κεφάλαιο

2

Μεταβλητές και Εκχωρήσεις

Ο στόχος μας σε αυτό το κεφάλαιο:

Να κατανοήσουμε την έννοια της *μεταβλητής* και μερικούς βασικούς τρόπους χειρισμού και χρήσης στο πρόγραμμα.

Προσδοκώμενα αποτελέσματα:

Θα μπορείς να γράψεις προγραμματάκια που θα έχουν περισσότερες δυνατότητες λόγω της χρήσης μεταβλητών.

Έννοιες κλειδιά:

- *μεταβλητή*
- *όνομα, τιμή* και *τύπος μεταβλητής*
- *μέγεθος και διεύθυνση μεταβλητής*
- *δήλωση μεταβλητής*
- *εντολή εκχώρησης*
- *εντολή εισόδου*
- *cin*
- *αριθμητικοί τύποι*
- *τυποθεώρηση*
- *σταθερές με όνομα*

Περιεχόμενα:

Εισαγωγικές Παρατηρήσεις – Η Ανάγκη για Μεταβλητές:
Ωραία και απλά τα προγραμματάκια με τις σταθερές αλλά... δεν μας βοηθάνε και πολύ!

- Ας πούμε ότι γράφουμε εντολές για υπολογισμό μιας παράστασης· μετά θέλουμε να ξαναϋπολογίσουμε την τιμή της παράστασης αφού αλλάξουμε δύο τιμές. Πρέπει να ξαναγράψουμε τις εντολές. Δεν θα ήταν καλύτερο να γράψουμε την παράσταση χρησιμοποιώντας *μεταβλητές* –δηλαδή αντικείμενα που μπορούμε να αλλάξουμε την τιμή τους– αντί για σταθερές;
- Αν ο υπολογισμός είναι μεγάλος θέλουμε να τον κάνουμε κατά τμήματα και να φυλάγουμε κάπου –σε *μεταβλητές*– τα ενδιάμεσα αποτελέσματα.
- Αν όταν γράφουμε το πρόγραμμα δεν ξέρουμε ορισμένες τιμές καλό θα ήταν να γράψουμε τις εντολές υπολογισμού με βάση κάποιες *μεταβλητές* που οι τιμές τους θα ορίζονται (στοιχεία εισόδου) κατά τη διάρκεια της εκτέλεσης.

Χρειαζόμαστε λοιπόν μια οντότητα, ας την πούμε **μεταβλητή** (variable), που η *τιμή* της μπορεί να αλλάζει.

Σε μια διαδικασιακή γλώσσα προγραμματισμού υψηλού επιπέδου (όπως η C++, η C, η C#, η Pascal, η Algol, η Fortran κλπ) *μια μεταβλητή υλοποιείται με μια θέση της μνήμης, που το περιεχόμενό της, η* **τιμή** (value) *της μεταβλητής, μπορεί να αλλάζει με την εκτέλεση του προγράμματος*. Μέσα στο πρόγραμμά μας χειριζόμαστε τη μεταβλητή με το **όνομά** της (name). Ο τρόπος που οργανώνεται η μνήμη για να φιλοξενήσει την τιμή της μεταβλητής καθορίζεται από τον **τύπο** της (type). Ο τύπος έχει σχέση με το τι είδος τιμών μπορεί να φιλοξενήσει: ακέραιους, πραγματικούς, κείμενα κλπ.

Το *όνομα*, η *τιμή* και ο *τύπος* είναι τα τρία *χαρακτηριστικά υψηλού επιπέδου* μιας μεταβλητής. Υπάρχουν δύο ακόμη *χαρακτηριστικά χαμηλού επιπέδου*:

- Το **μέγεθός** της (size), δηλαδή ο αριθμός *ψηφιολέξεων* (bytes) που απαιτούνται για την αποθήκευσή της.
- Η θέση ή η **διεύθυνσή** της (address) στη μνήμη.

Κατ' αρχήν δεν μας ενδιαφέρει πόσες ψηφιολέξεις απαρτίζουν μια μεταβλητή ούτε σε ποια διεύθυνση της μνήμης υλοποιείται. Δυστυχώς, αργότερα, αυτό θα αποδειχθεί ψέμα!

2.1 Μεταβλητές και Τύποι

Υπάρχουν γλώσσες (π.χ. Fortran) που δεν απαιτούν από τον προγραμματιστή να δηλώνει όλες τις μεταβλητές που θα χρησιμοποιήσει. Η C++ απαιτεί να έχουμε *δηλώσει* μια μεταβλητή πριν τη χρησιμοποιήσουμε. Η C (όπως και η Pascal, η Ada κλπ) απαιτεί να δηλώνουμε τις μεταβλητές μας στην αρχή του προγράμματός μας. Αυτή είναι μια καλή συνήθεια για αρχάριους προγραμματιστές και προς το παρόν θα την ακολουθούμε.

♦ ***Με τη δήλωση μιας μεταβλητής καθορίζουμε τα τρία χαρακτηριστικά υψηλού επιπέδου μιας μεταβλητής. Μπορεί όμως να μην ορίσουμε την (αρχική) τιμή.***

Μια **δήλωση μεταβλητών** (variable declaration) αποτελείται από:

- το όνομα ενός *τύπου στοιχείων*,
- μια λίστα ονομάτων μεταβλητών,
- τον χαρακτήρα `';'`.

Παραδείγματα

Τα παρακάτω είναι δηλώσεις μεταβλητών:

```
double  pi, e, distance, length;
int  i, counter, j, number, integer;
```

Τα ίδια θα μπορούσαν να δηλωθούν και ως εξής:

```
int  i;
double  pi, e;
int  metrntns, j, number, integer;
double  distance, length;
```

Τα `int` και `double`, που βλέπεις στο παραπάνω παράδειγμα είναι ονόματα τύπων της C++. Τι είναι ένας τύπος;

- *Είναι ένα σύνολο τιμών*, μαζί με
- τις πράξεις που μπορούμε να κάνουμε σε αυτές τις τιμές.

Όταν λοιπόν δηλώνουμε, για παράδειγμα, "`int i`" εννοούμε ότι η τιμή της μεταβλητής *i* θα ανήκει *πάντοτε* στο `int`. Δηλαδή, οι μεταβλητές, με τη δήλωσή τους, αποκτούν μια ιδιότητα που παραμένει *αναλλοίωτη* όσο εκτελείται το πρόγραμμα. Κάθε τύπος έχει ένα δικό του τρόπο οργάνωσης της μνήμης. Δηλαδή, πόση μνήμη χρειάζεται για την παράσταση μιας τιμής και πώς ερμηνεύονται οι τιμές των δυαδικών ψηφίων που υπάρχουν εκεί.

Σημείωση: ▶

Στον τύπο `int` έχουμε ακέραιες τιμές ενώ στον τύπο `double` έχουμε πραγματικές. Στη συνέχεια τα λέμε εν εκτάσει.◀

Η C++ σου επιτρέπει μαζί με τη δήλωση να δώσεις στη μεταβλητή σου και αρχική τιμή:

Παραδείγματα

Θα μπορούσαμε να δώσουμε στις δηλώσεις μας αρχικές τιμές ως εξής:

```
double  pi, e, distance( 0.0 ), length( 1.0 );
int  i, counter( 0 ), j, number( 0 ), integer( 375 );
```

ή ως εξής:

```
double  pi, e, distance = 0.0, length = 1.0;
int  i, counter = 0, j, number = 0, integer = 375;
```

Από την άποψη χρήσης μνήμης. μπορούμε να πούμε ότι με τη δήλωση των μεταβλητών κάνουμε –πάντοτε– δύο πράγματα:

- ζητούμε μνήμη που θα χρησιμοποιηθεί από τις μεταβλητές του προγράμματός μας,
- απαιτούμε συγκεκριμένο τρόπο οργάνωσης αυτής της μνήμης –τον τρόπο που αντιστοιχεί στον συγκεκριμένο τύπο.

Αν θέλουμε, έχουμε τη δυνατότητα να

- δίνουμε αρχικές τιμές στις μεταβλητές που δηλώνουμε.

Τις δηλώσεις μεταβλητών επεξεργάζεται ο μεταγλωττιστής. Χωρίς να βρίσκεσαι πολύ μακριά από την πραγματικότητα, σκέψου αυτήν την επεξεργασία ως εξής: κατά τη διάρκεια της μεταγλώττισης, κάνει έναν πίνακα όπου βάζει, για κάθε μεταβλητή, την διεύθυνση στην μνήμη και τον τύπο της –δηλαδή, πώς είναι οργανωμένη αυτή η θέση. Ο πίνακας του Σχ. 2-1, θα μπορούσε να αντιστοιχεί στις δηλώσεις του προηγούμενου παραδείγματος. Αυτόν τον πίνακα χρησιμοποιεί ο ΗΥ στην διάρκεια της εκτέλεσης του προγράμματος για να ξέρει πού θα βρεί την κάθε μεταβλητή και πώς θα ερμηνεύσει τα δυαδικά ψηφία που θα βρεί αποθηκευμένα.

Όταν αρχίζει η εκτέλεση του προγράμματος:

- αν, για κάποια μεταβλητή, έχει καθοριστεί με τη δήλωση της και αρχική τιμή, αυτή αποθηκεύεται στην αντίστοιχη θέση της μνήμης,

όνομα	διεύθυνση	τύπος	μέγεθος	τιμή
counter	1245028	`int`	4	0
distance	1245044	`double`	8	0
e	1245052	`double`	8	`???`
i	1245032	`int`	4	`???`
integer	1245016	`int`	4	375
j	1245024	`int`	4	`???`
length	1245036	`double`	8	1
number	1245020	`int`	4	0
pi	1245060	`double`	8	`???`

Σχ. 2-1 Οι πληροφορίες που χρειάζονται για να ορίζουμε ή να παίρνουμε την τιμή μιας μεταβλητής: *διεύθυνση* και *τύπος*. Στην τελευταία στήλη βλέπεις την τιμή που θα έχουν οι μεταβλητές μας όταν αρχίζει η εκτέλεση του προγράμματος. Όσες πήραν (αρχική) τιμή κατά τη δήλωση έχουν την τιμή αυτή. Οι άλλες είναι αόριστες (τιμή: `???`).

- αν δεν έχει καθοριστεί τιμή κατά τη δήλωση η τιμή της δεν είναι ορισμένη· λέμε ότι η μεταβλητή είναι *αόριστη*.

Ως αρχική τιμή μιας μεταβλητής μπορείς να δώσεις, όχι μόνον μια σταθερά, αλλά και μια παράσταση που, όμως, θα περιέχει σταθερές ή μεταβλητές που η τιμή τους έχει καθοριστεί πιο πριν. Π.χ.:

```
int x( 1 ), y( 2 ), z( x+y );
double q( 5 ), r( 1+sqrt(q) );
```

Στη συνέχεια θα δούμε πώς μπορούμε να δώσουμε τιμή σε μια μεταβλητή στη διάρκεια της εκτέλεσης του προγράμματος.

2.1.1 Ονόματα (Αναγνωριστικά)

Οι γλώσσες προγραμματισμού δίνουν τη δυνατότητα στον προγραμματιστή να δίνει συμβολικά ονόματα στα διάφορα αντικείμενα της γλώσσας που χρησιμοποιεί. Συνήθως τα λέμε **ονόματα** ή **αναγνωριστικά** ή **ταυτότητες** (identifiers).

Όπως ήδη είπαμε:

όνομα μεταβλητής = *όνομα*;

Ένα όνομα της C++ σχηματίζεται από χαρακτήρες της γλώσσας ως εξής:

- Ο πρώτος χαρακτήρας είναι γράμμα του Λατινικού αλφαβήτου ή ο *χαρακτήρας της υπογράμμισης* (**"_"**, underscore).
- Οι υπόλοιποι χαρακτήρες είναι γράμματα του Λατινικού αλφαβήτου ή ψηφία, δηλ. αλφαριθμητικοί χαρακτήρες.

όνομα = *μη ψηφίο* | *όνομα*, *μη ψηφίο* | *όνομα*, *ψηφίο*;
μη ψηφίο = "_" | *γράμμα*;

Ισχύουν ακόμα και οι εξής περιορισμοί:

- Μέσα σε ένα όνομα δεν μπορούμε να έχουμε κενά ή αλλαγές γραμμής.
- Δεν επιτρέπονται ως ονόματα οι **λέξεις-κλειδιά** (keywords) της C++[1].

  Ακόμη: *απόφευγε τη χρήση αναγνωριστικών που περιέχουν διπλή υπογράμμιση* ("__").

[1] Για τις λέξεις-κλειδιά δες το Παράρ. C.

Υπάρχουν ακόμη μερικά ονόματα που η χρήση τους επιτρέπεται κατ' αρχήν αλλά στην πραγματικότητα απογορεύεται: είναι αυτά τα οποία ήδη χρησιμοποιούνται από τη C++, όπως τα: `sqrt`, `cout` κλπ. Έχεις, π.χ., δικαίωμα να δηλώσεις:

```
double sqrt;
```

αλλά, με τον τρόπο αυτό, το όνομα `sqrt` αναφέρεται σε μια μεταβλητή και όχι πια στη συνάρτηση για την τετραγωνική ρίζα[2]. Προς το παρόν λοιπόν, δέξου τον κανόνα:

- ***Δεν επιτρέπεται η χρήση των `main`, `sqrt` και άλλων προκαθορισμένων αναγνωριστικών της γλώσσας παρά μόνο για τις προκαθορισμένες χρήσεις τους.***

Παραδείγματα δεκτών αναγνωριστικών είναι τα εξής:

```
ALFA  OMEGA  _omikron  Nikos  prwto_programma
QWERTY  aSdFgH  program_1
```

Δεν είναι δεκτά όμως τα:

`12thprogram` :	αρχίζει από ψηφίο
`SYNTHESH$` :	περιέχει το χαρακτήρα '`$`'
`int` :	είναι λέξη-κλειδί
`Prwto Progr` :	περιέχει κενό
`MakryAnagnw ristiko` :	περιέχει αλλαγή γραμμής

Όπως λέγαμε και στο προηγούμενο κεφάλαιο, η C++ (όπως και η C) διαφέρει από την πλειοψηφία των γλωσσών προγραμματισμού στο εξής: *ξεχωρίζει τα πεζά από τα κεφαλαία γράμματα*. Έτσι, τα ονόματα: `Paradeigma`, `PARADEIGMA`, `PaRaDeIgMa`, `pARADEIGMa` είναι *διαφορετικά* για τη C++.

Η C++ δεν βάζει όριο για τον αριθμό χαρακτήρων που μπορεί να έχει ένα όνομα. Αλλά, τέτοια όρια μπορεί να βάζουν οι διάφορες υλοποιήσεις.

Ένας άλλος, εύλογος, περιορισμός είναι ο εξής: *κάθε όνομα αναφέρεται σε ένα μόνον αντικείμενο του προγράμματός σου*. Δεν μπορείς, για παράδειγμα, να δίνεις το ίδιο όνομα σε δυο μεταβλητές του προγράμματός σου[3].

Υπακούοντας στους παραπάνω κανόνες εφάρμοσε και τον παρακάτω:

- ***Κάθε όνομα θα πρέπει να έχει σχέση με το αντικείμενο που παριστάνει.***

Έστω π.χ. ότι θέλεις, μέσα σε κάποιο πρόγραμμα να υπολογίσεις την περιφέρεια ενός κύκλου από την ακτίνα του. Αντί να τα παραστήσεις με τα ονόματα *a*, *b* είναι πολύ καλύτερο να χρησιμοποιήσεις τα *perifereia*, *aktina* (φυσικά και τα *C* και *r* είναι μια χαρά για κάποιον που ξέρει λίγη Γεωμετρία).

Αυτός ο κανόνας αποβλέπει στο να κάνει τα προγράμματα **ευανάγνωστα** (readable). Ένα πρόγραμμα πρέπει να είναι καλογραμμένο όχι μόνο για να το διαβάζει εύκολα ένας τρίτος, αλλά και ο ίδιος ο προγραμματιστής που το σύνταξε.

Η περιγραφή του τι κάνει ένα πρόγραμμα και πώς το κάνει, λέγεται **τεκμηρίωση** (documentation) του προγράμματος. Ένα πρόγραμμα που γράφεται σωστά, δεν έχει ανάγκη από πολλές περιγραφές και λέμε ότι είναι **αυτοτεκμηριωμένο** (self-documented). Η τεκμηρίωση του προγράμματος είναι ένα σημαντικό κεφάλαιο του προγραμματισμού και θα αναφερόμαστε σ' αυτό ξανά και ξανά. Η σωστή επιλογή των αναγνωριστικών, είναι βασική αρχή αυτού του κεφαλαίου.

---

[2] Όπως θα δούμε, μπορείς να «ξαναβρείς» την τετραγωνική ρίζα ως "`std::sqrt`".

[3] Αυτός ο κανόνας θα ανατραπεί (μερικώς) και θα αντικατασταθεί από έναν πιο ακριβή, αργότερα.

2.2 Εκχώρηση – Μεταβλητές στις Παραστάσεις

Αφού είδαμε πώς δηλώνουμε τις μεταβλητές μας ας δούμε τώρα τι κάνουμε με αυτές στο πρόγραμμά μας. Αλλά πρώτα να υπενθυμίσουμε ότι:

♦ ***Για να χρησιμοποιηθεί κάποια μεταβλητή θα πρέπει να έχει δηλωθεί προηγουμένως.***

Αυτό είναι που απαιτεί η C++. Εμείς, όπως είπαμε, θα κάνουμε κάτι που απαιτείται ή συνηθίζεται στις περισσότερες γλώσσες προγραμματισμού: *θα βάζουμε τις δηλώσεις στην αρχή του προγράμματος*.

Οι δουλειές που κάνουμε με μια μεταβλητή είναι:

- αποθήκευση κάποιας πληροφορίας, δηλαδή: καθορισμός της τιμής μιας μεταβλητής,
- ανάκτηση και χρησιμοποίηση της αποθηκευμένης πληροφορίας, δηλ.: χρήση της μεταβλητής.

Ας ξεκινήσουμε με την πρώτη δουλειά και με ένα παράδειγμα: Θέλουμε να φυλάξουμε την τιμή της παράστασης $1+\sqrt{5}$ για να τη χρησιμοποιήσουμε στη συνέχεια. Αν έχουμε δηλώσει:

```
double x;
```

μπορούμε να δώσουμε την εντολή:

```
x = 1 + sqrt( 5 );
```

Τι σημαίνει;

- Υπολόγισε την τιμή της παράστασης: `1 + sqrt(5)`.
- Μετά εκχώρησε την τιμή στη μεταβλητή *x* (ή αλλιώς αποθήκευσε αυτήν την τιμή στη θέση της μνήμης που έχεις ονομάσει *x*).

Ένας άλλος τρόπος να το δούμε είναι ο εξής: Από εδώ και πέρα, όπου βρίσκεις το όνομα της μεταβλητής *x* θα το *αντικαθιστάς* με την τιμή που έχει η παράσταση `1 + sqrt(5)`. Γι' αυτό θα δεις ότι η εντολή αυτή *λέγεται* και **εντολή αντικατάστασης**.

Θα ονομάζουμε εντολές τέτοιου είδους **εντολές εκχώρησης** (assignment statements) και ας δούμε τα συντακτικά και τα νοηματικά τους. Προς το παρόν, μια εντολή εκχώρησης θα έχει την εξής μορφή:

όνομα μεταβλητής "=" *παράσταση*

όπου "=" είναι το σύμβολο (ή ο *τελεστής*) της εκχώρησης[4]. Η εκτέλεσή της γίνεται ως εξής:

- Υπολογίζεται η τιμή της παράστασης.
- Η τιμή μετατρέπεται στον τύπο της μεταβλητής.
- Η τιμή φυλάγεται ως τιμή της μεταβλητής.

Παραδείγματα

Οι παρακάτω είναι εντολές εκχώρησης:

```
pi = 4*atan(1);
e = exp(1);
number = 4*10 + 2*30;
integer = (100 % 51)*4 + 7;
```

Οι παρακάτω *δεν είναι εντολές εκχώρησης*:

```
x + 1 = 7;        x + y = 12;        sqrt(x) = 12;
```

(αριστερά από το "=" θα έπρεπε να υπάρχει μεταβλητή).

Ας δούμε τώρα πώς χρησιμοποιούμε μια τιμή που έχουμε αποθηκεύσει στη μνήμη: ποια είναι η θέση της μεταβλητής μέσα σε μια παράσταση; *Οι μεταβλητές μπαίνουν στην*

[4] Γιατί «θα ονομάζουμε»; Δεν είναι εντολή εκχώρησης; Η C++ δεν έχει εντολή εκχώρησης! Και αριστερά του "=" μπορεί να υπάρχει ολόκληρη παράσταση! Αργότερα θα καταλάβεις...

παράσταση όπως ακριβώς οι σταθερές –ως πρωτογενείς παραστάσεις. Έτσι, τα παρακάτω είναι παραστάσεις:

```
integer + length*3        i + j/2
sqrt(pi) + 1.0
log(e+1) - 1/pow(e,2)
```

Στον υπολογισμό της παράστασης κάθε μεταβλητή αντικαθίσταται από την τιμή που έχει εκείνη τη στιγμή. Δηλαδή, ο ΗΥ παίρνει το περιεχόμενο της αντίστοιχης θέσης της μνήμης χωρίς όμως και να το αλλάζει. Αν η παράσταση είναι μέσα σε μια "**cout <<** …", τότε η τιμή της παράστασης τυπώνεται συμφώνως με αυτά που ξέρουμε. Να λοιπόν ένα απλό προγραμματάκι:

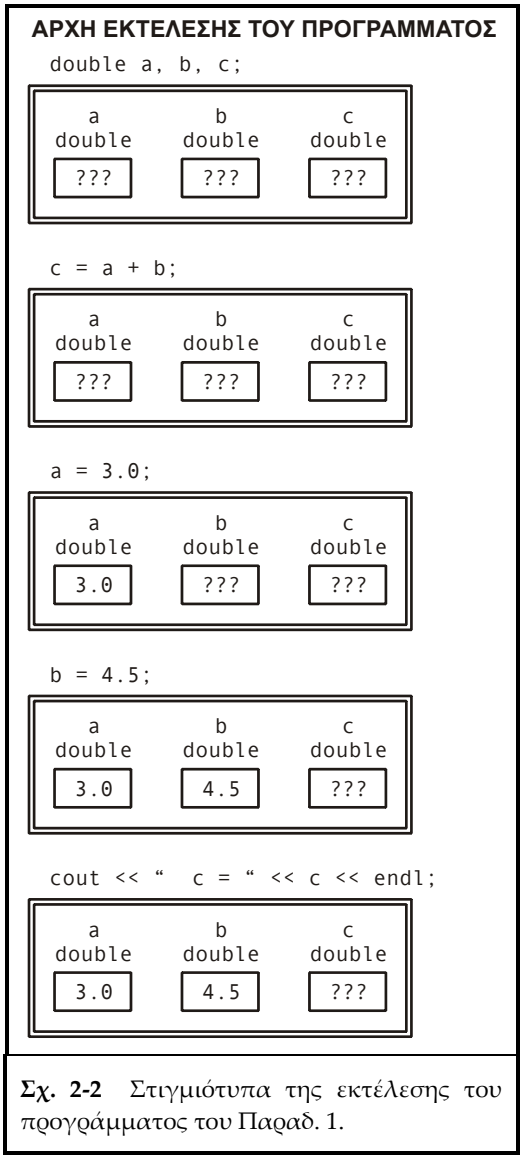
```
#include <iostream>
using namespace std;
int main()
{
   int k;

   k = 3;
   cout << " k = " << k << endl;
}
```

που δίνει:

```
 k = 3
```

Στην αρχή δηλώσαμε μόνο μια μεταβλητή τύπου **int** με το όνομα *k*. Με την πρώτη εντολή δίνουμε στη μεταβλητή *k* την τιμή 3. Στη **cout <<**… έχουμε δυο ορίσματα: τον ορμαθό **" k = "** και την παράσταση **k**. Από το πρώτο όρισμα τυπώνεται το «κείμενο» "␣k␣=␣" και από την παράσταση η τιμή της μεταβλητής *k*.

Ας δούμε δύο παραδείγματα, πολύ χαρακτηριστικά για την εντολή εκχώρησης:



Σχ. 2-2 Στιγμιότυπα της εκτέλεσης του προγράμματος του Παραδ. 1.

Παράδειγμα 1

Στα μαθηματικά μπορούμε να γράφουμε: $c = a + b$ και όταν πιο κάτω δώσουμε $a = 3$, $b = 4.5$, να είναι φανερό ότι $c = 7.5$. Αλλά, τι αποτέλεσμα θα δώσει το πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   double a, b, c;

   c = a + b;
   a = 3;  b = 4.5;
   cout << " c = " << c << endl;
}
```

Αποτέλεσμα:

```
  c = 4.24613e-314
```

Τι είναι αυτό;! Για να δούμε: Όταν αρχίζει η εκτέλεση του προγράμματος, οι τιμές των *a*, *b*, *c* δεν είναι ορισμένες. *Αυτό δεν σημαίνει ότι είναι* 0· απλώς έχουν τυχαίες τιμές, που βγαίνουν από τις τυχαίες καταστάσεις στις οποίες έτυχε να βρεθούν τα αντίστοιχα δυαδικά ψηφία της μνήμης. Πρώτη εντολή του προγράμματος είναι η:

```
  c = a + b;
```

Και πώς εκτελείται;

Παρακολούθησε την εκτέλεση στο Σχ. 2-2. Ο ΗΥ «πηγαίνει» στις θέσεις της μνήμης *a*, *b*, παίρνει αυτά (τα «σκουπίδια» ή «**???**» όπως τα παριστάνουμε) που έχουν μέσα, τα προσθέτει και αποθηκεύει το αποτέλεσμα στην *c*. Οι εντολές "**a = 3; b = 4.5**" εκτελούνται *μετά* την "**c = a + b**" και δεν την επηρεάζουν.

Πάντως ο μεταλωττιστής έδωσε προειδοποίηση (warning): «**possible use of 'a' before definition in function main()**» (δυνατή χρήση της "a" πριν από τον ορισμό της στη συνάρτηση **main()**, παρόμοια προειδοποίηση και για τη 'b')

Τα διδάγματα από τα παραπάνω είναι τα εξής:

- Ο υπολογισμός μιας παράστασης γίνεται με τις τιμές που έχουν οι μεταβλητές όταν γίνεται ο υπολογισμός. *Οι κατοπινές αλλαγές τιμών στις μεταβλητές δεν επηρεάζουν υπολογισμούς που έγιναν.*
- Μη χρησιμοποιείς μεταβλητές που δεν έχεις ορίσει τις τιμές τους.

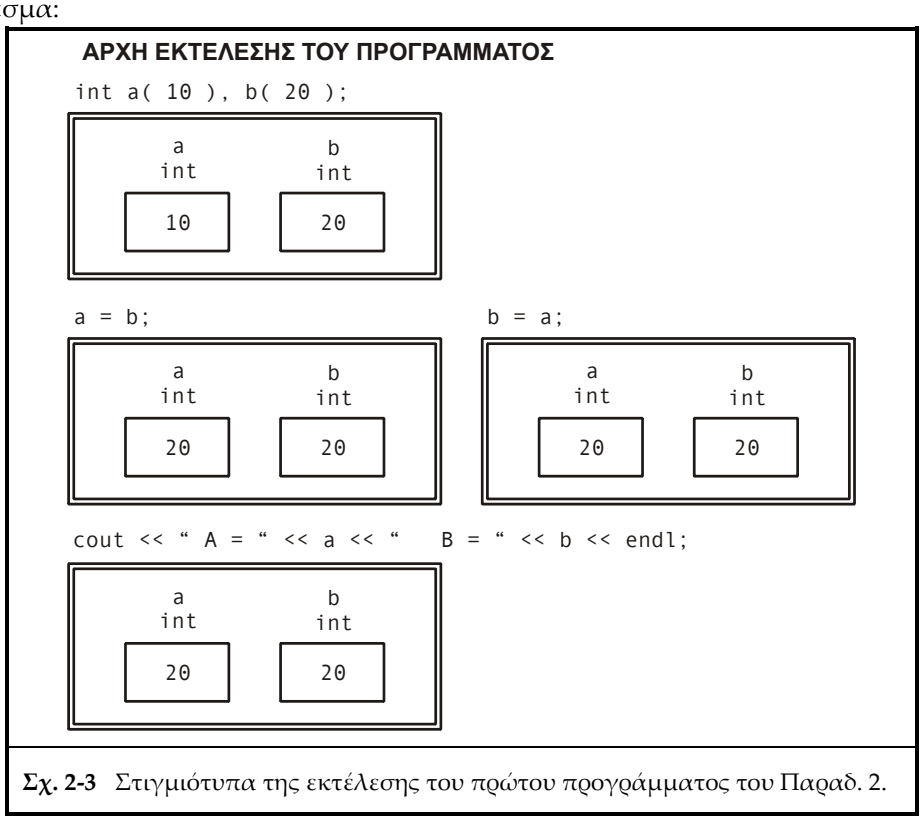
Παράδειγμα 2

Κάτι που θα χρειάζεται να κάνεις αρκετά συχνά στα προγράμματά σου, είναι να *αντιμεταθέτεις τις τιμές δύο μεταβλητών*. Ας δούμε πώς γίνεται κάτι τέτοιο.

Ας πούμε ότι θέλουμε να αντιμεταθέσουμε τις τιμές δύο μεταβλητών *a* και *b*. Η πρώτη ιδέα είναι: βάλε όπου *a* το *b* και όπου *b* το *a*. Ας δούμε τι θα βγάλει το πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   int a( 10 ), b( 20 );

   a = b;  b = a;
   cout << " a = " << a << "   b = " << b << endl;
}
```

Αποτέλεσμα:



Σχ. 2-3 Στιγμιότυπα της εκτέλεσης του πρώτου προγράμματος του Παραδ. 2.

```
 a = 20   b = 20
```

Ατυχία! Τι έγινε; Όταν εκτελέστηκε η εντολή `a = b`, η *a* πήρε την τιμή 20, που είχε η *b*. Στη συνέχεια εκτελέστηκε η εντολή `b = a` και η *b* (ξανα)πήρε την τιμή 20 από την *a*. Δες και το Σχ. 2-3 που τα δείχνει με πιο παραστατικό τρόπο.

Τι πρέπει να κάνουμε; Να «φυλάξουμε» κάπου την τιμή της *a*, πριν εκτελεστεί η "`a = b`" και από κει να δώσουμε μετά την τιμή στη *b*. Για τη φύλαξη θα χρειαστούμε μια βοηθητική μεταβλητή *s*, του ίδιου τύπου με τις *a*, *b*. Να το σωστό πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   int a( 10 ), b( 20 ), s;

   s = a;  a = b;  b = s;
   cout << " a = " << a << "   b = " << b << endl;
}
```

που μας δίνει:

```
 a = 20   b = 10
```

Το δίδαγμα και από εδώ είναι το εξής:

- ***Ο υπολογισμός μιας παράστασης γίνεται με τις τιμές που έχουν οι μεταβλητές όταν γίνεται ο υπολογισμός (η b πήρε την τιμή που είχε η a όταν εκτελέστηκε η εντολή `b = a`). Αν αλλάξεις την τιμή μιας μεταβλητής, η παλιά τιμή χάνεται (εκτός αν τη φυλάξεις εσύ σε κάποια άλλη θέση της μνήμης).***

Και τώρα μια ερώτηση: Γιατί είναι σωστό το πρόγραμμα που γράψαμε; Διότι δούλεψε το παράδειγμα; Ούτε να το σκέφτεσαι!

- ***Με ένα παράδειγμα ή με στιγμιότυπα εκτέλεσης μπορείς να ανακαλύψεις λάθη στο πρόγραμμά σου αλλά δεν μπορείς να αποδείξεις ότι το πρόγραμμα είναι σωστό.***

Στο επόμενο κεφάλαιο θα αποδείξουμε ότι το πρόγραμμα είναι σωστό χρησιμοποιώντας το *αξίωμα της εκχώρησης*.

2.3 Εισαγωγή Στοιχείων

Τα προγράμματα που είδαμε μέχρι τώρα, δουλεύουν με στοιχεία που ορίζονται μέσα σε αυτά και βγάζουν ορισμένα αποτελέσματα. Τέτοια προγράμματα είναι πολύ περιορισμένης χρησιμότητας γιατί δεν μπορούν να αλλάξουν τα στοιχεία με τα οποία γράφτηκαν.

Ένα πρόγραμμα μπορεί να επικοινωνεί με τον χρήστη του, *κατά τη διάρκεια της εκτέλεσής του*, με εντολές εισόδου που έχουν το συντακτικό:

`cin >>` *λίστα ορισμάτων*`;`

Προς το παρόν, η λίστα ορισμάτων θα απαρτίζεται από ονόματα μεταβλητών που διαχωρίζονται από "`>>`".

Παρομοίως με την έξοδο αποτελεσμάτων προς το ρεύμα `cout`, σε κάθε ΗΥ υπάρχει και μια προκαθορισμένη συσκευή –ή αρχείο– από όπου γίνεται είσοδος στοιχείων (default Input device (file))· σε κάθε υλοποίηση της C++, αυτή η συσκευή (αρχείο) συνδέεται με το πρόγραμμά μας με ένα προκαθορισμένο ρεύμα με το όνομα `cin`.

Αν δεν βάζεις την «τεμπέλικη» "`using namespace std`" να ξέρεις ότι θα πρέπει να δηλώνεις:

```
using std::cin;
```

Η "`cin >>` …" διαβάζει από το πληκτρολόγιο μια γραμμή με στοιχεία. Τι είναι μια γραμμή; Αν δουλεύεις σε τερματικό ενός ΗΥ (ή σε έναν μικροϋπολογιστή), μια γραμμή

είναι οι χαρακτήρες που γράφεις ανάμεσα σε δύο διαδοχικά πατήματα του πλήκτρου <enter>. Και τι περιέχει μια γραμμή με στοιχεία;

- Σε κάθε μεταβλητή της λίστας εισόδου πρέπει να αντιστοιχεί μια σταθερά (που μπορεί να έχει και πρόσημο) του ίδιου τύπου στη γραμμή με τα στοιχεία εισόδου.
- Το πρώτο στοιχείο της γραμμής αντιστοιχεί στην πρώτη μεταβλητή της λίστας εισόδου, το δεύτερο στοιχείο στη δεύτερη μεταβλητή κ.ο.κ.
- Το πλήθος των στοιχείων θα πρέπει να είναι τουλάχιστον ίσο με το πλήθος των μεταβλητών της λίστας εισόδου. Αν τα στοιχεία είναι λιγότερα από τις μεταβλητές, ο υπολογιστής θα περιμένει όλα τα στοιχεία, όσα <enter> και αν δώσεις. Αν είναι περισσότερα, όσα πλεονάζουν αγνοούνται από τη συγκεκριμένη εντολή.
- Τα στοιχεία θα πρέπει να διαχωρίζονται μεταξύ τους με ένα τουλάχιστον διάστημα. *Ο αριθμός των διαστημάτων που μεσολαβούν ανάμεσα στους αριθμούς δεν έχει σημασία*. Τα διαστήματα πριν από το πρώτο στοιχείο αγνοούνται.

  Το τι κάνει η εντολή εισόδου φαίνεται στο παρακάτω

Παράδειγμα 1

Γράφουμε το πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   int    k, j;
   double x, y;

   cin >> k >> j >> x >> y;
   cout << k << ' ' << j << ' '
        << x << ' ' << y << endl;
}
```

Αφού περάσει από το μεταγλωττιστή μας, το δίνουμε για εκτέλεση και βλέπουμε τον οθονοδείκτη (cursor) να περιμένει στην αρχή της πρώτης κενής γραμμής. Τώρα εκτελείται η εντολή `cin >> k >>`…. Ο ΗΥ περιμένει να του πληκτρολογήσουμε 4 τιμές. Πληκτρολογούμε λοιπόν:

```
17  -375      145.78   31e4
```

Στο τέλος πιέζουμε το πλήκτρο <enter>. Ο ΗΥ απαντάει αμέσως:

```
17 -375 145.78 310000
```

Η απάντηση ήρθε από την "`cout << k`…" όπου του ζητάμε να μας γράψει τις τιμές τεσσάρων μεταβλητών διαχωριζόμενες, ανά δύο, από ένα κενό. Τι έγινε όμως με τη "`cin >> k`…"; Ο πρώτος αριθμός που δώσαμε, το "`17`", έγινε τιμή της πρώτης μεταβλητής της λίστας εισόδου, δηλ. της *k*, ο δεύτερος, ο "`-375`", θα γίνει τιμή της *j*, ο τρίτος, ο "`145.78`", της *x* και ο τέταρτος, ο "`31e4`", της *y*, όπως φαίνεται παρακάτω:

```
  cin >> k >> j   >>    x   >>  y;
         ↑    ↑          ↑       ↑
        17  -375      145.78   31e4
```

Το ίδιο αποτέλεσμα παίρνουμε και με τα:

```
␣␣␣␣␣␣␣␣␣␣17␣-375␣␣␣␣␣␣␣␣␣145.78␣31E4
```

Αν όμως πληκτρολογήσουμε:

```
31e4    17  -375      145.78
```

κάνουμε λάθος! Η πρώτη τιμή προορίζεται για την *k*, που είναι τύπου `int` και εμείς δώσαμε "`31e4`"· η τιμή αυτή είναι ακέραιη, αλλά δεν είναι σταθερά τύπου `int`, όπως είδαμε στο προηγούμενο κεφάλαιο, §1.7, 1.8. Να λοιπόν τι διάβασε το πρόγραμμα όταν μεταγλωττίστηκε με τη gcc (Dev C++):

```
31 42 5.28418e-308 9.92631e-315
```

και να τι διάβασε όταν μεταγλωττίστηκε με τη Borland C++ v.5.5:

`31 0 2.12414e-314` (και runtime error)

Παρ' όλα αυτά, *σε μια μεταβλητή τύπου* `double` *μπορεί να αντιστοιχεί μια σταθερά τύπου* `int`.

Και τώρα ας ξαναεκτελέσουμε το πρόγραμμά μας, αλλά θα του δώσουμε ως στοιχεία εισόδου τα εξής:

`17  -375  145.78  31e4  123`

Αποτέλεσμα:

`17 -375 145.78 310000`

Τι έγινε με το "`123`"; Τίποτε απολύτως: το πρόγραμμα περίμενε να διαβάσει τέσσερις τιμές. Τις διάβασε και αγνόησε την πέμπτη! Αν ακολουθούσε άλλη εντολή εισόδου στη συνέχεια, θα ξεκινούσε την ανάγνωση από το "`123`".

Τι θα γίνει όμως αν βάλουμε τρεις τιμές; Ούτε να το σκέφτεσαι. Μπορεί να δίνεις κενά, να αλλάζεις γραμμή με το <enter>, αλλά ο ΗΥ είναι πιο επίμονος από σένα. Και θα επιμένει να πάρει όλες τις τιμές που περιμένει. Δοκίμασέ το!

Βάζοντας εντολές που διαβάζουν τιμές από το πληκτρολόγιο έχουμε τη δυνατότητα να καθορίσουμε (ή να αλλάξουμε) την τιμή μιας μεταβλητής την ώρα που εκτελείται το πρόγραμμα. Με την χρήση της μπορούμε να κάνουμε πιο «ευέλικτα» προγράμματα:

Παράδειγμα 2

Ας γράψουμε ένα προγραμματάκι που «διαβάζει» από το πληκτρολόγιο δύο ακέραιους και υπολογίζει το άθροισμά τους. Τώρα όμως έχουμε διδαχτεί από το προηγούμενο παράδειγμα: Θα βάλουμε το πρόγραμμά μας να δίνει κάποιο μήνυμα όταν θα περιμένει να πληκτρολογήσουμε στοιχεία εισόδου:

```
#include <iostream>
using namespace std;
int main()
{
   int a1, a2, sum;

   cout << "ΔΩΣΕ ΜΟΥ ΤΟΥΣ ΔΥΟ ΠΡΟΣΘΕΤΕΟΥΣ" << endl;
   cin >> a1 >> a2;
   sum = a1 + a2;
   cout << " (" << a1 << ") + (" <<  a2 << ") = "
        << sum << endl;
}
```

Μόλις αρχίσει η εκτέλεση του προγράμματός μας, θα δούμε πάνω στην οθόνη:

```
ΔΩΣΕ ΜΟΥ ΤΟΥΣ ΔΥΟ ΠΡΟΣΘΕΤΕΟΥΣ
_
```

Πληκτρολογούμε τα εξής:

`5  7`

και ο ΗΥ απαντάει:

`(5) + (7) = 12`

Είπαμε ότι το πρόγραμμά αυτό είναι ευέλικτο, γιατί τώρα *μπορούμε να ξαναζητήσουμε την εκτέλεσή του, χωρίς να χρειαστεί νέα μεταγλώττιση*, και αφού πληκτρολογήσουμε:

`57  -17`

να πάρουμε:

`(57) + (-17) = 40`

Όπως καταλαβαίνεις, αν αντί για τη `cin >> a1 >> a2` είχαμε τις: `a1 = 5;  a2 = 7` αυτό δεν θα ήταν δυνατό.

Αυτό ήταν εύκολο. Να δούμε κάτι πιο δύσκολο.

Παράδειγμα 3

Όπως ξέρεις, οι λύσεις της εξίσωσης: $ax^2 + bx + c = 0$ είναι οι:

$$x_{\pm}=\frac{-b\pm\sqrt{\Delta}}{2a} \quad \text{όπου: } \Delta = b^2 - 4ac$$

Το παρακάτω πρόγραμμα θα διαβάζει τα *a*, *b*, *c* και στη συνέχεια θα υπολογίζει και θα τυπώνει τις ρίζες της δευτεροβάθμιας εξίσωσης.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   double  a, b, c;
   double  x1, x2;
   double  delta;

   cout << "Δώσε τρεις πραγματικούς αριθμούς" << endl;
   cin >> a >> b >> c;
   cout << " a = " << a << "   b = " << b
        << "   c = " << c << endl;
   delta = b*b - 4*a*c;
   x1 = (-b + sqrt(delta))/(2*a);
   x2 = (-b - sqrt(delta))/(2*a);
   cout << " Λύση1 = " << x1
        << "   Λύση2 = " << x2 << endl;
}
```

Αφού τελειώσουμε με τη μεταγλώττιση, ζητούμε την εκτέλεση και βλέπουμε:

```
Δώσε τρεις πραγματικούς αριθμούς
_
```

Πληκτρολογούμε:

```
               1  3  -10
```

και παίρνουμε:

```
 a = 1   b = 3   c = -10
 Λύση1 = 2   Λύση2 = -5
```

Πολύ ωραία! Άλλη μια δοκιμή: Μόλις δούμε το μήνυμα: «Δώσε τρεις πραγματικούς αριθμούς» πληκτρολογούμε:

```
 2   3   4
```

και παίρνουμε (gcc - Dev C++):

```
 a = 2   b = 3   c = 4
 Λύση1 = -1.#IND   Λύση2 = -1.#IND
```

ή (Borland C++ 5.5):

```
 a = 2   b = 3   c = 4

sqrt: DOMAIN error

sqrt: DOMAIN error
 Λύση1 = +NAN   Λύση2 = +NAN
```

Τι είναι αυτό; Καλέσαμε (δύο φορές) τη συνάρτηση *sqrt()* με όρισμα εκτός του πεδίου ορισμού της. Δηλαδή, του ζητήσαμε να υπολογίσει την τετραγωνική ρίζα ενός αρνητικού αριθμού ($\Delta = 3^2 - 4\cdot2\cdot4 = -23$) και ο ΗΥ μας έβγαλε... τα είδες[5].

Σε επόμενο κεφάλαιο θα δούμε πώς, σε τέτοιες περιπτώσεις, μπορείς να παίρνεις τον έλεγχο στα χέρια σου (ή καλύτερα στο πρόγραμμά σου).

[5] Τι είναι το "**#IND**"; Θα τα πούμε αργότερα...

2.4 Σταθερές με Ονόματα

Στα επόμενα παραδείγματα θα χρησιμοποιήσουμε δύο πολύ γνωστές σταθερές: το π (= 3.14159...) και την επιτάχυνση της βαρύτητας κοντά στην επιφάνεια της Γής, *g* (= 9.81 *m/sec*²).

Μπορούμε βέβαια να γράφουμε

```
vP = -sqrt( 2*h*9.81 );
```

ή να απαλλάξουμε τον υπολογιστή από έναν πολλαπλασιασμό(!):

```
vP = -sqrt( h*19.62 );
```

Πόσο εύκολα θα καταλάβει κάποιος που διαβάζει το πρόγραμμα τι είναι αυτή η «μαγική σταθερά» "`9.81`" (και πολύ περισσότερο η "`19.62`"); Εκτός από αυτό, είναι πολύ πιθανό, σε μια διόρθωση ή επέκταση του προγράμματος να χρησιμοποιήσουμε ως τιμή της *g* το "`10`" ή το "`9.8`".

Η C++, όπως και οι άλλες γλώσσες προγραμματισμού, μας δίνουν τη δυνατότητα να χρησιμοποιήσουμε σταθερές με όνομα:

```
const double g( 9.81 ); // m/sec²
```

και να γράφουμε:

```
vP = -sqrt( 2*h*g );
```

Γιατί χρειαζόμαστε σταθερές με όνομα; Δεν θα μας έκανε μια μεταβλητή; Αντί για άλλη απάντηση, βάζουμε μια εντολή `g = 5.32` και δίνουμε το πρόγραμμά μας για μεταγλώττιση. Αποτέλεσμα: "`Cannot modify a const object in function main()`". Δηλαδή, ο μεταγλωττιστής δεν θα επιτρέψει μια κατά λάθος τροποποίηση τιμής της σταθεράς, πράγμα που δεν μπορεί να συμβαίνει για μια μεταβλητή.

Παρομοίως για το π μπορούμε να ορίσουμε:

```
const double pi( 4*atan(1) ); // = π
```

Εδώ βλέπεις ότι η τιμή της σταθεράς δίνεται από μια (σταθερή) παράσταση!

2.5 Οι Αριθμητικοί Τύποι της C++

Ο τύπος `int` της C++ είναι ένα υποσύνολο των ακεραίων· σε πολλές διαλέκτους της γλώσσας, είναι το σύνολο των ακεραίων αριθμών μεταξύ -2147483648 και 2147483647. Με το παρακάτω προγραμματάκι μπορείς να δεις τη μέγιστη και την ελάχιστη τιμή τύπου `int` στη C++ που χρησιμοποιείς:

```
#include <iostream>
#include <climits>
using namespace std;
int main()
{
   cout << "INT_MIN = " << INT_MIN << "    INT_MAX = "
        << INT_MAX << endl;
}
```

Στη gcc (Dev C++) και τη Borland C++ θα μας δώσει:

```
INT_MIN = -2147483648    INT_MAX = 2147483647
```

Αν λοιπόν έχεις δηλώσει, όπως είδαμε παραπάνω:

```
   int number;
```

έχουμε τη σιγουριά ότι η *number* έχει πάντοτε ακέραιη τιμή ανάμεσα στη μεγαλύτερη και τη μικρότερη που μπορεί να παρασταθεί στον τύπο `int`.

Εκτός από τον `int` η C++ μας δίνει και άλλους τύπους με ακέραιες τιμές. Στον Πίν. 2-1 βλέπεις τους **ακέραιους** (integral) τύπους. Διαλέγεις και παίρνεις: Θέλεις μεγάλες ακέραιες τιμές; Χρησιμοποίησε τον `long int` και αν θέλεις ακομα μεγαλύτερες διάλεξε τον `long`

Όνομα	Τιμές από .. μέχρι	Δυαδικά ψηφία
`signed char` ή `char`	-128 .. 127	8
`short int` ή `short`	-32768 .. 32767	16
`int`	-2147483648 .. 2147483647	32
`long int` ή `long`	-2147483648 .. 2147483647	32
`long long int`	–9223372036854775808 .. 9223372036854775807	64
`unsigned char`	0 .. 255	8
`unsigned short int`	0 .. 65535	16
`wchar_t`	0 .. 65535	16
`unsigned int`	0 .. 4294967295	32
`unsigned long int`	0 .. 4294967295	32
`unsigned long long int`	0 .. 18446744073709551615	64
`bool`	`false` (0) .. `true` (1)	8

Πίν. 2-1 Ακέραιοι τύποι στη **C++**. Ο `char` σε άλλες υλοποιήσεις είναι ίδιος με τον `signed char` (-128 .. 127) και σε άλλες σαν τον `unsigned char` (0 .. 255). Ο `wchar_t` είναι ίδιος με τον `unsigned short int`. Για τον `bool` θα τα πούμε αργότερα. Οι ακέραιοι `long long int` σε 64 δυαδικά ψηφία υπάρχουν στο C++11.

`long int`.[6] Έχεις μικρές ακέραιες τιμές: ο `short` θα σου λύσει το πρόβλημα και με οικονομία μνήμης (2 ψηφιολέξεις αντί για τις 4 του `long int`). Για πολύ μικρές τιμές έχεις τον `char` που πιάνει μια ψηφιολέξη μόνο![7] Για μη αρνητικές τιμές προτίμησε `unsigned int` ή `unsigned long int` ή `unsigned long long int`.

Εδώ πρόσεξε το εξής: τα μεγέθη εξαρτώνται από την υλοποίηση. Η C++ απαιτεί γενικώς:

`short int` "≤" `int` "≤" `long int` "≤" `long long int`

όπου το "≤" μπορείς να το καταλάβεις ως *μικρότερος ή ίσος χώρος αποθήκευσης* ή *μικρότερη ή ίση μέγιστη τιμή*. Αυτό δεν αποκλείει το να είναι ταυτόσημοι ανά δύο (ή και οι τρεις). Έτσι, στη Borland C++ (και στη gcc) έχουμε `short int` "<" `int` "=" `long int`.

Ο τύπος `double` είναι ένα υποσύνολο των πραγματικών· για την ακρίβεια ένα υποσύνολο των **ρητών**. Αυτό είναι συνέπεια του ότι μια τιμή τύπου `double` αποθηκεύεται στη μνήμη του ΗΥ σε πεπερασμένο πλήθος δυαδικών ψηφίων. Γενικώς, μια πραγματική τιμή, αποθηκεύεται στον ΗΥ με προσέγγιση. Ο τρόπος αποθήκευσης εκτέθηκε πολύ συνοπτικώς στην §1.7.1.

Το παρακάτω πρόγραμμα θα μας δώσει τη μέγιστη και την ελάχιστη θετική τιμή του τύπου `double` (και κάτι ακόμη):

```
#include <iostream>
#include <cfloat>
using namespace std;
int main()
{
   cout << "DBL_MIN = " << DBL_MIN
        << "    DBL_MAX = " << DBL_MAX << endl;
   cout << "DBL_EPSILON = " << DBL_EPSILON << endl;
}
```

Αποτέλεσμα:

```
DBL_MIN = 2.22507e-308    DBL_MAX = 1.79769e+308
DBL_EPSILON = 2.22045e-16
```

[6] Οι ακέραιοι σε 64 δυαδικά ψηφία υπάρχουν στο πρότυπο C++11. Θα τους βρεις στη gcc αλλά όχι στη Borland 5.5.

[7] Για τον `char`, τον `wchar_t` και τον `bool` θα τα πούμε στο Κεφ. 4.

Όνομα	Απόλυτη τιμή από .. μέχρι	Σημαντικά δεκ. ψηφία	Δυαδικά ψηφία
`double`	5.0e–324 .. 1.7e+308	15-16	64
`float`	1.5e–45 .. 3.4e+38	7-8	32
`long double`	1.9e–4951 .. 1.1e+4932	19-20	80

Πίν. 2-2 Οι τύποι κινητής υποδιαστολής της (Borland) C++.

Ένα χαρακτηριστικό ενός τύπου κινητής υποδιαστολής είναι ο ελάχιστος θετικός που αν προστεθεί στο 1 μας δίνει τιμή μεγαλύτερη από 1:

$$\varepsilon = min\{\, u : \mathbb{R} \mid u > 0 \wedge (1 + u)_{fp} \neq 1_{fp} \bullet u \,\}$$

(όπου το q_{fp} συμβολίζει την παράσταση της τιμής q στον `double`). Αυτό είναι το λεγόμενο **έψιλον** (epsilon) του τύπου κινητής υποδιαστολής. Για τον τύπο `double` της Borland C++:

$$\varepsilon = 2.220446e\text{-}16$$

Τι σημαίνει αυτό; Σημαίνει ότι όλες οι τιμές $1 + x$, για οποιοδήποτε $x \in [0,\varepsilon)$, αποθηκεύονται στον τύπο `double` όπως ακριβώς και το 1. Δες τις παρακάτω εντολές

```
cout << "DBL_EPSILON/2 = " << DBL_EPSILON/2 << endl;
cout << 1 + DBL_EPSILON/2 << endl;
```

που μας δίνουν:

```
DBL_EPSILON/2 = 1.11022e-016
1
```

Η δήλωση:

```
double  length;
```

μας εξασφαλίζει ότι η *length* θα έχει πάντοτε τιμή πραγματικό αριθμό από αυτούς που παριστάνονται στον `double`.

Η C++ έχει και άλλους τύπους **πραγματικών** ή **κινητής υποδιαστολής** (floating point) εκτός από τον `double`. Στον Πίν. 2-2 βλέπεις την ελάχιστη και τη μέγιστη απόλυτη τιμή για τον κάθε τύπο καθώς και το πλήθος των σημαντικών ψηφίων.

Όπως βλέπεις η C++ σου δίνει εργαλεία για να ξεπεράσεις –ακριβέστερα: για να *μετατοπίσεις*– τα προβλήματα των αριθμητικών τύπων.

Και ο τύπος μιας μεταβλητής καθορίζεται με τη δήλωσή της. Αν θέλουμε να καθορίσουμε τον τύπο μιας σταθεράς τι κάνουμε; Αυτό γίνεται με μια **κατάληξη** (suffix):

- σταθερά με κατάληξη "`l`" ή "`L`" είναι τύπου `long`,
- σταθερά με κατάληξη "`u`" ή "`U`" είναι τύπου `unsigned`,
- πραγματική σταθερά με κατάληξη "`f`" ή "`F`" είναι τύπου `float`,
- ακέραιη σταθερά χωρίς κατάληξη είναι τύπου `int`, ενώ πραγματική σταθερά χωρίς κατάληξη είναι τύπου `double`.

Παραδείγματα

```
123 (int)  123L (long)
123U (unsigned)  123LU (unsigned long)
1.23 (double)  1.23f (float)  1.23L (long double)
```

Ωραία λοιπόν, έχουμε τόσες επιλογές για τα δεδομένα μας. Και πώς διαλέγουμε τον τύπο; Δεν τα βάζουμε όλα `double` να τελειώνουμε; Όχι! Να γιατί:

- Η παράσταση των τιμών τύπου `double` (και `float` και `long double`) γίνεται *προσεγγιστικώς*, ενώ των τιμών ακέραιου τύπου με *ακρίβεια*.
- Οι πράξεις στους ακέραιους τύπους είναι ακριβείς, ενώ στους πραγματικούς τύπους (όπως ο `double`) δεν είναι, όπως είδες και πιο πάνω.

- Η επεξεργασία (π.χ. πράξεις) στοιχείων ακεραίων τύπων είναι ταχύτερη από αυτή των στοιχείων πραγματικού τύπου.

Αλλά:

- Στον τύπο `double` μπορούμε να παραστήσουμε *κλασματικές* τιμές ενώ στον τύπο `int` *μόνον ακέραιες*.
- Στον τύπο `double` μπορούμε να παραστήσουμε τιμές πολύ μεγαλύτερες, κατ' απόλυτη τιμή, από αυτές που μπορούμε να παραστήσουμε στον τύπο `int`.

Με βάση τα παραπάνω οδηγούμαστε στο συμπέρασμα ότι θα πρέπει να χρησιμοποιούμε ακέραιους τύπους όσο περισσότερο γίνεται. Μπορούμε λοιπόν να δώσουμε έναν πρώτο κανόνα για αποδοτικά προγράμματα:

♦ ***Όταν μια μεταβλητή θα παίρνει σίγουρα ακέραιες τιμές σε όλη την εκτέλεση του προγράμματος, ως τύπος της θα πρέπει να επιλέγεται κάποιος ακέραιος τύπος.***

2.6 Έξω από τα Όρια

Πολλοί οι τύποι λοιπόν, αλλά όλοι έχουν τα όριά τους. Έστω ότι έχουμε δηλώσει:

```
double d;
```

και στη συνέχεια έχουμε:

```
cin >> d;
cout << d << endl;
```

Όταν εκτελείται η `cin >> d` δίνουμε:

```
1e50000
```

Αποτέλεσμα:

```
4.24399e-314      (gcc - Dev C++)
+INF              (Borland C++)
```

Δηλαδή, στην περίπτωση αυτή η κάθε υλοποίηση αντιδρά με δικό της τρόπο.

Παρόμοια συμβαίνουν και με τους άλλους τύπους κινητής υποδιαστολής.

Με τους ακέραιους τύπους τα πράγματα είναι μάλλον χειρότερα. Αν η τιμή που δίνουμε είναι έξω από τα όρια του τύπου αποθηκεύεται κάποια τιμή που υπολογίζεται με αριθμητική υπολοίπων (modulo 2^n).[8]

2.7 Πότε Λύνεται το Πρόβλημα – Δύο Παραδείγματα

Ας σταματήσουμε για λίγο να δίνουμε νέα στοιχεία της γλώσσας κι ας προσπαθήσουμε με ένα παράδειγμα να κάνουμε μια επανάληψη σε όσα μάθαμε μέχρι τώρα.

Το πρόβλημα:

Από ύψος h αφήνεται να πέσει προς τη γή ένα σώμα. Να γραφεί πρόγραμμα που θα διαβάζει από το πληκτρολόγιο την τιμή του h και θα υπολογίζει και θα γράφει:

α) τον χρόνο που θα κάνει το σώμα μέχρι να φτάσει στην επιφάνεια της γής

β) η ταχύτητά του τη στιγμή της πρόσκρουσης.

Να αγνοηθεί η αντίσταση του αέρα. Επιτάχυνση βαρύτητας: g = 9.81 m/sec².

Αν πούμε *Oz* τον κατακόρυφο άξονα, με αρχή την επιφάνεια της γής και θετική φορά προς τα πάνω, έχουμε:

$$z=\frac{1}{2}at^2+v_0t+z_0$$

για τη θέση και:

[8] Αν αυτό σου λέει κάτι.

$$v = v_0 + at$$

για την ταχύτητα, όπου:

a : η επιτάχυνση,

v_0: η αρχική ταχύτητα,

z_0: η αρχική θέση.

Στην περίπτωσή μας:

$$a = -g \qquad v_0 = 0 \qquad z_0 = h$$

και οι παραπάνω εξισώσεις γίνονται:

$$z = -\frac{1}{2} g t^2 + h$$

για τη θέση και:

$$v = -gt$$

Όταν το σώμα προσκρούσει στην γή θα έχουμε $z = 0$ και τιμή της t θα είναι ακριβώς ο χρόνος πτώσης t_P. Και λύνοντας ως προς t_P την εξίσωση:

$$0 = -\frac{1}{2} g\, t_P^2 + h$$

παίρνουμε τον χρόνο πτώσης:

$$t_P = \sqrt{2h/g}$$

Η ταχύτητα την στιγμή της πρόσκρουσης είναι:

$$v_P = -g\, t_P = -\sqrt{2gh}$$

Αν λοιπόν δηλώσουμε:

```
const double g( 9.81 ); // m/sec²
double h, tP, vP;
```

έχουμε τις προδιαγραφές:

Προϋπόθεση: $(g == 9.81)$ && $(h \geq 0)$

Απαίτηση: $(t_P == \sqrt{2h/g})$ && $(v_P == -\sqrt{2gh})$

Εδώ όμως να τονίσουμε κάτι: Δεν θα πρέπει να σου έχει μείνει αμφιβολία ότι

- ***Όλη η δουλειά για την επίλυση του προβλήματος γίνεται όταν διατυπώνουμε τις προδιαγραφές.***

Το πρόγραμμα που θα γράψουμε από δω και πέρα είναι μόνο για τις πράξεις, που τις αφήνουμε στον υπολογιστή.

Ας έρθουμε τώρα στο πρόγραμμά μας. Το σχέδιο για το πρόγραμμα θα είναι:

```
Διάβασε το h
Υπολόγισε τα tP, vP
Τύπωσε τα tP, vP
```

Υποθέτουμε και πάλι ότι το πρόγραμμα θα εκτελεστεί σε διάλογο με το χρήστη. Γι' αυτό, πριν δοκιμάσει να διαβάσει το h, θα πρέπει να δώσει κατάλληλο μήνυμα προς τον χρήστη. Έτσι, η «*Διάβασε το h*» θα γίνει:

```
cout << " Δώσε μου το αρχικό ύψος σε m: ";  cin >> h;
```

Η «*Υπολόγισε τα t_P, v_P*» θα γίνει:

```
tP = sqrt(2*h/g);
vP = -sqrt(2*h*g);
```

σύμφωνα με αυτά που είπαμε παραπάνω.

Εδώ όμως πρόσεξε: Ο υπολογισμός του v_P μπορεί να γίνει και με την:

```
vP = -g*tP;
```

Η πρώτη είναι μεν σύμφωνη με αυτά που λένε τα βιβλία, αλλά χρειάζεται δύο πολλαπλασιασμούς και υπολογισμό μιας τετραγωνικής ρίζας. Η δεύτερη χρειάζεται μόνον έναν πολλαπλασιασμό. Θα προτιμήσουμε λοιπόν την «οικονομικότερη» δεύτερη.

Το τρίτο βήμα του σχεδίου «*Τύπωσε τα t_P, v_P*» μεταφράζεται εύκολα στις:

```
   cout << " Χρόνος Πτώσης = " << tP << " sec" << endl;
   cout << " Ταχύτητα τη Στιγμή της Πρόσκρουσης = "
        << vP << " m/sec" << endl;
```

Να λοιπόν ολόκληρο το πρόγραμμα:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   const double g( 9.81 ); // m/sec², η επιτάχυνση της βαρύτητας
   double h,   // m, αρχικό ύψος
          tP,  // sec, χρόνος πτώσης
          vP;  // m/sec, ταχύτητα τη στιγμή πρόσκρουσης
// Διάβασε το h
   cout << " Δώσε μου το αρχικό ύψος σε m: ";  cin >> h;
// Υπολόγισε τα tP, vP
// (g == 9.81) && (0 ≤ h ≤ DBL_MAX)
   tP = sqrt( 2*h/g );
   vP = -sqrt(2*h*g);
// (tP ≈ √(2h/g)) && (vP ≈ -√(2hg))
// Τύπωσε τα tP, vP
   cout << " Αρχικό ύψος = " << h << " m" << endl;
   cout << " Χρόνος Πτώσης = " << tP << " sec" << endl;
   cout << " Ταχύτητα τη Στιγμή της Πρόσκρουσης = "
        << vP << " m/sec" << endl;
}
```

και ένα παράδειγμα εκτέλεσης. Αρχικώς το πρόγραμμα ζητάει:

```
Δώσε μου το αρχικό ύψος σε m: _
```

Απαντούμε:

```
Δώσε μου το αρχικό ύψος σε m: 80
```

και παίρνουμε:

```
Χρόνος Πτώσης = 4.03855 sec
Ταχύτητα τη Στιγμή της Πρόσκρουσης = -39.6182 m/sec
```

Ας δούμε άλλο ένα

Παράδειγμα

Να γραφεί πρόγραμμα που θα διαβάζει την ακτίνα και το ύψος ενός κυλίνδρου σε *m*, και θα υπολογίζει και θα γράφει:

α) το εμβαδό της βάσης σε m^2,

β) τον όγκο του κυλίνδρου σε *lt*.

Ένα πρόγραμμα για τη δουλειά αυτήν είναι απλό. Ας καταγράψουμε τι θέλουμε να κάνει:

```
Διάβασε την ακτίνα της βάσης και το ύψος.
Υπολόγισε το εμβαδό.
Υπολόγισε τον όγκο και μετάτρεψέ τον σε λίτρα.
Τύπωσε τα αποτελέσματα.
```

Για τον υπολογισμό του εμβαδού μας χρειάζεται το π και καλό είναι να το υπολογίσουμε από την αρχή.

Μια καλή αρχή για τα προγράμματά σου είναι να *αντηχούν* (echo) τα στοιχεία εισόδου μόλις τα διαβάσουν.

Μετά από αυτές τις παρατηρήσεις ξαναγράφουμε το σχέδιό μας:

```
Υπολόγισε το π.
Διάβασε ακτίνα βάσης και ύψος και τύπωσε τις τιμές τους.
Υπολόγισε το εμβαδό.
Υπολόγισε τον όγκο και μετάτρεψέ τον σε λίτρα.
Τύπωσε τα αποτελέσματα.
```

Στο πρόγραμμα θα χρειαστούμε μεταβλητές για:

- το π
- την ακτίνα
- το ύψος
- το εμβαδό
- τον όγκο.

Αν χρησιμοποιήσουμε τα ονόματα: *pi*, *r*, *h*, *s*, *v*, αντιστοίχως, έχουμε κάνει μια καλή επιλογή, μια και αυτά παριστάνουν τέτοια μεγέθη στην απλή γεωμετρία. Θα μας χρειαστεί ακόμη μια μεταβλητή για τον παράγοντα μετατροπής κυβικών μετρων σε λίτρα. Ας την ονομάσουμε *m3SeLt*, και ας της δώσουμε την τιμή 1000 με τη δήλωσή της. Όλες οι μεταβλητές μας θα είναι τύπου `double`.

Να πώς θα μπορούσε να είναι το πρόγραμμα:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   const double  pi( 4*atan(1) ); // = π
   const double  m3SeLt( 1000 ); // lt/m³
   double  r, h, s, v;
// Διάβασε ακτίνα βάσης και ύψος και τύπωσε τις τιμές τους
   cout << " Δώσε την ακτίνα βάσης σε m: ";   cin >> r;
   cout << " Δώσε το ύψος σε m: ";   cin >> h;
   cout << " Διαστάσεις Κυλίνδρου:";
   cout << " Ακτ.Βάσης = " << r << " m  Ύψος = "
                        << h << " m" << endl;
// Υπολόγισε το εμβαδό
   s = pi*r*r;       // βάση
// Υπολόγισε τον όγκο και μετάτρεψέ τον σε λίτρα
   v = s * h;        // όγκος
   v = m3SeLt * v;  // μετατροπή σε lt
// Τύπωσε τα αποτελέσματα
   cout << " Εμβαδό Βάσης = " << s << " m²" << endl;
   cout << " Όγκος = " << v << " lt" << endl;
}
```

Όταν αρχίσει η εκτέλεσή του, βλέπουμε στην οθόνη μας:

```
Δώσε την ακτίνα βάσης σε μέτρα: _
```

Σε απάντηση πληκτρολογούμε:

```
Δώσε την ακτίνα βάσης σε μέτρα: 1.2
```

και στη συνέχεια βλέπουμε:

```
Δώσε το ύψος σε μέτρα: _
```

απαντούμε:

```
Δώσε το ύψος σε μέτρα: 1.8
```

και παίρνουμε το αποτέλεσμα:

```
Διαστάσεις Κυλίνδρου: Ακτ.Βάσης = 1.2 m  Ύψος = 1.8 m
Εμβαδό Βάσης = 4.52389 m²
Όγκος = 8143.01 lt
```

Δες όμως και ένα άλλο παράδειγμα εκτέλεσης:

```
Δώσε την ακτίνα βάσης σε μέτρα: 1.2  1.8
Δώσε το ύψος σε μέτρα:  Διαστάσεις Κυλίνδρου: Ακτ.Βάσης = 1.2 m  Ύψος = 1.8 m
Εμβαδό Βάσης = 4.52389 m²
Όγκος = 8143.01 lt
```

Εδώ τι έγινε; Όταν μας ζητήθηκε η ακτίνα βάσης πληκτρολογήσαμε εκτός από αυτήν (1.2) και την τιμή του ύψους (1.8). Το 1.8 αγνοήθηκε από την `cin >> r;` αλλά έγινε δεκτό από την `cin >> h;`. Το αποτέλεσμα είναι φανερό: Ενώ υπολογίστηκαν σωστά οι τιμές,

έγινε άνω κάτω η εμφάνιση αφού δεν πήρε το δεύτερο <enter> που θα του δίναμε μαζί με το ύψος.

2.8 Τα Χαρακτηριστικά της Μεταβλητής στη C++

Είδαμε πιο πριν ότι αναφέροντας το όνομα μιας μεταβλητής μπορούμε να πάρουμε την τιμή της για να τη γράψουμε ή για να τη χρησιμοποιήσουμε σε πράξεις (παραστάσεις). Στις προηγούμενες παραγράφους όμως συζητήσαμε και για άλλα χαρακτηριστικά μιας μεταβλητής, όπως η *διεύθυνση*, το *μέγεθος* (πόσες ψηφιολέξεις καταλαμβάνει στη μνήμη) και τον *τύπο*. Η C++ μας δίνει τα εργαλεία για να δούμε και αυτά τα χαρακτηριστικά.

Στις επόμενες υποπαραγράφους θα δούμε αυτά τα εργαλεία της C++ και στο τέλος θα γράψουμε ένα πρόγραμμα που, με βάση τις δηλώσεις (παράδ. στην §2.1):

```
double  pi, e, distance( 0.0 ), length( 1.0 );
int  i, counter( 0 ), j, number( 0 ), integer( 375 );
```

θα μας δώσει τον πίνακα του Σχ. 2.1.

Προς το παρόν, μπορείς να χρησιμοποιήσεις αυτά τα εργαλεία για να «σκαλίζεις» τη μνήμη και να βλέπεις πώς μεταφράζει ο μεταγλωττιστής της C++ τα προγράμματά σου. Αργότερα θα δεις ότι είναι χρήσιμα σε πολλές περιπτώσεις.

2.8.1 Ο Τύπος – Ο Τελεστής "typeid"

Αν δώσεις:

```
cout << typeid(pi).name() << "  "
     << typeid(integer).name() << endl;
```

θα πάρεις[9]:

```
double  int
```

Δηλαδή: ο τύπος της *pi* είναι **double** και ο τύπος της *integer* είναι **int**.

Γενικώς, μπορείς να δώσεις:

typeid(*παράσταση* **).name()**

και να πάρεις τον τύπο του αποτελέσματος της <παράστασης>. Στο πρόγραμμά σου θα πρέπει να περιλάβεις (**#include**) το "**typeinfo**".

Να τώρα *Η λύση της άσκ. 1-8*: Η

```
cout << typeid(4./2).name() << "  "
     << typeid(4/2).name() << endl;
```

μας δίνει την απάντηση:

```
double  int
```

2.8.2 Το Μέγεθος – Ο Τελεστής "sizeof"

Στις τελευταίες στήλες των Πίν. 2-1, 2-2 βλέπεις τη μνήμη, σε ψηφιολέξεις και δυαδικά ψηφία αντίστοιχα, που χρειάζονται οι μεταβλητές διαφόρων τύπων. Δεν θα ήθελες να δεις αυτές τις τιμές για τη δική σου εγκατάσταση της C++; Ο τελεστής "**sizeof**" σου δίνει αυτήν τη δυνατότητα. Γράφουμε:

```
cout << (sizeof number) << "  "
     << (sizeof length) << endl;
```

και παίρνουμε:

```
4  8
```

[9] Το πρόγραμμα από τον gcc (π.χ. Dev C++) θα δώσει: **d i**.

που σημαίνει: το μέγεθος της μεταβλητής *number* είναι 4 ψηφιολέξεις (= 32 δυαδικά ψηφία) και της *length* είναι 8 ψηφιολέξεις.

Γενικώς, μπορείς να γράφεις:

"`sizeof`", *όνομα μεταβλητής* ή

"`sizeof`", "`(`", *παράσταση*, "`)`" ή

"`sizeof`", "`(`", *όνομα τύπου*, "`)`"

Αν δώσεις:

```
cout << (sizeof (4./2)) << "  "
     << (sizeof (4/2)) << endl;
cout << (sizeof (double)) << "  "
     << (sizeof (int)) << endl;
```

θα πάρεις[10]:

```
8  4
8  4
```

2.8.3 Η Διεύθυνση – Οι Τελεστές "&" και "*"

Μπορούμε να δούμε τη διεύθυνση μιας μεταβλητής με τον τελεστή "`&`". Αν δώσεις π.χ.:

```
cout << &number << "  " << number << endl;
```

θα πάρεις απάντηση:

```
0x343f2744  0
```

ή κάτι παρόμοιο. Η δεύτερη τιμή (0) είναι η τιμή της *number*, όπως την περιμένουμε. Η πρώτη είναι μια ακέραιη τιμή –γραμμένη στο *δεκαεξαδικό* σύστημα– και είναι η διεύθυνση, στη μνήμη, όπου αρχίζει η μεταβλητή *number*. Δοκίμασε και στον δικό σου υπολογιστή, αλλά μην περιμένεις να πάρεις την ίδια τιμή!

Έτσι:

- γράφοντας `number` παίρνουμε αυτό που μας ενδιαφέρει, δηλ. την τιμή της μεταβλητής `number`, ενώ
- γράφοντας `&number` παίρνουμε τη διεύθυνση μιας θέσης μνήμης όπου υπάρχει η πληροφορία που θέλουμε· παίρνουμε δηλαδή μια *παραπομπή* προς αυτό που μας ενδιαφέρει.

Λέμε λοιπόν ότι η `&number` είναι μια **παραπέμπουσα** ή **αναφερόμενη** (referencing) τιμή –αφού παραπέμπει ή αναφέρεται σε κάτι– ή τιμή-**βέλος** (pointer) –αφού κατευθύνεται προς (δείχνει) κάτι. Όπως θα δούμε αργότερα, στον προγραμματισμό με τη C++, τα βέλη χρησιμοποιούνται πάρα πολύ.

Ας πούμε ότι έχουμε μια τιμή-βέλος *p*, δηλαδή μια διεύθυνση· πώς μπορούμε να δούμε την τιμή που είναι αποθηκευμένη στη θέση που μας δείχνει η *p*; Ή, με άλλα λόγια, ποια είναι η *αντίστροφη πράξη* της "`&`"; Η C++ τη συμβολίζει με "`*`": η τιμή που είναι απόθηκευμενη στη θέση που μας δείχνει η *p* παριστάνεται με "`*p`". Αν, λοιπόν, πάρουμε τη "`*(&number)`" είναι σαν να παίρνουμε τη "`number`". Πράγματι, η:

```
cout << &number << "  " << number
     << "  " << *(&number) << endl;
```

θα δώσει:

```
0x343f2744  0  0
```

Δες και αυτό:

```
*(&number) = 4;
cout << &number << "  " << number
     << "  " << *(&number) << endl;
```

[10] Αυτό μπορείς να το θεωρήσεις και ως λύση στην άσκ. 1-8.

Αποτέλεσμα:

```
0x343f2744  4  4
```

«Μα, αριστερά του "=" δεν υπάρχει μεταβλητή!» Υπάρχει! Αφού είπαμε ότι «*Αν πάρουμε την* `*(&number)` *είναι σαν να παίρνουμε τη* `number`.»

Λέμε ότι ο τελεστής "`*`" **απο-παραπέμπει** (dereferences) την τιμή-βέλος στην οποία δρα.

Εδώ όμως μπορεί να υπάρχουν αντιρρήσεις: «ο τελεστής "`*`" ξέρουμε ότι κάνει πολλαπλασιασμό! Τι αποπαραπομπές και κουραφέξαλα μας λες;» Ναι, *ο "`*`" κάνει πολλαπλασιασμό όταν δρα σε δύο αριθμητικές τιμές· κάνει αποπαραπομπή όταν δρα σε μια τιμή-βέλος.*

2.8.4 Πώς Παίρνουμε τον Πίνακα

Και τώρα ας έρθουμε να γράψουμε το πρόγραμμα που θα βγάζει τον πίνακα του Σχ. 2-1 και μάλιστα με μια στήλη παραπάνω όπου θα γράφεται το μέγεθος σε ψηφιολέξεις. Εδώ θα γράψουμε τις εντολές που βγάζουν τις δύο πρώτες γραμμές και εσύ θα το συμπληρώσεις για να βγάλει τις υπόλοιπες.

Φυσικά, ξεκινούμε με τις δηλώσεις:

```
   double pi, e, distance( 0.0 ), length( 1.0 );
   int    i, counter( 0 ), j, number( 0 ), integer( 375 );
```

Στη αρχή θα πρέπει να τυπώσει το όνομα σε 8 θέσεις και στη συνέχεια τη διεύθυνση:

```
   cout.width( 8 );
   cout << "integer" << "  " << &integer << "  ";
```

Τώρα σειρά έχει ο τύπος, σε 6 θέσεις, και μετά το μέγεθος:

```
   cout.width( 6 );
   cout << typeid(integer).name() << "  "
        << (sizeof integer) << "  ";
```

Τέλος βγάζουμε και την τιμή:

```
   cout << integer << endl;
```

Να λοιπόν το πρόγραμμα:

```
#include <iostream>
#include <typeinfo>
using namespace std;
int main()
{
   double pi, e, distance( 0.0 ), length( 1.0 );
   int    i, counter( 0 ), j, number( 0 ), integer( 375 );

   cout << " όνομα\tδιεύθυνση\tτύπος\tμεγ\tτιμή" <<endl;
   cout.width( 8 );
   cout << "counter" << '\t' << &counter << '\t';
   cout.width( 6 );
   cout << typeid(counter).name() << '\t'
        << (sizeof counter) << '\t';
   cout << counter << endl;

   cout.width(8);
   cout << "distance" << '\t' << &distance << '\t';
   cout.width(6);
   cout << typeid(distance).name() << '\t'
        << (sizeof distance) << '\t';
   cout << distance << endl;
// εσύ γράφεις τα υπόλοιπα
}
```

που (συμπληρωμένο θα) δίνει:

```
όνομα  διεύθυνση    τύπος  μεγ    τιμή
 counter     0x12FF64        int 4      0
distance     0x12FF74     double 8      0
```

```
       e      0x12FF7C      double 8      2.12414e-314
       i      0x12FF68         int 4      1245072
 integer      0x12FF58         int 4      375
       j      0x12FF60         int 4      1
  length      0x12FF6C      double 8      1
  number      0x12FF5C         int 4      0
      pi      0x12FF84      double 8      2.122e-314
```

Παρατηρήσεις: ▶

- Δεν σου αρέσει ως πίνακας, ε; Πέρνα τον στο Excel ή σε πίνακα του Word και θα γίνει πανέμορφος.
- Οι τιμές των *e*, *i*, *j* και *pi* δεν πρέπει να σε εκπλήσσουν: αυτές οι μεταβλητές είναι αόριστες. Έτσι, ας πούμε για την *e*, ο ΗΥ «βλέπει» τα δυαδικά ψηφία στην αντίστοιχη διεύθυνση και τα «ερμηνεύει» ως τιμή τύπου `double`. ◀

2.9 Αλλαγή Τύπου

Για κάθε τύπο *T* της C++ υπάρχει μια συνάρτηση, με όνομα

`static_cast<`*T*`>`

που μετατρέπει τιμές άλλων τύπων σε τιμή τύπου *T* (αν αυτή η μετατροπή είναι δυνατή). Εδώ θα ασχοληθούμε με μετατροπές αριθμητικών τύπων. Ας δούμε ένα παράδειγμα:

Παράδειγμα

Η εντολή:

```
cout << static_cast<int>(9.916) << "  "
     << static_cast<int>(-9.916) << endl;
```

θα δώσει:

```
9  -9
```

Δηλαδή: οι τιμές "`9.916`" και "`-9.916`" μετατράπηκαν σε τιμές τύπου `int` από τη συνάρτηση `static_cast<int>`. Αυτό έγινε με *αποκοπή του κλασματικού μέρους*.

Η εντολή:

```
cout << static_cast<double>(901234567L) << "  "
     << static_cast<float>(901234567L) << endl;
```

θα δώσει:

```
9.01235e+08  9.01235e+08
```

Δηλαδή: η τιμή `901234567L` μετατράπηκε

- σε μια τιμή τύπου `double` από τη `static_cast<double>` και
- σε μια τιμή τύπου `float` από τη `static_cast<float>`.

Αλλά, η ακέραιη τιμή είχε 9 ψηφία ενώ από τη μετατροπή μας έμειναν 6. Μήπως υπάρχουν τα ψηφία αλλά δεν γράφονται; Ας δοκιμάσουμε να αλλάξουμε την ακρίβεια:

```
cout.precision(10);
cout << static_cast<double>(901234567L) << "  "
     << static_cast<float>(901234567L) << endl;
```

Αποτέλεσμα:

```
901234567  901234560
```

Βλέπουμε δηλαδή ότι *από τη μετατροπή* `long` *σε* `float` *μπορεί να έχουμε απώλεια σημαντικών ψηφίων*.

Να δούμε τώρα μια άλλη περίπτωση: Ας υποθέσουμε ότι έχουμε τύπο `int` με μέγιστη τιμή 2147483647· τι θα γίνει αν ζητήσω την

`static_cast<int>( 2345678901.23 )`

Η C++ μας λέει ότι το αποτέλεσμα είναι αόριστο. Γενικώς: *αν έχεις αριθμητικό τύπο T που περιλαμβάνει τιμές από* min_T *μέχρι* max_T *μπορείς να χρησιμοποιείς την* `static_cast<T>(x)` *μόνον αν*

$$min_T \le \texttt{static\_cast<}T\texttt{>}(x) \le max_T$$

Η αλλαγή παράστασης μιας τιμής από έναν τύπο σε έναν άλλο ονομάζεται (στατική) **τυποθεώρηση** (typecasting ή casting)[11].

Εδώ θα πρέπει να τονίσουμε ότι η τιμή που έχει μια μεταβλητή δεν μεταβάλλεται από τη στατική τυποθεώρησή της. Π.χ. οι εντολές:

```
double x( 123.457 );
int    i;
i = static_cast<int>( x );
cout << "  x = " << x << "    i = " << i << endl;
```

δίνουν:

```
x = 123.457    i = 123
```

Αντί για `i = static_cast<int>( x )` μπορούμε να γράψουμε:

```
i = int( x );
```

και να πάρουμε το ίδιο αποτέλεσμα. Γενικώς, το να γράφουμε:

$T(v)$ αντί για `static_cast<`T`>(`v`)`

είναι πολύ συνηθισμένο αλλά όχι κατ' ανάγκη καλύτερο.

Πρόσεξε τώρα μια (πολύ συχνή) χρήση της τυποθεώρησης: Από όσα ξέρουμε, μετά τη δήλωση:

```
int k( 10 ), n( 3 );
```

η εντολή:

```
cout << "  k/n = " << k/n << endl;
```

θα δώσει:

```
k/n = 3
```

Πώς μπορούμε να πάρουμε το ακριβές αποτέλεσμα; Έτσι:

```
cout << "  k/n = " << static_cast<double>(k)/n << endl;
```

ή έτσι:

```
cout << "  k/n = " << double(k)/n << endl;
```

που δίνουν:

```
k/n = 3.33333
```

2.9.1 Η Τυποθεώρηση στη C

Η C++ έχει κληρονομήσει από τη C και έναν άλλο τρόπο γραφής της τυποθεώρησης· γράφουμε:

$(T)\,v$ που είναι ισοδύναμο[12] με `static_cast<`T`>(`v`)`

Έτσι, μπορεί να δεις γραμμένο

```
(unsigned long int) -1234567890L
```

αντί για:

```
static_cast<unsigned long int>(-1234567890L)
```

---

[11] Υπάρχουν και άλλα είδη τυποθεώρησης, που θα τα δούμε αργότερα.

[12] Ε, όχι ακριβώς· ο συμβολισμός αυτός καλύπτει και άλλες περιπτώσεις τυποθεώρησης, που θα μάθουμε αργότερα.

2.10 * Οι «Συντομογραφίες» της Εκχώρησης

Η C++ μας δίνει μερικές «συντομογραφίες» πολύ συνηθισμένων εκχωρήσεων. Πολύ συχνά στα προγράμματά μας θέλουμε να γράψουμε: αύξησε το *x* κατά *b*· αυτό το γράφουμε, σύμφωνα με όσα ξέρουμε:

```
   x = x + b;
```

δηλαδή: πάρε την τιμή της *x* και την τιμή της *b*, πρόσθεσέ τις και το αποτέλεσμα να το αποθηκεύσεις ως νέα τιμή της *x*. Η C++ μας δίνει τη δυνατότητα να το γράψουμε ως:

```
   x += b;
```

Παρόμοιες συντομογραφίες υπάρχουν και για τις άλλες πράξεις:

`x += Π;` είναι ισοδύναμη με: `x = x + Π;`
`x -= Π;` είναι ισοδύναμη με: `x = x - Π;`
`x *= Π;` είναι ισοδύναμη με: `x = x * Π;`
`x /= Π;` είναι ισοδύναμη με: `x = x / Π;`
`x %= Π;` είναι ισοδύναμη με: `x = x % Π;`

Για τις "`/=`" και "`%=`" η παράσταση *Π* θα πρέπει να παίρνει τιμή μη μηδενική. Για την τελευταία: η *x* και η τιμή της *Π* θα πρέπει να είναι ακέραιου τύπου.

Ειδικώς για την περίπτωση που θέλουμε να αυξήσουμε ή να μειώσουμε την τιμή μιας μεταβλητής κατά 1, υπάρχουν και άλλες συντομογραφίες:

`++x;` είναι ισοδύναμη με: `x = x + 1;` ή `x += 1;`
`--x;` είναι ισοδύναμη με: `x = x - 1;` ή `x -= 1;`

Τέλος, υπάρχει και μια παραλλαγή της τελευταίας συντομογραφίας: οι `x++` και `x--` αυξάνουν/μειώνουν την τιμή της *x* κατά 1. Τη διαφορά τους από τις `++x` και `--x` θα τη μάθουμε αργότερα.[13]

Στο παρακάτω παράδειγμα μπορείς να δεις όλα τα παραπάνω:

```
#include <iostream>
using namespace std;
int main()
{
   int x, y;
   int a = 1, b = 2;

   x = 5; x += a+b; cout << x << "  ";
   x = 5; x -= a+b; cout << x << endl;
   x = 5; x *= a+b; cout << x << endl;
   x = 5; x /= a+b; cout << x << "  ";
   x = 5; x %= a+b; cout << x << endl;

   x = 5;  ++x;   cout << x << "  ";
   x = 5;  --x;   cout << x << endl;
   x = 5;  x++;   cout << x << "  ";
   x = 5;  x--;   cout << x << endl;
}
```

Αποτέλεσμα:

```
8  2
15
1  2
6  4
6  4
```

Θα αρχίσουμε να χρησιμοποιούμε τις συντομογραφίες αυτές στο Β' Μέρος του βιβλίου.

---

[13] Τώρα προσπάθησε να καταλάβεις τη σχέση που έχει η C++ με τη «μητρική» της γλώσσα C!

2.11 * Υπολογισμός Παράστασης

Στον πίνακα του Παρ. Ε βλέπεις τα χαρακτηριστικά των πράξεων της C++. Ας δούμε αυτές που έχουμε μάθει μέχρι τώρα:

- Πρώτα εκτελούνται υπολογισμοί μέσα σε παρενθέσεις (προτ. 0).
- Στη συνέχεια υπολογίζονται οι κλήσεις συναρτήσεων (προτ. 2).
- Μετά γίνονται πράξεις με τα πρόσημα (προτ. 3) με προσεταιριστικότητα από δεξιά προς τα αριστερά. Δηλαδή; Στη C++ μπορείς να γράψεις:

  `+ - - +12` ή `- + - -x`

  που υπολογίζονται ως:

  `+(-(-(+12)))` και `-(+(-(-x)))`

  αντιστοίχως.
- Μετά γίνονται πολλαπλασιασμοί, διαιρέσεις και υπολογισμοί υπολοίπου (προτ. 5) με προσεταιριστικότητα από αριστερά προς τα δεξιά. Αυτό σημαίνει ότι η παράσταση: `2*x/y*z/5` υπολογίζεται ως: `(((2*x)/y)*z)/5`. Αλλά προσοχή: *δεν σημαίνει* ότι αν έχεις την `(x*y)/(w/u)` θα γίνει πρώτα ο πολλαπλασιασμός `x*y` και μετά η διαίρεση `w/u`.
- Στη συνέχεια γίνονται προσθέσεις και αφαιρέσεις (προτ. 6) από αριστερά προς τα δεξιά, με την ίδια έννοια όπως παραπάνω.

Όταν υπολογίζεται μια αριθμητική παράσταση γίνονται και ορισμένες μετατροπές τύπων, που θα τις ονομάζουμε *Συνήθεις Αριθμητικές Μετατροπές* (ΣΑΜ). Ας πούμε ότι έχουμε κάποια πράξη *a θ b*. Τότε:

1. Αν κάποια από τις τιμές *a*, *b* είναι τύπου `long double`, τότε μετατρέπεται στον τύπο `long double` και η άλλη τιμή.
2. Αλλιώς, αν κάποια από τις τιμές *a*, *b* είναι τύπου `double`, τότε μετατρέπεται στον τύπο `double` και η άλλη τιμή.
3. Αλλιώς, αν κάποια από τις τιμές *a*, *b* είναι τύπου `float`, τότε μετατρέπεται στον τύπο `float` και η άλλη τιμή.
4. Αλλιώς, τα *a*, *b* (είναι ακέραιου τύπου) υφίστανται *προώθηση ακεραίων*, δηλαδή: αν κάποια (ή και οι δύο) από τις τιμές *a*, *b* είναι «μικρού ακέραιου τύπου» (`char`, `short int`, `signed` ή `unsigned`), αυτή μετατρέπεται σε τύπο `int` ή `unsigned int` (αν δεν μπορεί να παρασταθεί στον `int`) και στη συνέχεια

   αν κάποια από τις τιμές *a*, *b* είναι τύπου `unsigned long`, τότε

   μετατρέπεται στον τύπο `unsigned long` και η άλλη τιμή.
5. Αλλιώς, αν κάποια από τις τιμές *a*, *b* είναι τύπου `long int` και η άλλη `unsigned int`, τότε αν κάθε τιμή `unsigned int` μπορεί να παρασταθεί σε `long int`

   η τιμή `unsigned int` μετατρέπεται στον τύπο `long int`

   αλλιώς

   και οι δύο μετατρέπονται σε `unsigned long int`.
6. Αλλιώς, αν κάποια από τις τιμές *a*, *b* είναι τύπου `long`, τότε μετατρέπεται σε `long` και η άλλη τιμή.
7. Αλλιώς, αν κάποια από τις τιμές *a*, *b* είναι τύπου `unsigned`, τότε μετατρέπεται στον τύπο `unsigned` και η άλλη τιμή.
8. Αλλιώς και οι δυο τιμές είναι τύπου `int`.

Οι ΣΑΜ εφαρμόζονται όταν γίνονται οι πράξεις `+`, `-`, `*`, `/` μεταξύ αριθμητικών τιμών και καθορίζουν τον τύπο του αποτελέσματος. Ακέραιη προώθηση γίνεται στις ενικές πράξεις `+`, `-` (πρόσημα). Ο τύπος του αποτελέσματος είναι αυτός που προκύπτει από την προώθηση.

Δηλαδή, ο μεταγλωττιστής κάνει τις κατάλληλες μετατροπές τύπου ώστε να έχει τον ακριβέστερο υπολογισμό. Αλλά, στο τέλος, αυτό μπορεί να «χαλάσει» από κάποια εκχώρηση που μπορεί να ακολουθεί. Δες το παρακάτω πρόγραμμα:

```
#include <iostream>
#include <climits>
using namespace std;
int main()
{
   short int a;

   a = SHRT_MAX + 10;
   cout << SHRT_MAX+10 << "  " << a << endl;
   cout << sizeof(SHRT_MAX+10) << "  " << sizeof(a) << endl;
}
```

που δίνει:

```
32777  -32759
4  2
```

Η πρώτη τιμή είναι ακριβής αλλά η δεύτερη όχι. Το τι έγινε το καταλαβαίνεις αν πάρεις υπόψη σου και τη δεύτερη γραμμή: Η τιμή `SHRT_MAX + 10` χρειάζεται 4 ψηφιολέξεις για να παρασταθεί αλλά η *a* έχει μόνο 2...

2.12 *scanf()*: Η Δίδυμη της *printf()*

Στη C, όπως γράφουμε με την *printf().*, διαβάζουμε με τη *scanf()*. Η αλλιώς: *ό,τι κάνει ο "`>>`" στο cin κάνει η scanf() στο stdin*, που είναι για τη C το ρεύμα από το πληκτρολόγιο προς το πρόγραμμά μας.

Ξαναγράφουμε το πρόγραμμα του Παραδ. 1 της §2.3 χρησιμοποιώντας τις *scanf()* και *printf()*:

```
#include <cstdio>
using namespace std;
int main()
{
   int    k, j;
   double x, y;

   scanf( "%d %d %lf %lf", &k, &j, &x, &y );
   printf( "%d %d %f %f\n", k, j, x, y );
}
```

Η σύνταξη της *scanf* μοιάζει με αυτήν της *printf* ως προς το ότι και οι δύο παίρνουν πρώτο όρισμα έναν *ορμαθό μορφοποίησης*. Τα υπόλοιπα ορίσματα διαφέρουν ως προς τον τύπο: η *scanf* περιμένει (παραστάσεις που οι τιμές τους είναι) *βέλη*, δηλαδή *διευθύνσεις στη μνήμη*. Σε κάθε προδιαγραφή μορφοποίησης αντιστοιχεί μια διεύθυνση. Πηγαίνοντας από αριστερά προς τα δεξιά βρίσκουμε

- Την προδιαγραφή "`%d`" που αντιστοιχεί στην πρώτη διεύθυνση "`&k`". Αυτό σημαίνει: θα διαβάσεις ψηφία που σχηματίζουν μια *ακέραιη* σταθερά· θα αποθηκεύσεις την τιμή (τύπου `int`) που θα προκύψει από τη μετατροπή σε εσωτερική παράσταση στη διεύθυνση `&`*k*.
- Παρόμοια ισχύουν για τη δεύτερη "`%d`" που αντιστοιχεί στη δεύτερη διεύθυνση "`&j`".
- Η πρώτη προδιαγραφή "`%lf`" που αντιστοιχεί στην τρίτη διεύθυνση "`&x`" και σημαίνει: θα διαβάσεις ψηφία που σχηματίζουν μια *πραγματική* σταθερά· θα αποθηκεύσεις την τιμή τύπου `double` που θα προκύψει από τη μετατροπή σε εσωτερική παράσταση στη διεύθυνση `&`*x*.
- Παρόμοια ισχύουν και για τη δεύτερη "`%lf`" που αντιστοιχεί στην τέταρτη διεύθυνση "`&y`".

Οι προδιαγραφές "**%d**" είναι γενικώς για ακέραιες τιμές. Μπροστά από το '**d**' μπορεί να υπάρχει ένας **τροποποιητής** (modifier) '**h**' αν θέλουμε η εσωτερική παράσταση να είναι **short int** ή '**l**' για να είναι **long int** (χωρίς τροποποιητή για **int**).[14]

Για πραγματικές τιμές έχουμε τις προδιαγραφές "**%f**" (ή "**%e**" ή "**%E**" ή "**%f**" ή "**%g**" ή "**%G**"). Μπροστά από το '**f**' μπορεί να υπάρχει τροποποιητής '**l**' αν θέλουμε η εσωτερική παράσταση να είναι **double** ή '**L**' για να είναι **long double** (χωρίς τροποποιητή για **float**).

Παρατήρηση: ▶

Το παραπάνω πρόγραμμα δεν είναι γραμμένο σε C, αλλά πρόγραμμα C++ που χρησιμοποιεί βιβλιοθήκες της C. Ο μεταγλωττιστής της C *δεν* θα το δεχτεί.

Να τι θα δεχθεί:

```
#include <stdio.h>

int main()
{
   int    k, j;
   double x, y;

   scanf( "%d %d %lf %lf", &k, &j, &x, &y );
   printf( "%d %d %f %f\n", k, j, x, y );
}
```

Φύλαξέ το στο αρχείο **test_C_IO.c** και δώσε το στον μεταγλωττιστή της C. Δεν θα σου φέρει αντίρρηση.◀

Ας ξαναγράφουμε και το πρόγραμμα του Παραδ. 3 της §2.3 με τις *scanf* και *printf*:

```
#include <cstdio>
#include <cmath>
using namespace std;
int main()
{
   double  a, b, c;
   double  x1, x2;
   double  delta;

   printf( "Δώσε τρεις πραγματικούς αριθμούς\n" );
   scanf( "%lf %lf %lf", &a, &b, &c );
   printf( " a = %f   b = %f   c = %f\n", a, b, c );
   delta = b*b - 4*a*c;
   x1 = (-b + sqrt(delta))/(2*a);
   x2 = (-b - sqrt(delta))/(2*a);
   printf( " Λύση1 = %f   Λύση2 = %f\n", x1, x2 );
}
```

Πρόσεξε ότι εδώ προτιμήσαμε να βάλουμε όλα τα σταθερά κείμενα στον ορμαθό μορφόποιησης. Δηλαδή:

- Αντι να γράψουμε

```
   printf( "%s\n", "Δώσε τρεις πραγματικούς αριθμούς" );
```

γράφουμε

```
   printf( "Δώσε τρεις πραγματικούς αριθμούς\n" );
```

- Αντι να γράψουμε

```
   printf( "%s %f %s %f %s %f\n",
           " a =", a, " b =", b, " c =", c );
```

γράφουμε

```
   printf( " a = %f   b = %f   c = %f\n", a, b, c );
```

[14] Για ακέραιες τιμές χωρίς πρόσημο έχουμε την προδιαγραφή "**%u**" με τους ίδιους τροποποιητές.

2.13 Λάθη, Λάθη ...

Αν ακολουθείς τις οδηγίες μας και περνάς τα προγράμματα και πειραματίζεσαι θα πρέπει να έχεις αγανακτήσει ήδη με τον μεταγλωττιστή σου και τα λάθη που βρίσκει. Κάνε υπομονή. Όσο προχωράς θα ελαττωθούν τα λάθη απροσεξίας καθώς και τα σοβαρότερα.

- ***Είναι βασικό να μάθεις να βρίσκεις και να διορθώνεις μόνος/η σου τα λάθη των προγραμμάτων σου.*** [15]

Μόνο έτσι θα μάθεις προγραμματισμό! (Ε, στην αρχή μπορεί να χρειαστείς και λίγη βοήθεια...)

Αλλά, ας τα πάρουμε από την αρχή:

- Το ";" είναι για τη C++ **τερματιστής εντολής** (statement terminator). Με αυτό τελειώνουν οι εντολές και οι δηλώσεις.
- Όλες οι μεταβλητές και οι σταθερές (με όνομα) πρέπει να δηλώνονται υποχρεωτικώς πριν χρησιμοποιηθούν· καλύτερα να τις δηλώνεις στην αρχή.
- Μη χρησιμοποιείς σε παραστάσεις μεταβλητές που δεν τους έχεις δώσει τιμή πιο πριν είτε με εντολή εκχώρησης, είτε με εντολή `cin >>` ....

Συνηθισμένα λάθη εκτέλεσης είναι αυτά της ανάγνωσης στοιχείων, όταν μάλιστα πρόκειται για αριθμούς:

- Αν πληκτρολογείς τιμή μεταβλητής ακέραιου τύπου, δώσε σταθερά του τύπου αυτού: πρόσημο, αν χρειάζεται, και στη συνέχεια ψηφία. Το 17.0 *δεν είναι* σταθερά ακέραιου τύπου, ούτε το 17,0 ούτε το 1.7e1. Το 17 είναι! Και, για όνομα του Θεού, μην πληκτρολογήσεις *INT_MAX*.
- Αν πληκτρολογείς τιμή μεταβλητής πραγματικού τύπου, δώσε μια σταθερά του τύπου αυτού. Η υποδιαστολή είναι "." όχι ",".

Ξανακοίταξε τώρα τα προγράμματα με τα ανεξήγητα λάθη, διόρθωσέ τα και συνέχισε την προσπάθεια. Καλή τύχη!

2.14 Τι (Πρέπει να) Έμαθες Μέχρι Τώρα

Η βασική έννοια αυτού του κεφαλαίου είναι η *μεταβλητή*, το βασικό εργαλείο για τη διαχείριση της κύριας μνήμης του ΗΥ. Πρέπει να έμαθες

- Να *δηλώνεις* μεταβλητές και –αν θέλεις– να *ορίζεις* την αρχική τιμή τους.
- Να ορίζεις/αλλάζεις την τιμή τους με την *εντολή εκχώρησης*.
- Να ορίζεις/αλλάζεις την τιμή τους με *εντολή εισόδου* από το πληκτρολόγιο.

Ακόμη, πρέπει να έμαθες μερικά πράγματα για τους βασικούς (αριθμητικούς) *τύπους* της C++ και πώς να «μετατρέπεις τον τύπο» μιας τιμής.

Τι προγράμματα μπορείς να γράψεις; Αυτά που λύνουν τις ασκήσεις της Β Ομάδας είναι χαρακτηριστικά.

---

[15] Ξέρεις πως λέγεται αυτή η δουλειά στα αγγλικά; «debugging», που θα μπορούσε να μεταφραστεί «απεντόμωση», ενώ σημαίνει «διόρθωση λαθών». Ρώτησε να μάθεις γιατί λέγεται έτσι και γιατί τα λάθη προγραμματισμού λέγονται «bugs» (ζωϋφια, κοριοί). Έχει ιστορική και χιουμοριστική αξία.

Ασκήσεις

Α Ομάδα

2-1 Δίνεται το πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{
   int i, j, k;

   i = 5;  j = 7;  k = 10;
   cout << i << ' ' << (i+1) << ' ' << (i+j)
        << ' ' << (i + j*k) << endl;
   k = i + j;         cout << k << endl;
   j = j + 1;         cout << j << endl;
   i = j + 2*i + j;   cout << i << endl;
}
```

Εκτέλεσε το πρόγραμμα (*εσύ*, όχι ο υπολογιστής), όπως κάναμε στο παράδειγμα της αντιμετάθεσης.

2-2 Το ίδιο για το πρόγραμμα:

```
#include <iostream>
using namespace std;
int main()
{  double first, second, realResult;
   int    third, intResult;

   cin >> first >> second >> third;
   realResult = first + second + third;
   intResult = first + second - third;
   realResult = realResult - intResult;
   intResult = intResult - realResult;
   cout << realResult << intResult << endl;
}
```

Δίνεται μια γραμμή με στοιχεία εισόδου:

1.5e1 8.1 5 7.41e-5 33.0

2-3 Γράψε πρόγραμμα που θα διαβάζει από το πληκτρολόγιο μια τιμή –ας πούμε x– και θα υπολογίζει και θα τυπώνει τις τιμές των παραστάσεων για την τιμή που διαβάστηκε:

$$\frac{1}{1+\sqrt{x}} \qquad 1+e^{-x/2} \qquad x^x + x^{x/2}$$

$$\frac{\sigma\upsilon\nu x - \eta\mu x}{1+\sqrt{2x}} \qquad \frac{e^{-x/2}+e^{x/2}}{2} \qquad x^{x/3} + x^{3x}$$

όπου e η βάση των φυσικών λογαρίθμων.

2-4 Γράψε πρόγραμμα που θα διαβάζει από το πληκτρολόγιο την τιμή μιας γωνίας σε μοίρες –ας πούμε x– και θα υπολογίζει και θα τυπώνει τα:

$$\eta\mu x,\ \sigma\upsilon\nu x,\ \varepsilon\varphi x,\ \sigma\varphi x,\ \tau\varepsilon\mu x,\ \sigma\tau\varepsilon\mu x$$

Β Ομάδα

2-5 Γράψε πρόγραμμα που διαβάζει από το πληκτρολόγιο δυο αριθμούς: β (θετικό πραγματικό, $\neq 1$) και x (θετικό πραγματικό) και θα υπολογίζει και θα τυπώνει τα $\log_\beta x$ και β^x.
Υπόδ.: Χρησιμοποίησε την ιδιότητα: $\log_\beta x = \ln x / \ln \beta$.

2-6 Γράψε πρόγραμμα που θα διαβάζει από το πληκτρολόγιο την τιμή κάποιου αντικειμένου, χωρίς ΦΠΑ και τον συντελεστή ΦΠΑ (%). Θα υπολογίζει και θα γράφει την τιμή με ΦΠΑ.

2-7 Γράψε πρόγραμμα που θα διαβάζει τις καρτεσιανές συντεταγμένες, x, y, ενός σημείου στο επίπεδο και θα υπολογίζει και θα τυπώνει τις πολικές r, ϕ. Υπόθεσε ότι $x > 0$. Υπολόγισε τη ϕ: α) με την *atan* β) με την *atan2*.

Υπόδ.: $r = \sqrt{x^2 + y^2}$, $\phi = τοξεφ(y/x)$.

2-8 Ας υποθέσουμε ότι ο μισθός ενός εργαζόμενου προσαυξάνεται κατά 2%, επί του βασικού μισθού, για κάθε χρόνο υπηρεσίας. Γράψε πρόγραμμα που θα διαβάζει τον βασικό μισθό και τα χρόνια υπηρεσίας ενός υπαλλήλου και θα υπολογίζει το χρονοεπίδομα και το συνολικό μισθό.

2-9 Αν συνδέσουμε δυο αντιστάσεις R_1 και R_2 σε σειρά, η συνολική αντίσταση είναι $R = R_1 + R_2$. Γράψε πρόγραμμα που θα διαβάζει τις τιμές των R_1 και R_2 και θα υπολογίζει και θα τυπώνει την R.

2-10 Αν συνδέσουμε δυο αντιστάσεις R_1 και R_2 παραλλήλως, η συνολική αντίσταση είναι:

$$R = \frac{R_1 R_2}{R_1 + R_2}$$

Γράψε πρόγραμμα που θα διαβάζει τις τιμές των R_1 και R_2 και θα υπολογίζει και θα τυπώνει την R.

2-11 Ένα κύκλωμα αποτελείται από πυκνωτή χωρητικότητας C και αντιστάτη αντίστασης R συνδεμένα παράλληλα. Στα κοινά τους άκρα συνδέουμε πηγή εναλλασσόμενης τάσης $U = U_0 ημ(2πνt)$.

Γράψε πρόγραμμα που θα διαβάζει από το πληκτρολόγιο τις τιμές των U_0 (V), $ν$ (Hz), C (F), R ($Ω$) και θα υπολογίζει και θα γράφει στην οθόνη:

- την ενεργή τιμή της τάσης $U_{εν} = U_0 / \sqrt{2}$,
- την κυκλική συχνότητά της $ω = 2πν$,
- την εμπέδηση $Z = \sqrt{\frac{1}{R^2} + (ωC)^2}$ του κυκλώματος,
- το πλάτος i_0 και την ενεργή τιμή $i_{εν}$ του ρεύματος,
- τη διαφορά φάσης μεταξύ U και i, $\phi = τοξεφ(ωCR)$.

2-12 Στην §2.5 είδαμε ένα πρόγραμμα που μας δίνει τη μέγιστη και την ελάχιστη τιμή του τύπου `int`. Αν στις εντολές του προγράμματος αντικαταστήσεις το `"INT"` με `"LONG"`, `"SHRT"`, `"CHAR"`, `"UINT"`, `"ULONG"`, `"USHRT"`, `"UCHAR"`, θα πάρεις τα στοιχεία του Πίν. 2-1 για τους τύπους `long int`, `short int`, `char`, `unsigned int`, `unsigned long int`, `unsigned short int`, `unsigned char`, αντιστοίχως. Γράψε πρόγραμμα που θα βγάζει αυτά τα στοιχεία.

Στην ίδια παράγραφο είδαμε και ένα πρόγραμμα που μας δίνει τις χαρακτηριστικές τιμές του τύπου `double`. Αν αλλάξεις το `"DBL"` σε `"FLT"` και `"LDBL"` θα πάρεις τα στοιχεία των τύπων `float` και `long double` αντιστοίχως. Γράψε πρόγραμμα που θα βγάζει τα στοιχεία του Πιν. 2-2.

Γ Ομάδα

2-13 Γράψε πρόγραμμα που θα λύνει το πρόβλημα της ελεύθερης πτώσης, στην περίπτωση που η αρχική ταχύτητα δεν είναι μηδέν και έχει συνιστώσες στον κατακόρυφο και τον οριζόντιο άξονα.