          Πρέπει να κρατήσουμε το δεξί ήμισυ
            fa = fm;                         του αρχικού διαστήματος, οπότε θέτουμε
                                             ως αριστερό άκρο το μέσον.
        }
        ++n;
    }

    xs = (b+a)/2;                            Η τελική λύση της εξίσωσης.
    fs = xs+exp(xs)-2;                       Η τιμή της συνάρτησης στο σημείο xs.
    cout << "Η λύση είναι x = "
         << xs << endl;
    cout << "Η τιμή της συνάρτησης είναι "
         << fs << endl;
    cout << Εγιναν " << n
         << " επαναλήψεις.\n";
}
```



Σχήμα 5.2: Υπολογισμός ολοκληρώματος με τον κανόνα του παραλληλογράμμου.

5.5.6 Ολοκληρώματα

5.5.6.1 Απλό ολοκλήρωμα

Δεδομένης μιας γνωστής συνάρτησης $f(x)$, θα γράψουμε πρόγραμμα που υπολογίζει προσεγγιστικά το ολοκλήρωμα

$$I = \int_a^b f(x)dx$$

όταν δίνονται τα όρια ολοκλήρωσης a, b χρησιμοποιώντας τον κανόνα του παραλληλογράμμου.

Το ανωτέρω ολοκλήρωμα είναι το εμβαδόν της επιφάνειας που περικλείεται από την $f(x)$ και τον άξονα x (σχ. 5.2). Ο κανόνας του παραλληλογράμμου έχει ως εξής: Χωρίζουμε το διάστημα $[a, b]$ σε N ίσα υποδιαστήματα μήκους $h = (b-a)/N$ το καθένα. Αν τα υποδιαστήματα είναι αρκετά μικρά μπορούμε σε ικανοποιητικό βαθμό να προσεγγίσουμε το εμβαδόν του καθενός από ένα παραλληλόγραμμο που η οριζόντια πλευρά του έχει μήκος h, ενώ το ύψος της κατακόρυφης πλευράς είναι η τιμή της συνάρτησης $f(x)$ υπολογισμένη στο μέσο του διαστήματος. Το μέσο του πρώτου υποδιαστήματος είναι $a + h/2$, το μέσου του δεύτερου $a + h + h/2$, του τρίτου $a + 2h + h/2$ και γενικά το μέσο του υποδιαστήματος υπ' αριθμόν k είναι $a + (k-1)h + h/2$ ή αλλιώς $a + kh - h/2$. Κατά συνέπεια το εμβαδόν του υποδιαστήματος υπ' αριθμόν k είναι $hf(a + kh - h/2)$. Αρκεί λοιπόν να αθροίσουμε τα N επιμέρους εμβαδά.

Το παρακάτω πρόγραμμα υπολογίζει με τον τρόπο αυτό το ολοκλήρωμα της συνάρτησης $f(x) = \sin x$.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`int main ( )`	

```
{
    double a, b;                          Τα όρια ολοκλήρωσης.
    double h;                             Το μήκος κάθε υποδιαστήματος.
    double x, f;
    double s;                             Το ολοκλήρωμα.
    int n, k;

    cout << "Εισάγετε τα όρια a, b\n";    Εισαγωγή των ορίων ολοκλήρωσης από το
    cin >> a >> b;                        πληκτρολόγιο.
    cout << "Πόσα διαστήματα\n";          Εισαγωγή του πλήθους των διαστημάτων.
    cin >> n;

    h = (b-a)/n;                          Υπολογισμός του βήματος ολοκλήρωσης
    s = 0.;
    for ( k=1; k<=n; ++k ) {              Επανάληψη για όλα τα υποδιαστήματα.
        x = a + k*h - h/2;                Υπολογισμός του μέσου του υποδιαστήματος.
        f = sin(x);                       Υπολογισμός της συνάρτησης στο μέσον.
        s += h*f;                         Υπολογισμός του επιμέρους εμβαδού και
    }                                     άθροιση.
    cout << s << endl;                    Εμφάνιση του αποτελέσματος.

}
```

Σημειώστε ότι για τη συνάρτηση $\sin x$ που χρησιμοποιήσαμε το αντίστοιχο ολοκλήρωμα είναι αναλυτικά γνωστό:

$$I = \int_a^b \sin x \; dx = \cos a - \cos b$$

Κατά συνέπεια εάν θέλουμε να ελέγξουμε την ορθότητα του προγράμματός μας μπορούμε στο τέλος να τυπώσουμε την ποσότητα $\cos a - \cos b$ για επιβεβαίωση. Φυσικά η επιβεβαίωση αυτή δεν είναι δυνατή για οποιαδήποτε συνάρτηση.

5.5.6.2 Διπλό ολοκλήρωμα

Χρησιμοποιώντας και πάλι τη μέθοδο του παραλληλογράμμου θα υπολογίσουμε το διπλό ολοκλήρωμα

$$I = \int_{y_a}^{y_b} dy \int_{x_a}^{x_b} dx f(x, y)$$

Μπορούμε να θεωρήσουμε το διπλό ολοκλήρωμα ως:

$$I = \int_{y_a}^{y_b} dy g(y)$$

όπου η συνάρτηση προς ολοκλήρωση $g(y)$ είναι

$$g(y) = \int_{x_a}^{x_b} dx f(x, y)$$

Κατά συνέπεια εφαρμόζουμε την τεχνική του απλού ολοκληρώματος δύο φορές. Μία φορά "εξωτερικά" για να ολοκληρώσουμε την $g(y)$ και μια φορά "εσωτερικά" για να υπολογίσουμε την $g(y)$ για ένα δεδομένο y.

Το παρακάτω πρόγραμμα υπολογίζει με τον τρόπο αυτό το ολοκλήρωμα της συνάρτησης $f(x, y) = x^2 \sin(xy)$.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( )`	
`{`	
`double xa, xb;`	Όρια ολοκλήρωσης στο x.
`double ya, yb;`	Όρια ολοκλήρωσης στο y.
`int n;`	Πλήθος υποδιαστημάτων.
`double xh, yh;`	Μήκη διαστημάτων στο x και y.
`double sx, s;`	
`int i, j;`	
`double x, y, f;`	
`cout << "Δώστε τα όρια για το x ";`	Εισαγωγή των ορίων για το x.
`cin >> xa >> xb ;`	
`cout << "Δώστε τα όρια για το y ";`	Εισαγωγή των ορίων για το y.
`cin >> ya >> yb ;`	
`cout << "Πόσα διαστήματα " ;`	Εισαγωγή του πλήθους των διαστημάτων.
`cin >> n ;`	Χρησιμοποιούμε το ίδιο πλήθος διαστημάτων και για τις δύο διευθύνσεις.
`xh = (xb-xa)/n;`	Υπολογισμός του μήκους των διαστημάτων στη διεύθυνση x.
`yh = (yb-ya)/n;`	Υπολογισμός του μήκους των διαστημάτων στη διεύθυνση y.

```
    s = 0;                              Αρχική τιμή του ολοκληρώματος.
    for ( j=1; j<=n; ++j ) {            Εξωτερική επανάληψη για τον υπολογισμό του
                                        ολοκληρώματος ως προς y.
        y = ya+j*yh-yh/2;
        sx = 0;
        for ( i=1; i<=n; ++i ) {        Εσωτερική επανάληψη για τον υπολογισμό του
                                        ολοκληρώματος ως προς x.
            x = xa+i*xh-xh/2;
            f = x*x*sin(x*y);           Υπολογισμός της συνάρτησης.
            sx += xh*f;
        }
        s += yh*sx;
    }

    cout << s << endl;                  Εμφάνιση του αποτελέσματος.
}
```

Κεφάλαιο 6

Μονοδιάστατοι πίνακες

Γνωρίζουμε ότι οι μεταβλητές είναι συμβολικά ονόματα που δίνουμε σε θέσεις αποθήκευσης στην κεντρική μνήμη του ΗΥ. Σε πολλές εφαρμογές υπάρχει ανάγκη αποθήκευσης πολλών δεδομένων του ιδίου τύπου και παρόμοιας σημασίας. Στις περιπτώσεις αυτές χρησιμοποιούμε πίνακες. Ένας *μονοδιάστατος πίνακας* είναι μια ομάδα διαδοχικών θέσεων αποθήκευσης στη μνήμη στις οποίες αναφερόμαστε με ένα κοινό όνομα. Για να ξεχωρίσουμε τα επιμέρους στοιχεία μεταξύ τους χρησιμοποιούμε ένα ακέραιο δείκτη.

6.1 Δήλωση

Κάθε πίνακας πρέπει να δηλωθεί, έτσι ώστε να καθοριστεί το όνομά του, το πλήθος των στοιχείων του και ο τύπος τους. Η δήλωση γίνεται μαζί με τις άλλες μεταβλητές και έχει τη μορφή:

τύπος *όνομα* `[` *πλήθος_στοιχείων* `]` `;`

όπου *τύπος* είναι ο τύπος των στοιχείων του πίνακα και μπορεί να είναι οποιοσδήποτε από τους απλούς τύπους `double`, `int`, `char` κλπ, καθώς και παράγωγοι τύποι που θα περιγραφούν σε επόμενα κεφάλαια. Για παράδειγμα, η δήλωση ενός πίνακα που ονομάζεται `x` και αποτελείται από 100 στοιχεία διπλής ακρίβειας είναι:

```
double x[100];
```

Ένας άλλος τρόπος δήλωσης είναι με τη χρήση συμβολικών σταθερών:

```
#define NMAX 100
double x[NMAX];
```

Με το `#define` ορίζουμε μια συμβολική σταθερά που ονομάζεται `NMAX` και έχει την τιμή 100. Κατόπιν χρησιμοποιούμε το `NMAX` αντί του σταθερού αριθμού 100 στη δήλωση. Η γραμμή `#define` δεν αποτελεί εντολή της γλώσσας C++ αλλά απευθύνεται στον *προεπεξεργαστή* (δείτε και παράγραφο 2.4).

Τα επιμέρους στοιχεία του πίνακα γράφονται με τη βοήθεια ενός ακέραιου δείκτη ο οποίος λαμβάνει τιμές από 0 έως 99. Έτσι το πρώτο στοιχείο είναι το `x[0]`, το δεύτερο το `x[1]` κοκ μέχρι το τελευταίο

που είναι το `x[99]`. Είναι προφανές ότι από το πλήθος των 100 στοιχείων μπορεί στο πρόγραμμα να χρειαστεί να χρησιμοποιήσουμε λιγότερα. Όλα τα στοιχεία του πίνακα στο ανωτέρω παράδειγμα είναι τύπου `double` και μπορούν να χρησιμοποιηθούν σε αριθμητικές παραστάσεις ακριβώς όπως και οι απλές μεταβλητές.

Παραδείγματα:

```
x[0] = 4;
a = 3*x[0]+2*x[1];
cin >> x[4];
```

6.2 Ανάθεση τιμών στα στοιχεία του πίνακα

6.2.1 Ανάθεση τιμών κατά τη δήλωση

Μπορούμε να αναθέσουμε τιμές στα στοιχεία ενός πίνακα μαζί με τη δήλωσή του. Πχ. η δήλωση

```
int a[10]={4,7,1,-3,0};
```

ορίζει ένα πίνακα ακεραίων με 10 στοιχεία, όπου οι τιμές των πέντε πρώτων γράφονται μέσα στα άγκιστρα. Οι τιμές αυτές αποτελούν αρχικές τιμές και μπορούν να τροποποιηθούν στη συνέχεια από το πρόγραμμα.

6.2.2 Απευθείας ανάθεση

Κάθε στοιχείο του πίνακα μπορεί να λάβει τιμή όπως και οποιαδήποτε άλλη μεταβλητή.

Παραδείγματα:

```
x[0] = 1.2;
x[1] = 2.5;
x[2] = z+2*x[0];
```

Επίσης μπορούμε να χρησιμοποιήσουμε κατάλληλες εντολές `for` έτσι ώστε να δώσουμε τιμές σε πολλά (ή όλα) στοιχεία του πίνακα.

Παράδειγμα:

```
for (k=0; k<100; ++k)
    x[k] = k*k/2;
```

6.2.3 Εισαγωγή τιμών από το πληκτρολόγιο

Μπορούμε να εισάγουμε ένα προς ένα τα στοιχεία του πίνακα από το πληκτρολόγιο όπως φαίνεται στο παρακάτω απόσπασμα προγράμματος:

Κώδικας C++	*Σχολιασμός*
⋮	
`#define NMAX 100`	Πλήθος στοιχείων του πίνακα.
`double x[NMAX];`	Δήλωση του πίνακα.
⋮	
`cout << "Πόσα στοιχεία ? ";`	Εισαγωγή του πλήθους των στοιχείων
`cin >> n;`	από το πληκτρολόγιο.
`if ( n>NMAX || n<1 ) {`	Έλεγχος αν δόθηκε σωστός αριθμός.
`  cout << "Λανθασμένος αριθμός \n";`	
`  return 1;`	Τερματισμός του προγράμματος.
`}`	
`for ( k=0; k<n; ++k ) {`	Επανάληψη για όλα τα στοιχεία του πίνακα.
`  cout << "Εισάγετε το στοιχείο "`	Προτρεπτικό μήνυμα.
`  << k << endl;`	
`  cin >> x[k];`	Εισαγωγή του στοιχείου με δείκτη `k`.
`}`	
⋮	

6.3 Έξοδος τιμών

Για να εμφανίσουμε τα στοιχεία ενός πίνακα στην οθόνη χρησιμοποιούμε εντολή `for` όπως φαίνεται παρακάτω:

```
for ( k=0; k<n; ++k )
    cout << x[k] << endl;
```

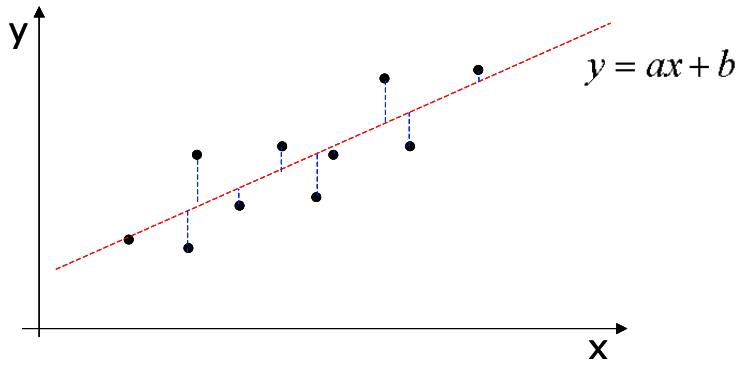
6.4 Παραδείγματα

6.4.1 Μέσος όρος και ελάχιστη τιμή

Το παρακάτω πρόγραμμα βρίσκει το μέσο όρο των στοιχείων ενός πίνακα, το μικρότερο στοιχείο του και τη θέση στην οποία βρίσκεται αυτό.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`#define NMAX 100`	Μέγιστο πλήθος στοιχείων.
`int main ( )`	
`{`	
`  double x[NMAX];`	Δήλωση του πίνακα.
`  int n;`	Το πλήθος των στοιχείων που θα χρησιμοποιήσουμε.
`  int k;`	
`  double s;`	Το άθροισμα των στοιχείων του πίνακα.
`  double avg;`	Ο μέσος όρος των στοιχείων του πίνακα.
`  double min;`	Το μικρότερο στοιχείο του πίνακα.
`  double kmin;`	Η θέση του μικρότερου στοιχείου.
`  cout << "Πόσα στοιχεία ? ";`	Εισαγωγή του πλήθους των στοιχείων του
`  cin >> n;`	πίνακα.
`  if ( n > NMAX ) {`	Έλεγχος αν δώσαμε λανθασμένο αριθμό.
`    cout << "Επιτρέπονται μέχρι ";`	
`    cout << NMAX << " στοιχεία\n";`	
`    return 1;`	Τερματισμός του προγράμματος.
`  }`	
`  for ( k=0; k<n; ++k) {`	Εισαγωγή όλων των στοιχείων του πίνακα
`    cout << "Δώστε το στοιχείο "`	ένα–προς–ένα.
`         << k << endl;`	
`    cin >> x[k];`	
`  }`	
`  s = 0;`	Αρχική τιμή αθροιστή.
`  for ( k=0; k<n; ++k)`	Υπολογισμός του αθροίσματος όλων των
`    s += x[k];`	στοιχείων.
`  avg = s/n;`	Υπολογισμός του μέσου όρου.
`  cout << "Μέσος όρος: "`	Εμφάνιση του μέσου όρου στην οθόνη.

```
        << avg << "\n";

    min = x[0];                          Αρχικά θέτουμε ως μικρότερο το πρώτο
                                         στοιχείο του πίνακα.
    kmin = 0;                            kmin είναι η θέση του μικρότερου.
    for ( k=1; k<n; ++k)                 Επανάληψη για όλα τα στοιχεία του πίνακα
                                         πλην του πρώτου.
        if ( x[k] < min ) {              Ελέγχουμε αν το x[k] είναι πιο μικρό από
                                         αυτό που θεωρούμε ως μικρότερο (min).
            min = x[k];                  Θέτουμε ως μικρότερο το x[k].
            kmin = k;                    Κρατάμε τη θέση του.
        }
    cout << "Ελάχιστο: " << min;         Εμφάνιση του μικρότερου και της θέσης
    cout << " στη θέση: " << kmin        του.
         << endl;
}
```



Σχήμα 6.1: Η ευθεία ελαχίστων τετραγώνων.

6.4.2 Ευθεία ελαχίστων τετραγώνων

Θεωρήστε μια σειρά N πειραματικών μετρήσεων (x_i, y_i). Οι συντελεστές της βέλτιστης ευθείας $y = ax + b$ που διέρχεται από τα σημεία (σχ. 6.1) δίνονται από:

$$a = \frac{N s_{xy} - s_x s_y}{N s_{xx} - s_x^2} \qquad b = \frac{s_{xx} s_y - s_x s_{xy}}{N s_{xx} - s_x^2}$$

όπου

$$s_x = \sum_{i=1}^{N} x_i \qquad s_y = \sum_{i=1}^{N} y_i \qquad s_{xx} = \sum_{i=1}^{N} x_i^2 \qquad s_{xy} = \sum_{i=1}^{N} x_i y_i$$

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`#define NMAX 100`	Μέγιστο πλήθος σημείων.
`int main ( )`	
`{`	
`  double x[NMAX], y[NMAX];`	Δήλωση των πινάκων.
`  int n, k;`	
`  double sx, sy, sxx, sxy;`	
`  double a, b;`	
`  cout << "Πόσα σημεία έχετε ";`	Εισαγωγή του πλήθους των σημείων.
`  cin >> n;`	
`  if ( n < 1 || n > NMAX ) {`	Έλεγχος αν δόθηκε σωστό n.
`    cout << "ΛΑΘΟΣ \n";`	
`    return 1;`	Τερματισμός του προγράμματος.
`  }`	

```
   for ( k=0; k<n; ++k ) {                  Εισαγωγή όλων των σημείων από το
      cout << "Δώστε το σημείο "            πληκτρολόγιο.
           << k << endl;
      cin >> x[k] >> y[k];
   }

   sx = sy = sxx = sxy = 0;                 Αρχική τιμή των αθροιστών.
   for ( k=0; k<n; ++k ) {                  Υπολογισμός των τεσσάρων αθροισμάτων
      sx += x[k];                           που απαιτούνται.
      sy += y[k];
      sxx += x[k]*x[k];
      sxy += x[k]*y[k];
   }
   a = (n*sxy-sx*sy)/                       Υπολογισμός της κλίσης της ευθείας.
       (n*sxx-sx*sx);
   b = (sxx*sy-sx*sxy)/                     Υπολογισμός του σταθερού όρου της
       (n*sxx-sx*sx);                       ευθείας.
   cout << "Οι συντελεστές είναι: \n";      Εμφάνιση των συντελεστών στης ευθείας.
   cout << a << endl;
   cout << b << endl;
}
```

6.4.3 Ταξινόμηση

Δεδομένου ενός πίνακα `x` το παρακάτω πρόγραμμα ταξινομεί τα στοιχεία του σε αύξουσα σειρά χρησιμοποιώντας τη μέθοδο της *απευθείας επιλογής*. Συγκεκριμένα ξεκινώντας από το `x[0]` και ψάχνοντας μέχρι το τελευταίο στοιχείο, βρίσκουμε το μικρότερο. Εναλλάσσουμε το μικρότερο στοιχείο με το `x[0]`. Έτσι το μικρότερο στοιχείο όλου του πίνακα έχει τοποθετηθεί στην τελική του θέση. Κατόπιν βρίσκουμε και πάλι το μικρότερο στοιχείο του πίνακα, όμως ξεκινώντας από το `x[1]`, και μόλις το βρούμε το εναλλάσσουμε με το `x[1]`. Η διαδικασία επαναλαμβάνεται μέχρι να βρούμε το μικρότερο ξεκινώντας από την προτελευταία θέση.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`#define NMAX 100`	Μέγιστο πλήθος στοιχείων.
`int main ( )`	
`{`	
`  double x[NMAX];`	Δήλωση του πίνακα.
`  int n;`	
`  int k, i, j;`	
`  double xmin, tmp;`	
`  int pos;`	
`  cout << "Πόσα στοιχεία ? ";`	Εισαγωγή του πλήθους των στοιχείων.
`  cin >> n;`	
`  if ( n < 1 || n > NMAX ) {`	Έλεγχος αν δώσαμε σωστό αριθμό.
`    cout << "Λάθος αριθμός\n";`	
`    return 1;`	Τερματισμός του προγράμματος.
`  }`	
`  for ( k=0; k<n; ++k ) {`	Εισαγωγή όλων των στοιχείων του πίνακα από το πληκτρολόγιο.
`    cout << "Δώστε το στοιχείο "`	
`         << k << "\n";`	
`    cin >> x[k];`	
`  }`	
`  for ( i=0; i<n-1; ++i ) {`	Εξωτερική επανάληψη που καθορίζει από ποιο στοιχείο θα αρχίσει η αναζήτηση του μικρότερου.
`    xmin = x[i];`	Αρχική τιμή για το μικρότερο.
`    pos = i;`	Αρχική τιμή για τη θέση του μικρότερου.
`    for ( j=i+1; j<n; ++j )`	Εσωτερική επανάληψη για την εύρεση του μικρότερου ξεκινώντας από τη θέση `i`.
`      if ( x[j] < xmin ) {`	Αν το `x[j]` είναι πιο μικρό από αυτό που

```
            xmin = x[j];                θεωρούμε μικρότερο μέχρι εδώ.
            pos = j;
        }
    tmp = x[i];                         Εναλλαγή του στοιχείου x[i] με το
    x[i] = x[pos];                      μικρότερο.
    x[pos] = tmp;
}

cout << "Ο ταξινομημένος πίνακας "      Εμφάνιση του ταξινομημένου πίνακα στην
     << "είναι: \n";                    οθόνη.
for ( k=0; k<n; ++k)
    cout << x[k] << endl;
}
```

6.4.4 Σύστημα φορτισμένων σωματιδίων

Θα μελετήσουμε ένα σύστημα n φορτισμένων σωματιδίων (ιόντων). Η θέση $\vec{r}_i$ κάθε σωματιδίου προσδιορίζεται από τρεις καρτεσιανές συντεταγμένες x_i, y_i, z_i. Επιπλέον κάθε σωματίδιο έχει μάζα m_i και φορτίο q_i. Να βρεθούν:

1. Οι συντεταγμένες (μ_x, μ_y, μ_z) του κέντρου μάζας του συστήματος που δίνονται από:

$$\mu_x = \frac{\sum_{i=1}^{n} x_i m_i}{\sum_{i=1}^{n} m_i} \qquad \mu_y = \frac{\sum_{i=1}^{n} y_i m_i}{\sum_{i=1}^{n} m_i} \qquad \mu_z = \frac{\sum_{i=1}^{n} z_i m_i}{\sum_{i=1}^{n} m_i}$$

2. Το ολικό φορτίο του συστήματος:

$$q_{tot} = \sum_{i=1}^{n} q_i$$

3. Η ηλεκτροστατική ενέργεια του συστήματος:

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{q_i q_j}{|\vec{r}_i - \vec{r}_j|}$$

Στο πρόγραμμα θα χρησιμοποιήσουμε συνολικά πέντε πίνακες. Τρεις για τις συντεταγμένες των σωματιδίων, έναν για τις μάζες και έναν για τα φορτία. Έτσι το σωματίδιο υπ' αριθμόν k θα έχει συντεταγμένες `x[k]`, `y[k]`, `z[k]`, μάζα `m[k]` και φορτίο `q[k]`. Επίσης πρέπει να λάβουμε υπ' όψη ότι οι δείκτες στα ανωτέρω αθροίσματα ξεκινούν από το 1, ενώ οι δείκτες των πινάκων ξεκινούν από το μηδέν.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`#define NMAX 100`	Μέγιστο πλήθος στοιχείων του πίνακα.
`int main ( )`	
`{`	
`int n;`	
`double x[NMAX], y[NMAX], z[NMAX];`	Οι πίνακες συντεταγμένων.
`double m[NMAX];`	Ο πίνακας μαζών.
`double q[NMAX];`	Ο πίνακας φορτίων.
`int k, i, j;`	
`double sx, sy, sz, sm;`	
`double mx, my, mz;`	Οι συντεταγμένες του κέντρου μάζας.
`double qtot;`	
`double d, e;`	
`cout << "Πόσα σωματίδια ? \n";`	Εισαγωγή του πλήθους των σωματιδίων.
`cin >> n;`	

```
    if ( n < 1 || n > NMAX ) {                Ελέγχουμε αν ο αριθμός είναι
        cout << "Λανθασμένος αριθμός\n";      λανθασμένος.
        return 1;                             Τερματισμός του προγράμματος.
    }
    for ( k=0; k<n; ++k ) {                   Εισαγωγή των συντεταγμένων, μαζών και
        cout << "Δώστε συντεταγμένες, μάζα"   φορτίων για όλα τα σωματίδια.
             << " και φορτίο για το ιόν "
             << k << "\n";
        cin >> x[k] >> y[k] >> z[k
            >> m[k] >> q[k]
    }

    sx = sy = sz = sm = 0;
    for ( k=0; k<n; ++k ) {                   Υπολογισμός των αθροισμάτων που
        sx += m[k]*x[k];                      απαιτούνται για το κέντρο μάζας.
        sy += m[k]*y[k];
        sz += m[k]*z[k];
        sm += m[k];
    }
    mx = sx/sm;                               Υπολογισμός των συντεταγμένων του
    my = sy/sm;                               κέντρου μάζας.
    mz = sz/sm;
    cout << "Το κέντρο μάζας είναι " << mx    Εμφάνιση του κέντρου μάζας στην οθόνη.
         << " " << my << " " << mz << endl;

    qtot = 0;
    for ( k=0; k<n; ++k )                     Υπολογισμός του ολικού φορτίου.
        qtot += q[k];
    cout << "Το ολικό φορτίο είναι "          Εμφάνιση του ολικού φορτίου στην οθόνη.
         << qtot << endl;

    e = 0;
    for ( i=0; i<n-1; ++i )                   Υπολογισμός της ηλεκτροστατικής
        for ( j=i+1; j<n; ++j ) {             ενέργειας.
            d = sqrt( pow(x[i]-x[j],2)+       d είναι η απόσταση μεταξύ των
                      pow(y[i]-y[j],2)+       σωματιδίων i και j.
                      pow(z[i]-z[j],2) );
            e += q[i]*q[j]/d;
        }
    cout << "Η ηλεκτροστατική ενέργεια "      Εμφάνιση της ηλεκτροστατικής ενέργειας
         << "είναι " << e << endl;            στην οθόνη.
}
```

6.5 Πίνακες χαρακτήρων

Θα δούμε εδώ την ειδική περίπτωση του πίνακα χαρακτήρων (συμβολοσειρά). Οι πίνακες αυτοί χρησιμεύουν στην αποθήκευση και επεξεργασία σειρών χαρακτήρων πχ. λέξεων ή κειμένου γενικότερα Δηλώνονται όπως οι υπόλοιποι πίνακες. Πχ:

```
#define MAX 100
char s[MAX];
```

Όμως για τους εισάγουμε από το πληκτρολόγιο ή να τους εμφανίσουμε στην οθόνη δεν χρησιμοποιούμε ένα–ένα τα στοιχεία, αλλά όλο τον πίνακα:

```
cin >> s;      // Εισαγωγή από το πληκτρολόγιο (μόνο μια λέξη).
cout << s;
```

Από το μέγιστο δηλωμένο μέγεθος ενός πίνακα χαρακτήρων μπορεί να χρησιμοποιήσουμε λιγότερες θέσεις. Για να γνωρίζουμε πόσες θέσεις χρησιμοποιήθηκαν τοποθετείται ο χαρακτήρας τερματισμού μετά το τέλος των χρησιμοποιημένων θέσεων. Ο χαρακτήρας τερματισμού είναι ο χαρακτήρας που αντιστοιχεί στον ακέραιο 0 και συμβολίζεται με το χαρακτήρα διαφυγής `\0`. Έτσι για παράδειγμα αν έχουμε μια εντολή

```
cin >> s;
```

και πληκτρολογήσουμε τη λέξη

```
HELLO
```

τότε οι χαρακτήρες της λέξης θα τοποθετηθούν στις πέντε πρώτες θέσεις του πίνακα `s` ενώ στη θέση `s[5]` θα τοποθετηθεί ο χαρακτήρας τερματισμού. Η τοποθέτησή του γίνεται αυτόματα από την `cin`. Εάν όμως για κάποιο λόγο δώσουμε απ' ευθείας τιμές στα στοιχεία του πίνακα `s` θα πρέπει να τοποθετήσουμε εμείς το χαρακτήρα τερματισμού, όπως στο παρακάτω παράδειγμα:

Παράδειγμα:

```
s[0] = 'H';
s[1] = 'E';
s[2] = 'L';
s[3] = 'L';
s[4] = 'O';
s[5] = '\0';
```

6.5.1 Συναρτήσεις για συμβολοσειρές

Υπάρχουν πολλές διαθέσιμες συναρτήσεις για την επεξεργασία συμβολοσειρών. Ορισμένες από αυτές περιγράφονται παρακάτω:

- `strlen ( s )`
  Επιστρέφει το πλήθος των χαρακτήρων της συμβολοσειράς `s` (δεν συμπεριλαμβάνεται ο χαρακτήρας τερματισμού).
- `strcpy ( s1, s2 )`
  Αντιγράφει τη συμβολοσειρά `s2` στην `s1`.

- `strcmp ( s1, s2 )`
  Συγκρίνει τις συμβολοσειρές `s1` και `s2` και επιστρέφει:

  -1 αν η `s1` είναι λεξικογραφικά μικρότερη από την `s2`.

  0 αν η `s1` είναι λεξικογραφικά ίση με την `s2`.

  1 αν η `s1` είναι λεξικογραφικά μεγαλύτερη από την `s2`.

- `gets ( s )`
  Εισαγωγή από το πληκτρολόγιο της συμβολοσειράς `s` (μια ολόκληρη φράση μέχρι να πατήσουμε Enter). Τοποθετεί αυτόματα και το χαρακτήρα τερματισμού.

Για να χρησιμοποιήσουμε τις τρεις πρώτες συναρτήσεις πρέπει να βάλουμε:

```
#include <cstring>
```

ενώ για την τελευταία:

```
#include <cstdio>
```

6.5.2 Παραδείγματα

6.5.2.1 Μήκος συμβολοσειράς

Το παρακάτω πρόγραμμα δέχεται ως είσοδο μια συμβολοσειρά και βρίσκει το μήκος της, δηλαδή πόσοι χαρακτήρες δόθηκαν. Για να γίνει αυτό ελέγχονται όλοι οι χαρακτήρες ένας προς ένα μέχρι να βρεθεί ο χαρακτήρας τερματισμού \0. Ουσιαστικά ο παρακάτω κώδικας κάνει ότι και η συνάρτηση `strlen`.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`#define MAX 100`	Μέγιστο πλήθος χαρακτήρων.
`int main ( )`	
`{`	
`  char s[MAX];`	Δήλωση του πίνακα χαρακτήρων.
`  int k;`	
`  cout << "Εισάγετε μια λέξη ";`	Εισαγωγή του πίνακα χαρακτήρων από το
`  cin >> s;`	πληκτρολόγιο.
`  for ( k=0; s[k] != '\0'; ++k )`	Επανάληψη ξεκινώντας από τον πρώτο χαρακτήρα. Η επανάληψη συνεχίζεται όσο δεν βρίσκουμε το χαρακτήρα τερματισμού.
`      ;`	Κενή εντολή.
`  cout << "Το μήκος είναι "`	Εμφάνιση του αποτελέσματος.
`       << k << endl;`	
`}`	

Σημειώστε ότι στην εντολή `for` δεν ακολουθεί κάποια εντολή που θα επαναλαμβάνεται. Για το λόγο αυτό τοποθετούμε απλώς ένα ερωτηματικό (κενή εντολή). Η εντολή `for` θα μπορούσε να αντικατασταθεί ισοδύναμα με την παρακάτω εντολή `while` (όμως όχι από μια εντολή `do while`):

```
k = 0;
while ( s[k] != '\0' )
    ++k;
```

6.5.2.2 Μετατροπή κεφαλαίων σε μικρά

Το παρακάτω πρόγραμμα δέχεται ως είσοδο μια συμβολοσειρά και μετατρέπει τυχόν κεφαλαίους χαρακτήρες σε μικρούς.

Για να μετατρέψουμε ένα χαρακτήρα από κεφαλαίο σε μικρό βασιζόμαστε στο ότι υπάρχει συγκεκριμένη αντιστοιχία μεταξύ χαρακτήρων και ακεραίων αριθμών. Δεν χρειάζεται να ξέρουμε την ακριβή αντιστοιχία κάθε χαρακτήρα, όμως στον πίνακα αντιστοίχησης υπάρχουν τρεις ομάδες χαρακτήρων: α) κεφαλαία, β) μικρά και γ) ψηφία. Οι χαρακτήρες κάθε ομάδας βρίσκονται σε διαδοχικές θέσεις του πίνακα, αντιστοιχούν δηλαδή σε διαδοχικούς ακέραιους. Για τη μετατροπή ενός χαρακτήρα (`char c`) από κεφαλαίο σε μικρό βρίσκουμε πόσο απέχει ο χαρακτήρας από την αρχή των κεφαλαίων (δηλαδή από το `A`):

```
d = c-'A';
```

Προσθέτουμε την απόσταση αυτή στην αρχή των μικρών (δηλαδή στο `a`):

```
new = 'a'+d;
```

ή συνολικά:

```
new = 'a'-'A'+c;
```

Στις αριθμητικές πράξεις χρησιμοποιούμε χαρακτήρες όμως εκεί εννοείται ο αντίστοιχος ακέραιος. Αυτό δείχνει και τη στενή σχέση χαρακτήρων και ακεραίων αριθμών.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cstdio>`	Χρειάζεται για την `gets`.
`#include <cstring>`	Χρειάζεται για την `strlen`.
`using namespace std;`	
`#define MAX 100`	Μέγιστο πλήθος χαρακτήρων.
`main ( )`	
`{`	
`  char s[MAX];`	Δήλωση του πίνακα χαρακτήρων.
`  char snew[MAX];`	Δήλωση πίνακα χαρακτήρων που θα περιέχει τη συμβολοσειρά μετά τη μετατροπή.
`  int len, k;`	
`  cout << "Δώστε μια συμβολοσειρά ";`	Εισαγωγή του πίνακα χαρακτήρων από το πληκτρολόγιο.
`  gets(s);`	
`  len = strlen(s);`	Προσδιορισμός του μήκους (πόσοι χαρακτήρες δόθηκαν).
`  for (k=0; k<len; ++k)`	Επανάληψη για όλους του χαρακτήρες του πίνακα.
`    if (s[k]>='A' && s[k]<='Z')`	Έλεγχος αν είναι κεφαλαίος.
`      snew[k] = s[k]-'A'+'a';`	Μετατροπή σε μικρό.
`    else`	
`      snew[k] = s[k];`	Ο χαρακτήρας παραμένει ως έχει.

```
    snew[len] = '\0';                    Τοποθετούμε στο νέο πίνακα το χαρακτήρα
                                         τερματισμού.
    cout << snew << endl;                Εμφάνιση του νέου πίνακα στην οθόνη.
}
```

6.5.2.3 Καταμέτρηση χαρακτήρων

Το παρακάτω πρόγραμμα δέχεται ως είσοδο μια συμβολοσειρά και μετράει πόσοι: 1) κεφαλαίοι χαρακτήρες 2) μικροί χαρακτήρες 3) αριθμητικά ψηφία 4) σύμβολα υπάρχουν στη συμβολοσειρά.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cstdio>`	Χρειάζεται για την `gets`.
`#include <cstring>`	Χρειάζεται για την `strlen`.
`using namespace std;`	
`#define MAX 100`	Μέγιστο πλήθος χαρακτήρων.
`int main ()`	
`{`	
`char str[MAX];`	Δήλωση του πίνακα χαρακτήρων.
`int len;`	Το μήκος της συμβολοσειράς.
`int n1;`	Πλήθος κεφαλαίων.
`int n2;`	Πλήθος μικρών.
`int n3;`	Πλήθος ψηφίων.
`int n4;`	Πλήθος άλλων συμβόλων.
`int k;`	
`cout << "Δώστε μια φράση ";`	Εισαγωγή της συμβολοσειράς από το
`gets(str);`	πληκτρολόγιο.
`len=strlen(s);`	Προσδιορίζεται το μήκος της συμβολοσειράς.
`n1 = n2 = n3 = n4 = 0;`	Αρχική τιμή μετρητών.
`for (k=0; k<len; ++k)`	Επανάληψη για όλους τους χαρακτήρες.
`if (str[k]>='A' && str[k]<='Z')`	Έλεγχος αν είναι κεφαλαίος.
`++n1;`	
`else if (str[k]>='a' && str[k]<='z')`	Έλεγχος αν είναι μικρός.
`++n2;`	
`else if (str[k]>='0' && str[k]<='9')`	Έλεγχος αν είναι ψηφίο.
`++n3;`	
`else`	
`++n4;`	
`cout << n1 << " Κεφαλαία \n";`	Εμφάνιση των μετρητών στην οθόνη.
`cout << n2 << " Μικρά \n";`	
`cout << n3 << " Ψηφία \n";`	
`cout << n4 << " Αλλα σύμβολα \n";`	
`}`	

Κεφάλαιο 7

Συναρτήσεις I

Οι συναρτήσεις είναι τμήματα κώδικα (υποπρογράμματα) που πραγματοποιούν μια καθορισμένη εργασία. Είναι χρήσιμες για περιπτώσεις που ο ίδιος υπολογισμός επαναλαμβάνεται πολλές φορές μέσα στο πρόγραμμα αλλά και για την τμηματική ανάπτυξη και έλεγχο μεγάλων προγραμμάτων. Ως παράδειγμα αναφέρουμε τις ενσωματωμένες συναρτήσεις **sin**, **cos**, **sqrt**, **abs**, κλπ. Κάθε συνάρτηση μπορεί να δέχεται ως είσοδο μία ή περισσότερες μεταβλητές και μπορεί να επιστρέφει ένα αποτέλεσμα (ή παραπάνω). Κάθε πρόγραμμα C++ αποτελείται από μία ή περισσότερες συναρτήσεις.

7.1 Πως γράφονται οι συναρτήσεις

Η δομή μιας συνάρτησης μοιάζει με αυτή του κυρίως προγράμματος **main** και έχει ως εξής:

τύπος_αποτελέσματος *όνομα_συνάρτησης* (*παράμετροι*)
{
    δηλώσεις τοπικών μεταβλητών
        ⋮
    εντολές
        ⋮
    return *τιμή αποτελέσματος* ;
{

Όπου *τύπος_αποτελέσματος* είναι ο τύπος του αποτελέσματος που επιστρέφει η συνάρτηση (πχ **int**, **double** κοκ), και *παράμετροι* είναι μια λίστα με τις παραμέτρους που δέχεται ως είσοδο η συνάρτηση. Η λίστα των παραμέτρων έχει τη μορφή:

τύπος μεταβλητή, *τύπος μεταβλητή*, ...

Με την εντολή **return** τερματίζεται η λειτουργία της συνάρτησης και επιστρέφεται το αποτέλεσμα.

Παράδειγμα:
Για να υλοποιήσουμε σε C++ τη συνάρτηση:

$$f(x) = \frac{7x^2 - 3x + 6}{\sqrt{1 + x^2}}$$

θα γράφαμε:

```
double f ( double x )
{
    return (7*x*x-3*x+6)/sqrt(1+x*x);
}
```

Η συνάρτηση αυτή ονομάζεται `f`, επιστρέφει αποτέλεσμα τύπου `double`, ενώ δέχεται ως είσοδο μια μεταβλητή που ονομάζεται `x` και είναι τύπου `double`.

Αφού γράψουμε τον κώδικα της συνάρτησης, για να την χρησιμοποιήσουμε πρέπει να δηλώσουμε το *πρωτότυπό* της. Το πρωτότυπο μιας συνάρτησης δηλώνει τι τύπου ορίσματα δέχεται η συνάρτηση και τι τύπου αποτέλεσμα επιστρέφει. Είναι ίδιο με την επικεφαλίδα της συνάρτησης χωρίς τα ονόματα των μεταβλητών (τα οποία όμως ακόμη και αν τα γράψουμε δεν πειράζει). Το πρωτότυπο χρειάζεται έτσι ώστε ο μεταφραστής να γνωρίζει τους τύπους των ορισμάτων και του αποτελέσματος. Τα πρωτότυπα των συναρτήσεων γράφονται πριν από το `main` και γενικά πριν χρησιμοποιήσουμε τις συναρτήσεις. Στο τέλος του πρωτότυπου μπαίνει ερωτηματικό.

Παράδειγμα:
Το πρωτότυπο της παραπάνω συνάρτησης `f` είναι:

```
double f ( double );
```

Σε μια συνάρτηση μπορούμε να δηλώσουμε και να χρησιμοποιήσουμε μεταβλητές (απλές ή πίνακες) όπως και στο κυρίως πρόγραμμα. Οι μεταβλητές που ορίζονται εσωτερικά σε μια συνάρτηση αποτελούν τοπικές μεταβλητές της συνάρτησης και δεν έχουν σχέση με μεταβλητές που έχουν το ίδιο όνομα σε άλλες συναρτήσεις ή στο `main`.

Παράδειγμα:
Κατασκευάστε συνάρτηση που θα δέχεται εκατοστά και θα τα μετατρέπει σε ίντσες. Υπενθυμίζεται ότι 1 in = 2.54 cm.

Κώδικας C++	*Σχολιασμός*
`double convert ( double cm )`	Επικεφαλίδα της συνάρτησης.
`{`	
`    double inch;`	Τοπική μεταβλητή.
`    inch = cm/2.54;`	Μετατροπή σε ίντσες.
`    return inch;`	Τερματισμός της συνάρτησης και
`}`	επιστροφή του αποτελέσματος.

Ένα πρόγραμμα που χρησιμοποιεί τη συνάρτηση `convert`:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`double convert ( double );`	Πρωτότυπο συνάρτησης.
`int main ( )`	
`{`	
`  double a, b;`	
`  cout << "Πόσα εκατοστά ? ";`	
`  cin >> a;`	Εισαγωγή των εκατοστών.
`  b = convert(a);`	Κλήση συνάρτησης
`  cout << a << " εκατοστά είναι "`	Εμφάνιση του αποτελέσματος σε
`  << b << " ίντσες \n";`	ίντσες.
`}`	

Για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να την καλέσουμε. Πχ η συνάρτηση με πρωτότυπο:

```
double convert ( double );
```

καλείται ως:

```
a = convert(b);
```

Οι μεταβλητές στην επικεφαλίδα της συνάρτησης (`cm`) ονομάζονται *τυπικές παράμετροι*. Οι μεταβλητές που δίνουμε κατά την κλήση (`b`) ονομάζονται *πραγματικές παράμετροι ή ορίσματα* της κλήσης. Το πλήθος και οι τύποι των ορισμάτων κατά την κλήση πρέπει να συμβαδίζουν με αυτά που δηλώθηκαν στο πρωτότυπο και την επικεφαλίδα της συνάρτησης.

Κάθε συνάρτηση μπορεί να κληθεί πολλές φορές με διαφορετικά ορίσματα. Πχ:

```
x = convert(y);
q = convert(3*w-t);
```

Η κλήση και η επιστροφή της συνάρτησης γίνεται ως εξής: Οι πραγματικές παράμετροι αντιγράφονται στις τυπικές παραμέτρους. Πχ αν η ανωτέρω συνάρτηση `convert` κληθεί ως:

```
a = convert(b);
```

τότε η τιμή της μεταβλητής `b` αντιγράφεται στην `cm`. Ο τρόπος αυτός μεταβίβασης τιμών ονομάζεται *μεταβίβαση κατ' αξία*. Κατόπιν ο υπολογιστής εγκαταλείπει το σημείο όπου βρισκόταν και πηγαίνει στον κώδικα της συνάρτησης. Εκτελούνται οι εντολές της συνάρτησης και μόλις βρεθεί η εντολή `return`, ο υπολογιστής επιστρέφει στο σημείο από όπου κλήθηκε η συνάρτηση.

Μια συνάρτηση μπορεί να μην επιστρέφει αποτέλεσμα. Λέμε ότι είναι συνάρτηση τύπου `void`. Μια συνάρτηση που δεν επιστρέφει αποτέλεσμα δεν καλείται με τον ίδιο τρόπο όπως οι άλλες συναρτήσεις, αλλά η κλήση της μοιάζει με εντολή.

Παράδειγμα:
Κατασκευάστε συνάρτηση που θα δέχεται ως είσοδο έναν ακέραιο n και θα εμφανίζει στην οθόνη n φορές τη φράση:
Κατανοώ τη γλώσσα C++

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`void katanoo ( int );`	Πρωτότυπο της συνάρτησης.
`int main ( )`	
`{`	
`  int k;`	
`  cout << "Πόσες φορές ? ";`	Εισάγουμε το πλήθος των φορών που
`  cin >> k;`	θέλουμε να εμφανιστεί η φράση.
`  katanoo(k);`	Κλήση της συνάρτησης.
`}`	
`void katanoo ( int n )`	Επικεφαλίδα της συνάρτησης.
`{`	
`  int i;`	Τοπική μεταβλητή της συνάρτησης.
`  for (i=1; i<=n; ++i)`	Επανάληψη n φορές.
`    cout << "Κατανοώ τη γλώσσα C++\n";`	Εμφάνιση του μηνύματος στην οθόνη.
`}`	

Υπάρχουν συναρτήσεις που εκτελούν μια συγκεκριμένη εργασία χωρίς να δέχονται παραμέτρους.

Παράδειγμα:
Κατασκευάστε συνάρτηση που θα υπολογίζει και θα επιστρέφει το λόγο της χρυσής τομής $(\sqrt{5}-1)/2$.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`double golden ( void );`	Πρωτότυπο συνάρτησης.
`int main ( )`	
`{`	
`  cout << golden() << endl;`	Κλήση της συνάρτησης και εμφάνιση του
`}`	αποτελέσματος στην οθόνη.

`double golden ( )`	Επικεφαλίδα συνάρτησης.
`{`	
`    return (sqrt(5.0)-1)/2;`	Υπολογισμός και επιστροφή του αποτελέσματος.
`}`	

7.2 Παραδείγματα

7.2.1 Ν–παραγοντικό

Θα γράψουμε τη συνάρτηση

```
int factorial ( int n )
```

που θα δέχεται τον ακέραιο n και θα υπολογίζει το $n!$ (δείτε και την παράγραφο 5.5.1).

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int factorial ( int );`	Πρωτότυπο της συνάρτησης.
`int main ( )`	
`{`	
`  int m, f;`	
`  cin >> m;`	
`  f = factorial(m)`	Κλήση της συνάρτησης.
`  cout << f << endl;`	Εμφάνιση του αποτελέσματος.
`}`	
`int factorial ( int n )`	Επικεφαλίδα της συνάρτησης.
`{`	
`  int f, k;`	Τοπικές μεταβλητές της συνάρτησης.
`  f = 1;`	Υπολογισμός του $n!$
`  for ( k=2; k<=n; ++k )`	
`    f *= k;`	
`  return f;`	Επιστροφή του αποτελέσματος.
`}`	

7.2.2 Συνδυασμοί

Το πλήθος των διαφορετικών τρόπων με τους οποίους μπορεί κανείς να διαλέξει k από n αντικείμενα συμβολίζεται με $\begin{pmatrix} n \\ k \end{pmatrix}$ και δίνεται από:

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!\ (n-k)!}$$

Χρησιμοποιώντας τη συνάρτηση `factorial` γράψτε τη συνάρτηση

```
int comb ( int n, int k )
```

που θα υπολογίζει τους συνδυασμούς $\begin{pmatrix} n \\ k \end{pmatrix}$ όταν δίνονται τα n και k αφού πρώτα υπολογίσει τα $n!$, $k!$ και $(n-k)!$

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int factorial ( int );`	Πρωτότυπα των δύο συναρτήσεων.
`int comb ( int, int );`	
`int main ( )`	
`{`	
`  int n, k;`	
`  cin >> n >> k;`	Εισαγωγή των `n` και `k` από το
`  cout << comb(n,k) << endl;`	πληκτρολόγιο.
`}`	
`int comb ( int n, int k )`	Επικεφαλίδα της συνάρτησης.
`{`	
`  int np, kp, nkp;`	Τοπικές μεταβλητές της συνάρτησης.
`  np = factorial(n);`	Υπολογισμός των παραγοντικών με κλήση
`  kp = factorial(k);`	της συνάρτησης `factorial`.
`  nkp = factorial(n-k);`	
`  return np/(kp*nkp);`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος.

Στο πρόγραμμα δεν φαίνεται η συνάρτηση `factorial` η οποία όμως πρέπει να συμπεριληφθεί προκειμένου να έχουμε ένα λειτουργικό πρόγραμμα.

7.2.3 Παράγωγος συνάρτησης

Δίνεται η συνάρτηση

$$f(x) = \frac{x^2 + 2x + 6}{\sqrt{x^2 + 1}}$$

Υπολογίστε μια προσέγγιση της πρώτης παραγώγου, από τον ορισμό

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

χρησιμοποιώντας ένα μικρό h, πχ $h = 10^{-5}x$.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`double f ( double );`	Πρωτότυπα των συναρτήσεων.
`double deriv ( double );`	
`int main ( )`	
`{`	
`    double x;`	
`    cout << "Ποιο x ? ";`	Εισαγωγή του `x` από το πληκτρολόγιο.
`    cin >> x;`	
`    cout << deriv(x) << endl;`	Κλήση της συνάρτησης και εμφάνιση του
`}`	αποτελέσματος στην οθόνη.
`double f ( double x )`	Επικεφαλίδα της συνάρτησης.
`{`	
`    return (x*x+2*x+6)/sqrt(1+x*x);`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος για την $f(x)$
`double deriv ( double x )`	Επικεφαλίδα της συνάρτησης.
`{`	
`    double h;`	Δήλωση τοπικής μεταβλητής.
`    if ( x == 0 )`	
`        h = 1.0e-5;`	
`    else`	
`        h = 1.0e-5*x;`	
`    return (f(x+h)-f(x-h))/(2*h);`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος.

7.2.4 Μέτρο, εσωτερικό γινόμενο και γωνία διανυσμάτων

Κατασκευάστε τη συνάρτηση

```
double gonia ( double x1, double y1, double z1,
               double x2, double y2, double z2 )
```

που θα υπολογίζει τη γωνία (σε μοίρες) μεταξύ δύο διανυσμάτων $\vec{r}_1, \vec{r}_2$ με συντεταγμένες (x_1, y_1, z_1) και (x_2, y_2, z_2). Υπενθυμίζεται ότι:

$$\cos\phi = \frac{\vec{r}_1 \cdot \vec{r}_2}{|\vec{r}_1||\vec{r}_2|} = \frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\,\sqrt{x_2^2 + y_2^2 + z_2^2}}$$

Επιπλέον της ζητούμενης συνάρτηση θα κατασκευάσουμε τις συναρτήσεις:

- `ginomeno` που θα υπολογίζει το εσωτερικό γινόμενο δύο διανυσμάτων.
- `metro` που θα υπολογίζει το μέτρο διανύσματος.
- `moires` που θα μετατρέπει ακτίνια σε μοίρες.

Η μετατροπή από ακτίνια σε μοίρες γίνεται ως:

μοίρες = ακτίνια*$180/\pi$.

Ο κώδικας της συνάρτησης φαίνεται παρακάτω:

Κώδικας C++	*Σχολιασμός*
`double moires ( double rad )`	Επικεφαλίδα της συνάρτησης.
`{`	
`  return rad*180/acos(-1.0);`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος.

Η συνάρτηση για τον υπολογισμό του μέτρου ενός διανύσματος όταν δίνονται οι συντεταγμένες του φαίνεται παρακάτω:

Κώδικας C++	*Σχολιασμός*
`double metro (double x,double y,double z)`	Επικεφαλίδα της συνάρτησης.
`{`	
`  return sqrt(x*x+y*y+z*z);`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος.

Η συνάρτηση για τον υπολογισμό του εσωτερικού γινομένου δύο διανυσμάτων όταν δίνονται οι συντεταγμένες τους φαίνεται παρακάτω:

Κώδικας C++	*Σχολιασμός*
`double ginomeno ( double x1, double y1,`	Επικεφαλίδα της συνάρτησης.
`  double z1, double x2,`	
`  double y2, double z2 )`	
`{`	
`  return x1*x2 + y1*y2 + z1*z2;`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος.

Τέλος δίνεται η ζητούμενη συνάρτηση `gonia` η οποία χρησιμοποιεί τις τρεις προηγούμενες συναρτήσεις.

Κώδικας C++	*Σχολιασμός*
`double gonia ( double x1, double y1,`	Επικεφαλίδα της συνάρτησης.
`  double z1, double x2,`	
`  double y2, double z2 )`	
`{`	
`  double m1, m2;`	Τα μέτρα των διανυσμάτων.
`  double g;`	Το εσωτερικό γινόμενο των διανυσμάτων.
`  double t;`	Το $\cos\phi$.
`  double phi;`	Η γωνία ϕ (σε ακτίνια).
`  double mphi;`	Η γωνία ϕ (σε μοίρες).
`  double pi;`	Ο αριθμός π.
`  pi = acos(-1.0);`	Ο αριθμός π.
`  m1 = metro(x1,y1,z1);`	Το μέτρο του πρώτου διανύσματος.
`  m2 = metro(x2,y2,z2);`	Το μέτρο του δεύτερου διανύσματος.
`  if ( m1*m2 == 0 )`	Αν κάποιο είναι το μηδενικό διάνυσμα.
`  mphi = 0;`	
`  else {`	
`  g = ginomeno(x1,y1,z1,x2,y2,z2);`	Το εσωτερικό γινόμενο των δύο διανυσμάτων.
`  t = g/(m1*m2);`	Υπολογισμός του $\cos\phi$
`  if ( t > 1 )  t = 1;`	
`  if ( t < -1 ) t = -1;`	
`  phi = acos(t);`	Η γωνία σε ακτίνια.
`  mphi = moires(phi);`	Μετατροπή της γωνίας σε μοίρες.
`  }`	
`  return mphi;`	Επιστροφή του αποτελέσματος.
`}`	

Οι παρακάτω γραμμές

```
if ( t > 1 )  t = 1;
if ( t < -1 ) t = -1;
```

είναι απαραίτητες διότι κατά τον υπολογισμό του $\cos\phi$ (δηλαδή της μεταβλητής `t`) είναι πιθανό λόγω αριθμητικών σφαλμάτων να προκύψει αριθμός εκτός του διαστήματος $[-1, 1]$ (πχ 1.0000000000000001). Αυτό θα προκαλέσει σφάλμα κατά την εκτέλεση της συνάρτησης `acos(t)`.

Ένα πρόγραμμα που χρησιμοποιεί τη συνάρτηση `gonia` δίνεται στη συνέχεια.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`double moires ( double );`	Τα πρωτότυπα των συναρτήσεων.
`double metro ( double,double,double );`	
`double ginomeno ( double,double,double,`	
`  double,double,double );`	
`double gonia ( double,double,double,`	
`  double,double,double );`	
`int main ( )`	
`{`	
`  double x1, y1, z1;`	Δήλωση τοπικών μεταβλητών του κυρίως
`  double x2, y2, z2;`	προγράμματος.
`  cin >> x1 >> y1 >> z1;`	Εισαγωγή των συντεταγμένων των δύο
`  cin >> x2 >> y2 >> z2;`	διανυσμάτων από το πληκτρολόγιο.
`  cout << "Η γωνία είναι "`	
`  << gonia(x1,y1,z1,x2,y2,z2)`	Κλήση της συνάρτησης και εμφάνιση των
`  << endl;`	αποτελεσμάτων.
`}`	

7.3 Τοπικές και καθολικές μεταβλητές

Κάθε συνάρτηση έχει τις δικές της μεταβλητές (*τοπικές μεταβλητές*). Οι τοπικές μεταβλητές ισχύουν μόνο μέσα στη συνάρτηση στην οποία δηλώθηκαν και για όσο λειτουργεί η συνάρτηση. Όταν η συνάρτηση τερματίσει τη λειτουργία της οι τοπικές μεταβλητές καταστρέφονται και χάνουν τα περιεχόμενά τους. Μπορούμε να δηλώσουμε μεταβλητές που δεν ανήκουν σε καμία συνάρτηση (*καθολικές μεταβλητές*). Οι καθολικές μεταβλητές μπορούν να χρησιμοποιηθούν από όλες τις συναρτήσεις και τα περιεχόμενά τους παραμένουν καθ' όλη τη διάρκεια του προγράμματος. Οι καθολικές μεταβλητές δηλώνονται εκτός οποιασδήποτε συνάρτησης.

Παράδειγμα:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`double x, y, z;`	Δήλωση καθολικών μεταβλητών.
`int main ( )`	
`{`	
`x = 15;`	Ανάθεση τιμής στην καθολική μεταβλητή.
⋮	
`}`	
`double add ( double a, double b )`	
`{`	
⋮	
`k = (a+b)/x;`	Χρήση της καθολικής μεταβλητής.
⋮	
`}`	

Εάν υπάρχει σε μια συνάρτηση δηλωμένη τοπική μεταβλητή με το ίδιο όνομα με κάποια καθολική μεταβλητή, τότε στη συνάρτηση αυτή χρησιμοποιείται η τοπική και όχι η καθολική μεταβλητή. Κατά συνέπεια η συνάρτηση αυτή δεν μπορεί να έχει πρόσβαση στην αντίστοιχη καθολική μεταβλητή. Με τη χρήση καθολικών μεταβλητών μπορούμε να μεταβιβάσουμε τιμές στις συναρτήσεις, κάτι όμως που αν γίνεται συστηματικά οδηγεί σε δυσνόητα προγράμματα με δυσκολία στην εκσφαλμάτωση.

7.4 Αναδρομικότητα

Μια συνάρτηση μπορεί να καλέσει το εαυτό της. Όταν συμβαίνει αυτό έχουμε μια *αναδρομική κλήση* της συνάρτησης. Αναδρομική κλήση επίσης μπορούμε να έχουμε όταν μια συνάρτηση καλέσει τον εαυτό της μέσω κάποιας άλλης συνάρτησης.

Παράδειγμα:

Η συνάρτηση a καλεί την b
Η συνάρτηση b καλεί την c
Η συνάρτηση c καλεί την a

Στην πραγματικότητα στις αναδρομικές κλήσεις δεν καλείται πάλι η ίδια συνάρτηση, αλλά ένα δεύτερο αντίγραφό της, το οποίο δημιουργείται αυτόματα όταν χρειαστεί.

7.4.1 Παραδείγματα

7.4.1.1 N–παραγοντικό με αναδρομικό τρόπο

Θα φτιάξουμε τη συνάρτηση

```
int factorial ( int n )
```

που θα για δεδομένο n θα υπολογίζει το $n!$ Ο μη αναδρομικός τρόπος έχει ήδη παρουσιαστεί στην παράγραφο 7.2.1. Για να φτιάξουμε την αναδρομική εκδοχή της συνάρτησης παρατηρούμε ότι
$n! = n(n-1)!$ για $n > 1$.

Κώδικας C++	*Σχολιασμός*
`int factorial ( int n )`	Επικεφαλίδα της συνάρτησης.
`{`	
`  if ( n > 1 )`	
`  return n*factorial(n-1);`	Αναδρομική κλήση της συνάρτησης για τον υπολογισμό του $(n-1)!$.
`  else`	
`  return 1;`	Σε περίπτωση που $n = 1$.
`}`	

7.5 Υπερφόρτωση συναρτήσεων

Η C++ επιτρέπει τον ορισμό συναρτήσεων με το ίδιο όνομα, αλλά διαφορετικές παραμέτρους και λειτουργία, στο ίδιο πρόγραμμα. Η τεχνική αυτή ονομάζεται *υπερφόρτωση συναρτήσεων*. Για παράδειγμα οι γνωστές μαθηματικές συναρτήσεις (`sqrt`, `sin`, `cos` κλπ.) είναι ήδη υπερφορτωμένες αφού μπορούν να κληθούν με ορίσματα τύπου `float` ή `double`. Τα αντίστοιχα πρωτότυπα για την `sqrt` θα ήταν:

```
float sqrt ( float ) ;
double sqrt ( double ) ;
```

Κατά την κλήση της συνάρτησης, ανάλογα με τον τύπο των ορισμάτων ο μεταφραστής αποφασίζει ποια από τις διαφορετικές εκδοχές της συνάρτησης θα καλέσει στην πραγματικότητα.

7.5.1 Παραδείγματα

7.5.1.1 Υπόλοιπο διαίρεσης

Θα κατασκευάσουμε συνάρτηση η οποία θα υπολογίζει το υπόλοιπο της διαίρεσης του a από τον b. Η συνάρτηση θα πρέπει να καλείται είτε με ακέραια ορίσματα ή με πραγματικούς διπλής ακρίβειας. Γι αυτό θα γράψουμε δύο διαφορετικές συναρτήσεις με το ίδιο όνομα (`mod`) οι οποίες όμως θα δέχονται διαφορετικούς τύπους παραμέτρων.

Για την περίπτωση ακεραίων:
Θα εκμεταλλευτούμε το γεγονός ότι για το υπόλοιπο διαίρεσης ακέραιων αριθμών υπάρχει ενσωματωμένος στη γλώσσα C++ ο τελεστής **%**

Κώδικας C++	*Σχολιασμός*
`int mod ( int a, int b )`	Επικεφαλίδα της συνάρτησης.
`{`	
`    return a % b;`	Υπολογισμός και επιστροφή του
`}`	αποτελέσματος.

Για την περίπτωση πραγματικών διπλής ακρίβειας:
Εδώ δεν υπάρχει αντίστοιχος τελεστής, οπότε θα πρέπει να κάνουμε τη διαίρεση και να βρούμε το υπόλοιπο. Σημειώστε ότι για να βρούμε το ακέραιο μέρος της διαίρεσης a/b χρησιμοποιούμε τον τελεστή αλλαγής τύπου `(int)`.

Κώδικας C++	*Σχολιασμός*
`double mod ( double a, double b )`	Επικεφαλίδα της συνάρτησης.
`{`	
`    int p;`	Δήλωση τοπικών μεταβλητών.
`    double y;`	
`    p = (int) (a/b);`	Το ακέραιο μέρος της διαίρεσης a/b.

```
    y = a - p*b;                    Υπολογισμός του υπολοίπου.
    return y;                       Επιστροφή του αποτελέσματος.
}
```

Το κυρίως πρόγραμμα:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int mod ( int, int );`	Τα πρωτότυπα για τις δύο εκδοχές της
`double mod ( double, double );`	συνάρτησης.
`int main ( )`	
`{`	
`    int k;`	
`    double d;`	
`    k = mod(25,7);`	Κλήση της συνάρτησης με ακέραιους.
`    cout << k << endl;`	
`    d = mod(7.4,2.2);`	Κλήση της συνάρτησης με πραγματικούς.
`    cout << d << endl;`	
`}`	

Κεφάλαιο 8

Δείκτες

Μέχρι εδώ έχουμε δει ότι η δημιουργία και χρησιμοποίηση μιας μεταβλητής σε πρόγραμμα προϋποθέτει την διευκρίνηση 2 ιδιοτήτων: τον τύπο της μεταβλητής και την αριθμητική τιμή της μεταβλητής. Το πρώτο γίνεται με εντολή δήλωσης, ενώ το δεύτερο με εντολή ανάθεσης. Στο παρασκήνιο αυτών των διεργασιών, βέβαια, ο μεταφραστής δεσμεύει τον ανάλογο χώρο (θέση) στην μνήμη του υπολογιστή, δηλαδή 4 bytes για ακέραιο ή 8 bytes για ρητό διπλής ακρίβειας. Η θέση αυτή στην μνήμη είναι συγκεκριμένη (δεν αλλάζει κατά την διάρκεια εκτέλεσης του προγράμματος) και αποτελεί την διεύθυνση της μεταβλητής. Έτσι λοιπόν, στην γενικότερη περίπτωση μια μεταβλητή αποτελείται/χαρακτηρίζεται από 3 ιδιότητες: διεύθυνση, τύπο, και αριθμητική τιμή. Σε όσα παραδείγματα έχουμε δει μέχρι τώρα η πρώτη ιδιότητα (διεύθυνση στην μνήμη) παρέμενε κρυφή από τον προγραμματιστή/χρήστη. Όμως, η C++ μας δίνει την δυνατότητα να χρησιμοποιήσουμε την διεύθυνση μιας μεταβλητής εάν αυτό βοηθάει στην ευκολότερη ανάπτυξη ή διαχείριση ενός προγράμματος. Αυτό γίνεται μέσω μιας νέας μεταβλητής που ονομάζεται *δείκτης*. Θα αναπτύξουμε πρώτα το τι είναι δείκτης, και κατόπιν (και σε επόμενα κεφάλαια) θα δούμε ποια είναι η χρησιμότητά του.

8.1 Η έννοια του δείκτη

Εάν είναι λοιπόν να χρησιμοποιήσουμε την διεύθυνση μιας μεταβλητής σε ένα πρόγραμμα, θα πρέπει καταρχήν να βρούμε ποια είναι, και κατά δεύτερον να βρούμε έναν τρόπο να μπορούμε να την ανακαλούμε (θυμόμαστε) όποτε χρειάζεται. Υπενθυμίζουμε εδώ, ότι το ποια διεύθυνση θα καταλάβει στην μνήμη μια μεταβλητή δεν καθορίζεται από τον προγραμματιστή, αλλά από τον μεταφραστή (compiler) την ώρα που εκτελείται η εντολή της δήλωσης. Άρα, το ζητούμενο είναι: εάν δηλώσουμε μια μεταβλητή, πως μετά θα βρούμε σε ποια θέση την έβαλε ο μεταφραστής, καθώς και που θα αποθηκεύσουμε αυτή την πληροφορία. Για τον σκοπό αυτό η C++ μας δίνει μια νέα μορφή μεταβλητής, τον λεγόμενο δείκτη.

Ο δείκτης είναι μια μεταβλητή που οι τιμές που παίρνει δεν είναι αριθμητικές, αλλά διευθύνσεις μνήμης. Κατά κάποιον τρόπο δηλαδή, ένας δείκτης "δείχνει" σε μια διεύθυνση μνήμης. Είναι προφανές ότι το όλο εγχείρημα έχει νόημα μόνο εάν ο δείκτης δείχνει σε κατειλημμένη θέση, δηλαδή σε θέση (διεύθυνση) όπου "κατοικεί" κάποια μεταβλητή του προγράμματος.

Όπως είπαμε και πριν, ένας δείκτης είναι μεν μια νέα μεταβλητή, αλλά δεν παύει να είναι μεταβλητή. Και όπως μια απλή μεταβλητή, έτσι και ένας δείκτης χαρακτηρίζεται από τον τύπο του, την τιμή του, και από την διεύθυνσή του (όχι της μεταβλητής στην οποία "δείχνει", αλλά την δικιά του θέση στην μνήμη). Καθώς το τελευταίο όμως είναι ικανό να προκαλέσει αρκετή δυσκολία κατανόησης, προς το παρόν δεν θα ασχοληθούμε με αυτό. Άρα μας ενδιαφέρουν αποκλειστικά ο τύπος και η τιμή του.

Ο τύπος ενός δείκτη δεν είναι τίποτα άλλο από τον τύπο της μεταβλητής στην οποία δείχνει. Για παράδειγμα, εάν ο δείκτης "δείχνει" σε μια ακέραια μεταβλητή, τότε είναι δείκτης σε ακέραιο, δηλαδή ο τύπος του είναι `int`. Η τιμή του προφανώς, είναι μια διεύθυνση στην μνήμη.

8.2 Δήλωση δείκτη

Η δήλωση ενός δείκτη γίνεται με τον ίδιο τρόπο που γίνεται και η δήλωση μιας οποιαδήποτε άλλης μεταβλητής, εκτός του ότι χρησιμοποιείται το ειδικό σύμβολο (τελεστής) αστέρι `*` (είναι ατυχής η σύμπτωση του τελεστή αυτού με τον τελεστή του πολλαπλασιασμού, οπότε προσοχή χρειάζεται να μην μπερδευτούν οι δύο έννοιες). Το αστέρι μπαίνει μπροστά από τον δείκτη. Ο τύπος του δείκτη είναι ο τύπος της μεταβλητής στην οποία προορίζουμε τον δείκτη να δείχνει.

Παράδειγμα:
Σε απόσπασμα προγράμματος, δηλώστε δύο ακέραιες μεταβλητές, καθώς και δύο δείκτες σε ακέραιες μεταβλητές.

`int n, m;`	Δύο ακέραιες μεταβλητές, `n` και `m`.
`int *k, *j;`	Δύο δείκτες σε ακέραιους, `k` και `j`.

Παρομοίως και για ρητούς. Στο παραπάνω παράδειγμα ορίσαμε δύο δείκτες `k` και `j`. Με την δήλωση του τύπου τους ως `int` προεξοφλούμε ότι δείχνουν σε ακέραιο, αλλά δεν έχουμε ακόμα πει σε ποιόν ακέραιο δείχνουν. Αυτό γίνεται με την ανάθεση τιμών σε δείκτη.

8.3 Ανάθεση τιμών σε δείκτη

Σε έναν δείκτη αναθέτουμε τη διεύθυνση μιας μεταβλητής. Για να μπορέσει να γίνει αυτό όμως, χρειαζόμαστε μια πράξη που να "εξάγει" τη διεύθυνση μιας μεταβλητής. Στην C++ υπάρχει ένας ειδικός τελεστής το `&` ο οποίος κάνει ακριβώς αυτή τη δουλειά: εξάγει την διεύθυνση μιας μεταβλητής. Ο τρόπος που το χρησιμοποιούμε είναι:

διεύθυνση_μεταβλητής = `&` *μεταβλητή* ;

Παράδειγμα:
Στο προηγούμενο παράδειγμα, έστω ότι θέλουμε ο δείκτης `k` να δείχνει στον ακέραιο `n`, και ο δείκτης `j` να δείχνει στον ακέραιο `m`. Το αντίστοιχο απόσπασμα προγράμματος:

`k = & n;`	Αναθέτουμε στον δείκτη `k` την διεύθυνση της μεταβλητής `n`
`j = & m;`	Αναθέτουμε στον δείκτη `j` την διεύθυνση της μεταβλητής `m`

Το ίδιο ισχύει και εάν οι μεταβλητές και δείκτες ήταν ρητοί. Χρειάζεται μεγάλη προσοχή λοιπόν να αναθέτουμε ακέραιους δείκτες σε ακέραιες μεταβλητές, και ρητούς δείκτες σε ρητές μεταβλητές.

8.4 Πράξεις με δείκτες

Αναφέραμε τους δύο τελεστές για δείκτες, το * και το &. Ο τελεστής *, πέρα από τη λειτουργία του για την δήλωση ενός δείκτη, έχει και μια δεύτερη λειτουργία, η οποία είναι να εξάγει, ή/και τροποποιεί, την αριθμητική τιμή της μεταβλητής στην οποία δείχνει ο δείκτης.

Παράδειγμα:

```
int n;          Ακέραια μεταβλητή n.
int *k;         Δείκτης k σε ακέραιο.

n = 5;          Αναθέτουμε στην μεταβλητή n την τιμή 5.
k = & n;        Αναθέτουμε στον δείκτη k την διεύθυνση της μεταβλητής n.

cout << k;      Εκτύπωση της διεύθυνσης της μεταβλητής n.
cout << *k;     Εκτύπωση της τιμής της μεταβλητής n.
```

Στο παραπάνω, η πρώτη εντολή εκτύπωσης θα έχει ως αποτέλεσμα την εμφάνιση μιας σειράς ψηφίων (αλφαριθμητικών) που αντιστοιχούν στην διεύθυνση στη μνήμη του n, ενώ η δεύτερη εντολή εκτύπωσης θα έχει ως αποτέλεσμα την εμφάνιση της τιμής του n, δηλαδή το 5.

Έστω τώρα ότι θέλουμε να αλλάξουμε την τιμή της μεταβλητής n από 5 σε 7. Υπάρχουν δύο ισοδύναμοι τρόποι για να γίνει αυτό. Ο πρώτος και γνωστός τρόπος είναι η απευθείας ανάθεση στην μεταβλητή n

```
n = 7;          Αναθέτουμε στην μεταβλητή n την τιμή 7.
```

Ο δεύτερος τρόπος είναι μέσω της διεύθυνσης της μεταβλητής:

```
*k = 7;         Αναθέτουμε στην μεταβλητή που δείχνει ο k την τιμή 7.
```

Η τελευταία έκφραση μεταφράζεται ως εξής: πήγαινε στην θέση μνήμης που δείχνει ο k και άλλαξε την τιμή εκεί σε 7. Οι παραπάνω δύο εκφράσεις είναι εντελώς ισοδύναμες, ουσιαστικά ο μεταφραστής κάνει την ίδια πράξη και στις δύο εκφράσεις.

Σαν σύνοψη, έχουμε δύο τελεστές για δείκτες:
- & : δίνει την διεύθυνση μιας μεταβλητής
- * : δίνει την τιμή στην διεύθυνση που δείχνει ο δείκτης

Οι προτεραιότητες των τελεστών δίνονται στο παράρτημα Α.

8.5 Παραδείγματα

8.5.1 Διάταξη μεταβλητών και δεικτών στην μνήμη

Έστω τρεις ακέραιες μεταβλητές και δύο δείκτες σε ακέραιους, όπως δηλώνονται στο επόμενο απόσπασμα προγράμματος:

```
int n1, n2, n3;     Ακέραιες μεταβλητές n1, n2, n3.
int *k1, *k2;       Δείκτες σε ακέραιους k1, k2.
k1 = &n1;           Ο δείκτης k1 να πάρει την διεύθυνση του n1.
```

`k2 = &n2;`	Ο δείκτης `k2` να πάρει την διεύθυνση του `n2`.
`n1 = 5;`	Η τιμή του `n1` να είναι 5.
`*k2 = 10;`	Εκεί που δείχνει ο `k2`, η τιμή να γίνει 10.
`n3 = 15;`	Η τιμή του `n3` να γίνει 15.

Το παρακάτω σχήμα δείχνει πως διατάσσονται τα παραπάνω στην μνήμη, και τι τιμές παίρνουν:

θέση μνήμης	A1	A2	A3	A4	A5
όνομα μεταβλητής	`n1`	`n2`	`n3`	`k1`	`k2`
τιμή μεταβλητής	5	10	15	A1	A2

Έστω ότι τώρα εκτελούμε τις εντολές:

`*k1 = 1;`	Εκεί που δείχνει ο `k1`, η τιμή να γίνει 1.
`*k2 = 2;`	Εκεί που δείχνει ο `k2`, η τιμή να γίνει 2.

Η νέα διάταξη στην μνήμη είναι:

θέση μνήμης	A1	A2	A3	A4	A5
όνομα μεταβλητής	`n1`	`n2`	`n3`	`k1`	`k2`
τιμή μεταβλητής	1	2	15	A1	A2

Ενώ εάν εκτελεστεί το παρακάτω:

`k1 = &n3;`	Ο δείκτης `k1` να πάρει την διεύθυνση του `n3`.
`*k1 = 11;`	Εκεί που δείχνει ο `k1`, η τιμή να γίνει 11.

η νέα διάταξη στην μνήμη είναι:

θέση μνήμης	A1	A2	A3	A4	A5
όνομα μεταβλητής	`n1`	`n2`	`n3`	`k1`	`k2`
τιμή μεταβλητής	1	2	11	A3	A2

8.5.2 Σωστές και λάθος εκφράσεις με δείκτες

Κώδικας C++	*Σχολιασμός*	*Τιμές* a1	a2
`#include <iostream>`			
`using namespace std;`			
`int main ( ) {`			
`  double a1, a2;`			
`  double *b1, *b2;`			
`  b1 = &a1;`	Σωστή	-	-
`  b2 = &a2;`	Σωστή	-	-
`  *a1 = 33;`	Λάθος: ο a1 δεν είναι δείκτης.		
`  a1 = 22;`	Σωστή	22	-
`  a2 = 20;`	Σωστή	22	20
`  b2 = *a1;`	Λάθος: το *a1 δεν έχει νόημα.		
`  b2 = a1;`	Λάθος: ο b2 είναι δείκτης, ο a1 μεταβλητή.		
`  b2 = 40;`	Λάθος: ο b2 είναι δείκτης, όχι μεταβλητή.		
`  b2 = b1;`	Σωστή	22	20
`  *b2 = 40;`	Σωστή	40	20
`  *b1 = 30;`	Σωστή	30	20
`  a2 = *b1;`	Σωστή	30	30
`}`			

Ένα συνηθισμένο λάθος είναι όταν ο δείκτης δεν δείχνει πουθενά (δεν του έχει γίνει κάποια ανάθεση διεύθυνσης) και παρ' όλα αυτά μέσα στο πρόγραμμα επιχειρείται να αλλάξουμε τιμή στην διεύθυνση που δείχνει. Εάν όμως δεν έχει γίνει ανάθεση διεύθυνσης ακόμα, τότε ποια τιμή θα αλλάξει; Η απάντηση είναι ότι κάποια τυχαία θέση στην μνήμη θα αλλάξει, η οποία πράξη μπορεί όμως μπορεί και να προκαλέσει κατάρρευση του προγράμματος. Άρα, θέλει προσοχή, ένας δείκτης πρέπει πρώτα να δηλωθεί σε ποια διεύθυνση αναφέρεται, και μετά να επιχειρηθεί η οποιαδήποτε τροποποίηση τιμής εκεί.

8.6 Δείκτες και πίνακες

Η σχέση μεταξύ δεικτών και πινάκων είναι πολύ στενή. Για να κατανοήσουμε την σχέση αυτή, ας δούμε κατ' αρχήν τι είναι ένας πίνακας: μια ομαδοποίηση μεταβλητών κάτω από ένα κοινό όνομα, με την επιβολή ότι θα καταλαμβάνουν διαδοχικές θέσεις στην μνήμη. Το τελευταίο είναι και η πιο σημαντική ιδιότητα του πίνακα: ο μεταφραστής χρειάζεται μόνο να ξέρει σε ποια θέση μνήμης βρίσκεται το πρώτο στοιχείο του πίνακα. Κατόπιν μπορεί να βρει όλα τα υπόλοιπα με προσπέλαση. Κατά κάποια έννοια δηλαδή, το μόνο που χρειαζόμαστε να γνωρίζουμε για έναν πίνακα, είναι η θέση στην μνήμη του πρώτου στοιχείου του. Η C++ το κάνει αυτό πιο εύκολο, με το να θεωρεί το ίδιο το όνομα του πίνακα (χωρίς τις αγκύλες) ως δείκτη στο πρώτο στοιχείο του πίνακα.

Παράδειγμα:

```
int n[4];
int *k;
for (i=0; i<4; i++) n[i] = i*10;
k = n;          Σωστή, γιατί ο n χωρίς τις αγκύλες είναι δείκτης.
```

Το αποτέλεσμα των παραπάνω είναι η παρακάτω διάταξη στην μνήμη:

θέση μνήμης	A1	A2	A3	A4	A5
όνομα μεταβλητής	`n[0]`	`n[1]`	`n[2]`	`n[3]`	`k`
τιμή μεταβλητής	0	10	20	30	A1

Η τελευταία εντολή είναι εντελώς ισοδύναμη με την:

```
k = &n[0];
```

όπου στον `k` ανατίθεται η διεύθυνση του πρώτου στοιχείου του πίνακα `n`, δηλαδή το `n` και το `&n[0]` είναι το ίδιο ακριβώς.

Μια πολύ ενδιαφέρουσα ιδιότητα των δεικτών εδώ είναι ότι μπορούμε να κάνουμε αριθμητική με τους δείκτες, χρήσιμο για την προσπέλαση ενός πίνακα. Όπως είδαμε παραπάνω, ο `k` δείχνει στο πρώτο στοιχείο του πίνακα `n`. Υπάρχει κάποια έννοια στην έκφραση `k+1;` ή `k+2;` Η απάντηση είναι ναι: ο μεταφραστής αναγνωρίζει ότι εάν ο `k` δείχνει σε κάποιον ακέραιο κάπου στην μνήμη, ο `k+1` δείχνει στον επόμενο ακέραιο (4 bytes παρακάτω), ο `k+2` στον μεθεπόμενο (8 bytes παρακάτω) κ.ο.κ. (εδώ καταλαβαίνουμε γιατί είναι σημαντικό η δήλωση ενός δείκτη να γίνεται με τον σωστό τύπο, δηλώνει πόσα bytes καλύπτει η μεταβλητή). Όμως ο επόμενος ακέραιος είναι το στοιχείο `n[1]`, ο μεθεπόμενος είναι ο `n[2]`, κ.ο.κ. Στην ουσία, μπορούμε να χρησιμοποιήσουμε ένα δείκτη σαν να ήταν και ο ίδιος πίνακας.

Παράδειγμα:

```
int m;
m = n[2];       Ανάθεση στον m του τρίτου στοιχείου του πίνακα n.
m = *(k+2);     Ανάθεση στον m της τιμής που εμφανίζεται στην μνήμη δύο θέσεις μετά.
                Την αρχή του πίνακα n.
m = k[2];       Ανάθεση στον m του τρίτου στοιχείου του πίνακα k.
```

Και οι τρεις παραπάνω εκφράσεις είναι τελείως ισοδύναμες. Οι παρενθέσεις στο `*(k+2)` είναι απαραίτητες καθώς ο `*` έχει μεγαλύτερη προτεραιότητα από το `+`. Εάν δεν τις βάλουμε παίρνουμε το `*k+2` που είναι το `n[0]+2` και όχι το `n[2]`. Επίσης, μπορούμε να βάλουμε αγκύλες σε έναν δείκτη σαν να ήταν πίνακας. Αυτό δείχνει την απόλυτα στενή σχέση μεταξύ πινάκων και δεικτών.

Η C++ λοιπόν επιτρέπει την προσπέλαση των στοιχείων ενός πίνακα με δύο τρόπους:

- την δεικτοδότηση του πίνακα (πχ `n[2]`)
- την αριθμητική δεικτών (πχ `*(k+2)`).

Πολλές φορές η χρησιμοποίηση αριθμητικής δεικτών επιτρέπει την πιο γρήγορη εκτέλεση ενός προγράμματος. Επίσης η παρακάτω πράξη είναι επιτρεπτή:

```
k = k + 1;
```

Τώρα ο k δείχνει στο δεύτερο στοιχείο του πίνακα n, και η διάταξη στη μνήμη είναι:

θέση μνήμης	A1	A2	A3	A4	A5
όνομα μεταβλητής	n[0]	n[1]	n[2]	n[3]	k
τιμή μεταβλητής	0	10	20	30	A2

Τώρα όμως οι παρακάτω εκφράσεις δεν είναι ισοδύναμες:

`m = n[2];`	Ανάθεση στον m του τρίτου στοιχείου του πίνακα n.
`m = k[2];`	Ανάθεση στον m του τρίτου στοιχείου του πίνακα k.

καθώς εάν ο k δείχνει το δεύτερο στοιχείο του n, ο k[2] δείχνει στο τέταρτο στοιχείο του n, δηλαδή το n[3]. Προφανώς οι εκφράσεις m=k[2] και m=*(k+2) παραμένουν ισοδύναμες.

8.7 Παραδείγματα

8.7.1 Ανάθεση τιμών πίνακα χρησιμοποιώντας αποκλειστικά αριθμητική δεικτών

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`int main ( ) {`	
`int a[100];`	Δήλωση πίνακα ακεραίων.
`int *b;`	Δήλωση δείκτη σε ακέραιο.
`b=a;`	Ανάθεση στον b της διεύθυνσης του πρώτου στοιχείου του πίνακα a.
`for (int i=0; i<100; i++) {`	
`*b = i;`	Στην θέση που δείχνει ο b αντικαθιστούμε την τιμή του i.
`b++;`	Μετακινούμε τον δείκτη b στην επόμενη θέση.
`}`	
`}`	

Κεφάλαιο 9

Συναρτήσεις II

Στην πρώτη ματιά που ρίξαμε πάνω στις συναρτήσεις στο κεφάλαιο 7 είδαμε την κλήση κατ' αξία, δηλαδή στην συνάρτηση μεταφέρονται, μέσω της λίστας εισόδου, μόνο οι τιμές των μεταβλητών. Αυτό δημιουργεί δύο περιορισμούς. Πρώτον, δεν μπορούμε να αλλάξουμε την τιμή μιας μεταβλητής εισόδου, και δεύτερον, δεν μπορούμε να εισάγουμε πίνακες στην συνάρτηση. Το πρώτο είναι εύκολο να αντιληφθούμε γιατί γίνεται: κατά την κλήση της συνάρτησης δημιουργούνται αντίγραφα των μεταβλητών εισόδου, τα οποίο είναι και αυτά που εισέρχονται μέσα στην συνάρτηση. Όταν η συνάρτηση επιστρέφει στο κυρίως πρόγραμμα όμως, όλα τα αντίγραφα καταστρέφονται. Έτσι, οποιαδήποτε αλλαγή έγινε στην αριθμητική τους τιμή, χάνεται. Η αναγκαία συνθήκη ώστε αυτό να μην συμβαίνει είναι να περνάμε στην συνάρτηση τις ίδιες τις μεταβλητές και όχι αντίγραφά τους, δηλαδή μέσα στην συνάρτηση να υπάρχει η διεύθυνση μνήμης των αρχικών μεταβλητών. Αυτό γίνεται με *κλήση κατ' αναφορά*, δηλαδή μέσω δεικτών. Αντί να εισάγουμε τις μεταβλητές στην λίστα εισόδου, εισάγουμε τις διευθύνσεις τους (ή με άλλα λόγια δείκτες προς αυτές). Παρομοίως και για τους πίνακες, η C++ επιβάλλει να περάσουμε μέσω της λίστας εισόδου μόνο την διεύθυνση μνήμης του πρώτου στοιχείου του πίνακα (αλλιώς θα έπρεπε να κάνει ένα πλήρες αντίγραφο όλου του πίνακα κάθε φορά που καλείται η συνάρτηση, το οποίο δεν είναι καθόλου πρακτικό). Άρα όταν θέλουμε να περάσουμε έναν πίνακα σε συνάρτηση, ουσιαστικά περνάμε πάλι έναν δείκτη. Κατόπιν φυσικά, μέσα στην συνάρτηση, η προσπέλαση των στοιχείων του πίνακα γίνεται όπως και στο κυρίως πρόγραμμα.

9.1 Δήλωση και κλήση συνάρτησης με δείκτες σε μεταβλητές

Κατά τη δήλωση της συνάρτησης πρέπει να οριστεί εάν κάποιες από τις μεταβλητές εισόδου είναι δείκτες. Αυτό γίνεται όπως και στο κυρίως πρόγραμμα, με την τελεστή *. Δηλαδή:

τύπος_αποτελέσματος *όνομα_συνάρτησης* (*τύπος* ***μεταβλητή***, *τύπος* ***μεταβλητή***, ...)

Μέσα στην συνάρτηση οι πράξεις όσον αφορά τους δείκτες γίνονται όπως είδαμε και στο προηγούμενο κεφάλαιο.

9.2 Παραδείγματα

9.2.1 Συνάρτηση που μετατρέπει μια μεταβλητή στο τετράγωνό της

Έστω συνάρτηση πού δέχεται έναν ρητό και υψώνει την τιμή του στο τετράγωνο. Καθώς το αποτέλεσμα γυρνάει μέσω της ίδιας της μεταβλητής εισόδου και δεν χρειάζεται επιπλέον επιστροφή, η συνάρτησή μας θα είναι τύπου `void`.

Κώδικας C++	*Σχολιασμός*
`void square ( double *x ) {`	Ο τύπος της συνάρτησης είναι void καθώς δεν χρειάζεται να γυρίσει αποτέλεσμα.
`    (*x) = (*x) * (*x);`	Πράξεις γίνονται πάνω στην αριθμητική τιμή
`}`	στη θέση μνήμης που δείχνει ο δείκτης.

Παρατηρήστε την χρήση των παρενθέσεων. Δεν είναι απαραίτητες (ο μεταφραστής δεν θα έμπλεκε τις προτεραιότητες και τη σημασία του κάθε αστερίσκου), αλλά βοηθάνε στην καθαρότητα και διαύγεια του προγράμματος.

Το πρωτότυπο της συνάρτησης δηλώνεται ως:

```
void square ( double * );
```

Για την κλήση της παραπάνω συνάρτησης θυμόμαστε ότι πρέπει να περάσουμε τη διεύθυνση μιας μεταβλητής (ή έναν δείκτη σε αυτή). Για παράδειγμα, ένα πρόγραμμα που ζητάει από τον χρήστη έναν ρητό και εκτυπώνει το τετράγωνό του, χρησιμοποιώντας την συνάρτηση `square`.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`void square ( double * );`	Πρωτότυπο συνάρτησης.
`int main ( ) {`	
`    double x;`	
`    cout << "Δώσε ένα ρητό ";`	
`    cin >> x;`	
`    square (&x);`	Κλήση συνάρτησης.
`    cout << "Το τετράγωνό του είναι "`	
`         << x << endl;`	
`}`	

Η κλήση της συνάρτησης έγινε με την διεύθυνση της μεταβλητής `x`, δηλαδή `square(&x);`

Πλήρως ισοδύναμη είναι η παρακάτω έκφραση:

`double *y;`	Το `y` είναι δείκτης.
`y = &x;`	Το `y` δείχνει στην διεύθυνση του `x`.

```
square (y);              Περνάμε στην συνάρτηση το y, δηλαδή τη διεύθυνση του x.
cout << "Το τετράγωνό του είναι " << x << endl;
```

9.2.2 Μετατροπή καρτεσιανών σε πολικές συντεταγμένες

Θα γράψουμε συνάρτηση που να δέχεται τις πολικές συντεταγμένες ρ και θ ενός διδιάστατου διανύσματος, και να τις μετατρέπει σε καρτεσιανές συντεταγμένες. Δεδομένου της ακτίνας ρ και γωνίας θ ενός διδιάστατου διανύσματος σε πολικές συντεταγμένες, οι καρτεσιανές βρίσκονται από τις σχέσεις:

$$x = \rho\cos\theta \qquad y = \rho\sin\theta$$

Κώδικας C++	*Σχολιασμός*
`void polar_to_cartesian(double *r,`	Στην είσοδο έχουμε έναν δείκτη στην ακτίνα ρ
`  double *t){`	και ένα δείκτη στην γωνία θ.
`  double x = (*r) * cos(*t);`	Δήλωση τοπικής μεταβλητής x.
`  double y = (*r) * sin(*t);`	Δήλωση τοπικής μεταβλητής y.
`  *r = x;`	Ανταλλαγή του ρ με το x.
`  *t = y;`	Ανταλλαγή του θ με το y.
`}`	

9.2.3 Ανταλλαγή δύο αριθμών μεταξύ τους

Θα γράψουμε συνάρτηση που ανταλλάσσει την τιμή δύο ρητών μεταξύ τους. Αυτή η συνάρτηση θα φανεί χρήσιμη όταν στο επόμενο μέρος γράψουμε την συνάρτηση για ταξινόμηση ενός πίνακα.

Κώδικας C++	*Σχολιασμός*
`void swap ( double *x, double *y ) {`	
`  double z = *x ;`	Προσωρινή μεταβλητή z παίρνει το x.
`  *x = *y ;`	Νέα τιμή του x η παλιά τιμή του y.
`  *y = z ;`	Νέα τιμή του y η παλιά τιμή του x.
`}`	

9.3 Δήλωση και κλήση συνάρτησης με δείκτες σε πίνακες

Όταν θέλουμε να περάσουμε ένα πίνακα σε μια συνάρτηση, αρκεί να περάσουμε τη διεύθυνση του πρώτου του στοιχείου. Κατόπιν η προσπέλαση των στοιχείων του πίνακα γίνεται απλά όπως και στο κυρίως πρόγραμμα, πχ με την χρήση των αγκυλών. Όπως είδαμε στο προηγούμενο κεφάλαιο, όταν έχουμε ένα πίνακα, το όνομά του (χωρίς τις αγκύλες) είναι δείκτης προς την διεύθυνση του πρώτου στοιχείου του πίνακα. Άρα, όταν θέλουμε να περάσουμε ένα πίνακα σε μια συνάρτηση, το μόνο που χρειάζεται να περάσουμε είναι το όνομα του πίνακα.

9.4 Παραδείγματα

9.4.1 Υπολογισμός μέσου όρου

Κώδικας C++	*Σχολιασμός*
`double mean ( double *x, int n ) {`	Συνάρτηση τύπου `double`.
`    if (n == 0) return 0;`	Επιστροφή εάν ο πίνακας είναι άδειος.
`    double s = 0 ;`	Αρχικοποίηση αθροιστή.
`    for (int i=0; i<n; i++)`	
`        s += x[i];`	Αθροιση τιμών πίνακα.
`    return s/n;`	Επιστροφή του μέσου όρου.
`}`	

Παρατηρήστε στο παραπάνω παράδειγμα ότι ο πίνακας εισέρχεται στην λίστα μεταβλητών ως ένας απλός δείκτης. Κατόπιν μέσα στην συνάρτηση ο πίνακας προσπελάσεται κανονικά ως πίνακας. Σημειώστε ότι όπως είπαμε και στο προηγούμενο κεφάλαιο, δείκτες και πίνακες έχουν πολύ στενή σχέση μεταξύ τους, και έτσι οι εκφράσεις `x[i]` και `*(x+i)` είναι πλήρως ισοδύναμες.

Παρακάτω γράφουμε το πρόγραμμα που θα καλούσε μια τέτοια συνάρτηση.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`double mean ( double *, int );`	Πρωτότυπο συνάρτησης.
`int main ( ) {`	
`    double x[1000], xmean;`	Δήλωση πίνακα και μεταβλητής.
`    int n;a`	
`    cout << "πόσα στοιχεία; ";`	
`    cin >> n;`	Πλήθος στοιχείων που θα εισαχθούν.
`    for (int i=0; i<n; i++)`	
`        cin >> x[i];`	Εισαγωγή τιμών στον πίνακα.
`    xmean = mean(x,n);`	Κλήση συνάρτησης για μέσο όρο. Στην είσοδο, ο `x` είναι δείκτης, το `n` απλή μεταβλητή.
`    cout << "ο μέσος όρος είναι"`	
`         << xmean << endl;`	Εκτύπωση αποτελέσματος.
`}`	

Στο κάλεσμα της συνάρτησης ο `x` είναι δείκτης στο πρώτο στοιχείο του πίνακα `x`. Άρα, τελείως ισοδύναμα θα μπορούσαμε επίσης να γράψουμε:

```
xmean = mean(&x[0], n);
```

Δηλαδή, το `x` και το `&x[0]` είναι ταυτόσημα.

9.4.2 Εσωτερικό γινόμενο

Η παρακάτω συνάρτηση δέχεται δύο πίνακες και το πλήθος των στοιχείων τους, και επιστρέφει το εσωτερικό τους γινόμενο.

Κώδικας C++	*Σχολιασμός*
`double array_prod(double *x,`	
`  double *y, int n) {`	
`  if (n == 0) return 0;`	Επιστρέφει 0 εάν οι πίνακες είναι άδειοι.
`  double s = 0 ;`	Αρχικοποίηση αθροιστή.
`  for (int i=0; i<n; i++)`	
`  s += x[i] * y[i];`	Αθροιση γινομένων στοιχείων.
`  return s ;`	Επιστροφή αποτελέσματος.
`}`	

και το πρόγραμμα που την καλεί:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`double array_prod ( double *, double *, int );`	
`int main ( ) {`	
`  double x[1000], y[1000], prod;`	
`  int n;`	
`  cout << "Πόσα στοιχεία; ";`	
`  cin >> n;`	Πλήθος στοιχείων που θα εισαχθούν.
`  for (int i=0; i<n; i++)`	
`  cin >> x[i] >> y[i];`	Εισαγωγή πινάκων.
`  prod = array_prod(x, y, n);`	Οι x, y είναι δείκτες, το n απλή μεταβλητή.
`  cout << "Το γινόμενο είναι"`	
`  << prod << endl;`	
`}`	

9.4.3 Ταξινόμηση φυσαλίδας

Θα γράψουμε συνάρτηση που δέχεται πίνακα και τον ταξινομεί με τη μέθοδο της φυσαλίδας. Η συνάρτηση είναι τύπου `void`, καθώς δεν γυρνάει κάποιο αποτέλεσμα. Ο ταξινομημένος πίνακας είναι ήδη αποθηκευμένος στην μνήμη.

Κώδικας C++	*Σχολιασμός*
`void sort ( double *x, int n ) {`	
`  for (int j=n-1; j>0; j--)`	Ξεκινώντας από το τέλος.
`  for (int i=0; i<j; i++)`	Πήγαινε σε όλα τα προηγούμενα στοιχεία.
`  if ( x[i] > x[i+1] )`	
`  swap(&x[i],&x[i+1]);`	Κάθε φορά που κάποιο είναι μεγαλύτερο
`}`	από το επόμενο, άλλαξέ του θέση.

και το πρόγραμμα που την καλεί:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`void swap(double *x, double *y);`	Πρωτότυπο της συνάρτησης `swap` που καλείται από την `sort`.
`void sort(double *, int );`	Πρωτότυπο συνάρτησης `sort`.
`main ( ) {`	
`  double x[1000];`	
`  int n;`	
`  cout << "Πόσα στοιχεία;\n";`	
`  cin >> n;`	
`  for (int i=0; i<n; i++)`	
`  cin >> x[i];`	
`  sort(x, n);`	
`  cout << "Μετά την ταξινόμηση:\n"`	
`  for (int i=0; i<n; i++)`	
`  cout << x[i] << endl;`	
`}`	

Κεφάλαιο 10

Δομές

10.1 Τι είναι οι δομές

Έχουμε δει μέχρι εδώ πως χρησιμοποιούνται οι συναρτήσεις: δέχονται κάποιες μεταβλητές, εκτελούν πράξεις, και επιστρέφουν το αποτέλεσμα. Όταν οι μεταβλητές στην είσοδο είναι λίγες, και το παραγόμενο αποτέλεσμα της συνάρτησης ένα, είμαστε επαρκείς. Τι γίνεται όμως σε αντίθετη περίπτωση; Υπάρχουν προβλήματα που ο αριθμός των μεταβλητών που απαιτούνται μπορεί να γίνει πολύ μεγάλος. Αλλά ακόμα και μικρός να είναι, σε πολλά προβλήματα κάποιες μεταβλητές είναι τόσο άρρηκτα δεμένες μεταξύ τους, που ουσιαστικά συμπεριφέρονται σαν μια νέα, σύνθετη μεταβλητή.

Για παράδειγμα, έστω οι συντεταγμένες ενός σημείου στον χώρο. Αυτές αποτελούνται από τις x, y, z συντεταγμένες του. Ουσιαστικά περιγράφουν ένα τριδιάστατο διάνυσμα. Έστω και μια συνάρτηση που θέλουμε να κάνει κάποιον υπολογισμό πάνω στο διάνυσμα. Εάν κάθε φορά πρέπει να περνάμε μία–μία αυτές μεταβλητές μπορεί να γίνει κουραστικό. Ο προγραμματισμός θα ήταν πολύ απλούστερος εάν μπορούσαμε να τις περνάμε όλες μαζί, σαν ένα διάνυσμα και όχι μια συλλογή τριών μεταβλητών. Η C++ μας δίνει την δυνατότητα να δημιουργούμε νέους τύπους μεταβλητών, οι οποίες να είναι το σύνολο κάποιων ήδη υπαρχουσών μεταβλητών.

10.2 Ορισμός δομής

Θα χρησιμοποιήσουμε το παράδειγμα του τριδιάστατου διανύσματος. Έστω λοιπόν ότι θέλουμε το τριδιάστατο διάνυσμα ως έναν νέο τύπο μεταβλητής. Το πρώτο που πρέπει να αποφασίσουμε είναι πως θα ονομάσουμε αυτόν τον νέο τύπο μεταβλητής. Ήδη χρησιμοποιούμε τους τύπους `double` και `int`. Μπορούμε να δώσουμε οποιοδήποτε όνομα εκτός των δεσμευμένων λέξεων της C++, αλλά καλό πάντα είναι το όνομα που επιλέγουμε να είναι όσο πιο περιγραφικό της νέας μεταβλητής γίνεται. Στην προκειμένη περίπτωση η προφανής επιλογή είναι να την ονομάσουμε vector. Το δεύτερο που πρέπει να αποφασίσουμε είναι τι περιλαμβάνει αυτή η νέα μεταβλητή. Στην προκειμένη, αρκούν οι 3 καρτεσιανές συνιστώσες. Ο ορισμός γίνεται με την δεσμευμένη λέξη `struct`:

```
struct vector {
    double x;
```

```
    double y;
    double z;
};
```

Ας τα δούμε ένα–ένα. Ο ορισμός ξεκινάει με `struct`. Ακολουθεί το όνομα που θέλουμε για την δομή. Αυτό το όνομα θα είναι και ο τύπος της νέας μεταβλητής. Μέσα σε αγκύλες μπαίνουν οι μεταβλητές που συνιστούν την δομή. Τα ονόματα που χρησιμοποιούμε (εδώ επιλέξαμε `x`, `y`, `z`) μπορούν να είναι οποιαδήποτε. Είναι όμως σημαντικό να είναι απολύτως περιγραφικά σχετικά με την ποσότητα στην οποία αντιστοιχούν. Τέλος προσέξτε ότι ο ορισμός της δομής κλείνει με ένα επιπλέον ερωτηματικό. Αυτό συμβαίνει γιατί η δήλωση μιας δομής είναι εντολή.

Μια δομή δεν είναι αναγκαστικό να περιέχει μεταβλητές του ίδιου τύπου. Αναλόγως την ανάγκη, μπορούμε να ενώσουμε πολλές διαφορετικές μεταβλητές μέσα σε μια δομή. Στη γενική περίπτωση, ο ορισμός μιας δομής γίνεται:

```
struct όνομα_δομής {
    τύπος όνομα ;
    τύπος όνομα ;
       ⋮
} ;
```

Ο ορισμός συνήθως γίνεται πριν ξεκινήσει το κυρίως πρόγραμμα (δηλαδή πριν την `main`).

Αυτό που κάνουμε με μια δομή είναι να δημιουργούμε νέους τύπους μεταβλητών. Αυτό μεταφέρει τον προγραμματισμό σε ένα νέο, πιο ψηλό επίπεδο. Θα δούμε παρακάτω πως τις χρησιμοποιούμε. Προς το παρόν, ας δούμε μερικά άλλα παραδείγματα δομών (μόνο η φαντασία του κάθε προγραμματιστή περιορίζει τον αριθμό και την μορφή των δομών που μπορούν να γραφτούν).

Παράδειγμα:
Δομή για μιγαδικούς αριθμούς, με πραγματικό και φανταστικό μέρος:
```
struct complex {
    double real, imag;
};
```

Παράδειγμα:
Δομή για ιόντα (πχ για προσομοίωση μοριακής δυναμικής), με θέση, μάζα και σθένος:
```
struct ion {
    double x, y, z, m, q;
};
```

Παράδειγμα:
Δομή για προϊόντα σε αποθήκη, που περιλαμβάνει κωδικό προϊόντος, έτος κατασκευής και τιμή πώλησης:
```
struct item {
    int code, year;
    double price;
};
```

10.3 Δήλωση μεταβλητών δομής

Μέσα σε ένα πρόγραμμα δηλώνουμε μεταβλητές του νέου αυτού τύπου όπως γίνεται και στις απλές μεταβλητές: είτε μέσα στο πρόγραμμα σαν τοπική, είτε απέξω σαν καθολική μεταβλητή. Για παράδειγμα, έστω ένα πρόγραμμα που δηλώνει δύο ρητούς και δύο διανύσματα:

Κώδικας C++

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

struct vector {
    double x, y, z;
};

int main ( ) {
    double a1, a2 ;
    vector v1, v2 ;
        ⋮
}
```

Τα a1, a2 είναι απλές μεταβλητές διπλής ακρίβειας, ενώ τα v1, v2 είναι διανύσματα, τα οποία αποτελούνται το καθένα από 3 ρητούς διπλής ακρίβειας.

10.4 Προσπέλαση των μελών μιας δομής

Το επόμενο βήμα είναι η προσπέλαση των μελών μιας δομής, είτε για ανάθεση τιμών, είτε για αριθμητική επεξεργασία. Αυτό γίνεται με τον τελεστή της τελείας "." χρησιμοποιώντας τα ονόματα των μελών που έχουν δοθεί στον ορισμό της δομής. Για παράδειγμα, απόσπασμα κώδικα όπου γίνεται ανάθεση τιμών σε διάνυσμα:

Κώδικας C++

```
int main ( ) {
    vector v1, v2;
    v1.x = 3;
    v1.y = 4;
    v1.z = 5;
    v2.x = 11;
    v2.y = 22;
    v3.z = 33;
}
```

Η ανάθεση μπορεί να γίνει και μέσω του πληκτρολογίου:

Κώδικας C++

```
int main ( ) {
    vector v1, v2;
    cout << "Δώσε συντεταγμένες του 1ου διανύσματος \n";
    cin >> v1.x >> v1.y >> v1.z;

    cout <<"Δώσε συντεταγμένες του 2ου διανύσματος \n";
    cin >> v2.x >> v2.y >> v2.z;

    cout << "Το άθροισμά τους είναι το διάνυσμα \n";
    cout << v1.x+v2.x << v1.y+v2.y << v1.z+v2.z << endl;
}
```

Όπως φαίνεται στο παραπάνω, με ανάλογο τρόπο γίνεται και η εκτύπωση των τιμών. Ουσιαστικά οι μεταβλητές που περνάνε μέσα στις εσωτερικές συναρτήσεις `cin` και `cout` πρέπει να είναι απλές μεταβλητές τύπου `int` ή `double`, δεν μπορούμε να περάσουμε μια δομή σε αυτές. Δηλαδή εκφράσεις όπως

```
cout << v1;  ή
cin >> v1;
```

δεν έχουν κανένα νόημα και θα προκαλέσουν λάθος κατά την μετάφραση του προγράμματος.

Όλες οι πράξεις (=, +, −, *, /) με τα μέλη της κάθε δομής είναι επιτρεπτές, καθώς τα ίδια είναι απλές μεταβλητές. Όμως εκτός από μία, οι συνήθεις αυτές πράξεις δεν επιτρέπονται σε ολόκληρες τις δομές. Η μόνη πράξη που επιτρέπεται είναι η ανάθεση =, και αυτό μόνο ανάμεσα σε ίδιες δομές. Στο παραπάνω παράδειγμα, έστω ότι δηλώνουμε και ένα τρίτο διάνυσμα v3, και θέλουμε να αντιγράψουμε τις τιμές του v1 στο v3. Η παρακάτω έκφραση είναι επιτρεπτή:

```
vector v1, v2, v3;
v3 = v1;
```

το οποίο είναι απολύτως ισοδύναμο με το παρακάτω (ουσιαστικά το παρακάτω είναι αυτό που εκτελείται έτσι και αλλιώς: ο μεταφραστής αντιγράφει τις τιμές των μελών της δομής v1 στην v3):

```
vector v1, v2, v3;
v3.x = v1.x;
v3.y = v1.y;
v3.z = v1.z;
```

Όμως, εάν θέλουμε να αναθέσουμε στο v3 το άθροισμα των v1 και v2, πρέπει αναγκαστικά να γράψουμε όλες τις πράξεις αναλυτικά:

```
vector v1, v2, v3;
v3.x = v1.x + v2.x;
v3.y = v1.y + v2.y;
v3.z = v1.z + v2.z;
```

Το παρακάτω δηλαδή είναι λάθος:

```
v3 = v1 + v2;
```
ΛΑΘΟΣ

Ο λόγος είναι ότι τέτοιου είδους πράξεις (δηλαδή εκτός του =) δεν είναι πάντα σαφώς προσδιορισμένες. Μπορεί σε διανύσματα η πρόσθεση και η αφαίρεση να είναι σχετικά προφανή, αλλά ποιο για παράδειγμα θα ήταν το νόημα του πολλαπλασιασμού; Ακόμα περισσότερο, φανταστείτε την δομή για προϊόντα σε αποθήκη, τι νόημα θα μπορούσε να έχει εκεί η πρόσθεση ή ο πολλαπλασιασμός; Θα δούμε σε επόμενα κεφάλαια ότι η C++ μας δίνει την δυνατότητα να προγραμματίσουμε εμείς τις πράξεις μεταξύ σύνθετων μεταβλητών μέσω μιας διαδικασίας που ονομάζεται *υπερφόρτωση τελεστή*. Αυτό είναι αναπόσπαστο κομμάτι του *αντικειμενοστραφούς* χαρακτήρα της C++, και θα το δούμε εφόσον κάνουμε μια εισαγωγή στις τάξεις στο επόμενο κεφάλαιο.

Από την στιγμή που θα οριστεί μια δομή, μπορούμε να την χρησιμοποιούμε όπως και κάθε απλή μεταβλητή, με μόνη ουσιαστική διαφορά ότι πρέπει να χρησιμοποιούμε τον τελεστή της τελείας για την προσπέλαση των μελών της. Μπορούμε για παράδειγμα να δηλώσουμε πίνακα με στοιχεία της δομής.

Τέλος, μια δομή μπορεί να εμπεριέχει ως μέλος μια άλλη δομή. Στην περίπτωση αυτή χρησιμοποιούμε τον τελεστή της τελείας δύο φορές. Στη γενική περίπτωση, χρησιμοποιούμε τον τελεστή της τελείας όσες φορές χρειάζεται ώστε να φτάσουμε σε απλές μεταβλητές.

Παράδειγμα:
Μια δομή για ιόντα που περιέχουν ένα διάνυσμα θέσης, φορτίο και μάζα.

```
struct vector { double x, y, z; };
struct ion { vector r; double m, q; };
```

10.5 Παραδείγματα

10.5.1 Διδιάστατα διανύσματα

Θα γράψουμε πρόγραμμα που δηλώνει πίνακα διδιάστατων διανυσμάτων (μόνο x και y συνιστώσες), τα διαβάζει από τον πληκτρολόγιο, και τυπώνει τον μέσο όρο των μέτρων τους.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`struct vector {`	
`  double x, y;`	Ορισμός δομής της δομής `vector`.
`};`	
`int main ( ) {`	
`  int n;`	
`  vector v[1000];`	Δήλωση πίνακα vector με 1000 στοιχεία (δηλαδή 1000 διανυσμάτων).
`  cout << "Δώστε πλήθος διανυσμάτων\n";`	
`  cin >> n;`	Πλήθος διανυσμάτων που θα εισαχθούν.
`  if ( n==0 || n>1000 )`	
`  return 1;`	Επιστροφή εάν το πλήθος είναι έξω από τα παραδεκτά όρια.
`  for (int i=0; i<n; i++) {`	Εισαγωγή διανυσμάτων.
`  cout << "Δώστε συντεταγμένες\n";`	
`  cin >> v[i].x >> v[i].y;`	
`  }`	
`  double s = 0;`	Αρχικοποίηση αθροιστή.
`  for (int i=0; i<n; i++)`	Αθροιση των μέτρων των διανυσμάτων.
`  s+=sqrt(v[i].x*v[i].x+v[i].y*v[i].y)`	
`  cout << "Ο μέσος όρος είναι"`	
`  << s/n << endl;`	Εμφάνιση του μέσου όρου των μέτρων.
`  }`	
`}`	

10.5.2 Δομή για ιόντα

Στο πρόγραμμα δηλώνουμε έναν πίνακα από ιόντα, και εισάγουμε τιμές.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`struct vector { double x, y, z; };`	
`struct ion { vector r; double m, q; };`	
`int main ( ) {`	
`  int n;`	
`  ion a[1000];`	Δήλωση πίνακα `ion` με 1000 στοιχεία (δηλαδή 1000 ιόντα).
`  cout <<"Δώστε το πλήθος ιόντων\n";`	
`  cin >> n;`	Πλήθος ιόντων που θα εισαχθούν.
`  if ( n==0 || n>1000 )`	
`  return 1;`	Επιστροφή εάν το πλήθος είναι έξω από τα παραδεκτά όρια.
`  for (int i=0; i<n; i++) {`	Εισαγωγή ιόντων.
`  cout << "Δώστε στοιχεία ιόντος\n";`	
`  cin >> a[i].r.x >> a[i].r.y >>`	
`  a[i].r.z >> a[i].m >> a[i].q;`	
`  }`	
⋮	
`}`	

10.6 Συναρτήσεις και δομές

Εάν σε μια συνάρτηση θέλουμε να περάσουμε μια δομή, τότε αυτό γίνεται με τον ίδιο ακριβώς τρόπο όπως και για απλές μεταβλητές, δηλαδή δηλώνουμε τύπο και όνομα μεταβλητής. Επιπρόσθετα, εάν η συνάρτηση γυρνάει πίσω μια δομή, τότε πρέπει η συνάρτηση να δηλωθεί με τον αντίστοιχο τύπο.

Συνάρτηση που δέχεται ένα διάνυσμα και γυρνάει πίσω το μέτρο του.

```
double magnitude ( vector v ) {
    double m = sqrt(v.x*v.x + v.y*v.y + v.z*v.z);
    return m;
}
```

Στο παραπάνω παράδειγμα, θέλουμε η συνάρτηση να επιστρέφει το μέτρο του διανύσματος. Το μέτρο είναι απλός ρητός, άρα και η συνάρτηση είναι τύπου `double`. Το όνομα της συνάρτησης είναι οποιαδήποτε μη δεσμευμένη λέξη της C++, και επιλέγουμε ένα περιγραφικό όνομα, για την προκειμένη περίπτωση επιλέγουμε το `magnitude`. Μέσα στην συνάρτηση θέλουμε να περάσουμε ένα διάνυσμα. Χρησιμοποιούμε την δομή `vector` που είχαμε περιγράψει στα προηγούμενα παραδείγματα, και έτσι ο τύπος της μεταβλητής εισόδου είναι `vector`. Το όνομα της μεταβλητής εισόδου είναι οποιοδήποτε, εδώ επιλέξαμε το `v`. Μέσα στην συνάρτηση δηλώσαμε την τοπική μεταβλητή `m` καθαρά για βοηθητικούς λόγους. Στο συγκεκριμένο παράδειγμα θα μπορούσαμε να το αποφύγουμε επιστρέφοντας κατευθείαν την τιμή του μέτρου:

```
double magnitude ( vector v ) {
    return sqrt(v.x*v.x + v.y*v.y + v.z*v.z);
}
```

Ερώτηση: Όταν επιστρέφουμε μέσω της εντολής `return` πρέπει να έχει τον ίδιο τύπο με την συνάρτηση. Εδώ η συνάρτηση είναι τύπου `double`. Με τον στον πρώτο ορισμό δηλώσαμε τον `m` ως `double` και τον επιστρέψαμε μέσω της εντολής `return`. Στον δεύτερο ορισμό, πως ξέρουμε τι τύπο γυρνάμε; Είναι `double`; Και γιατί;

10.7 Παραδείγματα

10.7.1 Άθροισμα διανυσμάτων

Η παρακάτω συνάρτηση δέχεται δύο διανύσματα και γυρνάει πίσω το άθροισμά τους. Το άθροισμα δύο διανυσμάτων είναι διάνυσμα, άρα η συνάρτηση θα πρέπει να είναι τύπου `vector`. Στην είσοδο θα πρέπει να δέχεται δύο διανύσματα.

Κώδικας C++	*Σχολιασμός*
`vector sum ( vector v, vector u ) {`	
`    vector w;`	Δήλωση τοπικού διανύσματος.
`    w.x = v.x + u.x;`	Η x συνιστώσα του `w` είναι η x του `v` συν την x του `u`.
`    w.y = v.y + u.y;`	Παρομοίως για y.
`    w.z = v.z + u.z;`	Παρομοίως για z.
`    return w;`	Επιστροφή του αποτελέσματος.
`}`	

10.7.2 Εσωτερικό γινόμενο διανυσμάτων

Πλήρες πρόγραμμα που δέχεται από το πληκτρολόγιο τις συνιστώσες δύο διανυσμάτων και τυπώνει το μέτρο του καθενός, το μέτρο του αθροίσματός τους, καθώς και και το εσωτερικό τους γινόμενο.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`struct vector { double x, y, z; };`	Ορισμός δομής διανύσματος.
`double magnitude ( vector );`	Πρωτότυπα συναρτήσεων.
`vector sum ( vector, vector );`	
`double product ( vector, vector );`	
`int main ( ) {`	
`    vector v1, v2, v3;`	Δήλωση 3 διανυσμάτων
`    double m1, m2, m3, p;`	
`    cout << "Δώσε το 1ο διάνυσμα\n";`	
`    cin >> v1.x >> v1.y >> v1.z;`	
`    cout << "Δώσε το 2ο διάνυσμα\n";`	
`    cin >> v2.x >> v2.y >> v2.z;`	
`    m1 = magnitude(v1);`	Το μέτρο του πρώτου διανύσματος.
`    m2 = magnitude(v2);`	Το μέτρο του δεύτερου διανύσματος.
`    v3 = sum(v1,v2);`	Το άθροισμα των διανυσμάτων.
`    m3 = magnitude(v3);`	Το μέτρο του αθροίσματος.
`    p = product(v1,v2);`	Το εσωτερικό γινόμενο των v1, v2.
`    cout << "Το μέτρο του 1ου είναι "`	
`         << m1 << endl;`	
`    cout << "Το μέτρο του 2ου είναι "`	
`         << m2 << endl;`	
`    cout << "Το μέτρο του αθροίσματος "`	
`         << "είναι " << m3 << endl;`	
`    cout << "Το εσωτερικό γινόμενο είναι"`	
`         << p << endl;`	
`}`	

```
double magnitude ( vector v ) {
    return sqrt(v.x*v.x+v.y*v.y+v.z*v.z);
}
```
Συνάρτηση για το μέτρο διανύσματος.

```
double product ( vector v, vector u ) {
    return v.x*u.x + v.y*u.y + v.z*u.z;
}
```
Συνάρτηση για εσωτερικό γινόμενο δύο διανυσμάτων .

```
vector sum ( vector v, vector u ) {
    vector w;
    w.x = v.x + u.x;
    w.y = v.y + u.y;
    w.z = v.z + u.z;
    return w;
}
```
Συνάρτηση για άθροιση δύο διανυσμάτων.

Μια συνάρτηση μπορεί να καλεί μια άλλη συνάρτηση. Στο παραπάνω παράδειγμα θα μπορούσαμε να εισάγουμε μια συνάρτηση για να υπολογίζει την γωνία που σχηματίζουν μεταξύ τους τα διανύσματα. Υπενθυμίζεται ότι ισχύει η σχέση για το εσωτερικό γινόμενο δύο διανυσμάτων:

$$\vec{v} \cdot \vec{u} = |\vec{v}||\vec{u}| \cos\theta \Rightarrow \theta = \text{acos}\left(\frac{\vec{v} \cdot \vec{u}}{|\vec{v}||\vec{u}|}\right)$$

10.7.3 Γωνία διανυσμάτων

Θα γράψουμε συνάρτηση για τον υπολογισμό της γωνίας μεταξύ δύο διανυσμάτων (υποθέτουμε ότι οι συναρτήσεις `magnitude` και `product` υπάρχουν ήδη όπως ορίστηκαν στο προηγούμενο παράδειγμα).

Κώδικας C++	*Σχολιασμός*
`double angle ( vector v, vector u ) {`	
`    double theta, m1, m2;`	Τοπικές μεταβλητές για τη γωνία και τα μέτρα.
`    m1 = magnitude(v);`	Το μέτρο του διανύσματος `v`.
`    m2 = magnitude(u);`	Το μέτρο του διανύσματος `u`.
`    if (m1 == 0 || m2 == 0)`	Εάν ένα από τα δύο είναι μηδέν,
`        theta = 0;`	Θέτουμε και την γωνία μηδέν.
`    else`	
`        theta=acos(product(v,u)/(m1*m2));`	Αλλιώς υπολογίζουμε την γωνία.
`        theta=theta*180/acos(-1.0);`	Μετατρέπουμε τη γωνία σε μοίρες.
`    return theta;`	
`}`	

10.8 Δείκτες και δομές

Στα προηγούμενα παραδείγματα περνούσαμε τις δομές στην συνάρτηση κατ' αξία, δηλαδή στην είσοδο στη συνάρτηση δημιουργείται ένα αντίγραφο ολόκληρης της δομής που περνάει. Αυτό δημιουργεί δύο προβλήματα: εάν η δομή είναι πολύ μεγάλη το πρόγραμμα γίνεται πολύ αργό, και επίσης, παραμένει ο περιορισμός ότι δεν μπορεί να γίνει καμία τροποποίηση τιμής σε κάποια από τα μέλη της δομής. Ο τρόπος για να λυθεί αυτό είναι να μην περνάμε την δομή ως έχει στην συνάρτηση, αλλά έναν δείκτη προς αυτήν.

Όπως και στις απλές μεταβλητές, μπορούμε να δηλώσουμε δείκτες σε στοιχεία μιας δομής. Ο δείκτης αυτός δείχνει στο πρώτο μέλος της δομής. Χρησιμοποιούμε και πάλι τους τελεστές * και &.

Παράδειγμα:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`struct vector {`	
`  double x, y, z;`	
`};`	
`int main ( ) {`	
`  vector v;`	Δήλωση διανύσματος.
`  vector *p;`	Δήλωση δείκτη σε διάνυσμα.
`  v.x = 3;`	Ανάθεση τιμών στο διάνυσμα v.
`  v.y = 4;`	
`  v.z = 5;`	
`  p = &v;`	Ανάθεση της διεύθυνσης του v στον δείκτη p.
`  cout << (*p).x << (*p).y`	Εκτύπωση τιμών του v μέσω του δείκτη p.
`  << (*p).z << endl;`	
`}`	

Μια σημαντική λεπτομέρεια: καθώς ο τελεστής της τελείας "." έχει μεγαλύτερη προτεραιότητα από τον τελεστή του αστεριού "*", εάν δεν βάζαμε τις παρενθέσεις, δηλαδή γράφαμε `*p.x` ο μεταφραστής θα το εκλάμβανε ως `*(p.x)`, οποίο είναι λάθος.

Μια ισοδύναμη και πιο εύχρηστη έκφραση που δεν χρειάζεται τις παρενθέσεις είναι με τον τελεστή του βέλους `->`, η οποία γράφεται:

```
cout << p->x << p->y << p->z << endl;
```

το οποίο σημαίνει: πήγαινε στην διεύθυνση που δείχνει ο δείκτης, και δώσε την αριθμητική τιμή του μέλους που δείχνει το βέλος. Με αυτή την μορφή ο δείκτης κυριολεκτικά "δείχνει" σε ένα μέλος της δομής. Οι δύο μορφές `p->x` και `(*p).x` είναι ισοδύναμες, και μπορεί να χρησιμοποιηθεί οποιαδήποτε από τις δύο ανάλογα με την αισθητική του προγραμματιστή. Στα επαγγελματικά προγράμματα, βέβαια, χρησιμοποιείται σχεδόν αποκλειστικά ο τελεστής του βέλους.

Μπορούμε να καλούμε συναρτήσεις με όρισμα δείκτη σε δομή. Παρακάτω επαναλαμβάνουμε το παράδειγμα 10.7.2, αλλά με τις συναρτήσεις να δέχονται δείκτες. Το αποτέλεσμα είναι και στις δύο περιπτώσεις το ίδιο.

10.9 Παραδείγματα

10.9.1 Εσωτερικό γινόμενο διανυσμάτων (με δείκτες)

Πλήρες πρόγραμμα που δέχεται από το πληκτρολόγιο τις συνιστώσες δύο διανυσμάτων και τυπώνει το μέτρο του καθενός, το μέτρο του αθροίσματός τους, καθώς και και το εσωτερικό τους γινόμενο. Οι συναρτήσεις που θα γραφτούν, να δέχονται δείκτες.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`struct vector { double x, y, z; };`	Ορισμός δομής για διάνυσμα.
`double magnitude ( vector * );`	Πρωτότυπα συναρτήσεων, που όμως τώρα δέχονται δείκτες σε διανύσματα.
`vector sum ( vector *, vector * );`	
`double product ( vector *, vector * );`	
`int main ( ) {`	
`    vector v1, v2, v3;`	
`    double m1, m2, m3, p;`	
`    cout << "Δώσε το 1ο διάνυσμα\n";`	
`    cin >> v1.x >> v1.y >> v1.z;`	
`    cout << "Δώσε το 2ο διάνυσμα\n";`	
`    cin >> v2.x >> v2.y >> v2.z;`	
`    m1 = magnitude(&v1);`	Κλήση συνάρτησης με δείκτη.
`    m2 = magnitude(&v2);`	Κλήση συνάρτησης με δείκτη.
`    v3 = sum(&v1,&v2);`	Κλήση συνάρτησης με δείκτες.
`    m3 = magnitude(&v3);`	Κλήση συνάρτησης με δείκτη.
`    p = product(&v1,&v2);`	Κλήση συνάρτησης με δείκτες.
`    cout << "Το μέτρο του 1ου είναι "`	
`         << m1 << endl;`	
`    cout << "Το μέτρο του 2ου είναι "`	
`         << m2 << endl;`	
`    cout << "Το μέτρο του αθροίσματος "`	
`         << "είναι " << m3 << endl;`	

```
    cout << "Το εσωτερικό γινόμενο είναι
         << p << endl;
}

double magnitude ( vector *v ) {            Συνάρτηση για μέτρο διανύσματος, που δέ-
                                            χεται δείκτη σε διάνυσμα.
    return sqrt(v->x*v->x + v->y*v->y + v->z*v->z);
}

double product ( vector *v, vector *u ) {   Συνάρτηση για εσωτερικό γινόμενο δια-
                                            νυσμάτων, που δέχεται δείκτες σε διανύ-
                                            σματα.
    return v->x*u->x + v->y*u->y + v->z*u->z;
}

vector sum ( vector *v, vector *u ) {       Συνάρτηση για άθροισμα διανυσμάτων, που
                                            δέχεται δείκτες σε διανύσματα.
    vector w;
    w.x = v->x + u->x;
    w.y = v->y + u->y;
    w.z = v->z + u->z;
    return w;
}
```

10.10 Εισαγωγή πίνακα δομής σε συνάρτηση

Εδώ ουσιαστικά δεν αλλάζει τίποτα σε σχέση με όσα είχαμε δει στην περίπτωση των πινάκων απλών μεταβλητών. Ο πίνακας εισάγεται απλά ως ένας δείκτης προς το πρώτο στοιχείο, ο οποίος δείκτης δεν είναι τίποτα άλλο παρά το όνομα του πίνακα χωρίς την δεικτοδότηση. Μέσα στην συνάρτηση χρησιμοποιούμε τον πίνακα κανονικά, με την προσπέλαση να γίνεται μέσω του τελεστή της τελείας.

10.11 Παραδείγματα

10.11.1 Πίνακας διανυσμάτων

Να γραφεί συνάρτηση που δέχεται έναν πίνακα διανυσμάτων και επιστρέφει το μέσο όρο των μέτρων τους. Επίσης να γραφεί το πλήρες πρόγραμμα C++ που θα καλούσε αυτήν την συνάρτηση.

Ο μέσος όρος των μέτρων είναι ρητός, άρα η συνάρτηση θα είναι τύπου `double`. Στην λίστα μεταβλητών εισάγουμε τον πίνακα ως έναν δείκτη, και επίσης εισάγουμε το πλήθος των διανυσμάτων για τα οποία θα υπολογίσουμε τον μέσο όρο.

Κώδικας C++	*Σχολιασμός*
`double ave_magn ( vector *v, int n ) {`	
`    if (n == 0) return 0;`	Επιστροφή εάν δεν υπάρχουν διανύσματα.
`    double s = 0;`	Αρχικοποίηση αθροιστή.
`    for (int i=0; i<n; ++i)`	Βρόγχος πάνω σε όλα τα διανύσματα.
`    s += magnitude(v[i]);`	Αθροιση των μέτρων τους. Χρησιμοποιούμε τη συνάρτηση `magnitude` που ορίσαμε προηγουμένως.
`    return s/n;`	Επιστροφή μέσου όρου.
`}`	

Το πλήρες πρόγραμμα που θα καλούσε την παραπάνω συνάρτηση:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`struct vector { double x, y, z; };`	Ορισμός δομής για διάνυσμα.
`double magnitude ( vector );`	Πρωτότυπο συνάρτησης.
`double ave_magn ( vector *, int );`	πρωτότυπο συνάρτησης.
`int main ( ) {`	
`    int n;`	

```
    vector v[1000];
    cout << "Αριθμός διανυσμάτων?";
    cin >> n;                                   Ο αριθμός των διανυσμάτων που θα εισαχθούν.

    if (n==0 || n>1000) return 1;               Εξοδος εάν ο αριθμός είναι έξω από τα επιτρεπτά όρια.

    for (int i=0; i<n; ++i)
        cin >> v[i].x >> v[i].y >> v[i].z;      Εισαγωγή διανυσμάτων.

    cout << ave_magn(v,n) << endl;              Κλήση συνάρτησης με ταυτόχρονη εκτύπωση του αποτελέσματος.
}
```

10.11.2 Ηλεκτροστατική ενέργεια ιόντων

Πρόγραμμα που διαβάζει n ιόντα και υπολογίζει το κέντρο μάζας, την ολική ηλεκτροστατική ενέργεια καθώς και τη δύναμη που ασκείται σε κάθε ιόν. Υπενθυμίζονται οι σχέσεις για το κέντρο μάζας:

$$\mu_x = \frac{\sum_{i=0}^{n-1} x_i m_i}{\sum_{i=0}^{n-1} m_i} \qquad \mu_y = \frac{\sum_{i=0}^{n-1} y_i m_i}{\sum_{i=0}^{n-1} m_i} \qquad \mu_z = \frac{\sum_{i=0}^{n-1} z_i m_i}{\sum_{i=0}^{n-1} m_i}$$

Η ολική ηλεκτροστατική ενέργεια του ιόντος j λόγω των υπόλοιπων ιόντων:

$$E_j = \sum_{i=0, i\neq j}^{n-1} \frac{q_i q_j}{|\vec{r}_j - \vec{r}_i|}$$

Η x συνιστώσα της συνολικής δύναμης που νοιώθει το ιόν j από όλα τα υπόλοιπα ιόντα, και παρομοίως για τις υπόλοιπες συνιστώσες:

$$F_j^x = \sum_{i=0, i\neq j}^{n-1} q_i q_j \frac{x_j - x_i}{|\vec{r}_j - \vec{r}_i|^3}$$

Για αυτό το πρόβλημα θα χρειαστούμε καταρχάς μια βολική δομή για ιόντα, που να περιέχει ένα διάνυσμα για την θέση, και δύο ρητούς για τη μάζα και το φορτίο. Θα ορίσουμε λοιπόν δύο δομές, μια για διανύσματα και μια για ιόντα. Η δεύτερη θα εμπεριέχει ως μέλος την πρώτη.

```
struct vector { double x, y, z; };        ορισμός δομής για διάνυσμα
struct ion { vector r; double m, q; };    ορισμός δομής για ιόντα
```

Μέσα στο κυρίως πρόγραμμα θα δηλώσουμε ένα πίνακα από ιόντα, και θα τους δώσουμε τιμές. Για να πάρουμε τα ζητούμενα αποτελέσματα θα χρειαστούμε να ορίσουμε συναρτήσεις.

Συνάρτηση για την διαφορά δύο διανυσμάτων. Η διαφορά εμφανίζεται σε αρκετά σημεία στο πρόβλημα, οπότε θα ορίσουμε μια συνάρτηση για τον υπολογισμό της:

```
vector dif ( vector v, vector u ) {
    vector w;
    w.x = v.x - u.x;
    w.y = v.y - u.y;
    w.z = v.z - u.z;
    return w;
}
```

Συνάρτηση για την απόσταση δύο διανυσμάτων. Για ευκολία, θα χρησιμοποιήσουμε την συνάρτηση `dif` παραπάνω, και τη συνάρτηση `magnitude` που ορίσαμε σε προηγούμενο παράδειγμα.

```
double dist ( vector v, vector u ) {
    return magnitude( dif(v,u) );
}
```

Τώρα ορίζουμε μια συνάρτηση που να επιστρέφει το κέντρο μάζας του συστήματος των ιόντων. Το κέντρο μάζας έχει x,y,z συνιστώσες, είναι δηλαδή ουσιαστικά ένα διάνυσμα. Άρα η συνάρτηση που θα ορίσουμε θα επιστρέφει ένα διάνυσμα, θα είναι δηλαδή τύπου `vector`.

Κώδικας C++	*Σχολιασμός*
`vector centermass ( ion *a, int n ) {`	Εισαγωγή πίνακα ιόντων καθώς και του πλήθους τους.
`  vector cm; double m;`	Τοπικοί αθροιστές.
`  cm.x = cm.y = cm.z = m = 0;`	Αρχικοποίηση αθροιστών.
`  if (n == 0) return cm;`	Επιστροφή αν ο πίνακας είναι άδειος.
`  for (int i=0; i<n; ++i) {`	Βρόγχος πάνω σε όλα τα ιόντα.
`  cm.x += a[i].r.x * a[i].m;`	Αθροιση των $x_i m_i$.
`  cm.y += a[i].r.y * a[i].m;`	Αθροιση των $y_i m_i$.
`  cm.z += a[i].r.z * a[i].m;`	Αθροιση των $z_i m_i$.
`  m += a[i].m;`	Αθροιση των m_i.
`  }`	
`  cm.x = cm.x / m;`	Υπολογισμός των συνιστωσών του κέντρου
`  cm.y = cm.y / m;`	μάζας.
`  cm.z = cm.z / m;`	
`  return cm;`	Επιστροφή του αποτελέσματος (ως
`}`	διάνυσμα).

Κατόπιν ορίζουμε μια συνάρτηση για τον υπολογισμό της ενέργειας ενός ιόντος λόγω των υπολοίπων:

Κώδικας C++	*Σχολιασμός*
`double energy ( ion *a, int n, int j ) {`	Εισαγωγή πίνακα ιόντων, του πλήθους τους, καθώς και του ιόντος του οποίου θέλουμε την ενέργεια.
`  double e = 0;`	Δήλωση και αρχικοποίηση αθροιστή.
`  for (int i=0; i<n; ++i)`	Βρόγχος πάνω σε όλα τα ιόντα,
`  if (i != j)`	εξαιρουμένου αυτού του οποίου υπολογίζουμε την ενέργεια.
`  e += a[i].q*a[j].q/`	Αθροιση επιμέρους ενεργειών.
`  dist(a[j].r,a[i].r);`	
`  return e;`	Επιστροφή αποτελέσματος.
`}`	

Τέλος, ορίζουμε την συνάρτηση που υπολογίζει την συνολική δύναμη που δέχεται ένα ιόν από όλα τα υπόλοιπα. Καθώς η δύναμη είναι διάνυσμα, ο τύπος της συνάρτησης θα είναι `vector`.

Κώδικας C++	*Σχολιασμός*
`vector force ( ion *a, int n, int j ) {`	Εισαγωγή πίνακα ιόντων, του πλήθους τους, καθώς και του ιόντος του οποίου θέλουμε την ενέργεια.
`   vector f, fi;`	Τοπικές μεταβλητές αθροιστές.
`   f.x = f.y = f.z = 0;`	Αρχικοποίηση αθροιστή
`   for (int i=0; i<n; ++i)`	Βρόγχος πάνω σε όλα τα ιόντα,
`      if (i != j) {`	εξαιρουμένου αυτού του οποίου υπολογίζουμε την δύναμη.
`         fi.x = a[i].q*a[j].q*dif(a[j].r,a[i].r).x/`	
`                  pow(dist(a[j].r,a[i].r),3);`	
`         fi.y = a[i].q*a[j].q*dif(a[j].r,a[i].r).y/`	
`                  pow(dist(a[j].r,a[i].r),3);`	
`         fi.z = a[i].q*a[j].q*dif(a[j].r,a[i].r).z/`	
`                  pow(dist(a[j].r,a[i].r),3);`	
`      }`	
`      f = sum(f,fi);`	Χρησιμοποιούμε την συνάρτηση `sum` που ορίσαμε στην παράγραφο 10.7.1 για να αθροίσουμε τις δυνάμεις.
`   }`	
`   return f;`	
`}`	

Στην παραπάνω συνάρτηση το

```
dif(a[j].r,a[i].r).x
```

σημαίνει το εξής: Η κλήση της συνάρτησης `dif(a[j].r,a[i].r)` επιστρέφει ένα διάνυσμα, και εφαρμόζοντας το `.x` εμείς παίρνουμε την x του συνιστώσα.

Μπορούμε τώρα να γράψουμε το πλήρες πρόγραμμα C++ που θα έκανε τους παραπάνω υπολογισμούς για το σύστημα των ιόντων:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`struct vector { double x, y, z; };`	Ορισμός δομής για διάνυσμα.
`struct ion { vector r; double m, q; }`	Ορισμός δομής για ιόν.
`vector sum ( vector, vector );`	Πρωτότυπα συναρτήσεων.

```
vector dif ( vector, vector );
double magnitude ( vector );
double dist ( vector, vector );
vector centermass ( ion *, int );
double energy ( ion *, int, int );
vector force ( ion *, int, int );

int main ( ) {
    int n;
    ion a[1000];                              Δήλωση πίνακα με 1000 ιόντα.
    vector f, m;                              Δήλωση τοπικών μεταβλητών για δύναμη
                                              και κέντρο μάζας.
    cout << "Δώστε το πλήθος των ιόντων ";
    cin >> n;                                 Αριθμός ιόντων που θα εισαχθούν.

    if (n < 1 || n > 1000) return 1;          Εξοδος εάν ο αριθμός τους είναι εκτός
                                              ορίων.
    for (int i=0; i<n; ++i)
        cin >> a[i].r.x >> a[i].r.y >> a[i].r.z
            >> a[i].m >> a[i].q;              Εισαγωγή ιόντων.

    m = centermass(a,n);                      Κλήση συνάρτησης για κέντρο μάζας.
    cout << "Κέντρο μάζας: \n";
    cout << m.x << " " << m.y << " "          Εξαγωγή αποτελέσματος.
         << m.z << endl;

    for (int i=0; i<n; ++i) {                 Βρόγχος πάνω σε όλα τα ιόντα.
        cout << "Το ιόν " << i;
        cout << " έχει ενέργεια ";
        cout << energy(a,n,i);                Υπολογισμός και εκτύπωση της ενέργειας
                                              του ιόντος i.
        f = force(a,n,i);                     Υπολογισμός της δύναμης πάνω στο ιόν i.
        cout << " και υφίσταται δύναμη ";
        cout << f.x << " " << f.y << " "      Εκτύπωση αποτελέσματος.
             << f.z << endl;
    }
}
```

Κεφάλαιο 11

Τάξεις

11.1 Τι είναι οι τάξεις

Οι τάξεις είναι μια πιο γενικευμένη μορφή των δομών. Μέσα σε μια τάξη εκτός από δεδομένα συμπεριλαμβάνεται και κώδικας (συναρτήσεις) που επεξεργάζεται αυτά τα δεδομένα. Γενικά, μια τάξη (class) ορίζει έναν νέο τύπο δεδομένων, ένα αντικείμενο (object) με συγκεκριμένες ιδιότητες: περιλαμβάνει δεδομένα (μεταβλητές), και συναρτήσεις που επεξεργάζονται τα δεδομένα αυτά. Δεδομένα και συναρτήσεις ονομάζονται μέλη της τάξης.

Στο προηγούμενο κεφάλαιο εξετάσαμε τις δομές. Βασικά χαρακτηριστικά τους είναι ότι περιέχουν μόνο μεταβλητές. Θα μπορούσαμε να πούμε ότι είναι σαν μια τάξη που δεν έχει συναρτήσεις. Όσον αφορά την προσπέλαση των μεταβλητών μιας δομής, είδαμε ότι αυτό γίνεται απλά με την τελεία. Κανένας περιορισμός δεν δόθηκε όσον αφορά το πότε μπορούμε να προσπελάσουμε τις μεταβλητές μιας δομής. Σε οποιοδήποτε σημείο του προγράμματος επιθυμούμε μπορούμε να επεξεργαστούμε τα μέλη μιας δομής (αρκεί βέβαια η δομή να ορίστηκε καθολικά, δηλαδή πριν την `main`). Κατά αυτήν την έννοια οι μεταβλητές μιας δομής είναι δημόσιες, "ορατές" από κάθε σημείο.

Αντίθετα, τα δεδομένα και οι συναρτήσεις μιας τάξης χωρίζονται σε δύο μέρη: *τα δημόσια* (`public`) και τα *ιδιωτικά* (`private`). Σε μια τάξη όλα τα μέλη θεωρούνται ιδιωτικά εκτός και εάν δηλωθούν δημόσια με τη δεσμευμένη λέξη–κλειδί `public`. Μέσα από το κυρίως πρόγραμμα μπορούμε να έχουμε απ' ευθείας αναφορά μόνο στα δημόσια μέλη. Τα ιδιωτικά μέλη μπορούμε να τα επεξεργαστούμε μόνο μέσω των δημόσιων συναρτήσεων του ίδιου αντικειμένου. Στην πράξη, οι μεταβλητές μιας τάξης ορίζονται συνήθως ως ιδιωτικές, ενώ οι συναρτήσεις–μέλη μιας τάξης είναι δημόσιες. Αυτή η διαδικασία ονομάζεται *ενθυλάκωση*. Οι μεταβλητές "προστατεύονται" μέσα στο αντικείμενο, και η επεξεργασία τους επιτρέπεται μόνο μέσω προκαθορισμένων τρόπων που ορίζουν οι δημόσιες συναρτήσεις της τάξης.

11.2 Ορισμός τάξης

Μια τάξη δηλώνεται με την δεσμευμένη λέξη `class`. Η γενική μορφή είναι:

```
class όνομα {
    ιδιωτικά_δεδομένα_και_συναρτήσεις
public:
    δημόσια_δεδομένα_και_συναρτήσεις
} ;
```

Στην πιο απλοϊκή της μορφή, όπως είπαμε παραπάνω, μια τάξη περιέχει ιδιωτικές μεταβλητές και δημόσιες συναρτήσεις. Άρα ο ορισμός μιας τάξης στην πιο απλή και πρακτική μορφή της είναι:

```
class όνομα {
    ιδιωτικά_δεδομένα
public:
    δημόσιες_συναρτήσεις
} ;
```

Όπως και στις δομές, μια τάξη ορίζεται πριν την `main`, και αποτελεί κάτι παραπάνω από απλά ένα νέο είδος μεταβλητής. Αποτελεί ένα αντικείμενο.

Παράδειγμα:
Ας ορίσουμε μια τάξη ονόματι `vector` για τριδιάστατο διάνυσμα, που να περιέχει μόνο δημόσιες μεταβλητές (δηλαδή το ακριβές αντίστοιχο της δομής `vector` του προηγούμενου κεφαλαίου).

```
class vector {
public:
    double x, y, z;
};
```

Για την προσπέλαση των δημόσιων μεταβλητών χρησιμοποιούμε (όπως και στις δομές) χρησιμοποιούμε τον τελεστή της τελείας.

11.3 Ορισμός συναρτήσεων τάξης

Μέσα στην τάξη ορίζουμε και συναρτήσεις–μέλη. Η βασική τους λειτουργία είναι να διαχειρίζονται τις ιδιωτικές μεταβλητές της τάξης. Ο βασικός λόγος ύπαρξής τους είναι ώστε η επεξεργασία των μεταβλητών να γίνεται με αυστηρό και δομημένο τρόπο. Εάν σκεφτούμε ότι μια τάξη ορίζει ένα αντικείμενο, τότε ουσιαστικά οι δημόσιες συναρτήσεις δίνουν τις ιδιότητες του αντικειμένου. Προφανώς μπορεί κανείς να ορίσει και ιδιωτικές συναρτήσεις, αλλά για απλότητα θα το αποφύγουμε αυτό εδώ.

Παράδειγμα:
Έστω πάλι η τάξη `vector` που ορίσαμε πριν, αλλά με ιδιωτικές τις συνιστώσες `x`, `y`, `z` και δύο δημόσιες συναρτήσεις, μία για να δίνει τιμές στις μεταβλητές (πχ ονόματι `set`) και μια για να επιστρέφει το μέτρο του διανύσματος (πχ ονόματι `magn`).

```
class vector {
    double x, y, z;
public:
    void set ( double x1, double y1, double z1 );
```

```
    double magn();
};
```

Κατόπιν πρέπει να ορίσουμε τις συναρτήσεις της τάξης. Το πρωτότυπο της συνάρτησης είναι ήδη δηλωμένο μέσα στην τάξη, οπότε δεν χρειάζεται να το ξανακάνουμε. Ο ορισμός της συνάρτησης γίνεται μετά τον ορισμό της τάξης και έξω από την **main**, όπως και στις κοινές συναρτήσεις.

Αντίθετα όμως με τις κοινές συναρτήσεις που δεν ανήκουν πουθενά, οι συναρτήσεις μιας τάξης ανήκουν στην τάξη, και ως εκ τούτου έχουν άμεση πρόσβαση στις ιδιωτικές μεταβλητές της τάξης. Άρα η τάξη στην οποία ανήκουν πρέπει να δηλωθεί, και γίνεται ως εξής:

```
τύπος τάξη::συνάρτηση ( λίστα_εισόδου ) {
    ... εντολές ...
}
```

Ο *τύπος* είναι ο τύπος της μεταβλητής επιστροφής, ο οποίος μπορεί να είναι **void**, **int**, **double**, μια δομή, ή και μια τάξη. Προσέξτε τις δύο άνω–κάτω τελείες. Πρώτα δηλώνουμε την τάξη στην οποία ανήκει η συνάρτηση, και μετά το όνομα της συνάρτησης. Στο προηγούμενο παράδειγμα, οι συναρτήσεις θα είναι:

```
void vector::set ( double x1, double y1, double z1 ) {
    x=x1;                                  Αντιγραφή των μεταβλητών εισόδου
    y=y1;                                  στις εσωτερικές μεταβλητές.
    z=z1;
}


double vector::magn ( ) {
    return sqrt(x*x + y*y + z*z);          Υπολογισμός και επιστροφή του μέτρου.
}
```

Όπως παρατηρούμε οι συναρτήσεις ανήκουν στο αντικείμενο και έτσι έχουν απόλυτη και άμεση γνώση την ιδιωτικών μεταβλητών της τάξης, δηλαδή των εσωτερικών μεταβλητών **x**, **y**, **z** (χωρίς την ανάγκη του τελεστή της τελείας). Με τον ίδιο τρόπο, έχουν απόλυτη και άμεση γνώση των υπόλοιπων συναρτήσεων της τάξης. Στο μυαλό μας όταν χτίζουμε μια τάξη, θα πρέπει να έχουμε ότι χτίζουμε ένα αυτοσυνεπές αντικείμενο, με μεταβλητές (ποσότητες) και συναρτήσεις (ιδιότητες), το οποίο θα μπορεί να χειρίζεται τον εαυτό του.

11.4 Κλήση της τάξης

Για προσπέλαση των δημόσιων δεδομένων χρησιμοποιήσαμε τον τελεστή της τελείας. Το ίδιο κάνουμε και για τις δημόσιες συναρτήσεις. Σημαντικό να μην ξεχαστεί: την συνάρτηση πάντα συνοδεύουν οι παρενθέσεις, οι οποίες περιέχουν τη λίστα εισόδου. Εάν δεν υπάρχει λίστα εισόδου (όπως πχ με τη συνάρτηση **magn** παραπάνω, τότε οι παρενθέσεις παραμένουν, αλλά είναι άδειες.

11.4.1 Παράδειγμα

Η τάξη για διάνυσμα με δύο συναρτήσεις που περιγράψαμε παραπάνω, και το πλήρες πρόγραμμα που την δηλώνει και την καλεί.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`class vector {`	Ορισμός τάξης `vector`.
`  double x, y, z;`	3 ιδιωτικές μεταβλητές.
`public:`	
`  void set(double, double, double);`	Δημόσια συνάρτηση για εισαγωγή τιμών.
`  double magn();`	Δημόσια συνάρτηση για υπολογισμό
`};`	του μέτρου.
`void vector::set (double x1, double y1,`	Δήλωση της συνάρτησης `set` που ανήκει
`  double z1) {`	στην τάξη `vector`.
`  x = x1;`	Ανάθεση στα εσωτερικά `x`, `y`, `z`
`  y = y1;`	των μεταβλητών εισόδου.
`  z = z1;`	
`}`	
`double vector::magn() {`	Δήλωση της συνάρτησης `magn` που ανήκει στην τάξη `vector`.
`  return sqrt(x*x + y*y +z*z);`	Υπολογισμός μέτρου διανύσματος.
`}`	
`int main ( ) {`	
`  vector v, u;`	Δήλωση δύο αντικειμένων–διανυσμάτων.
`  double a, b, c;`	Δήλωση τριών ρητών.
`  cout << "Δώστε συνιστώσες "`	
`  cin >> a >> b >> c;`	Εισαγωγή συνιστωσών 1ου διανύσματος.
`  v.set(a, b, c);`	Ανάθεση 1ου διανύσματος.
`  cout << "Δώστε συνιστώσες "`	
`  cin >> a >> b >> c;`	Εισαγωγή συνιστωσών 2ου διανύσματος.
`  u.set(a,b,c);`	Ανάθεση 2ου διανύσματος.
`  cout << "Το μέτρο του πρώτου: "`	
`  cout << v.magn() << endl;`	Εκτύπωση μέτρου 1ου διανύσματος.

```
    cout << "Το μέτρο του δεύτερου: "
    cout << u.magn() << endl;                Εκτύπωση μέτρου 2ου διανύσματος.
}
```

11.5 Συνάρτηση δόμησης

Στο προηγούμενο παράδειγμα χρησιμοποιήσαμε την συνάρτηση `set` για να δώσουμε αρχικές τιμές στο διάνυσμα. Σε πολλές περιπτώσεις μπορεί να χρειαστεί να δίνουμε τις ίδιες αρχικές τιμές κάθε φορά που δηλώνουμε ένα νέο αντικείμενο. Στο διάνυσμα για παράδειγμα, ίσως θέλουμε εξ ορισμού (αυτόματα) με την δήλωσή του να παίρνει συγκεκριμένες αρχικές τιμές, πχ (0,0,0), χωρίς να απαιτείται να καλεστεί καμιά συνάρτηση. Η C++ μας δίνει αυτήν την δυνατότητα μέσω της λεγόμενης *συνάρτησης δόμησης*. Η συνάρτηση δόμησης:

1. Έχει υποχρεωτικά το ίδιο όνομα με την τάξη.
2. Δεν έχει τύπο (καλείται αυτόματα κατά την δήλωση αντικειμένου, και άρα εξ ορισμού δεν δύναται να επιστρέφει κάτι).
3. Μπορεί να έχει ή όχι μεταβλητές εισόδου.

Παράδειγμα:
Στην τάξη `vector`, έστω ότι θέλουμε να παίρνει εξ ορισμού τις αρχικές τιμές (0,0,0). Τότε η τάξη γράφεται:

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`#include <cmath>`	
`using namespace std;`	
`class vector {`	Ορισμός τάξης `vector`.
`  double x, y, z;`	3 ιδιωτικές μεταβλητές,
`public:`	
`  vector();`	Πρωτότυπο συνάρτησης δόμησης.
`  ⋮`	
υπόλοιπες συναρτήσεις μέλη	
`  ⋮`	
`};`	
`vector::vector ( ) {`	Δήλωση συνάρτησης δόμησης.
`  x = y = z = 0;`	
`}`	

Με αυτό τον τρόπο, κάθε φορά που δηλώνουμε ένα νέο διάνυσμα, αυτομάτως θα λαμβάνει την συγκεκριμένη αρχική τιμή. Για παράδειγμα, η δήλωση:

```
vector v;
```

δημιουργεί τώρα ένα διάνυσμα με τιμή (0,0,0). Βεβαίως, κατόπιν μέσα στο πρόγραμμα μπορούμε να ξαναλλάξουμε την τιμή του χρησιμοποιώντας την εσωτερική συνάρτηση `set`.

Η συνάρτηση δόμησης μπορεί να δηλωθεί και με λίστα εισόδου. Για παράδειγμα:

```
vector::vector ( double x1, double y1, double z1 ) {
    x = x1;
    y = y1;
    z = z1;
}
```

Η συνάρτηση δόμησης τώρα παίζει τον ίδιο ρόλο που παίζει και η συνάρτηση set. Η διαφορά είναι ότι ενώ τη συνάρτηση set πρέπει να την καλέσουμε μέσα στο πρόγραμμα, η συνάρτηση δόμησης εκτελείται αυτόματα κατά τη δήλωση του αντικείμενου. Στη δήλωση βέβαια πρέπει να δώσουμε και τις συνιστώσες. Για παράδειγμα οι δηλώσεις:

`vector v(0,0,0);`	Δήλωση διανύσματος με συντεταγμένες (0,0,0).
`vector u(1,1,1);`	Δήλωση διανύσματος με συντεταγμένες (1,1,1).

11.6 Παραδείγματα

Περνάμε πλέον σε πιο ολοκληρωμένα παραδείγματα σχεδιασμού λογισμικού. Προφανώς οι προτεινόμενες λύσεις είναι στην πλέον απλουστευμένη τους μορφή, και καθιστούν απλά εκπαιδευτικά παραδείγματα.

11.6.1 Οργάνωση και χειρισμός αντικειμένων σε εμπορικό κατάστημα

Θέλουμε να διαχειριστούμε την αποθήκη και το ισοζύγιο ενός εμπορικού καταστήματος. Θα χρειαστούμε ένα λογισμικό που για κάθε αντικείμενο θα έχει πληροφορίες για:

1. Τις τιμές αγοράς και πώλησης.
2. Την ποσότητα αντικειμένων που υπάρχουν στην αποθήκη.
3. Το ισοζύγιο (κέρδος ή ζημιά) στο ταμείο για κάθε αντικείμενο.

Επίσης, το λογισμικό θα πρέπει να προβλέπει συγκεκριμένες πράξεις–ενέργειες:

1. Πώληση αντικειμένου.
2. Αγορά αντικειμένου.
3. Αναπροσαρμογή τιμών αγοράς και πώλησης.
4. Έλεγχο κατάστασης–ισοζυγίου.

Η πρώτη λοιπόν αναγκαιότητα είναι η δημιουργία μιας τάξης. Θα την ονομάζουμε `item`.

Θα περιέχει ιδιωτικές μεταβλητές:

- `cost` (για το κόστος αγοράς προϊόντος)
- `price` (για την τιμή πώλησης προϊόντος)
- `stock` (για τον αριθμό των προϊόντων που παραμένουν στην αποθήκη)
- `cash` (για το εμπορικό ισοζύγιο έσοδα–έξοδα του προϊόντος).

Θα περιέχει δημόσιες συναρτήσεις:

- `sell` (που εκτελεί την πώληση προϊόντων)
- `buy` (που εκτελεί την αγορά προϊόντων)
- `reset` (που επιτρέπει την αναπροσαρμογή των τιμών αγοράς και πώλησης)
- `info` (που επιδεικνύει πληροφορίες για το προϊόν)
- Θα χρησιμοποιήσουμε και μια συνάρτηση δόμησης, που να δίνει εξ ορισμού αρχική τιμή μηδέν σε όλες τις μεταβλητές.

Κώδικας C++	*Σχολιασμός*

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

class item {
    double cost;                                  Τιμή αγοράς.
    double price;                                 Τιμή πώλησης.
    int stock;                                    Αριθμός στην αποθήκη.
    double cash;                                  Εμπορικό ισοζύγιο.
public:
    item();                                       Συνάρτηση δόμησης.
    void sell();                                  Συνάρτηση για πώληση.
    void buy();                                   Συνάρτηση για αγορά.
    void reset();                                 Συνάρτηση επαναπροσδιορισμού τιμών.
    void info();                                  Συνάρτηση εκτύπωσης.
};

item::item ( ) {
    cost = price = stock = cash = 0;              Συνάρτηση δόμησης, θέτει εξ ορισμού
}                                                 αρχική τιμή μηδέν σε όλες τις μεταβλητές.

void item::sell ( ) {                             Συνάρτηση για πώληση.
    int n;
    cout << "Πόσα κομμάτια για πώληση;\n";
    cin >> n;
    if ( stock<n )                                Εάν δεν υπάρχουν αρκετά στην αποθήκη
        cout << "Δυστυχώς έχουμε μόνο"            δεν μπορεί να γίνει η πώληση.
             << stock << endl;
    else {                                        Εάν υπάρχουν αρκετά στην αποθήκη.
        stock -= n;                               Αφαίρεσε n από την αποθήκη.
        cash += n * price;                        Πρόσθεσε n*τιμή πώλησης στο ισοζύγιο
    }                                             του προϊόντος.
}

void item::buy ( ) {                              Συνάρτηση για αγορά.
    int n;
    cout << "Πόσα κομμάτια για αγορά;\n"  ;
    cin >> n;
    stock += n;                                   Πρόσθεσε n στην αποθήκη.
    cash -= n * cost;                             Αφαίρεσε n*τιμή αγοράς από το ισοζύγιο
}                                                 του προϊόντος.
```

```
void item::reset ( ) {
    cout << "Παλιές τιμές αγοράς/πώλησης ";
    cout << cost << " " << price << endl;    Τύπωσε τις παλιές τιμές.
    cout << "Εισάγετε νέες τιμές\n";
    cin >> cost >> price;                    Διάβασε τις νέες τιμές.
}

void item::info ( ) {
    cout << stock << " έχουν μείνει στην αποθήκη\n";
    cout << cost << " ευρώ η τιμή αγοράς\n";
    cout << price << " ευρώ η τιμή πώλησης\n";
    cout << cash << " ευρώ το ισοζύγιο\n";
}
```

Και το κυρίως πρόγραμμα:

Κώδικας C++	*Σχολιασμός*
`int main ( ) {`	
`    item a[1000];`	Δήλωση πίνακα 1000 αντικειμένων.
`    int i, j;`	
`    do {`	Συνέχισε να εκτελείς τα παρακάτω.
`        cout << "Αριθμός προϊόντος;\n";`	
`        cin >> i;`	Διάβασε τον αριθμό του αντικειμένου.
`        cout << "Εντολή; 0-4: e-s-b-r-i"\n;`	
`        cin >> j;`	Διάβασε τον αριθμό της εντολής που θα εκτελεστεί: 0 για exit, 1 για sell, 2 για buy, 3 για reset, 4 για info.
`        switch (j) {`	
`                case 1:`	
`                    a[i].sell();`	Για j=1, πώληση.
`                    break;`	
`                case 2:`	
`                    a[i].buy();`	Για j=2, αγορά.
`                    break;`	
`                case 3:`	
`                    a[i].reset();`	Για j=3, επαναπροσδιορισμός τιμών.
`                    break;`	
`                case 4:`	
`                    a[i].info();`	Για j=4, πληροφορίες.
`                    break;`	
`        }`	
`    }`	

```
    while( j > 0 );                          Όσο η εντολή j είναι μεγαλύτερη του 0.
}
```

Δίνεται ένα παράδειγμα εκτέλεσης του παραπάνω προγράμματος:

```
Αριθμός προϊόντος;
1
Εντολή; 0-4: e-s-b-r-i
3
Παλιές τιμές αγοράς/πώλησης
0  0
Εισάγετε νέες τιμές
3 4

Αριθμός προϊόντος;
1
Εντολή; 0-4: e-s-b-r-i
2
Πόσα κομμάτια προς αγορά;
5

Αριθμός προϊόντος;
1
Εντολή; 0-4: e-s-b-r-i
1
Πόσα κομμάτια προς πώληση;"
3

Αριθμός προϊόντος;
1
Εντολή; 0-4: e-s-b-r-i
4
2 έχουν μείνει στην αποθήκη
3 ευρώ η τιμή αγοράς
4 ευρώ η τιμή πώλησης
-3 ευρώ το ισοζύγιο
```

11.6.2 Λίστα προτεραιότητας

Θέλουμε να δημιουργήσουμε μια λίστα αναμονής, του τύπου "πρώτο μπαίνει–πρώτο βγαίνει". Θα δέχεται αριθμούς, και θα τους προσθέτει στο τέλος της ουράς. Όταν ζητείται ένας αριθμός από την λίστα, θα αφαιρείται από μπροστά. Η λίστα θα πρέπει να θυμάται ποιανού είναι η σειρά να τραβηχθεί, καθώς και σε ποια θέση θα πρέπει να μπει ένας νέος αριθμός.

Από μεταβλητές θα χρειαστούμε:

1. Έναν πίνακα για να αποθηκεύουμε τα νούμερα.
2. Έναν ακέραιο για να ξέρουμε σε ποια θέση βάζουμε τον επόμενο.
3. Έναν ακέραιο για να ξέρουμε από ποια θέση τραβάμε τον επόμενο.

Από συναρτήσεις θα χρειαστούμε:

1. Μια συνάρτηση όταν ένας νέος αριθμός μπαίνει στη λίστα.
2. Μια συνάρτηση για όταν βγάζουμε έναν αριθμό από την λίστα (και οι δύο πρέπει να ελέγχουν μην ξεπεραστούν όρια).
3. Μια συνάρτηση να δίνει αρχικές τιμές.

Θα ονομάσουμε την τάξη μας `queue` (λίστα αναμονής). Θα περιέχει:

1. Έναν πίνακα ακεραίων `q` που θα σώζονται τα νούμερα που εισέρχονται στην λίστα.
2. Έναν ακέραιο `sloc` που δείχνει ποια είναι η επόμενη θέση στην οποία θα πρέπει να προσθέσουμε έναν νέο αριθμό.
3. Έναν ακέραιο `rloc` που δείχνει από ποια θέση τραβάμε τον επόμενο αριθμό.

Επίσης, θα περιέχει:

1. Μια συνάρτηση δόμησης που θα δίνει αρχικές τιμές μηδέν στις εσωτερικές μεταβλητές `sloc` και `rloc`.
2. Μια συνάρτηση `qput` που θα προσθέτει έναν αριθμό.
3. Μια συνάρτηση `qget` που θα τραβάει έναν αριθμό.

Κώδικας C++	*Σχολιασμός*

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

class queue {
    int q[1000];                          Πίνακας 1000 στοιχείων, ο οποίος είναι ουσιαστικά η λίστα προτεραιότητας.
    int sloc, rloc;                       Η θέση του επόμενου στην αναμονή, και η θέση του επόμενου να τραβηχθεί.
public:
    queue();                              Συνάρτηση δόμησης.
    void qput(int i);                     Συνάρτηση που προσθέτει έναν ακέραιο στη λίστα.
    int qget();                           Συνάρτηση που τραβάει έναν ακέραιο από την λίστα.
};

queue::queue ( ) {
    rloc = sloc = 0;
}                                         Συνάρτηση δόμησης.

void queue::qput ( int i ) {              Συνάρτηση που προσθέτει έναν αριθμό στο τέλος της ουράς.
    if ( sloc==1000 ) {                   sloc είναι η επόμενη κενή θέση. Εάν αυτή είναι στο τέλος του πίνακα, τότε επιστροφή.
        cout << "Η λίστα είναι γεμάτη\n";
        return;
    }
    q[sloc]=i;                            Αλλιώς πρόσθεσε τον αριθμό στην θέση sloc.
    sloc++;                               Αύξησε την sloc κατά ένα.
}

int queue::qget ( ) {                     Συνάρτηση που τραβάει έναν αριθμό από την αρχή της ουράς.
    if ( rloc==sloc ) {                   Εάν η αρχή της ουράς συμπίπτει με το τέλος
        cout << "Η λίστα είναι άδεια\n";  της, τότε είναι άδεια και άρα επιστρέφουμε.
        return 0;
    }
    int r = q[rloc];                      Αλλιώς τράβα έναν αριθμό από τη θέση rloc.
    rloc++;                               Αύξησε την rloc κατά ένα.
```

```
    return r;                                   Επέστρεψε τον αριθμό.
}
```

Και ένα παράδειγμα προγράμματος που την καλεί:

```
int main ( ) {
    queue a;
    int i, j;
    do {                                        Συνέχισε να εκτελείς τα παρακάτω.
        cout << "Επόμενη εντολή; 0-2 e-p-g ";
        cin >> i;                               Διάβασε την επόμενη εντολή.
        switch (i) {
          case 1:                               Εάν η εντολή είναι 1.
            cout<< "Δώστε νέο αριθμό ";
            cin >> j;                           Διάβασε τον νέο αριθμό.
            a.qput(j);                          Πρόσθεσέ τον στην ουρά.
            break;
          case 2:                               Εάν η εντολή είναι 2.
            cout << "Επόμενος αριθμός στην ουρά ";
            cout << a.qget() << endl;           Τράβα έναν αριθμό
            break;
        }
    }
    while ( i > 0 );                            Όσο η εντολή είναι θετική.
}
```

Ένα παράδειγμα εκτέλεσης του παραπάνω προγράμματος:

```
Επόμενη εντολή; 0-2 e-p-g
1
Δώστε αριθμό
4

Επόμενη εντολή; 0-2 e-p-g
1
Δώστε αριθμό
7

Επόμενη εντολή; 0-2 e-p-g
2
Επόμενος αριθμός στην ουρά 4

Επόμενη εντολή; 0-2 e-p-g
```

```
1
Δώστε αριθμό
5

Επόμενη εντολή; 0-2 e-p-g
2
Επόμενος αριθμός στην ουρά 7

Επόμενη εντολή; 0-2 e-p-g
2
Επόμενος αριθμός στην ουρά 5

Επόμενη εντολή; 0-2 e-p-g
2
Η λίστα είναι άδεια 0

Επόμενη εντολή; 0-2 e-p-g
1
Δώστε αριθμό
15

Επόμενη εντολή; 0-2 e-p-g
2
Επόμενος αριθμός στην ουρά 15
```

11.7 Υπερφόρτωση τελεστών

Σε προηγούμενα κεφάλαια εξετάσαμε εκτενώς τις σημαντικές δυνατότητες που προσφέρει η C++ με την υπερφόρτωση συναρτήσεων. Προφανώς το ίδιο ισχύει και για τις συναρτήσεις–μέλη μιας τάξης. Η διαδικασία είναι ακριβώς η ίδια και δεν θα την εξετάσουμε ξανά εδώ. Αντίθετα, θα εξετάσουμε μια άλλη συναρπαστική ιδιότητα της C++, την δυνατότητα να υπερφορτώσουμε τους τελεστές.

Οι συνήθεις τελεστές είναι και αυτοί κατά μιαν έννοια παρόμοιοι με συναρτήσεις. Για παράδειγμα, οι τελεστές `+`, `-`, `*`, `/`. Σκεφτείτε την πράξη `c = a+b`. Ο τελεστής `+` δέχεται δύο ορίσματα, τους `a` και `b`, έναν από τα αριστερά και έναν από τα δεξιά του, και επιστρέφει το αποτέλεσμα της πράξης, το οποίο και ανατίθεται στον `c`. Εάν η πράξη πρόσθεση ήταν μια συνάρτηση όπως τις έχουμε γνωρίσει μέχρι τώρα, πχ ονόματι `sum`, θα γράφαμε:

```
c = sum(a,b);
```

έχοντας ορίσει την συνάρτηση `sum` ως:

```
double sum(double a, double b) {
    return a + b;
}
```

Σε αυτό και το προηγούμενο κεφάλαιο ορίσαμε νέες σύνθετες μεταβλητές με την μορφή δομών ή τάξεων. Καθώς οι συνήθεις πράξεις (πχ `+`, `-`, `*`) δεν επιτρέπονται, ορίσαμε και συναρτήσεις που εκτελούν τις συνήθεις πράξεις για τις σύνθετες μεταβλητές, πχ ορίσαμε τη συνάρτηση `sum` για την πρόσθεση δύο διανυσμάτων.

```
vector sum ( vector v, vector u ) {
    vector w;
    w.x = v.x + u.x;
    w.y = v.y + u.y;
    w.z = v.z + u.z;
    return w;
}
```

Η ίδια συνάρτηση επίσης υπερφορτώθηκε ώστε να μπορεί να χρησιμοποιηθεί και για άλλου τύπου μεταβλητές, πχ μιγαδικούς κτλ. Η ερώτηση λοιπόν είναι η εξής: ορίσαμε και υπερφορτώσαμε συναρτήσεις για να εκτελούν συνήθεις πράξεις σε σύνθετες μεταβλητές. Μπορούμε να κάνουμε το ανάποδο, δηλαδή να υπερφορτώσουμε, κατά κάποια έννοια να ανα–προγραμματίσουμε, τους συνήθεις τελεστές έτσι ώστε να εκτελούν αυτοί τις πράξεις μεταξύ των σύνθετων μεταβλητών, απαλείφοντας έτσι την αναγκαιότητα για δήλωση συναρτήσεων; Η ίδια ερώτηση θα μπορούσε να διατυπωθεί και ως εξής: ορίσαμε την συνάρτηση `sum`. Θα μπορούσαμε να την ονομάσουμε `+`, δηλαδή το ίδιο όνομα με τον απλό τελεστή της πρόσθεσης; Η απάντηση είναι ναι, μέσω της δυνατότητας που μας δίνει η C++ με την υπερφόρτωση τελεστών.

Δηλώνουμε πάλι την τάξη `vector`. Για να δηλώσουμε την υπερφόρτωση ενός τελεστή, πχ τον τελεστή `+`, τον δηλώνουμε ως συνάρτηση ονόματι `operator+`. Αυτός ο ορισμός δεν περιορίζεται μόνο στους συνήθεις τελεστές, μπορούμε να δημιουργήσουμε και νέους τελεστές. Έστω ότι θέλουμε να υπερφορτώσουμε τους τελεστές `+`, `-`, `*` για διανύσματα. Η τάξη `vector` τώρα γράφεται:

```
class vector {
public:
    double x, y, z;
    vector();
    void set ( double, double, double );
    double operator* ( vector );
    vector operator+ ( vector );
    vector operator- ( vector );
};
```

Παρατηρήστε ότι τώρα οι x, y, z συντεταγμένες έχουν οριστεί να είναι δημόσιες. Αυτό είναι απαραίτητο τώρα που θέλουμε να προγραμματίσουμε πράξεις μεταξύ δύο διανυσμάτων. Εάν ήταν ιδιωτικές, η συνάρτηση–τελεστής θα είχε άμεση πρόσβαση μόνο στο ένα διάνυσμα (το δικό της), αλλά όχι στο άλλο. Για να καταλάβουμε καλύτερα, ας ορίσουμε τον τελεστή +.

```
vector vector::operator+ ( vector v ) {
    vector u;
    u.x = x + v.x;
    u.y = y + v.y;
    u.z = z + v.z;
    return u;
}
```

Η παραπάνω συνάρτηση μπορεί να φαίνεται περίεργη. Για καλύτερη κατανόηση, θα κάνουμε δύο βήματα πίσω, και θα ξαναγράψουμε τη συνάρτηση–τελεστή + σαν απλή συνάρτηση ονόματι sum. Μόλις κατανοήσουμε πως λειτουργεί η απλή συνάρτηση, θα είναι πιο εύκολο να κατανοήσουμε πως λειτουργεί και η συνάρτηση–τελεστής.

α) Με εξωτερική συνάρτηση

Έστω η εξωτερική συνάρτηση sum (που δεν ανήκει στην τάξη)

```
vector sum ( vector v, vector u ) {
    vector w;
    w.x = v.x + u.x;
    w.y = v.y + u.y;
    w.z = v.z + u.z;
    return w;
}
```

Μέσα στο κυρίως πρόγραμμα θα γράφαμε:

```
vector a, b, c;          Δήλωση τριών διανυσμάτων.
c = sum(a,b);            Το c είναι το άθροισμα του a και b.
```

β) Με συνάρτηση–μέλος

Τι αλλάζει εάν η sum γίνει συνάρτηση–μέλος της τάξης; Σε αυτήν την περίπτωση, αναγκαστικά καλείται μέσω κάποιου αντικειμένου με τον τελεστή της τελείας. Ως συνάρτηση–μέλος, δηλαδή, δεν είναι αυτόνομη, αλλά άρρηκτα δεμένη με το ίδιο το αντικείμενο που την καλεί. Αυτό επηρεάζει

την δήλωσή της; Η απάντηση είναι ναι, την επηρεάζει και πολύ μάλιστα. Αφού καλείται μέσω ενός αντικειμένου, τότε οι x, y, z συνιστώσες αυτού είναι αυτομάτως γνωστές. Τότε, χρειάζονται στην είσοδο μόνο οι συνιστώσες του δεύτερου διανύσματος. Άρα για την συνάρτηση–μέλος sum

```
vector vector::sum ( vector v ) {
    vector w;
    w.x = x + v.x;
    w.y = y + v.y;
    w.z = z + v.z;
    return w;
}
```

όπου για το πρώτο διάνυσμα χρησιμοποιούμε άμεσα τις x, y, z συντεταγμένες. Πως χρησιμοποιείται αυτή η συνάρτηση στο κυρίως πρόγραμμα;

```
vector a, b, c;                 Δήλωση τριών διανυσμάτων.
c = a.sum(b);                   Το c είναι το άθροισμα του a και b.
```

Δηλαδή καλούμε τη sum μέσω του διανύσματος a με τον τελεστή της τελείας, και εισάγουμε από την λίστα μεταβλητών το διάνυσμα b.

γ) Με υπερφόρτωση τελεστή

Τώρα είμαστε έτοιμοι για την υπερφόρτωση του τελεστή +. Ο ορισμός δόθηκε παραπάνω. Όσον αναφορά την κλήση, η μόνη διαφορά με την συνάρτηση–μέλος sum είναι ότι με τον τελεστή δεν χρειάζεται η τελεία και οι παρενθέσεις. Πολύ απλά, το διάνυσμα που καλεί τον τελεστή γράφεται όπως και πριν από τα αριστερά του τελεστή, και το εισερχόμενο διάνυσμα από τα δεξιά. Στο κυρίως πρόγραμμα:

```
vector a, b, c;                 Δήλωση τριών διανυσμάτων.
c = a + b;                      Το c είναι το άθροισμα του a και b.
```

Δηλαδή το .sum ανάμεσα στα a και b αντικαθίσταται πολύ απλά με το +. Δείτε πόσο απλή είναι πλέον η έκφραση της πρόσθεσης δύο διανυσμάτων.

Δίνουμε τώρα και τους ορισμούς των τελεστών - και *

Αφαίρεση διανυσμάτων

```
vector vector::operator- ( vector v ) {
    vector u;
    u.x = x - v.x;
    u.y = y - v.y;
    u.z = z - v.z;
    return u;
}
```

Εσωτερικό γινόμενο διανυσμάτων

```
double vector::operator* ( vector v ) {
    return x*v.x + y*v.y +z*v.z;
}
```

Όπως είπαμε και προηγουμένως, η διαδικασία υπερφόρτωσης τελεστή δεν εξαντλείται μόνο σε υπάρχοντες τελεστές, αλλά επεκτείνεται και στον ορισμό νέων τελεστών. Ένα καλό παράδειγμα, είναι ο ορισμός ενός νέου τελεστή για το εξωτερικό γινόμενο δύο διανυσμάτων. Θα επιλέξουμε το σύμβολο ^ για το εξωτερικό γινόμενο, αφήνοντας το * για το εσωτερικό γινόμενο. Το εξωτερικό γινόμενο δύο διανυσμάτων είναι διάνυσμα το ίδιο, και ορίζεται ως

$$\vec{a} \times \vec{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

Η δήλωση του τελεστή ^

```
vector vector::operator^ ( vector v ) {
    vector u;
    u.x = y * v.z - z * v.y;
    u.y = z * v.x - x * v.z;
    u.z = x * v.y - y * v.x;
    return u;
}
```

Γινόμενο διανύσματος με ρητό

```
vector vector::operator* ( double m ) {
    vector u;
    u.x = x * m;
    u.y = y * m;
    u.z = z * m;
    return u;
}
```

11.7.1 Παραδείγματα

11.7.1.1 Διανυσματικός λογισμός

Πλήρες πρόγραμμα C++ που επιτρέπει πράξεις με διανύσματα. Για παράδειγμα, πρόγραμμα που υπολογίζει την έκφραση:

$$(\vec{v}_1 \times \vec{v}_2) \times (\vec{v}_1 \times \vec{v}_3) \cdot |(\vec{v}_1 + \vec{v}_2) \times (\vec{v}_1 - \vec{v}_3)|$$

για τρία διανύσματα $\vec{v}_1$, $\vec{v}_2$, $\vec{v}_3$.

Στο παρακάτω, όπως θα παρατηρήσετε, έχει γίνει εκτενής υπερφόρτωση συναρτήσεων και τελεστών, ώστε να είναι όσο το δυνατόν πιο εύχρηστη και διαφανής η τάξη.

Κώδικας C++	*Σχολιασμός*
`#include <iostream>`	
`using namespace std;`	
`class vector {`	
`public:`	
`  double x, y, z;`	
`  vector();`	Συνάρτηση δόμησης.
`  vector(double, double, double);`	Υπερφορτωμένη συνάρτηση δόμησης.
`  void set(double, double, double);`	Συνάρτηση–μέλος set.
`  vector operator+(vector);`	Υπερφόρτωση τελεστή αθροίσματος διανυσμάτων.
`  vector operator-(vector);`	Υπερφόρτωση τελεστή διαφοράς διανυσμάτων.
`  double operator*(vector);`	Υπερφόρτωση τελεστή εσωτερικού γινομένου διανυσμάτων.
`  vector operator*(double);`	Υπερφόρτωση τελεστή γινομένου διανύσματος–ρητού.
`  vector operator^(vector);`	Υπερφόρτωση τελεστή εξωτερικού γινομένου διανυσμάτων.
`  double magn();`	Συνάρτηση–μέλος για μέτρο διανύσματος.
`};`	
`vector::vector ( ) {`	Συνάρτηση δόμησης που δίνει αρχικές τιμές (0,0,0) στο διάνυσμα.
`  x = y = z  = 0;`	
`}`	
`vector::vector (double x1, double y1,`	Υπερφορτωμένη συνάρτηση δόμησης που δίνει τις επιθυμητές (μη–μηδενικές) αρχικές τιμές στο διάνυσμα.
`  double z1) {`	
`  x = x1;`	
`  y = y1;`	
`  z = z1;`	

```
}

void vector::set(double x1, double y1,
                 double z1) {
    x = x1;
    y = y1;
    z = z1;
}

vector vector::operator+ (vector v){
    vector u;
    u.x = x + v.x;
    u.y = y + v.y;
    u.z = z + v.z;
    return u;
}

vector vector::operator- (vector v){
    vector u;
    u.x = x - v.x;
    u.y = y - v.y;
    u.z = z - v.z;
    return u;
}

double vector::operator* (vector v){
    return x*v.x + y*v.y +z*v.z;
}

vector vector::operator* (double m){
    vector u;
    u.x = x * m;
    u.y = y * m;
    u.z = z * m;
    return u;
}

vector vector::operator^ (vector v){
    vector u;
    u.x = y * v.z - z * v.y;
    u.y = z * v.x - x * v.z;
    u.z = x * v.y - y * v.x;
```

Συνάρτηση–μέλος set που δίνει τις επιθυμητές (μη–μηδενικές) αρχικές τιμές στο διάνυσμα.

Υπερφόρτωση τελεστή αθροίσματος διανυσμάτων.

Υπερφόρτωση τελεστή διαφοράς διανυσμάτων.

Υπερφόρτωση τελεστή εσωτερικού γινομένου διανυσμάτων.

Υπερφόρτωση τελεστή γινομένου διανύσματος–ρητού.

Υπερφόρτωση τελεστή εξωτερικού γινομένου διανυσμάτων.

```
    return u;
}

double vector::magn ( ) {                Συνάρτηση–μέλος για μέτρο διανύσματος
    return sqrt(x*x + y*y + z*z);
}

int main ( ) {
    double x, y, z;
    cout << "Εισάγετε διάνυσμα v1 ";
    cin >> x >> y >> z;                  Εισαγωγή συνιστωσών.
    vector v1(x,y,z);                    Δήλωση διανύσματος.

    cout << "Εισάγετε διάνυσμα v2 ";
    cin >> x >> y >> z;                  Εισαγωγή συνιστωσών.
    vector v2(x,y,z);                    Δήλωση διανύσματος.

    cout << "Εισάγετε διάνυσμα v3 ";
    cin >> x >> y >> z;                  Εισαγωγή συνιστωσών.
    vector v3(x,y,z);                    Δήλωση διανύσματος.

    vector r = ((v1^v2)^(v1^v3)) *
               ((v1+v2)^(v1-v3)).magn();

    cout << "Το ζητούμενο διάνυσμα είναι ";
    cout << r.x <<" "<< r.y <<" "<< r.z << endl;
}
```

Παράρτημα Α

Προτεραιότητες τελεστών

ΑΑ	*Τελεστής*	*Σημαίνει*	*Προσεταιριστικότητα*
1	`( )`	Κλήση συνάρτησης	Από αριστερά προς δεξιά
	`[ ]`	Δείκτης πίνακα	
	`->`	Αναφορά μέλους δομής από δείκτη	
	`.`	Μέλος δομής	
	`++`	Επιθεματικός τελεστής μοναδιαίας αύξησης	
	`--`	Επιθεματικός τελεστής μοναδιαίας μείωσης	
2	`!`	Λογική αντιστροφή	Από δεξιά προς αριστερά
	`++`	Προθεματικός τελεστής μοναδιαίας αύξησης	
	`--`	Προθεματικός τελεστής μοναδιαίας μείωσης	
	`+`	Πρόσημο θετικών αριθμών	
	`-`	Πρόσημο αρνητικών αριθμών	
	`(τύπος)`	Αλλαγή τύπου	
	`*`	Περιεχόμενο δείκτη	
	`&`	Διεύθυνση μεταβλητής	
3	`*`	Πολλαπλασιασμός	Από αριστερά προς δεξιά
	`/`	Διαίρεση	
	`%`	Υπόλοιπο διαίρεσης	
4	`+`	Πρόσθεση	Από αριστερά προς δεξιά
	`-`	Αφαίρεση	
5	`<`	Μικρότερο	Από αριστερά προς δεξιά
	`<=`	Μικρότερο ή ίσο	
	`>`	Μεγαλύτερο	
	`>=`	Μεγαλύτερο ή ίσο	
6	`==`	Ίσο	Από αριστερά προς δεξιά
	`!=`	Διάφορο	
7	`&&`	Λογικό “και”	Από αριστερά προς δεξιά
8	`||`	Λογικό “ή”	Από αριστερά προς δεξιά

9	=	Ανάθεση τιμής	Από δεξιά προς αριστερά
	+=	Ανάθεση με πρόσθεση	
	-=	Ανάθεση με αφαίρεση	
	*=	Ανάθεση με πολλ/μο	
	/	Ανάθεση με διαίρεση	
	%=	Ανάθεση με υπόλοιπο διαίρεσης	