κεφάλαιο

8

Αρχεία I – Text

Ο στόχος μας σε αυτό το κεφάλαιο:
Να μάθεις να διαχειρίζεσαι τη βοηθητική μνήμη με εργαλείο το *σειριακό μορφοποιημένο αρχείο* ή *αρχείο-κείμενο*.

Προσδοκώμενα αποτελέσματα:
Θα μπορείς να γράφεις προγράμματα που θα φυλάγουν δεδομένα σε αρχείο ή αρχεία ώστε να τα χρησιμοποιήσεις αργότερα με άλλα προγράμματα. Με λίγη δουλειά παραπάνω θα μπορείς να επεξεργαστείς και αρχεία text που βγάζουν διάφορα «πακέτα» (έτοιμα προγράμματα) ή να ετοιμάσεις στοιχεία εισόδου για τέτοια «πακέτα».

Έννοιες κλειδιά:

- *σειριακό αρχείο*
- *μορφοποιημένο αρχείο ή αρχείο-κείμενο* (*text*)
- *μη-μορφοποιημένο ή δυαδικό* (*binary*) *αρχείο*

Περιεχόμενα:

Εισαγωγικές Παρατηρήσεις:
Στις συνηθισμένες εφαρμογές των ΗΥ, τα στοιχεία που έχει να επεξεργαστεί ένα πρόγραμμα είναι τόσο πολλά, που δεν είναι δυνατό να χωρέσουν στην κύρια μνήμη του ΗΥ, όλα μαζί. Ακόμη, τα στοιχεία αυτά δεν πρέπει να χάνονται όταν ο ΗΥ δεν λειτουργεί,

πράγμα που δεν συμβαίνει με τα στοιχεία που υπάρχουν στην κύρια μνήμη. Για το λόγο αυτόν, τα στοιχεία αποθηκεύονται σε βοηθητικές μονάδες μνήμης –δίσκους, ταινίες κλπ– και έχουν μια λογική οργάνωση ώστε να μπορούν να χρησιμοποιηθούν εύκολα από τα προγράμματα.

Ο πιο απλός τρόπος οργάνωσης είναι το **σειριακό αρχείο** (serial ή sequential file).

Μπορούμε να φανταστούμε το σειριακό αρχείο σαν μια πεπερασμένη ακολουθία από όμοιες –δηλ. του ίδιου τύπου– τιμές[1]. Ονομάζουμε **μήκος** του αρχείου τον αριθμό των στοιχείων που έχει. Το μήκος του *κενού αρχείου* είναι 0.

Σημείωση:▶

Εκτός από *μήκος* υπάρχει και το **μέγεθος** (size) του αρχείου που είναι ο αριθμός των ψηφιολέξεων που καταλαμβάνει. Για τα αρχεία που θα δούμε σε αυτό το κεφάλαιο –*αρχεία χαρακτήρων*– έχουμε *μήκος* == *μέγεθος*.◀

Παράδειγμα

Το:

```
< -7, -15, 0, 14, 33, -8, 16, 114, 375 >
```

είναι ένα αρχείο με στοιχεία τύπου `int` και μήκος 9.

Το:

```
<'m', 'a', 'i', 'n', '(', ')', '{', '}'>
```

είναι ένα αρχείο με στοιχεία τύπου `char` και μήκος 8 (είναι ένα πρόγραμμα C++).

Το:

```
<true, true, false, true, false, false, true>
```

είναι ένα αρχείο με στοιχεία τύπου `bool` και μήκος 7.

Πώς μπορούμε να «δούμε» το περιεχόμενο ενός αρχείου; Αν είναι ένα αρχείο σαν το δεύτερο του παραδείγματος ξέρεις ήδη την απάντηση: το παίρνουμε σε κάποιον κειμενογράφο και το βλέπουμε. Αυτό είναι ένα παράδειγμα **κειμένου** (text) ή **μορφοποιημένου αρχείου** (formatted file)[2]. Υπάρχει όμως και μια άλλη κατηγορία αρχείων: αυτά που τα στοιχεία τους είναι αντίγραφα των εσωτερικών παραστάσεων του υπολογιστή. Αν, ας πούμε, το πρώτο παράδειγμα είναι γραμμένο έτσι, μπορείς να το πάρεις στον κειμενογράφο αλλά δεν θα μπορείς να καταλάβεις το περιεχόμενό του. Αυτά τα αρχεία λέγονται **δυαδικά** (binary) ή **μη μορφοποιημένα** (unformatted).

Ένα αρχείο-κείμενο:

- Μπορείς να το μεταφέρεις εύκολα σε διαφορετικά υπολογιστικά περιβάλλοντα και διαφορετικούς υπολογιστές.
- Μπορείς να το διαχειριστείς με πρόγραμμα, αλλά και με έναν απλό κειμενογράφο.

Αλλά:

- Αν έχει αριθμητικά δεδομένα, η διαχείρισή τους επιβραδύνεται από το ότι
  - οι τιμές της εσωτερικής παράστασης θα πρέπει να μεταφράζονται σε χαρακτήρες, όταν γράφονται στο αρχείο
  - οι χαρακτήρες (ψηφία) που διαβάζονται από το αρχείο θα πρέπει να μεταφράζονται στην εσωτερική παράσταση.

Ένα δυαδικό αρχείο:

---

[1] Θα χρησιμοποιήσουμε τα σύμβολα "<", ">", "," για να παραστήσουμε ένα αρχείο στο χαρτί και μόνο. Φυσικά: α) δεν τα χρησιμοποιεί ο ΗΥ όταν αποθηκεύει το αρχείο β) δεν θα πρέπει να τα μπερδεύεις με τα ειδικά σύμβολα της C++. Με αυτόν το συμβολισμό, το **κενό** αρχείο –δηλ. αυτό που δεν έχει στοιχεία– παριστάνεται ως: <>.

[2] Καμιά φορά θα το δεις και ως *αρχείο ascii*.

- Παίρνει τα στοιχεία του από τη μνήμη (ή τα δίνει σε αυτή) χωρίς καμμιά μετατροπή και έτσι η επεξεργασία του είναι πιο γρήγορη.

Αλλά:

- Δεν μπορείς να το μεταφέρεις εύκολα.
- Η διαχείρισή του απαιτεί ειδικό πρόγραμμα.

Για τους παραπάνω λόγους τα διάφορα «πακέτα», συνηθέστατα, μπορούν να γράφουν και να διαβάζουν αρχεία-κείμενα, ώστε να μπορούν να ανταλλάσσουν στοιχεία από τη μια εγκατάστασή τους στην άλλη. Άλλα αρχεία που δημιουργούν, για να διαβαστούν από άλλο «πακέτο»

- μπορεί να είναι αρχεία-κείμενα,
- μπορεί να είναι δυαδικά αρχεία,
- συχνά χρειάζονται ειδικά προγράμματα (φίλτρα) είτε είναι δυαδικά είτε είναι κείμενα.

Στο κεφάλαιο αυτό θα ασχοληθούμε αποκλειστικά με αρχεία-κείμενα.

8.1 Σειριακά Αρχεία στην C++

Σε κάθε Λειτουργικό Σύστημα (ΛΣ) υπάρχει ένα κομμάτι που λέγεται **διαχειριστής αρχείων** (file manager) ή **σύστημα για τα αρχεία** (file system). Ο διαχειριστής αρχείων κρατάει **καταλόγους** (directories) όπου για κάθε αρχείο υπάρχουν στοιχεία όπως: η θέση του πάνω στο μέσο αποθήκευσης, το μέγεθος του αρχείου, πότε δημιουργήθηκε το αρχείο, πότε ενημερώθηκε τελευταία φορά κλπ. Μέσα στον κατάλογο, το αρχείο έχει κάποιο *όνομα* που το έχει ορίσει αυτός που το δημιούργησε.

Παράδειγμα

Παρακάτω, βλέπεις την εικόνα καταλόγου που μας δίνει ο διαχειριστής αρχείων του ΛΣ Windows:

```
Volume in drive C has no label
 Serial Number of Volume is  50DA-631A

 Directory of C:\0809bea\t02

14/04/2009  03:10 AM    <DIR>          .
14/04/2009  03:10 AM    <DIR>          ..
13/04/2009  09:59 AM             7,875 AircraftType.cpp
13/04/2009  09:57 AM             3,319 AircraftType.h
24/03/2009  02:52 PM               250 aircraftTypes.txt
13/04/2009  09:05 AM            14,016 ask01a.cpp
13/04/2009  03:28 AM           522,116 ask01a.exe
13/04/2009  09:41 AM            14,538 ask01b.cpp
13/04/2009  09:41 AM           520,081 ask01b.exe
24/03/2009  03:36 PM             7,636 flightHours.txt
17/11/2008  10:08 PM             7,090 MyLib.cpp
13/04/2009  10:00 AM             3,856 nAircraftTypes.txt
13/04/2009  09:34 AM             3,856 nPilots.txt
13/04/2009  10:16 AM           233,472 ooPr_t02.doc
13/04/2009  10:18 AM               604 ooPr_t02.log
13/04/2009  10:20 AM           362,918 ooPr_t02.pdf
13/04/2009  10:17 AM         1,146,318 ooPr_t02.prn
13/04/2009  10:21 AM           371,671 ooPr_t02.ZIP
13/04/2009  06:50 AM           367,616 ooPr_t02b.doc
13/04/2009  03:23 AM             7,126 Pilot.cpp
13/04/2009  03:19 AM             3,441 Pilot.h
24/03/2009  04:03 PM             1,583 pilots.txt
09/04/2009  02:12 PM           677,110 t02.rar
              20 file(s)     4,276,492 bytes
               2 Directories 50,009,026,560 bytes free
```

Κάθε καταχώριση (γραμμή) του καταλόγου έχει: όνομα αρχείου, μέγεθος (σε ψηφιολέξεις), την ημερομηνία και την ώρα που ενημερώθηκε για τελευταία φορά.

Για να χειριστούμε ένα αρχείο μέσα από το πρόγραμμά μας θα πρέπει:

1. το ΛΣ να μας επιτρέψει πρόσβαση στο αρχείο αυτό,
2. να δημιουργήσουμε ένα κανάλι ή, όπως το λέει η C++, ένα **ρεύμα** (stream) από το αρχείο προς το πρόγραμμά μας (όταν διαβάζουμε) ή από το πρόγραμμά μας προς το αρχείο (όταν γράφουμε).
3. να βρούμε την αρχή του, από όπου θα αρχίσει η ανάγνωση ή η εγγραφή στο αρχείο.

Λέμε ότι πρέπει να **ανοίξουμε** (open) το ρεύμα του αρχείου.

Εδώ θα γνωρίσουμε τρεις τύπους (κλάσεις) ρευμάτων της C++:

- τον `ifstream` για ρεύματα από το αρχείο προς το πρόγραμμά μας (**i**nput **f**ile **stream**),
- τον `ofstream` για ρεύματα από το πρόγραμμά μας προς το αρχείο (**o**utput **f**ile **stream**),
- τον `fstream` για ρεύματα που μπορεί να είναι και διπλής κατεύθυνσης.

Για να χρησιμοποιήσεις αυτούς τους τύπους θα πρέπει να περιλάβεις στο πρόγραμμά σου το αρχείο fstream (`#include <fstream>`).

Πριν προχωρήσουμε πιο κάτω θα κάνουμε μια παρένθεση για να πούμε δύο λόγια για τις κλάσεις. Μια **κλάση** (class) είναι ένας τύπος στοιχείων. Οι μεταβλητές και οι τιμές μιας κλάσης ονομάζονται και **αντικείμενα** (objects). Αυτά έχουν **περικλεισμένες** (encapsulated) **μεθόδους** (methods) ή **συναρτήσεις-μέλη** (member functions), για τη διαχείριση των στοιχείων τους. Π.χ. οι τρεις κλάσεις που αναφέραμε πιο πάνω έχουν μια μέθοδο που ονομάζεται *open* για άνοιγμα ρεύματος. Αν λοιπόν έχουμε δηλώσει:

```
ifstream a;
```

ανοίγουμε το ρεύμα *a* από το αρχείο text1.txt προς το πρόγραμμά μας με την:

```
a.open( "text1.txt" );
```

Ένα άλλο χαρακτηριστικό των κλάσεων είναι η **κληρονομιά** (inheritance) που μπορεί να αφήσει μια κλάση σε μια άλλη κλάση-παιδί της, π.χ. διάφορες μέθοδοι, τελεστές κλπ. Όπως θα δεις στη συνέχεια, μετά από τις παραπάνω δηλώσεις θα μπορούμε να διαβάζουμε από το αρχείο text1.txt ως εξής:

```
a >> x;
```

όπως ακριβώς διαβάζουμε από το πληκτρολόγιο με τη:

```
cin >> x;
```

παρ' όλο που το *cin* είναι ρεύμα κλάσης *istream*. Η *ifstream* είναι απόγονος της *istream* και –μέσω αυτής– μιας άλλης κλάσης, της *ios_base*, από την οποίαν κληρονόμησαν και οι δύο τον τελεστή "`>>`".

Αυτά τα ολίγα για τις κλάσεις προς το παρόν· θα τις ξαναδούμε αργότερα.

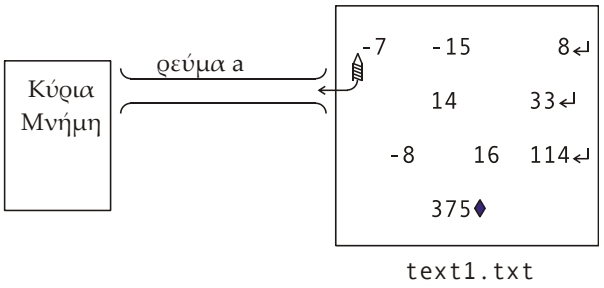
8.2 Αρχεία και Ρεύματα – Μια Εικόνα

Θα δώσουμε τώρα μια εικόνα για το (σειριακό) αρχείο-κείμενο και με αυτή θα δούμε τη διαχείρισή του.

Ας πούμε λοιπόν ότι το αρχείο είναι «γραμμένο» πάνω σε μια μαγνητοταινία (Σχ. 8-1).

Το αρχείο έχει μια αρχή και ένα τέλος. Το τέλος, εδώ στην εικόνα, το σημειώνουμε με το "♦". Αυτό δεν σημαίνει ότι ο ΗΥ σημειώνει το τέλος αρχείου με τον ίδιο τρόπο, αλλά, όπως θα δεις παρακάτω, αυτό δεν μας ενδιαφέρει. Το αρχείο-κείμενο είναι χωρισμένο σε γραμμές· στο σχήμα μας σημειώνουμε το τέλος γραμμής με το "↵".

Μια μαγνητική κεφαλή (παρόμοια με αυτήν του μαγνητοφώνου) διαβάζει το (γράφει στο) αρχείο. Λέγεται **κεφαλή ανάγνωσης/εγγραφής** (read/write head).



Σχ. 8-1 Παράσταση αρχείου-κειμένου (text1.txt) και ρεύματος *a* για την ανάγνωσή του. Παριστάνουμε το αρχείο όπως περίπου θα το δεις σε έναν κειμενογράφο. Τα "↵" και "♦" σημειώνουν το τέλος γραμμής και αρχείου αντιστοίχως. Το "⇧" παριστάνει την κεφαλή ανάγνωσης/εγγραφής

Η παράσταση του Σχ. 8-1 είναι απλουστευμένη αλλά θα μας βοηθήσει να καταλάβουμε τη διαχείριση του αρχείου-κειμένου. Βέβαια κρύβουμε λεπτομέρειες και περιπλοκές. Μια από αυτές είναι η ύπαρξη και ο ρόλος του *ενταμιευτή* που θα μας χρειαστεί σε κάποιο σημείο στη συνέχεια. Παρακάτω λέμε μερικά πράγματα γι' αυτόν.

Με τον όρο **ενταμιευτής** αποδίδουμε την αγγλική λέξη «buffer». Στα αγγλικά σημαίνει: αυτός που απαλλάσσει κάποιον από μια ενοχλητική δουλειά. Και αυτό ακριβώς κάνει ο ενταμιευτής: απαλλάσσει την ΚΜΕ (Κεντρική Μονάδα Επεξεργασίας, CPU) από τη συνεχή απασχόληση με τις περιφερειακές μονάδες –στην περίπτωσή μας μονάδες βοηθητικής μνήμης. Ο έλεγχος των μονάδων αυτών είναι αρκετά χρονοβόρος. Γι' αυτό, το ρεύμα περιλαμβάνει μια περιοχή μνήμης αρκετά μεγάλη (συνηθισμένες τιμές 1 *kB*, 2 *kB*). Όταν λοιπόν ζητήσουμε ανάγνωση μιας τιμής από το αρχείο δεν φέρνει μόνον αυτό που ζητήσαμε (π.χ. `'-'` και `'7'`) αλλά, π.χ., 1024 χαρακτήρες. Έτσι, οι επόμενες αναγνώσεις δεν απαιτούν πρόσβαση στο δίσκο αλλά μεταφορές μέσα στην κύρια μνήμη. Παρομοίως, όταν γράφουμε στο αρχείο, στην πραγματικότητα τα στοιχεία γράφονται στον ενταμιευτή. Όταν αυτός «γεμίσει», το περιεχόμενό του αντιγράφεται στο αρχείο.

8.3 Πώς Διαβάζουμε Ένα Αρχείο

Ας πούμε λοιπόν ότι στο δίσκο μας υπάρχει το αρχείο-κείμενο text1.txt. Τώρα θα δούμε πώς θα γράψουμε ένα πρόγραμμα που θα διαβάζει αυτό το αρχείο.

Όπως είπαμε και παραπάνω, μας χρειάζεται ένα ρεύμα κλάσης *ifstream*.Θα πρέπει λοιπόν να δηλώσουμε:

```
ifstream  a;
```

και να το ανοίξουμε ώστε να συνδέσει το αρχείο με το πρόγραμμά μας:

```
a.open( "text1.txt" );
```

Το στιγμιότυπο που βλέπεις στο Σχ. 8-1 δείχνει την κατάσταση που βρισκόμαστε μετά την *open()*. Η κεφαλή ανάγνωσης/εγγραφής βρίσκεται στην αρχή του αρχείου.

Ας πούμε ότι έχουμε δηλώσει ακόμη:

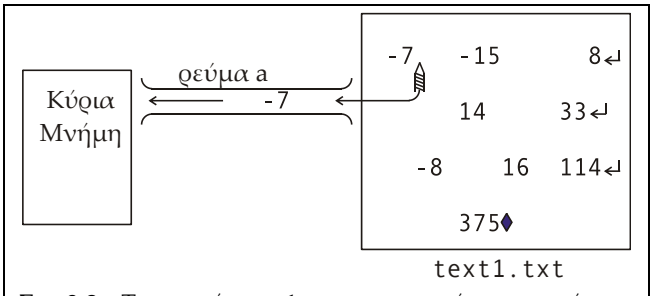
```
short int x, y;
```

και θέλουμε να διαβάσουμε την πρώτη τιμή του αρχείου και να την αποθηκεύσουμε στη *x*. Δίνουμε την εντολή:

```
a >> x;
```

Με την εκτέλεση αυτής της εντολής γίνονται τα εξής:

1. Η κεφαλή περνάει το αρχικό κενό και φτάνει στο `'-'`.
2. Η κεφαλή διαβάζει τους χαρακτήρες `'-'` και `'7'` και σταματάει όταν βρει το κενό.
3. Οι χαρακτήρες διαβιβάζονται στην κύρια μνήμη.



Σχ. 8-2 Το αρχείο text1.txt και το ρεύμα *a* μετά την ανάγνωση της πρώτης τιμής.

4. Οι δύο χαρακτήρες μεταφράζονται στην εσωτερική παράσταση του αριθμού "**-7**" που αποθηκεύεται στη θέση της μνήμης *x*.

Στο Σχ. 8-2 δείχνουμε την κατάσταση μετά την ανάγνωση της πρώτης τιμής.

Στη συνέχεια μπορούμε να επεξεργαστούμε αυτήν την τιμή· μπορούμε να δώσουμε, για παράδειγμα:

```
y = x*x;
cout << "   x = " << x << "   x^2 = " << y << endl;
```

Παίρνουμε αποτέλεσμα:

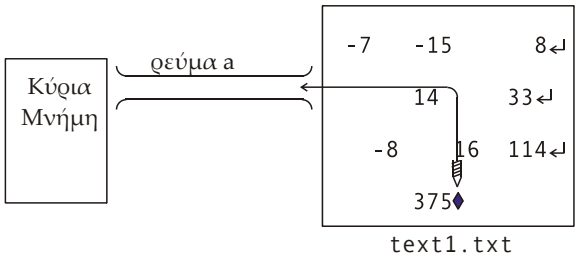
```
x = -7   x^2 = 49
```

Αν τώρα θέλουμε να επεξεργαστούμε τη δεύτερη τιμή, ζητούμε και πάλι εκτέλεση της εντολής:

```
a >> x;
```

Η κεφαλή θα ξεπεράσει τα κενά και θα διαβάσει και θα στείλει μέσω του *a* τους χαρακτήρες '**-**', '**1**', '**5**'. Με τον ίδιο τρόπο διαβάζεται και το 8. Αν ξαναζητήσουμε ανάγνωση τιμής, η κεφαλή θα ξεπεράσει το σημάδι τέλους γραμμής και τα κενά για να διαβάσει τους '**1**', '**4**'.

Συνεχίζοντας με αυτόν τον τρόπο φτάνουμε κάποτε στην κατάσταση που βλέπεις στο Σχ. 8-3. Αν ζητήσεις και πάλι εκτέλεση της `a >> x` αυτή θα αποτύχει και η τιμή της *x* δεν θα αλλάξει. Κάπου μέσα στον υπολογιστή σου «θα σηκωθεί μια σημαία» –κάποιο δυαδικό ψηφίο θα αλλάξει από 0 σε 1– που θα δείχνει ότι φτάσαμε στο τέλος του αρχείου. Μπορούμε να δούμε αυτή τη σημαία; Ναι! Η *ifstream* έχει μια μέθοδο, που ονομάζεται *eof* (χωρίς παραμέτρους) από τις αγγλικές λέξεις **e**nd **o**f **f**ile (τέλος αρχείου). Αν ελέγξεις την τιμή της `a.eof()`



Σχ. 8-3 Τι θα γίνει αν στην κατάσταση αυτή εκτελεστεί η εντολή `a >> x`; Η τιμή της *x* δεν θα αλλάξει. Αν ελέγξουμε την τιμή της `a.eof()` θα είναι μη μηδενική (`true`).

- μετά από μια προσπάθεια ανάγνωσης που απέτυχε διότι φτάσαμε στο τέλος του αρχείου αυτή θα είναι μη μηδενική (ως συνθήκη μεταφράζεται σε `true`)
- μετά από μια επιτυχή προσπάθεια ανάγνωσης αυτή θα είναι μηδενική (που ως συνθήκη θεωρείται `false`).

Φυσικά, αν η ανάγνωση δεν γίνει δεν έχει νόημα να επεξεργαστούμε την τιμή της *x*.

Το παρακάτω απλό πρόγραμμα κάνει μια στοιχειώδη επεξεργασία στο αρχείο που χρησιμοποιήσαμε στα παραδείγματά μας μέχρι τώρα. Υπολογίζει το τετράγωνο του κάθε στοιχείου. Επειδή το πρόγραμμα εκτελείται σε μια υλοποίηση της C++ όπου *SHRT_MAX* = 32767, αν η απόλυτη τιμή κάποιου στοιχείου είναι μεγαλύτερη από 181 (που είναι η ακέραια προσέγγιση της $\sqrt{SHRT\_MAX}$) τότε υπολογίζει το τετράγωνο του `x % 181`.

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

int main()
{
   ifstream  a;
   short int x, xr, y;

   a.open( "text1.txt" );
   a >> x;
   while ( !a.eof() )
```

```
   {
      if ( abs(x) > 181 )  xr = x % 181;
                     else  xr = x;
      y = xr*xr;
      cout << "   x =";  cout.width(4);  cout << x;
      cout << "    xr =";  cout.width(4);  cout << xr;
      cout << "    xr^2 =";  cout.width(5);  cout << y << endl;
      a >> x;
   } // while
   a.close();
} // main
```

Αυτό το πρόγραμμα δίνει τελικώς:

```
x =  -7    xr =  -7    xr^2 =   49
x = -15    xr = -15    xr^2 =  225
x =   8    xr =   8    xr^2 =   64
x =  14    xr =  14    xr^2 =  196
x =  33    xr =  33    xr^2 = 1089
x =  -8    xr =  -8    xr^2 =   64
x =  16    xr =  16    xr^2 =  256
x = 114    xr = 114    xr^2 =12996
x = 375    xr =  13    xr^2 =  169
```

Πρόσεξε την τελευταία εντολή:

```
   a.close();
```

Με την εκτέλεσή της το ρεύμα *a* αποσυνδέεται από το αρχείο text1.txt. Αν θέλουμε μπορούμε να το χρησιμοποιήσουμε για να επεξεργαστούμε άλλο αρχείο.

Το πρόγραμμα αυτό είναι παράδειγμα επεξεργασίας σειριακού αρχείου. Στο Πλ. 8.1 βλέπεις το γενικό σχήμα αυτής της επεξεργασίας. *Ξεκινώντας με a.open(..) και προχωρώντας μέχρις ότου η* `a.eof()` *δώσει μη μηδενική τιμή είναι σίγουρο ότι θα διαβάσουμε όλες τις τιμές του αρχείου*. Αυτό δεν σημαίνει ότι πρέπει να τις επεξεργαστείς όλες υποχρεωτικώς· μπορείς να κάνεις επιλεκτική επεξεργασία. Όπως βλέπεις, το σχήμα επεξεργασίας σειριακού αρχείου είναι όμοιο με το σχήμα επανάληψης με φρουρό.

Παρατηρήσεις ▶

1. Ας δούμε τώρα το εξής πρόβλημα: θέλουμε να διαβάσουμε την *πέμπτη* τιμή του αρχείου (το 33). Πρέπει να κάνουμε τα εξής:

i. Να φέρουμε το αρχείο στην αρχή του (Σχ. 8-1).

ii. Να διαβάσουμε και να αγνοήσουμε τις τέσσερις πρώτες τιμές του αρχείου.

iii. Να διαβάσουμε την πέμπτη τιμή.

Στις διαδικασίες αυτές φαίνεται πολύ καλά ένα βασικό χαρακτηριστικό των σειριακών αρχείων:

- ***Για να διαβάσουμε (ή να γράψουμε) το ν-οστό στοιχείο ενός σειριακού αρχείου πρέπει να περάσουμε κατ' ανάγκη τις ν – 1 προηγούμενες τιμές.***

Για να γίνει όμως αυτό,

- ***Η επεξεργασία κάθε σειριακού αρχείου ξεκινάει πάντα από την αρχή του.***

Πλαίσιο 8.1

Επεξεργασία Σειριακού Αρχείου

```
ifstream a;
   :
a.open( όνομα αρχείου );
a >> x;
while (!a.eof())
{
   Εντολές επεξεργασίας της x
   a >> x;
} // while
a.close();
```

2. Κάτι άλλο που φαίνεται στο πρόγραμμά μας είναι το εξής: μετά από την εκτέλεση της "`a >> x`" υπολογίζεται η "`!a.eof()`" και μόνο αν βρεθεί `true`, αν η ανάγνωση έγινε κανονικώς, επεξεργαζόμαστε την τιμή της *x*. Και αυτή είναι μια γενική αρχή:

- ***Πριν επεξεργαστούμε μια τιμή που (υποτίθεται ότι) διαβάσαμε από ένα αρχείο ελέγχουμε την eof για να δούμε αν η ανάγνωση έγινε κανονικά.***

3. Όταν προσπαθείς να πάρεις μια τιμή από ένα αρχείο-κείμενο με τον "`>>`", εκτός από τα κενά (διαστήματα) και τα σημάδια τέλους γραμμής, «τρώγονται» και οι στηλοθέτες (tabs). Όλα αυτά η C++ τα ονομάζει **λευκά διαστήματα** (white spaces).

4. Αν το αρχείο που επεξεργάζεσαι δεν είναι στον ίδιο υποκατάλογο με το πρόγραμμά σου θα πρέπει να περιλάβεις στο όνομα και τη **διαδρομή** (path). Αν δουλεύεις σε περιβάλλον Windows η διαδρομή θα περιλαμβάνει τον χαρακτήρα `'\'` που, όπως έχουμε πει, πρέπει να γράφεται δύο φορές. Αν έχεις π.χ.:

```
c:\students\datfl\text1.txt
```

θα πρέπει να δώσεις:

```
a.open( "c:\\students\\datfl\\text1.txt" );
```

4. Έχεις τη δυνατότητα να ανοίξεις το αρχείο όταν το δηλώνεις, π.χ.:

```
ifstream a( "text1.txt" );
```

Στην περίπτωση αυτή, δεν θα δώσεις `a.open("text1.txt")` στη συνέχεια.◀

8.3.1 *cin.eof()*

Μπορούμε να ελέγχουμε για «τέλος αρχείου» όταν παίρνουμε δεδομένα από το πληκτρολόγιο μέσω του *cin*; Ναι! Ο χρήστης «πληκτρολογεί το eof» δίνοντας

- `char(3)` (**E**nd of **Te**x**t**, ETX) με πληκτρολόγηση <ctrl-C> ή
- `char(4)` (**E**nd **O**f **T**ransmission, EOT) με πληκτρολόγηση <ctrl-D> ή
- `char(26)` με πληκτρολόγηση <ctrl-Z> (συνήθως σε περιβάλλον Windows).

Δυστυχώς η πληκτρολόγηση εξαρτάται από το ΛΣ και τον μεταγλωττιστή. Για παράδειγμα σε εφαρμογές «κονσόλας» στα Windows ο ETX δεν μπορεί να χρησιμοποιηθεί διότι του «έχει ανατεθεί» άλλος ρόλος (διακόπτει την εκτέλεση.) Χρησιμοποιείται, από την εποχή του MS-DOS, ο <ctrl-Z>. Θα το δεις να δουλεύει αν χρησιμοποιείς gcc (π.χ. Dev C++) ή BC++ 5.02.

8.3.2 Για να Ξαναχρησιμοποιήσεις το Ρεύμα

Ας πούμε ότι μετά την

```
ifstream a( "text1.txt" );
```

διάβασες το αρχείο `text1.txt` μέχρι το τέλος του και σταμάτησες την ανάγνωση μόλις πήρες *a.eof()*. Σε ποια κατάσταση είναι το ρεύμα; Σε «κακή κατάσταση»[3] ή, με άλλα λόγια, έχει ανασταλεί η λειτουργία του: δεν μπορεί να δεχθεί οποιαδήποτε εντολή· σε ορισμένες περιπτώσεις (μεταγλωττιστές) δεν εκτελεί ούτε την *close()*! Αν το ρεύμα έχει ανοιχτεί για διάβασμα δεν πάθαμε ζημιά εκτός από μια περίπτωση: Αν θέλουμε να ξαναχρησιμοποιήσουμε το ρεύμα για να δουλέψουμε με το ίδιο αρχείο ή με κάποιο άλλο.

Για να μπορούμε να ξαναχρησιμοποιήσουμε το ρεύμα θα πρέπει το *ανατάξουμε* δίνοντας την εντολή:

```
a.clear();
```

Ας πούμε ότι μετά την *a.eof()* θέλουμε να χρησιμοποιήσουμε το *a* για να διαβάσουμε ένα άλλο αρχείο, ας πούμε το `text2.txt`. Στην περίπτωση αυτή οι εντολές:

```
a.close();
a.open( "text2.txt" );
```

δεν αρκούν. Το σωστό είναι να γράψουμε:

```
a.clear();
a.close();
a.open( "text2.txt" );
```

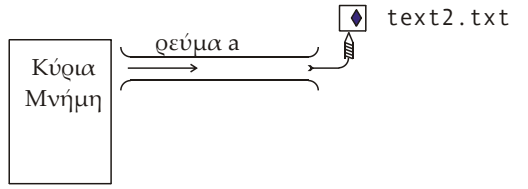
8.4 Πώς Γράφουμε Ένα Αρχείο

Τώρα θα δούμε πώς δημιουργούμε ένα αρχείο-κείμενο. Όπως καταλαβαίνεις τώρα θα πρέπει να δημιουργήσουμε ένα ρεύμα *από το πρόγραμμά μας προς το αρχείο*. Θα πρέπει λοιπόν να δηλώσουμε:

```
ofstream a;
```

Και η κλάση *ofstream* έχει μέθοδο *open()* για το άνοιγμα του αρχείου:

```
a.open( "text2.txt" );
```



Σχ. 8-4 Παράσταση αρχείου-κειμένου (text2.txt) και ρεύματος *a* για τη δημιουργία του. Το στιγμιότυπο δείχνει την κατάσταση ακριβώς μετά την εκτέλεση της `a.open('text2.txt')`. Το αρχείο είναι άδειο.

Εδώ όμως έχουμε διαφορά από την *open()* της *ifstream*: *Αν το αρχείο text2.txt υπήρχε, με την εκτέλεση της open() χάθηκε το περιεχόμενό του!* Το αρχείο καθαρίζεται και είναι έτοιμο για γράψιμο. Στο Σχ. 8-4 βλέπεις σχηματικά την κατάσταση μετά την εκτέλεση της *open()*.

Και τώρα θέλω να γράψω στο αρχείο την τιμή "`23`". Πώς θα γίνει αυτό; Μάλλον το έχεις μαντέψει: όπως η *ifstream* έχει τον "`>>`", που τον ξέρουμε από το *cin*, και η *ofstream* έχει τον, γνωστό μας από τη *cout*, "`<<`". Αν λοιπόν δώσουμε:

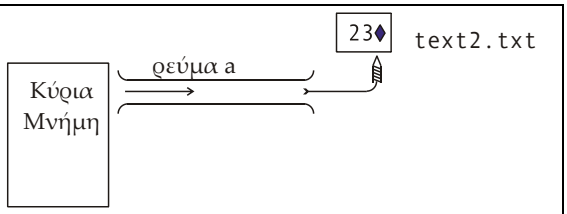
```
a << 23;
```

θα γίνουν αυτά που προσπαθούμε να δείξουμε στο Σχ. 8-5:

1. Η ακέραιη σταθερά «μεταφράζεται» στους χαρακτήρες `'2'`, `'3'`.
2. Οι χαρακτήρες διαβιβάζονται στη μαγνητική κεφαλή.
3. Οι χαρακτήρες γράφονται στο αρχείο.

Στη συνέχεια δίνουμε την εντολή:

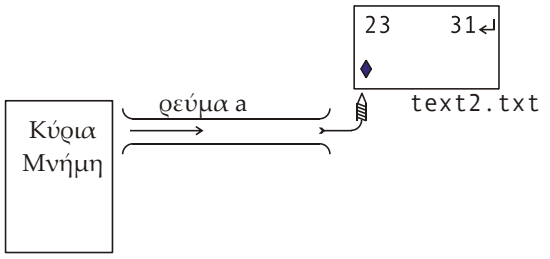
```
a << "    " << 31 << endl;
```



Σχ. 8-5 Παράσταση αρχείου-κειμένου (text2.txt) και ρεύματος *a* μετά την εκτέλεση της εντολής: `a << 23`.

[3] Όρος της C++ "(*std*::*ios*::)*bad*", θα τον δούμε αργότερα...

και το αποτέλεσμα είναι αυτό του Σχ. 8-6. Πρόσεξε τα τέσσερα διαστήματα: ζητήσαμε να γραφούν και γι' αυτό γράφηκαν. Αν δίναμε: "`a << 31`", το "`31`" θα «κολλούσε» στο "`23`" και θα είχαμε "`2331`".



Σχ. 8-6 Η κατάσταση μετά την εκτέλεση της εντολής: "`a << "  " << 31 << endl`". Λόγω του "`endl`" γράφηκε το σημάδι τέλους γραμμής και η κεφαλή είναι έτοιμη να γράψει στην αρχή της επόμενης γραμμής.

Το `endl` μας πηγαίνει στην αρχή νέας γραμμής. Είχαμε πει ότι γράφοντας στο *cout* τον `'\n'` έχουμε το ίδιο αποτέλεσμα· ισχύει αυτό και για τα αντικείμενα της κλάσης *ofstream*, όπως είναι το *a*; Ναι! Για την ακρίβεια το `'\n'` (= `char`(10)) είναι το σημάδι τέλους γραμμής για αρχεία-κείμενα που γράφει η C++. Τώρα όμως μπορούμε να δούμε τη διαφορά:

- Ζητώντας να γραφεί το `endl` πετυχαίνεις δύο πράγματα:
  - να πας σε νέα γραμμή και
  - να αντιγραφεί το όποιο περιεχόμενο του ενταμιευτή (η γραμμή που συμπληρώθηκε) στο αρχείο.
- Ζητώντας να γραφεί το `'\n'` πετυχαίνεις μόνο να πας σε νέα γραμμή. Το περιεχόμενο του ενταμιευτή θα αντιγραφεί όταν αυτός γεμίσει ή όταν ζητηθεί κάτι τέτοιο.

Το `endl` κάνει το πρόγραμμά σου *πιο αργό αλλά ασφαλέστερο*. Δηλαδή: αν, ας πούμε, έχεις διακοπή στην παροχή ηλεκτρικής ενέργειας την ώρα που δημιουργείς το αρχείο, όσες γραμμές ζήτησες να γραφούν, θα έχουν γραφεί στο αρχείο. Σε μια τέτοια περίπτωση, αν γράφεις με το `'\n'`, θα χάσεις όλο το περιεχόμενο του ενταμιευτή.

Όταν τελειώσεις το γράψιμο του αρχείου μην ξεχάσεις να το κλείσεις με μια *close()*. Η *close()* θα φροντίσει να αντιγράψει στο αρχείο όλο το περιεχόμενο του ενταμιευτή.

Το παρακάτω πρόγραμμα διαβάζει από το πληκτρολόγιο ζεύγη ακέραιων τιμών και τα γράφει, ένα ζεύγος ανά γραμμή σε αρχείο-κείμενο με όνομα text2.txt. Η ανάγνωση τελειώνει με το ζεύγος (0,0).

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

int main()
{
   ofstream a;
   int x, y;

   a.open( "text2.txt" );
   cout << " Δωσε τα x, y. 0,0 για τελος: ";  cin >> x >> y;
   while ( x != 0 || y != 0 )
   {
      a.width( 5 );  a << x;
      a.width( 5 );  a << y << endl;
      cout << " Δωσε τα x, y. 0,0 για τελος: ";  cin >> x >> y;
   } // while
   a.close();
} // main
```

Παρατηρήσεις: ▶

1. "Τι είναι το `a.width(5)`;" Ακριβώς σαν το `cout.width(5)`. Ζητάει να γραφεί το επόμενο στοιχείο στο αρχείο που είναι συνδεμένο με το *a* σε 5 θέσεις τουλάχιστον. Παρομοίως, για τα μέλη της κλάσης *ofstream* δουλεύουν όπως ξέρουμε οι μέθοδοι *precision* και *setf*.
2. Ας αλλάξουμε τη συνθήκη της **while**:

```
   cout << " Δωσε τα x, y. <ctrl-Z> για τελος: ";  cin >> x >> y;
```

```
   while ( !cin.eof() )
   {
      a.width( 5 );  a << x;
      a.width( 5 );  a << y << endl;
      cout << " Δωσε τα x, y. <ctrl-Z> για τελος: ";
      cin >> x >> y;
   } // while
```

Η εκτέλεση των επαναλήψεων θα τερματισθεί αν πληκτρολογήσουμε <ctrl-Z>.[4] Έτσι, δεν μας χρειάζεται φρουρός! ◄

8.5 Ένα «Πραγματικό» Πρόβλημα

Το παράδειγμα επεξεργασίας αρχείου που είδαμε στις προηγούμενες παραγράφους δεν είχε και πολύ νόημα. Εκεί θέλαμε να τραβήξουμε την προσοχή σου στις πράξεις του αρχείου. Τώρα ας δούμε ένα πιο «πραγματικό» πρόβλημα.

Έχουμε ένα αρχείο –με όνομα στο δίσκο exp4.txt– με τιμές τύπου **double**. Το πλήθος τους δεν είναι γνωστό. Θέλουμε να βρούμε και να τυπώσουμε το άθροισμα, τη Μέση Αριθμητική Τιμή και το πλήθος των πραγματικών αριθμών που υπάρχουν στο αρχείο. Ακόμη θέλουμε: το πλήθος, το άθροισμα και τη μέση τιμή όσων από τις τιμές που διαβάζει είναι θετικές και μέχρι 10.

Η αντίδρασή σου θα πρέπει να είναι: «Ωχ, πάλι αυτό!» ή «Μα αυτό το ξέρω!». Ας υποθέσουμε ότι είναι η δεύτερη και ας προχωρήσουμε. Γύρισε στο Κεφ. 6 (§6.1.2) για να δεις το πρόγραμμα *Μέση Τιμή* 4, ξαναδές το γενικό σχήμα επεξεργασίας σειριακού αρχείου (Πλ. 8.1) και ας γράψουμε το *Μέση Τιμή* 5.

Ας πούμε ότι το αρχείο θα το δουλέψουμε με το ρεύμα *a*. Θα πρέπει να το δηλώσουμε:

```
   ifstream a;
```

Με την τιμή - φρουρό τι θα κάνουμε; Τίποτε! *Δεν μας χρειάζεται!* Αντί να ψάχνουμε για φρουρό θα ελέγχουμε μήπως φτάσαμε στο τέλος του αρχείου. Να το γενικό σχήμα επεξεργασίας του αρχείου:

```
   a.open( "exp4.txt" );
   a >> x;
   while ( !a.eof() )
   {
      E
      a >> x;
   } // while
   a.close();
```

Ποιές είναι οι *E*; Νάτες:

```
      n = n + 1;               // Αυτά γίνονται για
      sum = sum + x;           // όλους τους αριθμούς
      if ( 0 < x && x <= 10 )  // 'Ελεγχος - Επιλογή
      {
         selSum = selSum + x;       // Αυτά γίνονται μόνο για
         selN = selN + 1;           // τους επιλεγόμενους αριθμούς
      } // if
```

Στη συνέχεια βλέπεις το πρόγραμμα *Μέση Τιμή* 5. Οι διαφορές του από το *Μέση Τιμή* 4 είναι πολύ μικρές. Και ο μόνος λόγος που ξαναπαραθέτουμε ολόκληρο, είναι ακριβώς για να δεις αυτήν την ομοιότητα και να επισημάνεις τις διαφορές.

```
// πρόγραμμα: Μέση Τιμή 5
#include <iostream>
#include <fstream>
using namespace std;
int main()
```

[4] Δοκίμασέ το και αν δουλέψει…

```
{
   ifstream a;

   int n;          // Μετρητής όλων των στοιχείων
   int selN;       // Μετρητής επιλεγόμενων στοιχείων
   double x;       // Εδώ αποθηκεύουμε κάθε τιμή που διαβάζουμε
   double sum;     // Το άθροισμα όλων των στοιχείων
   double selSum;  // Άθροισμα επιλεγόμενων στοιχείων
   double avrg;    // Μέση Αριθμητική Τιμή όλων των στοιχείων
   double selAvrg; // Μέση Αριθμητική Τιμή επιλεγόμενων στοιχείων

   sum = 0;  n = 0;
   selSum = 0;  selN = 0;
   a.open( "exp4.txt" );
   a >> x;
   while ( !a.eof() )
   {
      n = n + 1;               // Αυτά γίνονται για
      sum = sum + x;           // όλους τους αριθμούς
      if ( 0 < x && x <= 10 )  // 'Ελεγχος - Επιλογή
      {
         selSum = selSum + x;       // Αυτά γίνονται μόνο για
         selN = selN + 1;           // τους επιλεγόμενους αριθμούς
      } // if
      a >> x;
   } // while
   a.close();
   cout << " Διάβασα " << n << " αριθμούς" << endl;
   if ( n > 0 )
   {
      avrg = sum / n;
      cout << " ΑΘΡΟΙΣΜΑ = " << sum << "  <x> = " << avrg << endl;
   } // if ( n
   cout << " Διάλεξα " << selN << " αριθμούς ε (0,10]" << endl;
   if ( selN > 0 )
   {
      selAvrg = selSum/selN;
      cout << " ΑΘΡΟΙΣΜΑ = " << selSum
           << "  <x> = " << selAvrg << endl;
   } // if ( selN
} // main
```

Πρόσεξε ότι κλείνουμε το αρχείο αμέσως μόλις παύουμε να ασχολούμαστε μαζί του.

Ας πούμε τώρα ότι έχουμε το εξής πρόβλημα: Θέλουμε τα αποτελέσματα της επεξεργασίας όχι στην οθόνη αλλά σε ένα αρχείο-κείμενο με το όνομα report.txt. Τι θα πρέπει να αλλάξουμε στο *Μέση Τιμή* 5;

- Θα πρέπει να δηλώσουμε ένα ρεύμα *b* κλάσης *ofstream*, που θα το ανοίξουμε προς το report.txt.
- Ό,τι στέλνουμε στην οθόνη μέσω του *cout* να το στείλουμε στο report.txt μέσω του *b*.

```
// πρόγραμμα: Μέση Τιμή 6
#include <fstream>
using namespace std;
int main()
{
   ifstream a;
   ofstream b;
// . . .
   b.open( "report.txt" );
   b << " Διάβασα " << n << " αριθμούς" << endl;
   if ( n > 0 )
   {
      avrg = sum / n;
      b << " ΑΘΡΟΙΣΜΑ = " << sum << "  <x> = " << avrg << endl;
   } // if ( n
   b << " Διάλεξα " << selN << " αριθμούς ε (0,10]" << endl;
```

```
   if ( selN > 0 )
   {
      selAvrg = selSum/selN;
      b << " ΑΘΡΟΙΣΜΑ = " << selSum
        << "  <x> = " << selAvrg << endl;
   } // if ( selN
```

Ε! Ξεχάσαμε την "`#include <iostream>`"! Δεν την ξεχάσαμε! Την παραλείψαμε διότι δεν τη χρειαζόμαστε: ούτε το *cin* ούτε το *cout* υπάρχει στο πρόγραμμά μας.

Να πώς περίπου θα είναι το περιεχόμενο του report.txt:

```
Διάβασα 37 αριθμούς
ΑΘΡΟΙΣΜΑ = 560.767  <x> = 15.1559
Διάλεξα 12 αριθμούς ε (0,10]
ΑΘΡΟΙΣΜΑ = 46.6869  <x> = 3.89057
```

8.6 Και Διάβασμα και Γράψιμο

Ας δούμε τώρα το εξής πρόβλημα: Θέλουμε να γράψουμε ένα πρόγραμμα που θα διαβάζει άγνωστο πλήθος πραγματικών αριθμών x_1, x_2, ... –που καθένας τους είναι πολύ μικρότερος από 10000– και θα τους αποθηκεύει σε ένα αρχείο fltnum.txt. Θέλουμε ακόμη:

α) να υπολογίζει και να τυπώνει τη Μέση Τιμή τους <*x*>,

β) να υπολογίζει και να τυπώνει τις τιμές της συνάρτησης: $e^{\frac{<x>-x}{<x>}}$ για τους αριθμούς αυτούς. Δηλαδή τα $y_k = e^{\frac{<x>-x_k}{<x>}}$.

Ξέρουμε να κάνουμε τα πάντα! Ένα σημείο χρειάζεται προσοχή: για να υπολογίσουμε τη μέση τιμή θα πρέπει να έχουμε όλους τους αριθμούς· δηλαδή *θα την έχουμε υπολογίσει όταν θα έχουμε γράψει όλους τους αριθμούς στο αρχείο*. Για να υπολογίσουμε τα y_k θα πρέπει να ξαναδιαβάσουμε τους αριθμούς, όχι βέβαια από το πληκτρολόγιο (όχι και να τους ξαναπληκτρολογήσουμε), αλλά από το αρχείο.

Να λοιπόν το σχέδιό μας:

```
Πέρασε τους αριθμούς στο αρχείο και υπολόγισε πλήθος και μέση τιμή
Διάβασε όλους τους αριθμούς από το αρχείο και για κάθε έναν από
                    αυτούς (xk) υπολόγισε και γράψε το yk = e^((<x>-xk)/<x>)
```

Φυσικά θα πρέπει να προσέξουμε: Το δεύτερο βήμα θα εκτελεσθεί μόνο αν το πλήθος των αριθμών που δώσαμε (*n*) δεν είναι μηδέν:

```
Πέρασε τους αριθμούς στο αρχείο και υπολόγισε πλήθος και μέση τιμή
if ( n > 0 )
{  Διάβασε όλους τους αριθμούς από το αρχείο και για κάθε έναν από
                    αυτούς (xk) υπολόγισε και γράψε το yk = e^((<x>-xk)/<x>)  }
```

Το πρώτο βήμα είναι κατά βάση το *Μέση Τιμή* 3 με τις εξής διαφορές:

- Θα πρέπει να φυλάγουμε στο αρχείο κάθε τιμή που δίνεται.
- Θα πρέπει να αλλάξουμε φρουρό: το 0 δεν αποκλείεται ενώ μια καλή επιλογή είναι το 9999.

```
   const double sentinel = 9999.0;
   ofstream  a;
// . . .
   sum = 0;  n = 0;
   a.open( "fltnum.txt" );
   cout << " Δώσε αριθμό - 9999 για ΤΕΛΟΣ: ";  cin >> x;
   while ( x != sentinel )
   {
   // γράψε στο αρχείο την τιμή που πήρες
      a << x << endl;
   // πρόσθεσέ την στην sum και μέτρησέ την
```

```
      n = n + 1;
      sum = sum + x;
      cout << " Δώσε αριθμό - 9999 για ΤΕΛΟΣ: ";  cin >> x;
   } // while
   a.close();
   cout << " Διάβασα " << n << " αριθμούς" << endl;
   if ( n > 0 )
   {
      avrg = sum / n;
      cout << " ΑΘΡΟΙΣΜΑ = " << sum
           << "  <x> = " << avrg << endl;
   }
```

Για το δεύτερο βήμα τροποποιούμε το πρόγραμμα της προηγούμενης παραγράφου:

```
   ifstream  b;      // για διάβασμα αρχείου

      b.open( "fltnum.txt" );   m = 0;
      b >> x;
      while ( !b.eof() )
      {
         m = m + 1;
         y = exp( (avrg - x)/avrg );
         cout << " x[";
         cout.width(3);  cout << m << "] = ";
         cout.width(6);  cout << x << "   y[";
         cout.width(3);  cout << m << "] = " ;
         cout.width(6);  cout << y << endl;
         b >> x;
      } // while
      b.close();
```

Όλα αυτά θα γίνουν βέβαια αν και μόνον αν $n > 0$.

Να ολόκληρο το πρόγραμμα:

```
// πρόγραμμα: Μέση Τιμή 3+
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;
int main()
{
   const double sentinel = 9999.0;
   ofstream  a;
   ifstream  b; // για διάβασμα αρχείου
   int n;       // Μετρητής των επαναλήψεων. Η τελική
                // τιμή είναι το πλήθος των στοιχείων
   double x;    // Εδώ αποθηκεύουμε κάθε τιμή που διαβάζουμε
   double sum;  // Το μερικό (τρέχον) άθροισμα.
                // Στο τέλος έχει το ολικό άθροισμα.
   double avrg; // Μέση Αριθμητική Τιμή των x (<x>)
   int m;
   double y;

   sum = 0;  n = 0;
   a.open( "fltnum.txt" );
   cout << " Δώσε αριθμό - 9999 για ΤΕΛΟΣ: ";  cin >> x;
   while ( x != sentinel )
   {
   // γράψε στο αρχείο την τιμή που πήρες
      a << x << endl;
   // πρόσθεσέ την στην sum και μέτρησέ την
      n = n + 1;
      sum = sum + x;
      cout << " Δώσε αριθμό - 9999 για ΤΕΛΟΣ: ";  cin >> x;
   } // while
   a.close();
   cout << " Διάβασα και έγραψα " << n << " αριθμούς" << endl;
   if ( n > 0 )
```

```
   {
      avrg = sum / n;
      cout << " ΑΘΡΟΙΣΜΑ = " << sum
           << "  <x> = " << avrg << endl;
      b.open( "fltnum.txt" );   m = 0;
      b >> x;
      while ( !b.eof() )
      {
         m = m + 1;
         y = exp( (avrg - x)/avrg );
         cout << " x[";
         cout.width(3);  cout << m << "] = ";
         cout.width(6);  cout << x << "   y[";
         cout.width(3);  cout << m << "] = " ;
         cout.width(6);  cout << y << endl;
         b >> x;
      } // while
      b.close();
   }
} // main
```

Στο πρόγραμμα αυτό χρησιμοποιήσαμε δύο διαφορετικά ρεύματα για να διαχειριστούμε το ίδιο αρχείο· δύο διαφορετικούς «μονόδρομους». Δεν θα μπορούσαμε να χρησιμοποιήσουμε έναν μόνο «δρόμο διπλής κατεύθυνσης»; Ναι! Αρκεί να δηλώσουμε ρεύμα κλάσης *fstream*. Τι διαφορές θα έχουμε στην περίπτωση αυτή;

- Στη δήλωση:

```
   fstream  a;
```

- Στο άνοιγμα: Η *open()* παίρνει και μια δεύτερη παράμετρο, που καθορίζει τον τρόπο διαχείρισης του αρχείου: αν δώσουμε "`ios_base::in`" ανοίγουμε το αρχείο για διάβασμα, αν δώσουμε "`ios_base::out`" το ανοίγουμε για γράψιμο, αν δώσουμε

  "`ios_base::in|ios_base::out`"

  το ανοίγουμε και για διάβασμα και για γράψιμο.[5] Θα πρέπει λοιπόν να το ανοίξουμε ως εξής:

```
   a.open( "fltnum.txt", ios_base::in|ios_base::out );
```

- Στο κλείσιμο και την επαναφορά: Θα κλείσουμε το ρεύμα μια φορά μόνον στο τέλος. Πώς όμως θα επαναφέρουμε το αρχείο στην αρχή του για να το ξαναδιαβάσουμε; Η *fstream* (και η *ifstream*) έχει μια μέθοδο, τη *seekg()*, που όταν κληθεί με όρισμα 0 – στην περίπτωσή μας: "`a.seekg(0)`"– ξαναφέρνει το ρεύμα στην αρχή του αρχείου για να (ξανα)διαβαστεί.

  Τα υπόλοιπα παραμένουν ίδια:

```
// πρόγραμμα: Μέση Τιμή 3+ inout
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;
int main()
{
   const double sentinel = 9999.0;
   fstream  a;  // για διάβασμα αρχείου και γράψιμο
   int n;       // Μετρητής των επαναλήψεων. Η τελική
                // τιμή είναι το πλήθος των στοιχείων
   double x;    // Εδώ αποθηκεύουμε κάθε τιμή που διαβάζουμε
   double sum;  // Το μερικό (τρέχον) άθροισμα.
                // Στο τέλος έχει το ολικό άθροισμα.
   double avrg; // Μέση Αριθμητική Τιμή των x (<x>)
```

[5] Μοιάζουν λιγάκι με κινέζικα; Δεν πειράζει! Θα τα χρησιμοποιείς όπως ακριβώς τα δίνουμε και αργότερα θα καταλάβεις τι ακριβώς συμβαίνει. Όπως θα μάθουμε αργότερα, αυτό μπορεί να το δεις και ως: "`ios_base::in+ios_base::out`".

```
   int m;
   double y;

   sum = 0;  n = 0;
   a.open( "fltnum.txt", ios_base::in|ios_base::out );
   cout << " Δώσε αριθμό - 9999 για ΤΕΛΟΣ: ";  cin >> x;
   while ( x != sentinel )
   {
   // γράψε στο αρχείο την τιμή που πήρες
      a << x << endl;
   // πρόσθεσέ την στην sum και μέτρησέ την
      n = n + 1;
      sum = sum + x;
      cout << " Δώσε αριθμό - 9999 για ΤΕΛΟΣ: ";  cin >> x;
   } // while
   cout << " Διάβασα και έγραψα " << n << " αριθμούς" << endl;
   if ( n > 0 )
   {
      avrg = sum / n;
      cout << " ΑΘΡΟΙΣΜΑ = " << sum
           << "  <x> = " << avrg << endl;
      a.seekg( 0 );
      a >> x;
      while ( !a.eof() )
      {
         m = m + 1;
         y = exp( (avrg - x)/avrg );
         cout << " x[";
         cout.width(3);  cout << m << "] = ";
         cout.width(6);  cout << x << "   y[";
         cout.width(3);  cout << m << "] = " ;
         cout.width(6);  cout << y << endl;
         a >> x;
      } // while
   }
   a.close();
} // main
```

8.7 Μυστικά και Ψέματα

Το πρόγραμμα της §8.5 ήταν «ψεύτικο», όπως «μισοψεύτικα» είναι και τα προγράμματα της προηγούμενης παραγράφου! Δηλαδή; δεν δουλεύουν; Δουλεύουν μια χαρά, αλλά κανείς δεν θα καθήσει να δημιουργήσει ένα αρχείο-κείμενο τροφοδοτώντας ένα πρόγραμμα σαν τα παραπάνω! Και πώς γίνεται η δουλειά;

Ας πούμε ότι έχεις κάνει κάποιες μετρήσεις και έχεις μαζέψει μερικές σελίδες με νούμερα που θέλεις να τα επεξεργαστείς με κάποιο πρόγραμμα. Τι θα κάνεις; *Θα περάσεις τα νούμερα σε ένα αρχείο-κείμενο από κάποιον κειμενογράφο!* Αλλά, προσοχή: αν δεν χρησιμοποιήσεις έναν απλόν κειμενογράφο, σαν το Notepad των Windows (ή κάτι παρόμοιο) και προτιμήσεις έναν επεξεργαστή κειμένου, σαν το Word, φρόντισε *να φυλάξεις το αρχείο σου σε μορφή text* (ή ascii ή όπως αλλιώς το λέει).

Αυτός ο τρόπος έχει τα εξής πλεονεκτήματα:

- Δεν χρειάζεται να γράψεις πρόγραμμα.
- Έχεις δυνατότητα να κάνεις διορθώσεις[6].

Δηλαδή, τζάμπα μάθαμε να δημιουργούμε αρχεία με πρόγραμμα; Όχι βέβαια! Αλλά θα δημιουργούμε αρχεία με στοιχεία που θα δημιουργούνται από το πρόγραμμά μας ή θα

[6] Αν προσπαθήσεις να κάνεις το πρόγραμμά σου με δυνατότητα διόρθωσης, γίνεται πιο πολύπλοκο και… άσε καλύτερα.

προκύπτουν από επεξεργασία άλλων αρχείων. Όχι αρχεία που θα τα γεμίζουμε από το πληκτρολόγιο.

8.8 Πάγια Ρεύματα

Εκτός από τις κλάσεις *fstream*, *ifstream*, *ofstream*, που δηλώνονται στο `fstream`, υπάρχουν: η *istream* για ρεύματα από το αρχείο προς το πρόγραμμα και η *ostream* για ρεύματα από το πρόγραμμα προς το αρχείο.

Όλες οι διάλεκτοι της C++ παρέχουν τρία ρεύματα τα οποία μπορείς να υποθέσεις ότι δηλώνονται ως εξής:

```
istream  cin;
```

Αυτό είναι το **πάγιο ρεύμα εισόδου** (standard input) και, όπως έχουμε δει, αποκαθιστά ροή από το πληκτρολόγιο προς το πρόγραμμά μας.

```
ostream  cout;
```

Πρόκειται για το **πάγιο ρεύμα εξόδου** (standard output) και, όπως έχουμε δει, αποκαθιστά ροή από το πρόγραμμα προς την οθόνη. Τέλος:

```
ostream  cerr;
```

Αυτό είναι το ρεύμα όπου γράφονται τα μηνύματα λάθους (standard error)· συνήθως αντιστοιχεί στην οθόνη. Για την ίδια δουλειά υπάρχει και ένα τέταρτο πάγιο ρεύμα, το:

```
ostream  clog;
```

Αυτό είναι το ρεύμα προς το **πάγιο αρχείο καταγραφών** (standard log file)· συνήθως αντιστοιχεί στην οθόνη.

Μπορείς να θεωρήσεις ότι αυτά τα ρεύματα ανοίγονται αυτόματα, όταν αρχίζει η εκτέλεση του (κάθε) προγράμματός σου, το πρώτο για διάβασμα και τα άλλα για γράψιμο.

8.9 Αρχείο–Κείμενο: Άλλες Επεξεργασίες

Ο τρόπος που παρουσιάσαμε το αρχείο-κείμενο (text) είναι πρωτοφανής διεθνώς: κανείς δεν ξεκινάει να δουλεύει με τέτοια αρχεία διαβάζοντας και γράφοντας αριθμούς. Τώρα θα δούμε και τις επεξεργασίες που κάνει όλος ο κόσμος.

Όπως είπαμε, μπορούμε να σκεφτόμαστε ένα αρχείο-κείμενο ως εξής::

γραμμή
γραμμή
. . .
γραμμή, τέλος αρχείου

και *γραμμή*:

χαρακτήρας, ..., χαρακτήρας, τέλος γραμμής

ή

τέλος γραμμής

Η δεύτερη περίπτωση είναι η **κενή γραμμή**.

Για να χειριστείς αρχεία-κείμενα μπορείς να χρησιμοποιήσεις, εκτός από τους τελεστές "`<<`" και "`>>`" που ήδη είδαμε, τις μεθόδους *get()*, *peek()*, των κλάσεων *ifstream* και *fstream* και *put()* των κλάσεων *ofstream* και *fstream*.

Το τέλος της γραμμής τί είναι; Κάθε κειμενογράφος έχει ένα ειδικό σημάδι για να το σημειώνει. Όπου χρησιμοποιείται το σύνολο χαρακτήρων ASCII, συνήθως, το τέλος γραμμής σημειώνεται με δυο (μη-εκτυπώσιμους) χαρακτήρες:

- CR (Carriage Return) που είναι 13ος στον πίνακα (`'\r'` για τη C++) και
- LF (Line Feed), που είναι 10ος στο πίνακα (`'\n'` για τη C++).

Με αυτόν τον τρόπο αποθηκεύονται στο κείμενο οι λειτουργίες της οθόνης: όταν πιέζουμε το <enter> για να αλλάξουμε γραμμή: πρέπει να πάμε στην αρχή της γραμμής (CR) και να κατέβουμε στην επόμενη γραμμή (LF).

Όταν διαβάζουμε ένα αρχείο ως κείμενο (text mode) η C++ κάνει τα εξής: όταν βρίσκει ζευγάρι CR LF μας δίνει στο πρόγραμμα μόνο το LF (`'\n'`). Όταν γράφουμε κάνει το αντίστροφο: κάθε φορά που γράφουμε `'\n'`, βάζει CR LF.

Μερικοί κειμενογράφοι βάζουν για τέλος γραμμής τον έναν μόνον από τους δύο χαρακτήρες ή, με την αντίστροφη σειρά, LF CR.

Ας υποθέσουμε λοιπόν ότι θέλουμε να διαβάσουμε και να επεξεργαστούμε χαρακτήρα προς χαρακτήρα, ένα αρχείο text (με τη βοήθεια του `ifstream t`). Ήδη στην §4.4 είδαμε ότι αυτή η δουλειά δεν μπορεί να γίνει με τον τελεστή "`>>`" διότι αυτός «τρώει» τα κενά, τις αλλαγές γραμμής κλπ. Όπως εκεί, έτσι και εδώ θα χρησιμοποιήσουμε τη μέθοδο *get()*. Δίνοντας:

```
t.get( ch );
```

ζητάμε από το ρεύμα *t* να μας φέρει τον επόμενο χαρακτήρα από το αρχείο και να τον αποθηκεύσει ως τιμή της μεταβλητής *ch* (τύπου `char`).

Συνηθέστατα, όταν επεξεργαζόμαστε ένα αρχείο-κείμενο πρέπει να ξεχωρίζουμε τις γραμμές· εκτός από την επεξεργασία κάθε χαρακτήρα χωριστά, κάνουμε και άλλες επεξεργασίες συνολικά στη γραμμή. Πώς ανιχνεύουμε το *τέλος γραμμής*; Σύμφωνα με αυτά που είπαμε, τέλος γραμμής έχουμε όταν: "`ch == '\n'`".

Σημείωση: ▶
Συνήθως –και για διάφορα «σημάδια» τέλους γραμμής– η C++ θα σου επιστρέψει ένα απλό `'\n'`· τουλάχιστον όταν αυτό το σημάδι περιλαμβάνει και τον `'\n'`.◀

Μπορείς να σκεφτείς την επεξεργασία αρχείου text ως εξής:

```
while ( δεν τελείωσε το αρχείο )
{
   Διάβασε μια γραμμή
   Επεξεργάσου τη γραμμή
}
```

Το γενικό σχήμα επεξεργασίας ενός αρχείου text, που βλέπεις στο Πλ. 8.2, έχει περισσότερες λεπτομέρειες. Όπως βλέπεις, έχουμε δύο **while**, τη μια φωλιασμένη μέσα στην άλλη. Η εσωτερική **while** είναι αυτό που λέμε «Διάβασε γραμμή» αλλά έχει ενσωματωμένη την επεξεργασία του κάθε χαρακτήρα που διαβάζεται.

Πλαίσιο 8.2

Επεξεργασία Αρχείου Text

```
ifstream t;
    :
t.open( όνομα αρχείου );
t.get( ch );
while ( !t.eof() )
{
// προχώρα μέχρι το τέλος της γραμμής
   while ( !t.eof() && δεν βρήκαμε τέλος γραμμής )
   {
      Εντολές επεξεργασίας του ch
      t.get( ch );
   } // while (δεν βρήκαμε τέλος γραμμής…
   Εντολές επεξεργασίας γραμμής
   if ( !t.eof() ) πήγαινε στην αρχή της επόμενης γραμμής;
   t.get( ch );
}// while (!t.eof())
t.close();
```

Η εσωτερική **while** γράφεται:

```
while ( !t.eof() && ch != '\n' )
{
   Εντολές επεξεργασίας του ch
   t.get(ch);
} // while (δεν βρήκαμε τέλος γραμμής…
```

Για το πέρασμα στη νέα γραμμή αρκεί η `t.get(ch)` που υπάρχει στο τέλος της περιοχής της εξωτερικής **while**.

Όταν τελειώσει η εκτέλεση της εσωτερικής **while** θα συμβαίνει ένα από τα παρακάτω:

- `t.eof()` (τέλος αρχείου),
- `ch == '\n'` (τέλος γραμμής).

Φυσικά, θα προσπαθήσουμε να περάσουμε σε νέα γραμμή αν δεν έχουμε την πρώτη περίπτωση. Γι' αυτό, στο Πλ. 8.2 λέμε πιο προσεκτικά:

```
   if ( !t.eof() ) πήγαινε στην αρχή της επόμενης γραμμής;
```

Σημείωση: ▶

Με την ευκαιρία να πούμε ότι: Αν είμαστε στο τέλος του αρχείου, δηλαδή η `t.eof()` δίνει `true`, η ακριβώς προηγούμενη `t.get(ch)` έχει βάλει στη *ch* τιμή `char(-1)` (ή `unsigned char(255)`).◀

Όπως είπαμε και παραπάνω, στις κλάσεις *ofstream* και *fstream* υπάρχει η μέθοδος *put()*. Αν λοιπόν γράφουμε σε κάποιο αρχείο-κείμενο μέσω του ρεύματος *s* (ας πούμε, κλάσης *ofstream*), η

```
      s.put( ch );
```

γράφει την τιμή της *ch* (τύπου `char`) στο αρχείο.

Η μέθοδος *get()*, όπως ξέρουμε, υπάρχει και στην κλάση *istream* (την έχει το ρεύμα *cin*). Η μέθοδος *put()* υπάρχει και στην κλάση *ostream* (την έχει και το ρεύμα *cout*).

Στη συνέχεια βλέπεις ένα πολύ απλό πρόγραμμα: αντιγράφει ένα αρχείο-κείμενο (το text1.txt) σε ένα άλλο (το numdta.txt).

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

int main()
{
   ifstream s( "text1.txt" );
   ofstream t( "numdta.txt" );
   char ch;

   s.get(ch);
   while ( !s.eof() )
   {  t.put(ch);  s.get(ch);  } // while
   t.close();  s.close();
} // main
```

Στην επόμενη παράγραφο θα δεις παραδείγματα με πιο πλήρη εφαρμογή του γενικού σχήματος.

8.10 Παραδείγματα

Θα δούμε τώρα δύο χαρακτηριστικά παραδείγματα επεξεργασίας αρχείου text. Και τα δύο θα επεξεργαστούν το εξής κείμενο, που είναι αντιγραμμένο από τον πρόλογο του C.A.R. Hoare στο (Hoare & Shepherdson 1985).

```
The strong connection between the formalization of
mathematical logic and the formalization of computer
programming languages was clearly recognized by Alan
Turing as early as 1947, when, in a talk on 20 February
to the London Mathematical Society, he reported his
```

```
expectation
   'that digital computing machines will eventually
   stimulate a considerable interest in symbolic logic
   and mathematical philosophy. The language in which
   one communicates with these machines, i.e. the language
   of instruction tables, forms a sort of symbolic logic'.
```

Χρησιμοποιώντας έναν κειμενογράφο, γράψε αυτό το κείμενο και φύλαξέ το σε ένα αρχείο με όνομα alturing.txt.

Παράδειγμα 1

Να γραφεί πρόγραμμα που θα διαβάζει το αρχείο που έχει το παραπάνω κείμενο και θα μετράει: πόσες γραμμές έχει, πόσα κεφαλαία γράμματα έχει και πόσα ψηφία έχει.

Το τι θα κάνουμε μας είναι γνωστό. Μας χρειάζονται τρεις μετρητές, για τα τρία μεγέθη που θα μετράμε, που αρχικά θα πρέπει να μηδενιστούν. Τα αποτελέσματά μας δεν μπορούν να βγουν πριν από το τέλος της επεξεργασίας του αρχείου. Το πρόγραμμά μας θα είναι περίπου:

```
int main()
{
   ifstream t;    // ρευμα απο το αρχειο με το κειμενο
   int  nRows;    // μετρητής γραμμών
   int  nUpCase;  // μετρητής κεφαλαίων
   int  nDigits;  // μετρητής ψηφίων
   char ch;       // χαρακτήρας που διαβάζουμε

   Μηδένισε τους μετρητές;
   Επεξεργάσου το αρχείο;
   Λέγε τα αποτελέσματα;
}
```

Η πρώτη «εντολή» υλοποιείται εύκολα:

```
// Μηδένισε τους μετρητές
   nRows = 0;  nUpCase = 0;  nDigits = 0;
```

το ίδιο και η τρίτη:

```
// Λέγε τα αποτελέσματα
   cout << " Διάβασα " << nRows << " γραμμές" << endl;
   cout << " Μέτρησα " << nUpCase << " κεφαλαία γράμματα και "
        << nDigits << " ψηφία" << endl;
```

Ας δούμε τώρα τι γίνεται με την «Επεξεργάσου το αρχείο». Αυτή θα ακολουθεί το γενικό σχήμα επεξεργασίας αρχείου text:

```
// Επεξεργάσου το αρχειο
   t.open( "alturing.txt" );
   t.get( ch );
   while ( !t.eof() )
   {
   // προχώρα μέχρι το τέλος της γραμμής
      while ( !t.eof() && ch == '\n' )
      {
         Εντολές επεξεργασίας του ch
         t.get( ch );
      } // while (δεν βρήκαμε τέλος γραμμής…
      Εντολές επεξεργασίας γραμμής
      if ( !t.eof() ) t.get( ch );
   }// while (!t.eof())
   t.close();
```

Το πρόβλημά μας τώρα είναι να δούμε τι θα είναι οι «*Εντολές επεξεργασίας του ch*» και οι «*Εντολές επεξεργασίας γραμμής*».

Οι «*Εντολές επεξεργασίας του ch*» τι θα κάνουν; Θα ελέγχουν τον κάθε χαρακτήρα που διαβάζεται και εφόσον είναι κεφαλαίο γράμμα ή ψηφίο θα αυξάνεται ο αντίστοιχος μετρητής. Μπορούμε να την υλοποιήσουμε με την:

```
         if ( isupper(ch) ) nUpCase = nUpCase + 1;
```

```
         else if ( isdigit(ch) )  nDigits = nDigits + 1;
```

Τις *isupper* και *isdigit* τις θυμάσαι; Αν όχι, γύρισε πίσω, στον Πίν. 4-3. Για να τις χρησιμοποιήσουμε θα πρέπει να βάλουμε `#include <cctype>`.

Τέλος, ας δούμε και τις «*Εντολές επεξεργασίας γραμμής*»: Κάθε φορά που τελειώνει μια γραμμή, θα πρέπει να αυξάνουμε το μετρητή γραμμών κατά 1:

```
      nRows = nRows + 1;
```

Ολόκληρο το πρόγραμμα:

```
#include <iostream>
#include <fstream>
#include <cctype>

using namespace std;

int main()
{
   ifstream  t;  // ρεύμα απο το αρχείο με το κείμενο
   int  nRows;   // μετρητής γραμμών
   int  nUpCase; // μετρητής κεφαλαίων
   int  nDigits; // μετρητής ψηφίων
   char ch;      // χαρακτήρας που διαβάζουμε
// Μηδένισε τους μετρητές
   nRows = 0;  nUpCase = 0;  nDigits = 0;
// Επεξεργάσου το αρχείο
   t.open( "alturing.txt" );
   t.get( ch );
   while ( !t.eof() )
   {
      while ( !t.eof() && ch != '\n' )
      {
         if ( isupper(ch) ) nUpCase = nUpCase + 1;
         else if ( isdigit(ch) )  nDigits = nDigits + 1;
         t.get(ch);
      }
      nRows = nRows + 1;
      if ( !t.eof() )  t.get(ch);
   } // while (!t.eof())
   t.close();
// Λέγε τα αποτελέσματα
   cout << " Διάβασα " << nRows << " γραμμές" << endl;
   cout << " Μέτρησα " << nUpCase << " κεφαλαία γράμματα και "
        << nDigits << " ψηφία" << endl;
} // main
```

Το αποτέλεσμα που μας δίνει είναι:

```
 Διάβασα 11 γραμμές
 Μέτρησα 8 κεφαλαία γράμματα και 6 ψηφία
```

Παράδειγμα 2

Θέλουμε να επεξεργαστούμε ξανά το κείμενο που μετρήσαμε στο προηγούμενο παράδειγμα, αλλά με διαφορετικό στόχο: Θέλουμε να πάρουμε το ίδιο κείμενο γραμμένο με κεφαλαία.

Κατ' αρχάς θα πρέπει να δηλώσουμε τα δύο ρεύματα με τα οποία θα χειριστούμε τα αρχεία:

```
   ifstream  a;    // το αρχείο με το κείμενο
   ofstream  b;    // το αρχείο με τα κεφαλαία
```

και να τα ανοίξουμε:

```
   a.open( "alturing.txt" );
   b.open( "upcaltur.txt" );
```

Την άσκ. 7-8 την έλυσες; Ζητούσαμε να γράψεις μια:

```
   // upCase -- Αν ο ch είναι μικρό Λατινικό γράμμα επιστρέφει
   //           το αντίστοιχο κεφαλαίο, αλλιώς τον ch
   char upCase( char ch )
```

Αν την έλυσες, τότε τα πράγματα είναι πολύ απλά:

- Οι «*Εντολές επεξεργασίας του ch*» θα μετατρέπουν το χαρακτήρα σε κεφαλαίο και θα τον γράφουν στο νέο αρχείο, δηλαδή:

  `co = upCase(ch);  b.put(co);`

- Οι «*Εντολές επεξεργασίας γραμμής*» δεν είναι τίποτε ιδιαίτερο: απλώς μετά το τέλος της γραμμής θα πρέπει να βάζουμε ένα `endl`.

  Να ολόκληρο το πρόγραμμα:

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

   char upCase( char ch );

int main()
{
   ifstream  a;  // το αρχείο με το κείμενο
   ofstream  b;  // το αρχείο με τα κεφαλαία
   char ch, co;

   a.open( "alturing.txt" );
   b.open( "upcaltur.txt" );
   a.get( ch );
   while ( !a.eof() )
   {
      while ( !a.eof() && (ch != '\n') )
      {  co = upCase(ch);  b.put(co);  a.get(ch);  }
      b << endl;
      if ( !a.eof() )  a.get( ch );
   }
   b.close();
   a.close();
} // main

   // upCase -- Αν ο ch είναι μικρό Λατινικό γράμμα επιστρέφει
   //           το αντίστοιχο κεφαλαίο, αλλιώς τον ch
   char upCase( char ch )
   {
      char fv;

      if ( islower(ch) )  fv = char( int(ch)-32 );
                    else  fv = ch;
      return fv;
   } // upCase
```

Παρατήρηση: ▶

Φυσικά, θα μπορούσαμε πάρουμε τη λύση πιο απλά:

- Θυμήσου ότι η C++ σου δίνει την *toupper* που κάνει αυτά που ζητούμε από την *upCase*.
- Πάρε το πρόγραμμα αντιγραφής της προηγούμενης παραγράφου και άλλαξε την:

```
{  t.put(ch);  s.get(ch);  }
```

σε:

```
{  b.put(toupper(ch));  a.get(ch);  }
```

Δηλαδή:

```
   a.open( "alturing.txt" );
   b.open( "upcaltur.txt" );
   a.get( ch );
   while ( !a.eof() )
   {  b.put( toupper(ch) );  a.get( ch );  }
```

```
   b.close();  a.close(); ◀
```

🖘🖘🖘

8.11 Δουλεύοντας με Σιγουριά

Όταν κάνεις μια πράξη εισόδου/εξόδου (I/O) πολλά άσχημα μπορεί να συμβούν. Π.χ.

- να προσπαθήσεις να ανοίξεις για διάβασμα ένα αρχείο που δεν υπάρχει,
- να προσπαθήσεις να γράψεις σε μια δισκέτα που δεν έχει χώρο κ.ο.κ.

Η C++ σου δίνει τη δυνατότητα να ελέγχεις αν όλα πηγαίνουν καλά. Η μέθοδος *fail()* επιστρέφει `true` αν η προηγούμενη πράξη εισόδου/εξόδου απέτυχε αλλιώς `false`. Ας πούμε ότι έχεις δηλώσει:

```
   ifstream s;
```

και στη συνέχεια ζητάς:[7]

```
   s.open( "text1.txt" );
   if ( s.fail() )
      cerr << "δεν μπορώ να ανοίξω το αρχείο text1.txt" << endl;
   else // ok, προχώρα
   { // . . .
```

Τον ίδιο έλεγχο μπορείς να κάνεις και μετά την πράξη "`s.get(ch)`" ή την "`s >> x`".

Στη συνέχεια βλέπεις το πρόγραμμα της αντιγραφής αρχείου με πολλούς ελέγχους:

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

int main()
{
   ifstream s;
   ofstream t;
   char ch;

   s.open( "text1.txt" );
   if ( s.fail() )
      cerr << "δεν μπορώ να ανοίξω το αρχείο text1.txt" << endl;
   else // ok, προχώρα
   {
      t.open( "numdta.txt" );
      if ( t.fail() )
      {
         s.close();
         cerr << "δεν μπορώ να δημιουργήσω το numdta.txt" << endl;
      }
      else // ok και τα δύο αρχεία ανοικτά
      {
         s.get(ch);
         while ( !s.fail() && !t.fail() )
         {  t.put( ch );  s.get(ch);  } // while
         if ( !s.eof() )
            cerr << "πρόβλημα κατά την αντιγραφή" << endl;
         t.close();
         s.close();
      } // if ( t.fail
   } // if ( s.fail
} // main
```

[7] Πάντως, σε πολλά βιβλία, θα δεις και έναν άλλον τρόπο: Αν δεις την *s* ως *συνθήκη*: αν το άνοιγμα του αρχείου έγινε επιτυχώς θα έχει τιμή που ερμηνεύεται ως `true`, αλλιώς θα έχει τιμή που ερμηνεύεται ως `false`. Έτσι, αντί για `if (s.fail())...` θα δεις το `if (!s)...`

Εξηγούμε και με λόγια το τι γίνεται:

- Αν δεν μπορέσαμε να ανοίξουμε το ρεύμα *s* από το αρχείο που διαβάζουμε δεν προχωρούμε.
- Αν τα καταφέραμε, ελέγχουμε αν ανοίξαμε το ρεύμα προς το αρχείο *t* που γράφουμε. Αν δεν άνοιξε, *κλείνουμε και το s* και σταματάμε.
- Αν όλα πήγαν καλά αντιγράφουμε με τη **while**. Πρόσεξε τη συνθήκη της που ελέγχει τα δύο ρεύματα. Η εκτέλεση των επαναλήψεων διακόπτεται αν εμφανιστεί πρόβλημα σε κάποιο από τα δύο.
- Αν τελειώσει το αρχείο η `s.fail()` θα δώσει `true` και οι επαναλήψεις θα σταματήσουν.
- Η **while** τελειώνει αν βρούμε πρόβλημα στο *s* ή στο *t*. Αν όμως το «πρόβλημα» είναι η *s.eof*`()` τότε όλα πήγαν καλά! Το μήνυμα "*πρόβλημα κατά την αντιγραφή*" εμφανίζεται μόνον αν βγήκαμε από τη **while** χωρίς να έχουμε *s.eof*`()`.

Ένας άλλος τρόπος για να δούμε αν το ρεύμα είναι ανοικτό είναι η κλήση της μεθόδου *is_open()*. Αυτή επιστρέφει τιμή `true` αν το ρεύμα είναι ανοικτό, αλλιώς `false`:

```
// . . .
   s.open( "text1.dta" );
   if ( !s.is_open() )
      cerr << "δεν μπορώ να ανοίξω το αρχείο text1.dta" << endl;
   else
   {
      t.open( "numdta.txt" );
      if ( !t.is_open() )
      {
         s.close();
         cerr << "δεν μπορώ να δημιουργήσω το numdta.txt" << endl;
      }
      else // ok και τα δύο αρχεία ανοικτά
// . . .
```

8.12 Τρόποι Ανοίγματος (Ρεύματος) Αρχείου

Είπαμε παραπάνω ότι η *open()* δέχεται και μια δεύτερη παράμετρο που έχει να κάνει με τον τρόπο χρήσης του αρχείου και είδαμε ήδη τις εξής περιπτώσεις:

- "`ios_base::in`" για διάβασμα από αρχείο (δεν χρειάζεται αν το ρεύμα είναι *ifstream*).
- "`ios_base::out`" για γράψιμο σε αρχείο (δεν χρειάζεται αν το ρεύμα είναι *ofstream*).
- Ο συνδυασμός "`ios_base::in | ios_base::out`" για αρχείο που το ανοίγουμε για διάβασμα και για γράψιμο (δεν χρειάζεται αν το ρεύμα είναι *ofstream*).

Παρ' όλο που εδώ δεν έχουμε μεταβλητές τύπου `bool`, οι "`ios_base::in`" και "`ios_base::out`" ονομάζονται **σημαίες** (flags) της *open()*. Το γιατί θα το καταλάβεις αργότερα. Θα τις δεις ακόμη και ως *τιμές τύπου* (*ios_base*`::`)*openmode*.

Εκτός από αυτούς μπορείς να χρησιμοποιήσεις τα εξής (και συνδυασμούς τους):""

"`ios_base::binary`"

για αρχεία που έχουν το περιεχόμενό τους όχι σε μορφή κειμένου αλλά όπως είναι στην εσωτερική παράσταση. Θα τα δούμε στη συνέχεια.

"`ios_base::trunc`"

για να σβήσεις το περιεχόμενο του αρχείου με το άνοιγμά του (όπως γίνεται με το άνοιγμα ενός *ofstream*).

Ας πούμε ότι έχουμε ένα αρχείο –το `temp.txt`– και θέλουμε να το ανοίξουμε για γράψιμο και διάβασμα. Πριν από όλα όμως θέλουμε να σβήσουμε το παλιό του περιεχόμενο. Πώς μπορούμε να το κάνουμε;

- Με όσα ξέρουμε μέχρι τώρα: Το ανοίγουμε ως

```
   fstream temp( "temp.txt", ios_base::out );
```

```
temp.close();
```

για να σβήσουμε το αρχικό του περιεχόμενο και μετά το ανοίγουμε πάλι ως:

```
fstream temp( "temp.txt", ios_base::in|ios_base::out );
```

- Με χρήση της "`ios_base::trunc`": Το ανοίγουμε, μια φορά μόνον, ως

```
fstream temp( "temp.txt", ios_base::in|ios_base::out|ios_base::trunc );
```

"`ios_base::ate`"
με το άνοιγμα του αρχείου τοποθετείται στο τέλος του.
"`ios_base::app`"
το αρχείο δεν σβήνεται και κάθε γράψιμο γίνεται στο τέλος του.

Στη συνέχεια δίνουμε ένα παράδειγμα για να συγκρίνεις τις δύο τελευταίες. Χρησιμοποιούμε την εντολή "`temp.seekp(0)`" που σημαίνει «πήγαινε *να γράψεις* στην αρχή του αρχείου». Η μέθοδος *seekp()* είναι διαθέσιμη για ρεύματα *ofstream* και *fstream*.

Παράδειγμα

Έστω ότι έχουμε το αρχείο `temp.txt` με περιεχόμενο:

```
Ιανουάριος
Φεβρουάριος
Μάρτιος
Απρίλιος
```

Μετά το άνοιγμα:

```
fstream temp( "temp.txt",
              ios_base::in|ios_base::out|ios_base::ate );
```

και τις εντολές:

```
temp << "Μάιος" << endl;
temp.seekp( 0 );
temp << "Ιούνιος" << endl;
temp.close();
```

το περιεχόμενο του αρχείου γίνεται:

```
Ιούνιος
ς
Φεβρουάριος
Μάρτιος
Απρίλιος
Μάιος
```

Δηλαδή:

- Με το άνοιγμα πηγαίνουμε στο τέλος του αρχείου και γράφουμε "`Μάιος`".
- Μετά την `temp.seekp(0)`, το γράψιμο της λέξης "`Ιούνιος`" γίνεται στην αρχή του αρχείου.

Αν ανοίξουμε το *temp* με την:

```
fstream temp( "temp.txt", ios_base::out|ios_base::app );
```

οι ίδιες εντολές θα κάνουν το αρχείο:

```
Ιανουάριος
Φεβρουάριος
Μάρτιος
Απρίλιος
Μάιος
Ιούνιος
```

Δηλαδή, παρά την `temp.seekp(0)`, όταν έρχεται η στιγμή να γραφεί η λέξη "`Ιούνιος`" το γράψιμο γίνεται στο τέλος του αρχείου.

Σχετικώς με τη χρήση συνδυασμών από τέτοιες σημαίες να επισημάνουμε τα εξής:

- Όλοι οι επιτρεπτοί συνδυασμοί φαίνοντα στον Πιν. 8-1.

- Μπορείς να τους χρησιμοποιήσεις σε ρεύματα *fstream*, *ifstream*, *ofstream*. Αν κατά το άνοιγμα δεν βάλεις δεύτερη παράμετρο
  - Το *fstream* ανοίγει με `ios_base::in | ios_base::out`.
  - Το *ifstream* ανοίγει με `ios_base::in`.
  - Το *ofstream* ανοίγει με `ios_base::out`.
- Η ύπαρξη της `ios_base::out` στον συνδυασμό έχει ως αποτέλεσμα να σβηστεί το αρχείο, αν υπάρχει. Αυτό *αναιρείται* αν ο συνδυασμός περιέχει και κάποιαν από τις `ios_base::in`, `ios_base::app` ενώ *επιβάλλεται* αν περιέχει την `ios_base::trunc`.

8.13 Χειρισμός Αρχείων με τα Εργαλεία της C

Παρ' όλο που η C++ διαθέτει ένα πολύ πλούσιο και ευέλικτο σύστημα για τη διαχείριση αρχείων, θα δεις συχνά προγράμματα που χρησιμοποιούν τα αντίστοιχα εργαλεία της C. Θεωρούμε λοιπόν αναγκαίο να ρίξουμε μια ματιά στα εργαλειά αυτά αν και δεν θα τα χρησιμοποιούμε στη συνέχεια.

Στη συνέχεια βλέπεις (ξανά) το πρόγραμμα «*Μέση Τιμή 6*» με διαχείριση αρχείων όπως την κάνει η C:

```
0: // πρόγραμμα: Μέση Τιμή 6 C
1: #include <cstdio>
2: using namespace std;
3: int main()
4: {
```

Συνδυασμός τιμών `ios_base`					Ισοδύναμος ορμαθός `cstdio`
`binary`	`in`	`out`	`trunk`	`app`	x
		+			"w"
		+		+	"a"
				+	"a"
		+	+		"w"
	+				"r"
	+	+			"r+"
	+	+	+		"w+"
	+	+		+	"a+"
	+			+	"a+"
+		+			"wb"
+		+		+	"ab"
+				+	"ab"
+		+	+		"wb"
+	+				"rb"
+	+	+			"r+b"
+	+	+	+		"w+b"
+	+	+		+	"a+b"
+	+			+	"a+b"

Πίν. 8-1 (Από το πρότυπο της C++) Αντίστοιχοι τρόποι ανοίγματος αρχείων στη C++ και στη C.
Ένα παράδειγμα για το διάβασμά του: Η τελευταία γραμμή σημαίνει: αν στην *open* της C++ βάλεις
`ios_base::binary | ios_base::in | ios_base::app`
στη *fopen()* της C θα βάλεις `"a+b"`.

```
 5:    FILE* a;
 6:    FILE* b;
 7:
 8:    int n;         // Μετρητής όλων των στοιχείων
 9:    int selN;      // Μετρητής επιλεγόμενων στοιχείων
10:    double x;      // Εδώ αποθηκεύουμε κάθε τιμή που διαβάζουμε
11:    double sum;    // Το άθροισμα όλων των στοιχείων
12:    double selSum; // Άθροισμα επιλεγόμενων στοιχείων
13:    double avrg;   // Μέση Αριθμητική Τιμή όλων των στοιχείων
14:    double selAvrg;
15:                   // Μέση Αριθμητική Τιμή επιλεγόμενων στοιχείων
16:    sum = 0;  n = 0;
17:    selSum = 0;  selN = 0;
18:    a = fopen( "exp4.dta", "r" );
19:    fscanf( a, "%lf", &x );
20:    while ( !feof(a) )
21:    {
22:       n = n + 1;              // Αυτά γίνονται για
23:       sum = sum + x;          // όλους τους αριθμούς
24:       if ( 0 < x && x <= 10 ) // 'Ελεγχος - Επιλογή
25:       {
26:          selSum = selSum + x;  // Αυτά γίνονται μόνο για
27:          selN = selN + 1;      // τους επιλεγόμενους αριθμούς
28:       } // if
29:       fscanf( a, "%lf", &x );
30:    } // while
31:    fclose( a );
32:    b = fopen( "report.txt", "w" );
33:    fprintf( b, " Διάβασα %d αριθμούς\n", n );
34:    if ( n > 0 )
35:    {
36:       avrg = sum / n;
37:       fprintf( b, " ΑΘΡΟΙΣΜΑ = %lf   <x> = %lf\n",
38:                   sum, avrg );
39:    }
40:    fprintf( b, " Διάλεξα %d αριθμούς ε (0,10]\n", selN );
41:    if ( selN > 0 )
42:    {
43:       selAvrg = selSum/selN;
44:       fprintf( b, " ΑΘΡΟΙΣΜΑ = %lf  <x> = %lf\n",
45:                   selSum, selAvrg );
46:    }
47:    fclose( b );
48: } // main
```

Ας δούμε τα κρίσιμα σημεία:

- Στη γρ. 1 έχουμε "`#include <cstdio>`" αντί της γνωστής μας "`#include <fstream>`". Θυμίσου ότι το ίδιο είχαμε κάνει και όταν χρησιμοποιούσαμε τις `printf` (§1.12) και `scanf` (§2.12).
- Στις γρ. 5-6 βλέπεις τις δηλώσεις:

```
FILE* a;
FILE* b;
```

Με αυτά τα δύο βέλη θα χειριστούμε τα δύο ρεύματα. Αργότερα θα πάρουμε μια ιδέα για το τι μπορεί να είναι ο τύπος "`FILE`".

- Στη γρ. 18 ανοίγουμε το πρώτο ρεύμα με την "`a = fopen( "exp4.dta", "r" )`" που δίνει τιμή στο βέλος *a*. Η *fopen()* είναι μια συνάρτηση που παίρνει δύο ορίσματα:
  - το όνομα του αρχείου (στην περίπτωσή μας `"exp4.dta"`) και
  - τον τρόπο ανοίγματος (στην περίπτωσή μας `"r"`).

  ανοίγει το ρεύμα και επιστρέφει τιμή τύπου *FILE*`*`. Αν αποτύχει το άνοιγμα επιστρέφει 0 (`NULL`). Στον πίνακα 8.1 μπορείς να δεις πώς γράφεται ένας τρόπος ανοίγματος της C

αντίστοιχος τρόπου ανοίγματος της C++. Στην περίπτωσή μας (`"r"`) έχουμε το ισοδύναμο του "`ios_base::in`".

- Στη γρ. 19 κάνουμε το πρώτο διάβασμα: "`fscanf( a, "%lf", &x )`". Αυτή είναι ίδια με τη `scanf` (§2.12) αλλά έχει ένα επι πλέον όρισμα, στην αρχή, που καθορίζει το ρεύμα (στη `scanf` το ρεύμα είναι το `stdin`). Στη γρ. 29 διαβάζουμε ξανά με τον ίδιο ακριβώς τρόπο.
- Στη γρ. 20, στη συνθήκη της **while**, ελέγχουμε για τέλος αρχείου καλώντας την *feof*. Όπως βλέπεις, της δίνουμε ως όρισμα το βέλος *a*, που καθορίζει το ρεύμα.
- Στη γρ. 31 κλείνουμε τπ ρεύμα με την "`fclose( a )`".
- Στη γρ. 32 ανοίγουμε ρεύμα για να γράψουμε το αρχείο `report.txt` με την "`b = fopen( "report.txt", "w" )`". Ο τρόπος χρήσης του ρεύματος (`"w"`) είναι ισοδύναμος του "`ios_base::out`".
- Στις γρ. 33, 37, 40, 44 γράφουμε στο αρχείου χρησιμοποιώντας την `fprintf`. Αυτή είναι σαν την `printf` αλλά έχει ένα επί πλέον όρισμα στην αρχή που καθορίζει το ρεύμα.
- Θα μπορούσαμε να είχαμε ανοίξει τα ρεύματα με τις δηλώσεις των *a*, *b*; Βεβαίως, έτσι:

```
FILE* a( fopen("exp4.dta", "r") );
FILE* b( fopen("report.txt", "w") );
```

Στη συνέχεια (ξανα)δίνουμε και το πρόγραμμα αντιγραφής αρχείου με χρήση των εργαλείων της C:

```
 0: #include <cstdio>
 1: using namespace std;
 2:
 3: int main()
 4: {
 5:    FILE* s;
 6:    FILE* t;
 7:    int ch, res( 0 );
 8:
 9:    s = fopen( "text1.dta", "r" );
10:    if ( !s )
11:       fprintf( stderr,
12:                "δεν μπορώ να ανοίξω το αρχείο text1.dta\n" );
13:    else
14:    {
15:       t = fopen( "numdta.txt", "w" );
16:       if ( !t )
17:       {
18:          fclose( s );
19:          fprintf( stderr,
20:                   "δεν μπορώ να δημιουργήσω το numdta.txt\n" );
21:       }
22:       else // ok και τα δύο αρχεία ανοικτά
23:       {
24:          ch = getc( s );
25:          while ( ch != EOF && res != EOF )
26:          {  res = putc( ch, t );  ch = getc( s );  } // while
27:          if ( !feof(s) )
28:             fprintf( stderr,
29:                      "πρόβλημα κατά την αντιγραφή\n" );
30:          fclose( t );
31:          fclose( s );
32:       } // if ( t.fail
33:    } // if ( s.fail
34: } // main
```

Εδώ, τα καινούρια πράγματα είναι στις γρ. 24, 25, 26:

- Η συνάρτηση *getc()* παίρνει ένα όρισμα –το βέλος που καθορίζει το ρεύμα (που διαβάζουμε)– και μας επιστρέφει τον επόμενο χαρακτήρα από το αρχείο αλλά σε τύπο `int`. Αν υπάρξει πρόβλημα επιστρέφει `EOF` (-1).

Μέθοδος / Τελεστής	Κλάση	Τι κάνει, πού την είδαμε
`>>`	`ifstream`	Φέρνει τιμή από το αρχείο σε μεταβλητή. §8.3, 2.3, 4.5, 4.6
`<<`	`ofstream`	Γράφει τιμές στο αρχείο. §8.4, 1.2, 1.11, 4.4.1
`clear`	`ifstream, ofstream`	Ανατάσσει το ρεύμα μετά από *eof* ή άλλη αποτυχία κάποιας λειτουργίας. §8.3.2
`close`	`ifstream, ofstream`	Αποσυνδέει ένα ρεύμα από το αρχείο. §8.3, 8.4
`endl`	`ofstream`	Γράφει τέλος γραμμής και αντιγράφει τον ενταμιευτή στο αρχείο. §8.4
`eof`	`ifstream`	Μας πληροφορεί αν φτάσαμε στο τέλος του αρχείου. §8.3, 8.3.1, 8.11
`fail`	`ifstream, ofstream`	Επιστρέφει `true` αν η προηγούμενη λειτουργία στο ρεύμα απέτυχε. §8.11
`get`	`ifstream`	Φέρνει ένα χαρακτήρα από το αρχείο στο όρισμά της. §8.9, 8.12, 4.5
`open`	`ifstream, ofstream`	Συνδέει το ρεύμα με το αρχείο. §8.1, 8.3, 8.4
`is_open`	`ifstream, ofstream`	Επιστρέφει τιμή `true` αν το ρεύμα είναι ανοικτό, αλλιώς `false`, §8.11.
`peek`	`ifstream`	Δίνει το χαρακτήρα που θα διαβάσει η επόμενη *get()*. §8.9
`put`	`ofstream`	Γράφει το όρισμά της στο αρχείο. §8.9
`seekg(0)`	`ifstream`	Το ρεύμα ετοιμάζεται να διαβάσει από την αρχή του αρχείου. §8.6
`seekp(0)`	`ofstream`	Το ρεύμα ετοιμάζεται να γράψει στην αρχή του αρχείου. §8.12

Πίν. 8-2 Οι μέθοδοι για τη διαχείριση αρχείων με ρεύματα που μάθαμε. Όλες οι μέθοδοι υπάρχουν και στην κλάση `fstream`. Οι μέθοδοι της `ifstream` υπάρχουν και στην `istream` (τύπος του `cin`). Οι μέθοδοι της `ofstream` υπάρχουν και στην `ostream` (τύπος των `cout`, `cerr`, `clog`).

- Η *putc()* παίρνει δύο ορίσματα: τον χαρακτήρα που θέλουμε να γράψουμε στο αρχείο και το βέλος για το ρεύμα. Αν γράψει τον χαρακτήρα τον επιστρέφει και ως τιμή. Αν αποτύχει επιστρέφει `EOF`. Όπως βλέπεις, στη *res* κρατούμε την τιμή που επιστρέφει κάθε φορά η *putc()*. Όσο δεν παίρνουμε `EOF` ούτε στη *res* ούτε στη *ch* ("`while (ch != EOF && res != EOF)`") συνεχίζουμε τη διαδικασία αντιγραφής.

8.14 Σύνοψη

Στο κεφάλαιο αυτό μάθαμε πώς να χειριζόμαστε αρχεία-κείμενα (text) είτε έχουν αριθμητικά δεδομένα είτε τα θεωρούμε ως αρχεία χαρακτήρων. Τα εργαλεία μας είναι ρεύματα των κλάσεων *ifstream*, *ofstream* και *fstream* και οι μέθοδοί τους που βλέπεις στον Πίν. 8-2.

Θα επαναλάβουμε ακόμη και μερικές βασικές αρχές που μάθαμε στο κεφάλαιο αυτό.

- ***Για να διαβάσουμε (ή να γράψουμε) το ν-οστό στοιχείο ενός σειριακού αρχείου πρέπει να περάσουμε κατ' ανάγκη τις ν −1 προηγούμενες τιμές.***
- ***Η επεξεργασία κάθε σειριακού αρχείου ξεκινάει πάντα από την αρχή του.***
- ***Πριν επεξεργαστούμε μια τιμή που (υποτίθεται ότι) διαβάσαμε από ένα αρχείο ελέγχουμε την eof() ή τη fail() για να βεβαιωθούμε ότι η ανάγνωση έγινε κανονικώς.***

Λύσε οπωσδήποτε τις Ασκ. 8-5, 8-6 (και αργότερα την 9-13) και δες προσεκτικά τις λύσεις που προτείνουμε. Αναφέρονται στην *ενημέρωση* (μεταβολή, εισαγωγή, διαγραφή στοιχείων) σειριακού αρχείου με βάση την αρχή:

- ***Για να ενημερώσουμε ένα σειριακό αρχείο το αντιγράφουμε σε ένα άλλο και κατά την αντιγραφή κάνουμε τις αλλαγές που θέλουμε.***

Ασκήσεις

Α Ομάδα

8-1 Γράψε ένα πρόγραμμα που θα αντιγράφει ένα αρχείο text στην οθόνη. Αριστερά από κάθε γραμμή θα γράφεται ο αριθμός της, ξεκινώντας από 1.

8-2 Σε ένα αρχείο-κείμενο, με όνομα στο δίσκο real.txt, έχουμε πραγματικούς αριθμούς. Γράψε ένα πρόγραμμα που θα μετράει:

- πόσοι αριθμοί μεταξύ 4.5 και 5.5 υπάρχουν στο αρχείο,
- πόσες φορές υπάρχει η τιμή 5.0

Ακόμα, θα δίνει τα ποσοστά των παραπάνω αριθμών σε σχέση με τον συνολικό αριθμό τιμών του αρχείου.

8-3 Σε δύο αρχεία text, με ονόματα 'mydata.txt', 'moredata.txt' και το ίδιο πλήθος γραμμών, υπάρχουν αριθμοί, ένας σε κάθε γραμμή. Γράψε πρόγραμμα που θα τα διαβάζει και θα δημιουργεί ένα νέο αρχείο text, με όνομα "comb.txt", που θα έχει το ίδιο πλήθος γραμμών με κάθε ένα από τα αρχικά και στη ν-οστή γραμμή του θα έχει τέσσερις αριθμούς:

- αυτόν που υπάρχει στην ν-οστή γραμμή του πρώτου αρχείου,
- αυτόν που υπάρχει στην ν-οστή γραμμή του δεύτερου αρχείου,
- το άθροισμα των δύο προηγούμενων,
- το γινόμενο των δύο προηγούμενων.

8-4 Σε ένα αρχείο text, με όνομα protmet.txt, υπάρχουν πρότυπες μετρήσεις από τον έλεγχο μιας συσκευής –ένας αριθμός σε κάθε γραμμή. Το αρχείο testmet.txt είναι ίδιο και έχει τον ίδιο αριθμό γραμμών με το πρώτο αλλά όχι και τους ίδιους αριθμούς· περιέχει δοκιμαστικές μετρήσεις που κάναμε στη συσκευή για να δούμε αν είναι εντάξει. Έστω α ο αριθμός που υπάρχει στην k γραμμή του πρώτου αρχείου και β αυτός που υπάρχει στην k γραμμή του δεύτερου αρχείου. Η συσκευή μας είναι εντάξει αν

για όλα τα k έχουμε: $0.95\alpha \le \beta \le 1.05\alpha$

δηλαδή, αν υπάρχουν αποκλίσεις που δεν υπερβαίνουν το 5%.

Γράψε πρόγραμμα που θα διαβάζει τα δυο αρχεία και

- για κάθε γραμμή (k) που βρίσκει απόκλιση μεγαλύτερη από 5% θα μας τυπώνει το k, το α και το β,
- αν δεν βρει απόκλιση μεγαλύτερη από 5% θα μας τυπώνει στο τέλος το μήνυμα "ΣΥΣΚΕΥΗ ΕΝΤΑΞΕΙ".

Β Ομάδα

8-5 (Ενημέρωση – μεταβολή στοχείων) Σε ένα αρχείο-κείμενο, με όνομα στο δίσκο idid.txt, έχουμε σε κάθε γραμμή δύο πραγματικούς αριθμούς που παριστάνουν μια τάση (ο πρώτος) και ένα ρεύμα (ο δεύτερος), όπως μετρήθηκαν στο εργαστήριο. Αλλά οι τιμές του ρεύματος διαβάστηκαν λαθεμένα στο αμπερόμετρο. Στις πρώτες πέντε (5) γραμμές η τιμή που υπάρχει στο αρχείο είναι 1000πλάσια της πραγματικής και στις επόμενες 11 είναι 10πλάσια της πραγματικής. Οι υπόλοιπες γραμμές είναι σωστές.

Γράψε λοιπόν μια συνάρτηση που θα παίρνει τη θέση της γραμμής (k) και το ρεύμα (i) και θα μας δίνει το σωστό ρεύμα:

```
double correctI( int k, double i )
```

Θέλουμε να αλλάξουμε κάθε τιμή ρεύματος (*i*) του αρχείου σε *correctI*(*k*, *i*), όπου *k* (1η, 2η, 3η κλπ) η γραμμή του αρχείου όπου βρίσκεται η *i*. Αυτό θα το κάνεις με ένα πρόγραμμα που θα αντιγράφει μια προς μια τις γραμμές του idid.txt σε ένα άλλο, με το όνομα midid.txt, αλλά, κάθε φορά θα αλλάζει την τιμή ρεύματος (*i*) σε *correctI*(*k*, *i*) με τη συνάρτηση που έγραψες όπως είπαμε παραπάνω.

8-6 (Ενημέρωση – διαγραφή στοιχείων) Έστω τώρα ότι θέλουμε να σβήσουμε από το αρχείο idid.όλες τις γραμμές με λαθεμένα στοιχεία, δηλαδή τις πρώτες δεκαέξη (16) γραμμές.

Και πάλι, θα πρέπει να γράψεις ένα πρόγραμμα που να δημιουργεί ένα νέο αρχείο με όνομα dідid.txt, όπου θα αντιγράψει μόνον τις γραμμές με σωστά στοιχεία (από τη 17η και μετά). Φυσικά, τα στοιχεία που θα διαγράψεις δεν θα πάνε στα σκουπίδια: θα αντιγραφούν σε ένα άλλο αρχείο με όνομα xidid.txt.

8-7 Ένα αρχείο text, με όνομα στο δίσκο misthoi.txt, είναι γραμμένο ως εξής: Σε κάθε γραμμή έχει τρεις αριθμούς. Ο πρώτος είναι ο βασικός μισθός ενός υπαλλήλου, ο δεύτερος ο αριθμός χρόνων υπηρεσίας του ίδιου υπαλλήλου και ο τρίτος ο αριθμός των παιδιών του. Το πλήθος των στοιχείων είναι άγνωστο. Το αρχείο θα χρησιμοποιηθεί για τη μισθοδοσία των υπαλλήλων ενός οργανισμού.

Το περιεχόμενο του αρχείου δεν έχει ελεγχθεί. Θα γράψεις ένα πρόγραμμα που θα ελέγχει τα εξής:

- αν 800 ≤ βασικός μισθός ≤ 3500,
- αν 0 ≤ χρόνια υπηρεσίας ≤ 35,
- αν 0 ≤ αριθμός παιδιών ≤ 12.

Για κάθε λάθος που θα βρίσκει, θα πρέπει να τυπώνει τον αριθμό του υπαλλήλου, δηλαδή τη σειρά που έχουν τα στοιχεία του στο αρχείο και τη λαθεμένη τιμή. Π.χ.:

```
 7ΟΣ ΥΠΑΛΛΗΛΟΣ. ΜΙΣΘΟΣ: 35650
 7ΟΣ ΥΠΑΛΛΗΛΟΣ. ΑΡΙΘΜΟΣ ΠΑΙΔΙΩΝ: -4
10ΟΣ ΥΠΑΛΛΗΛΟΣ. ΧΡΟΝΙΑ ΥΠΗΡΕΣΙΑΣ: 51
23ΟΣ ΥΠΑΛΛΗΛΟΣ. ΜΙΣΘΟΣ: 955000
```

8-8 Με βάση τα στοιχεία του διορθωμένου αρχείου, dmisthoi.txt, της προηγούμενης άσκησης, υπολογίζονται οι αποδοχές των υπαλλήλων ως εξής:

- για κάθε χρόνο υπηρεσίας ο μισθός προσαυξάνεται κατά 2% πάνω στο βασικό,
- αν ο εργαζόμενος έχει μέχρι τρία (3) παιδιά ο μισθός προσαυξάνεται κατά 30 € για κάθε παιδί, αλλιώς, αν έχει περισσότερα από τέσσερα (4) ο μισθός προσαυξάνεται κατά 50 € για κάθε παιδί.

Γράψε πρόγραμμα που θα διαβάζει το αρχείο και θα υπολογίζει και θα τυπώνει την κατάσταση αποδοχών των υπαλλήλων.

Γ Ομάδα

8-9 Δίνεται ένα αρχείο text. Γράψε ένα πρόγραμμα που θα αντιγράφει το αρχείο σε ένα άλλο, αφού πρώτα πετάξει τα περιττά κενά του πρώτου. Δηλαδή, όπου υπάρχουν στο παλιό ένα ή περισσότερα συνεχόμενα κενά, στο νέο θα υπάρχει μόνο ένα.

8-10 Γράψε πρόγραμμα που θα διαβάζει ένα αρχείο κείμενο που περιέχει πρόγραμμα C++ και θα το αντιγράφει στην οθόνη με αρίθμηση των γραμμών. Π.χ. θα μας δείχνει:

```
1: #include <iostream>
2: #include <math>
3:
4: int main()
5: {
```

```
6:    int a[100], b[25];
       . . .
```

8-11 Γράψε πρόγραμμα που θα διαβάζει ένα αρχείο κείμενο που περιέχει πρόγραμμα C++ και θα το αντιγράφει σε δύο αρχεία ως εξής:

α) Το ένα θα έχει μόνον τις γραμμές που περιέχουν μόνον σχόλια, δηλαδή γραμμές που ξεκινούν με `//`.

β) Το άλλο θα έχει όλες τις υπόλοιπες γραμμές.

8-12 Τεχνικοί, που σχεδίασαν και κατασκεύασαν μια συσκευή, πιστεύουν ότι δύο βασικά της μεγέθη q και u συνδεόνται με τον απλό νόμο: $q = \eta\mu u$. Μια άλλη ομάδα τεχνικών πιστεύει ότι η εξάρτηση είναι πιο πολύπλοκη: $q = g(u)$, όπου g συνάρτηση περιοδική, με περίοδο 2π, που στο $[-\pi,\pi]$ ορίζεται ως εξής:

$$g(u) = \begin{cases} \frac{4}{\pi^2}u(u+\pi), & -\pi \le u \le 0 \\ -\frac{4}{\pi^2}u(u-\pi), & 0 \le u \le \pi \end{cases}$$

Γράψε συνάρτηση που να υλοποιεί τη g σε C++.

Σε ένα αρχείο text, με όνομα στο δίσκο irnapr.txt, υπάρχουν στοιχεία μετρήσεων της συσκευής. Σε κάθε γραμμή του αρχείου υπάρχουν δύο πραγματικοί αριθμοί: ο πρώτος αντιπροσωπεύει τιμή από μέτρηση της u, ενώ ο δεύτερος είναι η αντίστοιχη τιμή της q. Θέλουμε, με επεξεργασία αυτών των τιμών, να δούμε ποιο από τα δύο μοντέλα είναι καλύτερο.

Ας πούμε λοιπόν ότι διαβάζουμε μια γραμμή του αρχείου και αποθηκεύουμε τις τιμές που διαβάσαμε στις u_k και q_k. Υπολογίζουμε τα $\eta\mu u_k$ και $g(u_k)$ και θα λέμε:

- ότι «είναι καλύτερο το μοντέλο της g» αν $|g(u_k) - q_k| < |\eta\mu u_k - q_k|$·

αλλιώς θα λέμε

- ότι «καλύτερο είναι το μοντέλο του ημιτόνου».

Θέλουμε ένα πρόγραμμα που θα διαβάζει ολόκληρο το αρχείο και θα μας λέει στο τέλος:

- πόσες φορές (για πόσες γραμμές του αρχείου) ήταν καλύτερο το μοντέλο του ημιτόνου και πόσες φορές το μοντέλο της g,
- τις μέσες τιμές των $|g(u_k)-q_k| = \frac{1}{N}\sum_{k=1}^{N}|g(u_k)-q_k|$ και $|\eta\mu u_k-q_k| = \frac{1}{N}\sum_{k=1}^{N}|\eta\mu u_k-q_k|$ όπου N το πλήθος των γραμμών του αρχείου.

Ακόμη, το πρόγραμμα θα αντιγράφει το αρχείο σε δύο άλλα αρχεία text με ονόματα στο δίσκο girnapr.txt, sirnapr.txt, ως εξής:

- Αν για μια γραμμή είναι καλύτερο το μοντέλο της g τότε η γραμμή θα αντιγράφεται «εμπλουτισμένη» στο girnapr.txt. Για την ακρίβεια θα γράφονται: u_k, q_k, $g(u_k)$, $|g(u_k) - q_k|$.
- Αν για μια γραμμή είναι καλύτερο το μοντέλο του ημιτόνου τότε μια «εμπλουτισμένη» γραμμή με τα u_k, q_k, $\eta\mu u_k$, $|\eta\mu u_k - q_k|$ θα γράφεται στο αρχείο sirnapr.txt.

8-13 Για το παρόν πρόβλημα θεωρούμε ως λέξη μια ακολουθία χαρακτήρων που είναι γράμματα του λατινικού αλφαβήτου. Π.χ. στο κείμενο

```
if <x, f, x'> is a state transition
```

υπάρχουν 8 λέξεις με μήκη 2, 1, 1, 1, 2, 1, 5, 10 αντιστοίχως.

Μας δίνεται το αρχείο-κείμενο alturing.txt. Γράψε πρόγραμμα που θα το διαβάζει και θα μας λέει α) το πλήθος των λέξεων που έχει β) το μέγιστο μήκος λέξης που συνάντησε.