Αρχεία Δεδομένων

Μόνιμη αποθήκευση δεδομένων

- Η μνήμη (RAM) του Η/Υ κρατά δεδομένα μόνο **όσο** της δίνεται μια ικανοποιητική παροχή **ρεύματος**.
- Τα περιεχόμενα της μνήμης χάνονται (αμέσως) με το που διακοπεί η τροφοδοσία ρεύματος στον Η/Υ.
- Υπάρχει ανάγκη για την **μόνιμη** αποθήκευση δεδομένων (των προγραμμάτων) ώστε αυτά να εξακολουθούν να υφίστανται αφού σβήσει ο Η/Υ.
- Αυτό επιτυγχάνεται με μαγνητικά/οπτικά μέσα.
- Τα δεδομένα «γράφονται» αλλάζοντας είτε την μαγνητική κατεύθυνση είτε την επιφανειακή δομή του υλικού (μαγνητική ταινία, μαγνητικός δίσκος, οπτικός δίσκος, κλπ) έτσι ώστε να μπορεί να γίνει διαχωρισμός μεταξύ των καταστάσεων 0 και 1.

Μοντέλο πρόσβασης

- Το μοντέλο μνήμης που δίνεται στον προγραμματιστή είναι του **πίνακα** από bytes με **άμεση** πρόσβαση σε όλα τα στοιχεία με το ίδιο κόστος (direct access).
- Η πρόσβαση στα μόνιμα μέσα αποθήκευσης είναι (πολύ) πιο αργή και διαφορετική στη φύση της, απαιτώντας πολύπλοκες λειτουργίες διαχείρισης.
- Για αυτό το λόγο (κυρίως) τα μέσα μόνιμης αποθήκευσης χρησιμοποιούνται μέσα από ένα ειδικό υποσύστημα λογισμικού: το **σύστημα αρχείων**.
- Το μοντέλο που δίνεται στον προγραμματιστή για τα μέσα αποθήκευσης είναι μια **ακολουθία** από bytes με **σειριακή** πρόσβαση (sequential access).

Σχετικές συναρτήσεις

```
#include <stdio.h>

FILE fopen(const char *fname, const char *mode);
void fclose(FILE *f);
void fflush(FILE *f);
int fgetc(FILE *f);
int fputc(int c, FILE *f);
int fscanf(FILE *f, const char *fmt,…);
int fprintf(FILE *f, const char *fmt,…);
size_t fread(const void *ptr, size_t size,
                              size_t n, FILE *f);
size_t fwrite(const void *ptr, size_t size,
                               size_t n, FILE *f);
int fseek(FILE *f, long int offset, int whence);
int feof(FILE *f);
int ferror(FILE *f);
void clearerr(FILE *f);
```

Βασικές συναρτήσεις

- `FILE *fopen(const char *fname,`
  `const char *mode)`

  άνοιγμα του αρχείου με όνομα `fname` σύμφωνα με τον προσδιορισμό `mode` και επιστροφή δείκτη σε αντίστοιχη δομή πρόσβασης
- `void fclose(FILE *f)`

  κλείσιμο της δομής πρόσβασης αρχείου `f` (που έχει επιστραφεί από επιτυχημένη κλήση της `fopen`)
- `void fflush(FILE *f)`

  μονιμοποίηση αλλαγών που έγιναν στο αρχείο, χωρίς να γίνει κλείσιμο του αρχείου – σε κάποια συστήματα αρχείων, η `fclose` **δεν** κάνει `fflush`

Προσδιορισμοί της `fopen`

"r"	άνοιγμα αρχείου (που ήδη υπάρχει) για ανάγνωση
"w"	άνοιγμα αρχείου που ήδη υπάρχει για εγγραφή (με διαγραφή των δεδομένων που υπάρχουν) ή δημιουργία νέου αρχείου αν αυτό δεν υπάρχει
"a"	άνοιγμα αρχείου που υπάρχει για εγγραφή, αποκλειστικά με προσθήκη στο τέλος του αρχείου ή δημιουργία νέου αρχείου αν αυτό δεν υπάρχει
"r+"	"r" με εγγραφή, χωρίς δημιουργία νέου αρχείου
"w+"	"w" με ανάγνωση
"a+"	"a" με ανάγνωση

- Μη επιτρεπτός προσδιορισμός (mode) οδηγεί σε επιστροφή λάθους (τιμής 0) από την `fopen`.

Βασικές συναρτήσεις

- `int fgetc(FILE *f)`

  ανάγνωση του «επόμενου» byte στο αρχείο (ή EOF)
- `int fputc(int c, FILE *f)`

  γράψιμο του «επόμενου» byte στο αρχείο
- `int fscanf(FILE *f, const char *frmt,…)`

  ανάγνωση bytes από το αρχείο σύμφωνα με τον προσδιορισμό `frmt` και αποθήκευση τιμών στις μεταβλητές των οποίων οι διευθύνσεις δίνονται σαν παράμετροι – σε αντιστοιχία με την `scanf`
- `int fprintf(FILE *f, const char *frmt,…)`

  γράψιμο bytes στο αρχείο σύμφωνα με τον προσδιορισμό `frmt` – σε αντιστοιχία με την `printf`

Βασικές συναρτήσεις

- `size_t fread(const void *p, size_t size, size_t n, FILE *f)`

  ανάγνωση από `n` αντικείμενα μεγέθους `size` bytes και αποθήκευση στη μνήμη, αρχίζοντας από την διεύθυνση `p` – επιστρέφει τον αριθμό των αντικειμένων που διαβάστηκαν επιτυχώς

- `size_t fwrite(const void *p, size_t size, size_t n, FILE *f)`

  γράψιμο από `n` αντικείμενα μεγέθους `size` bytes στο αρχείο, αρχίζοντας από την διεύθυνση `p` – επιστρέφει τον αριθμό των αντικειμένων που γράφτηκαν επιτυχώς

Βασικές συναρτήσεις

- `int feof(FILE *f)`

  επιστρέφει τιμή !0 αν η επόμενη πράξη θα προσπαθήσει να «περάσει» το μέγεθος του αρχείου, διαφορετικά 0
- `int ferror(FILE *f)`

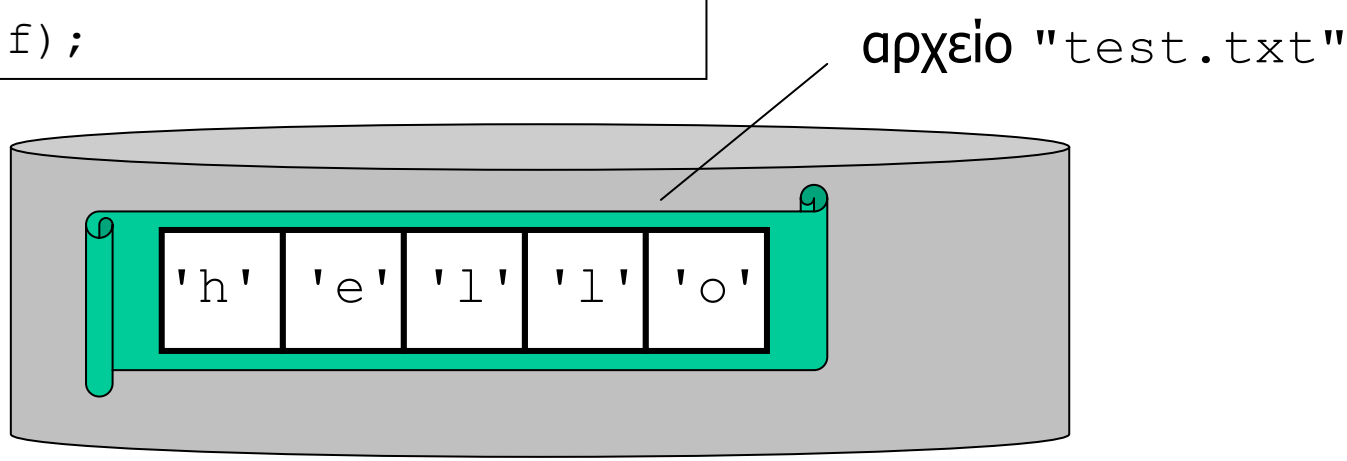
  επιστρέφει τιμή !0 αν παρουσιαστεί κάποιο πρόβλημα με τις πράξεις πρόσβασης για το συγκεκριμένο αρχείο (ο λόγος της αποτυχίας αποθηκεύεται ως ακέραιος στην καθολική μεταβλητή `errno`, και το αντίστοιχο μήνυμα λάθους επιστρέφεται μέσω της `strerror`)
- `int clearerr(FILE *f)`

  επαναφέρει την κατάσταση πρόσβασης σε «κανονική», αγνοώντας τυχόν προηγούμενο πρόβλημα πρόσβασης

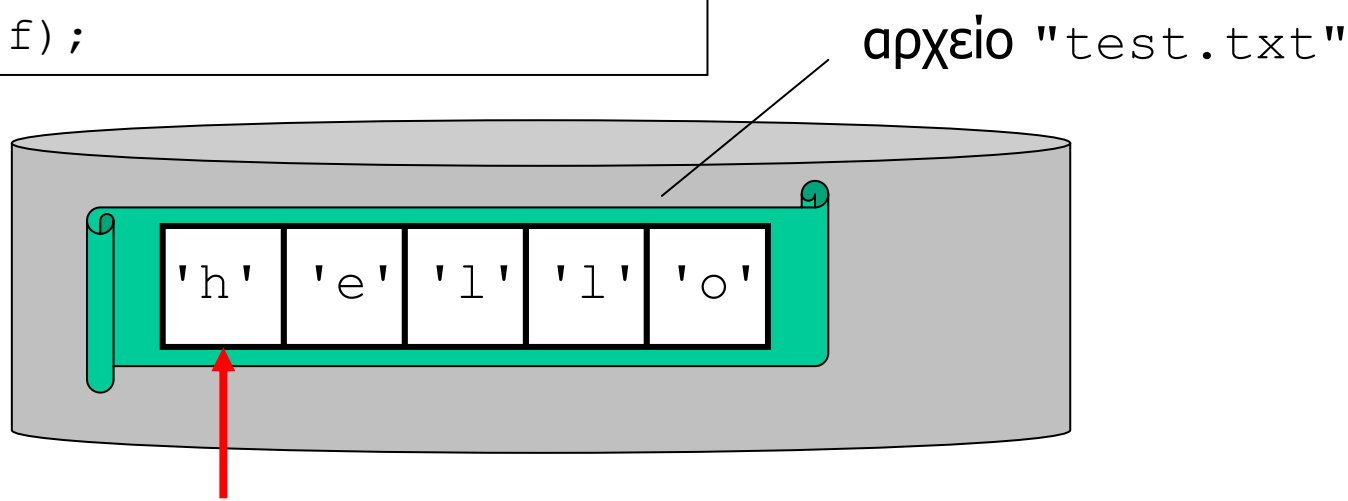
Σειριακή πρόσβαση

- Για κάθε αρχείο διατηρείται (εσωτερικά) μια μεταβλητή που υποδεικνύει την θέση στο αρχείο από την οποία θα αρχίσει η **επόμενη** πράξη (ανάγνωσης/εγγραφής).
- Κάθε πράξη ανάγνωσης ή εγγραφής **αυξάνει** την τιμή του δείκτη κατά τον **αριθμό** των bytes που διαβάστηκαν από το αρχείο ή γράφτηκαν στο αρχείο.
- Όταν ο δείκτης φτάσει το τέλος του αρχείου, οι πράξεις ανάγνωσης δεν εκτελούνται και επιστρέφουν EOF ή/και κωδικό λάθους (οι πράξεις εγγραφής απλά επεκτείνουν το μέγεθος του αρχείου όσο χρειάζεται).
- Πρέπει **πάντα** να γίνεται έλεγχος της τιμής που επιστρέφουν οι πράξεις πρόσβασης σε αρχεία.

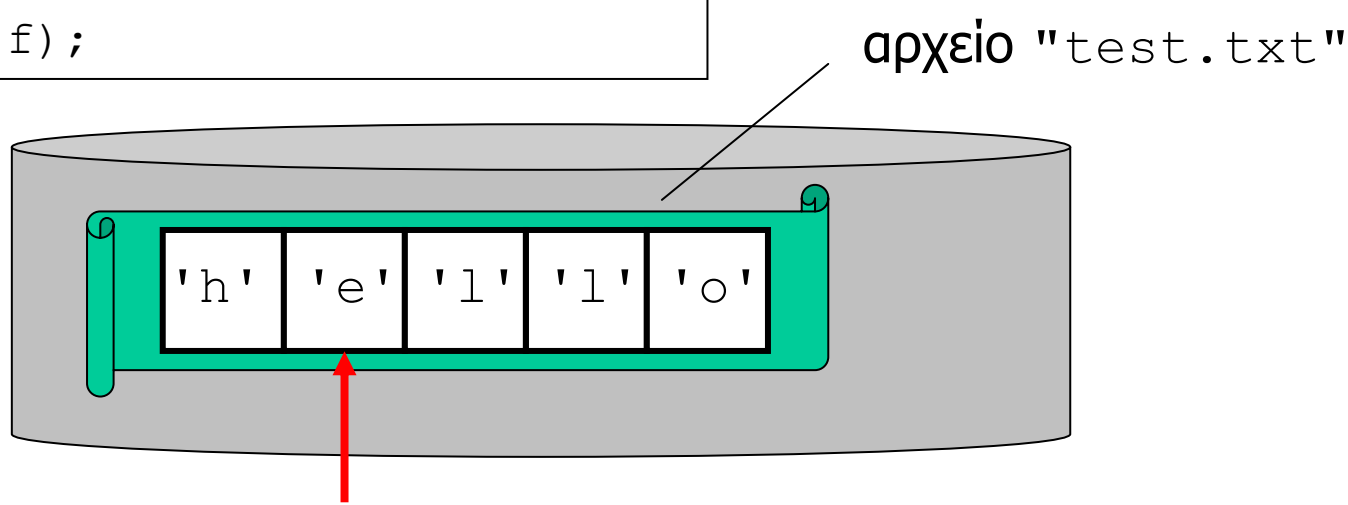
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



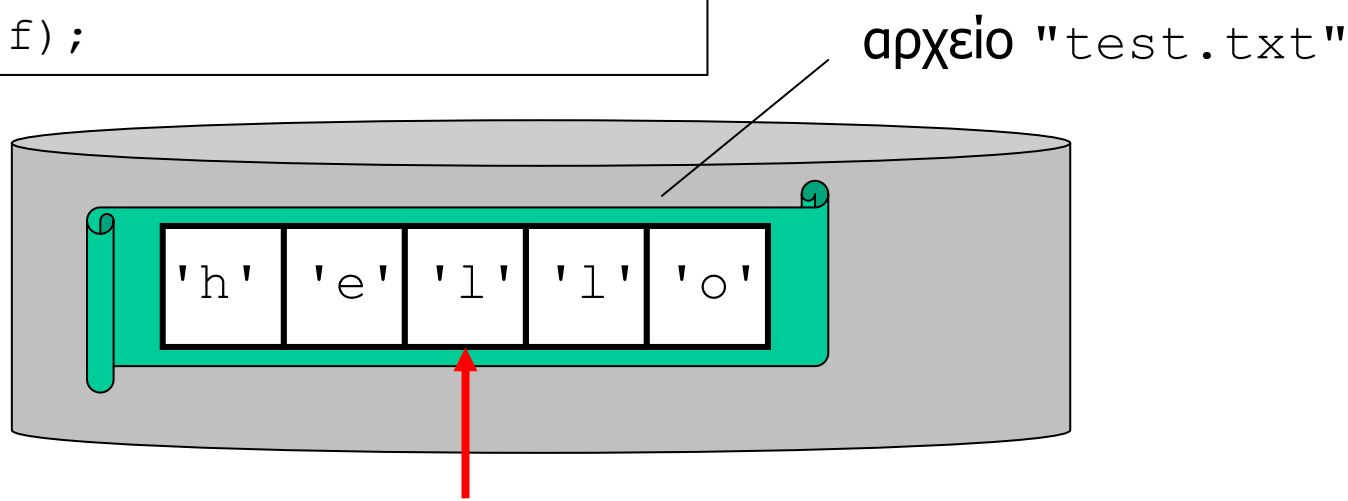
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



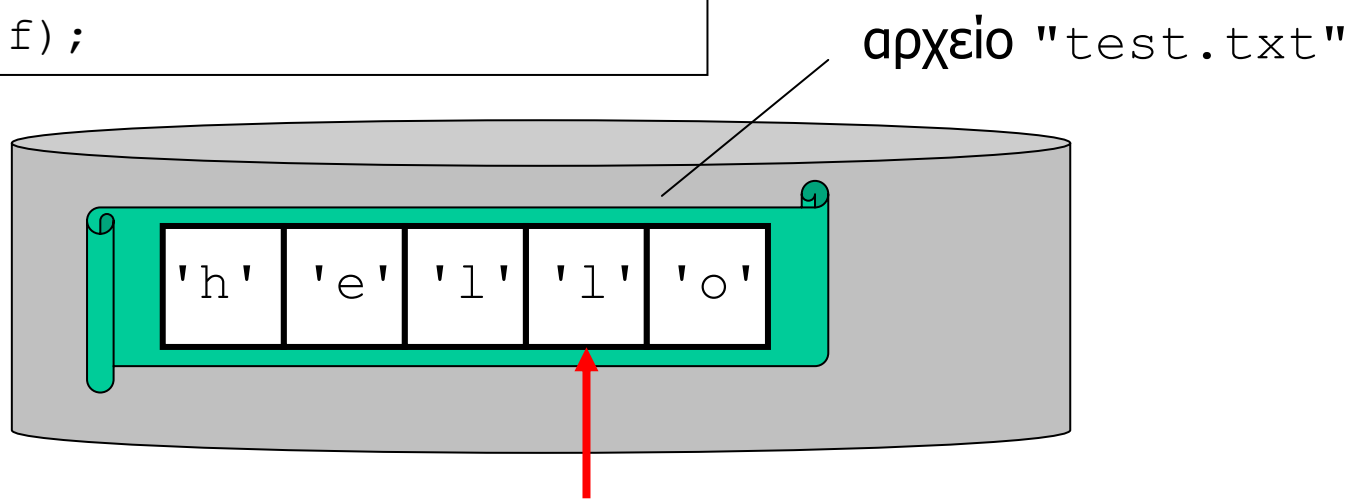
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



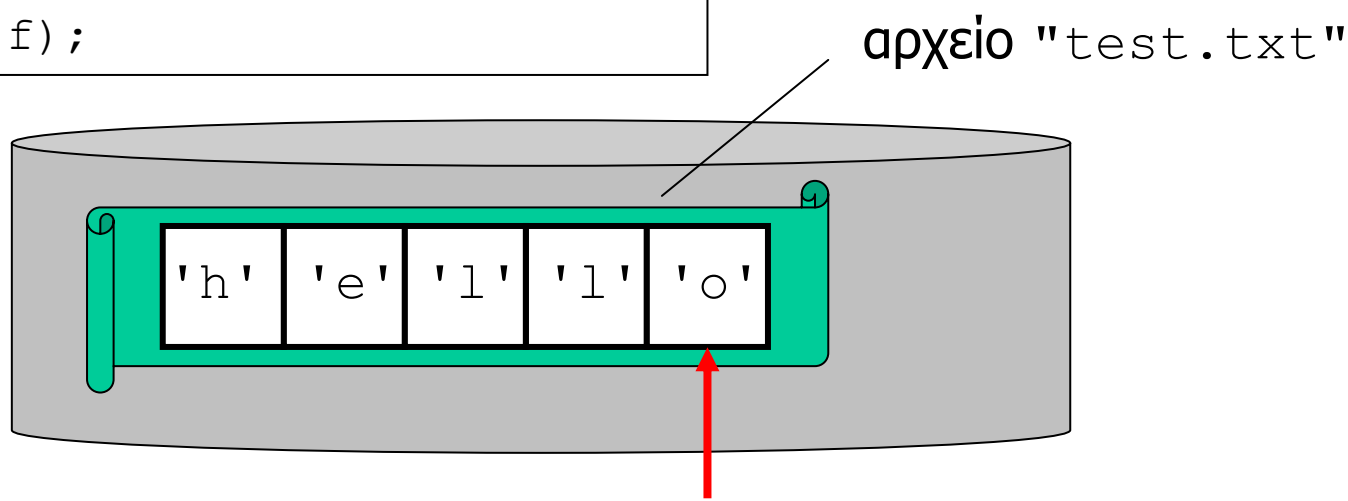
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



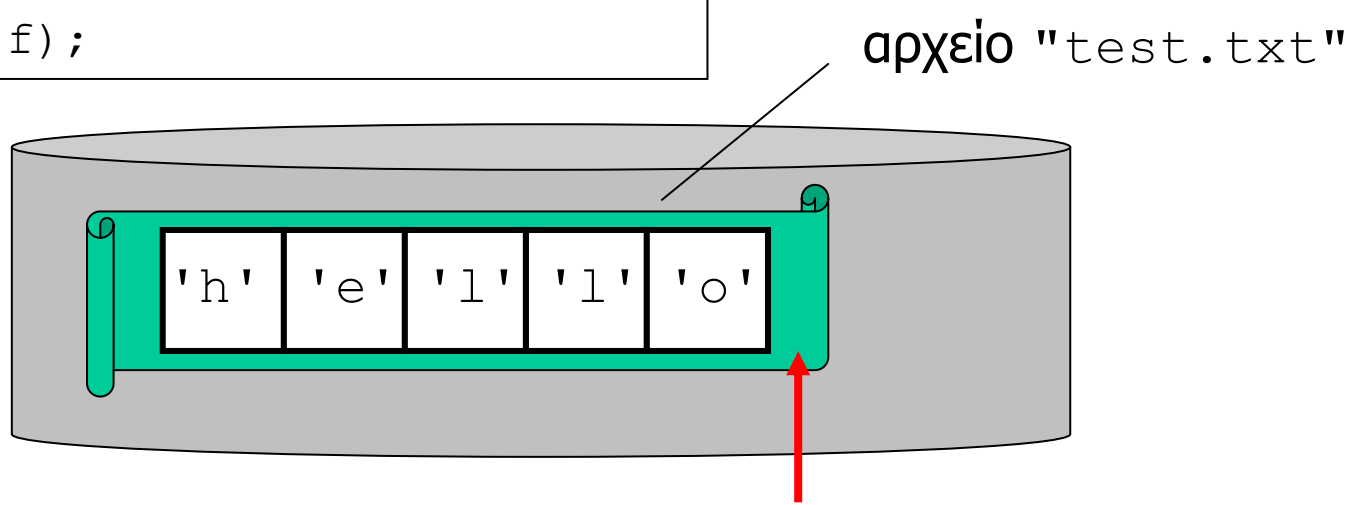
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



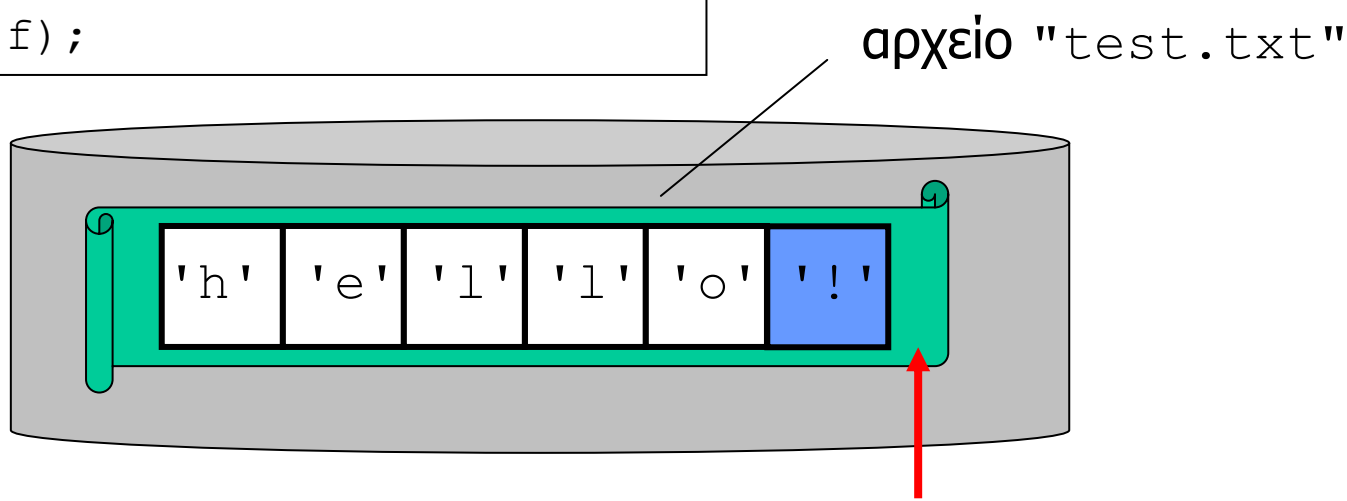
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



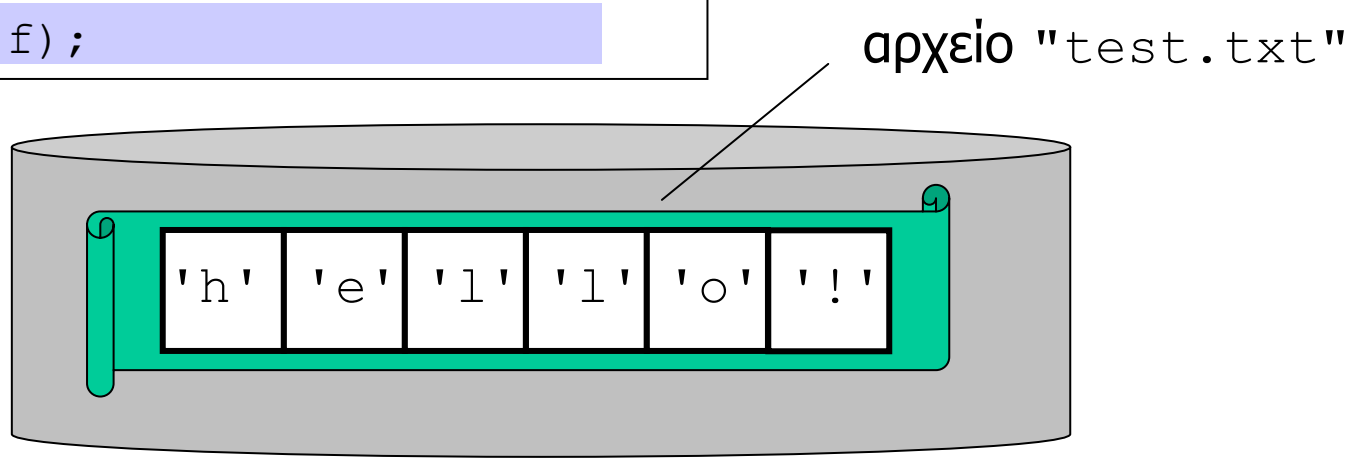
```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



```
FILE *f; char c;

f = fopen("test.txt","r+");

c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);
c = fgetc(f); putchar(c);

fputc('!',f); fflush(f);

fclose(f);
```



Βασικές συναρτήσεις

- Ο δείκτης μπορεί να επανατοποθετηθεί σε κάποιο σημείο του αρχείου μέσω της πράξης `fseek`.
- `int fseek(FILE *f, long offset,`
  `int whence)`

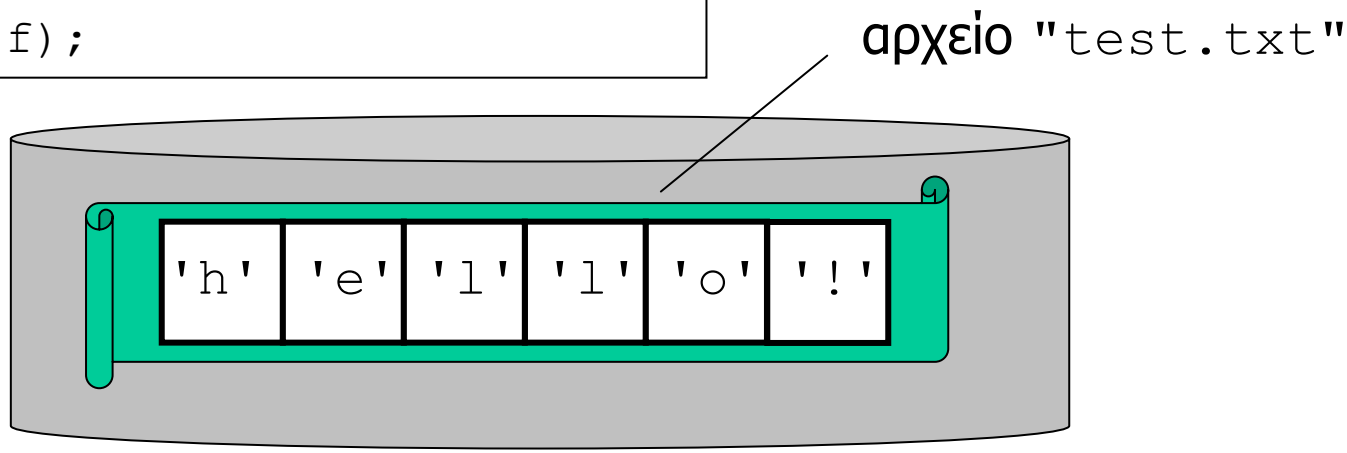
  τοποθέτηση σημείου ανάγνωσης/εγγραφής στην θέση `offset` και **σε σχέση** με τον προσδιορισμό `whence`
- Η τιμές για την παράμετρο `whence`:

  SEEK_SET (από την αρχή του αρχείου)

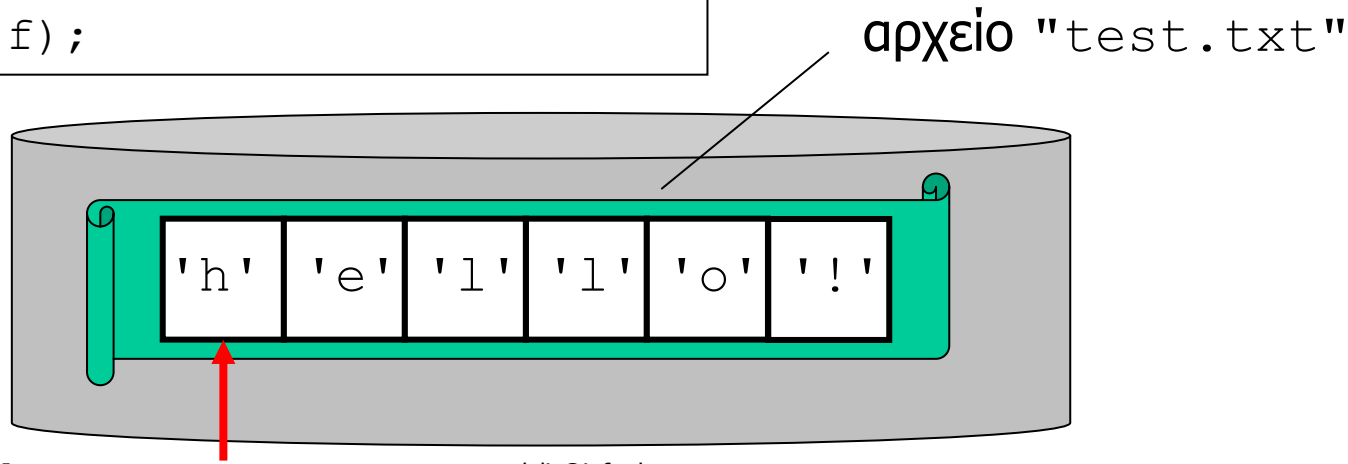
  SEEK_CUR (από την τρέχουσα θέση)

  SEEK_END (από το τέλος του αρχείου)
- Επιστρέφει 0 εάν επιτύχει, διαφορετικά -1.

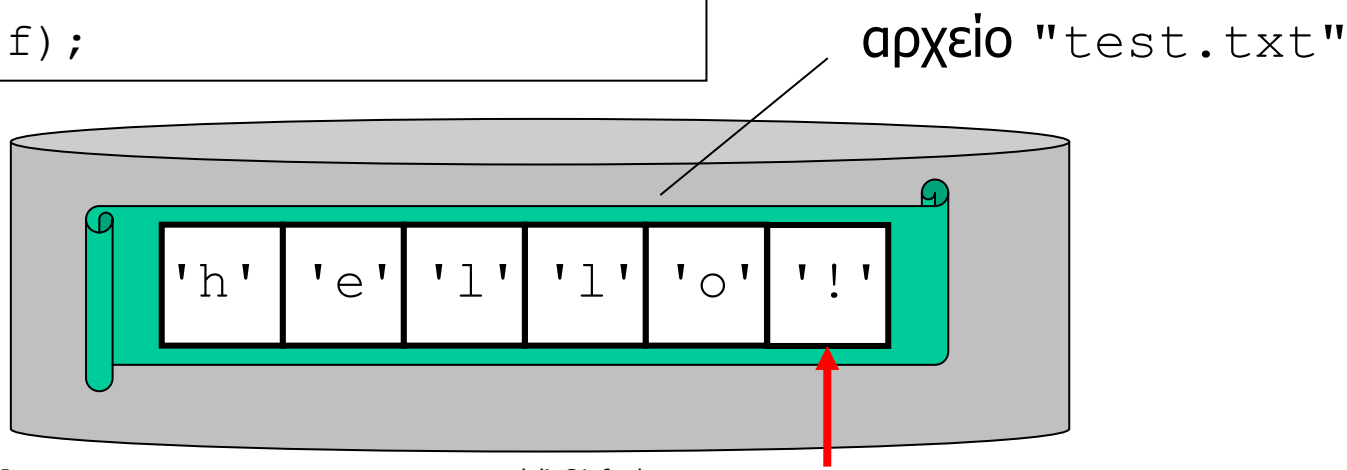
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



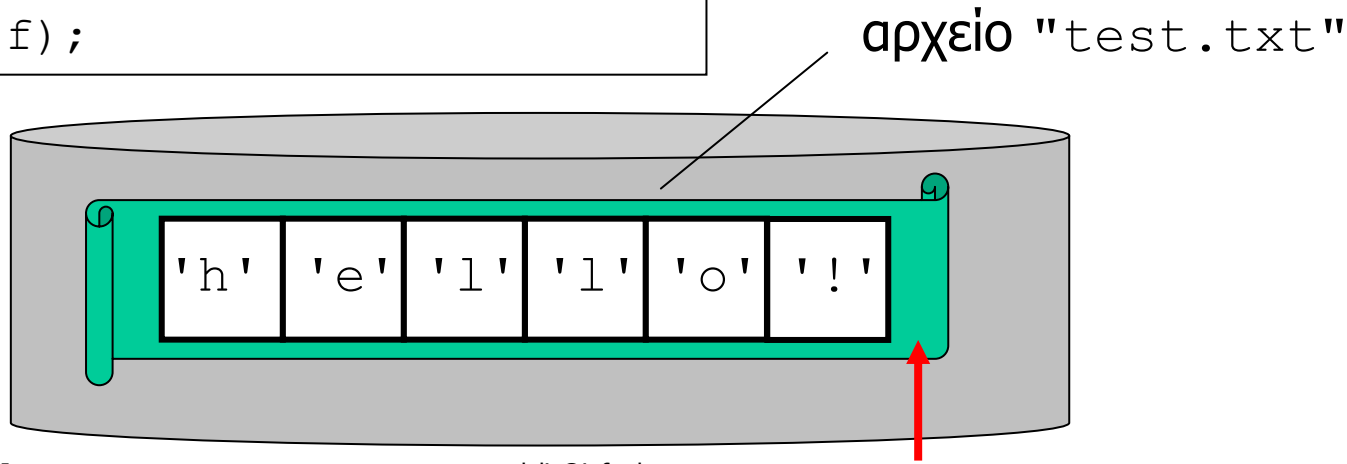
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



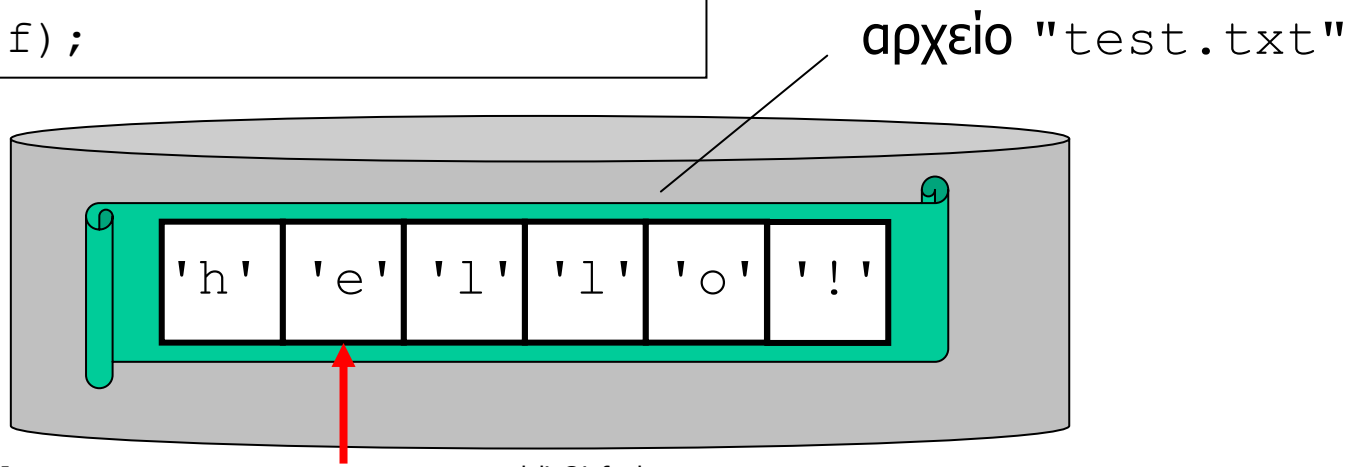
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



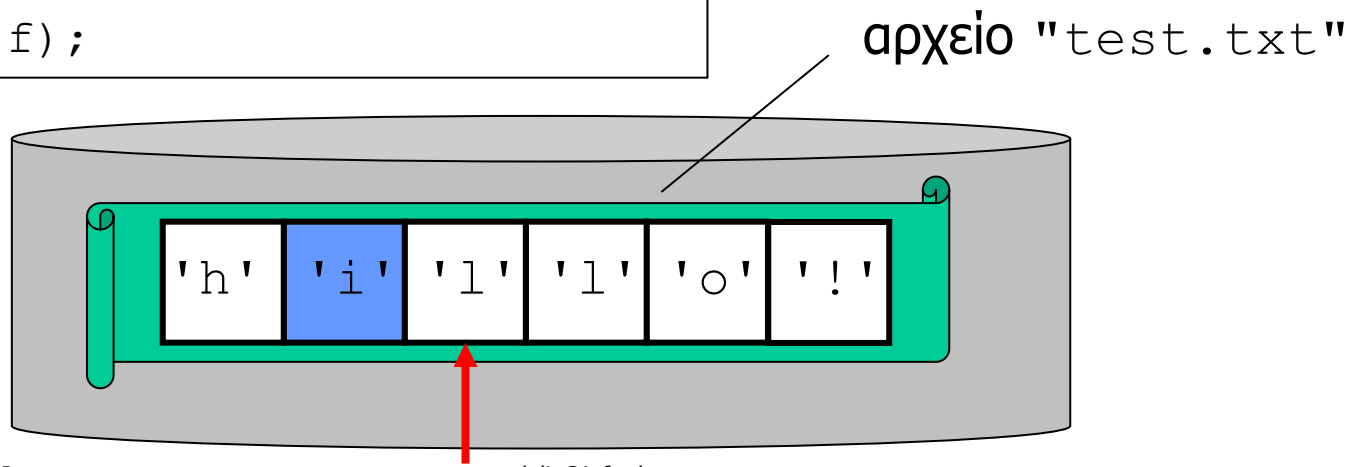
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



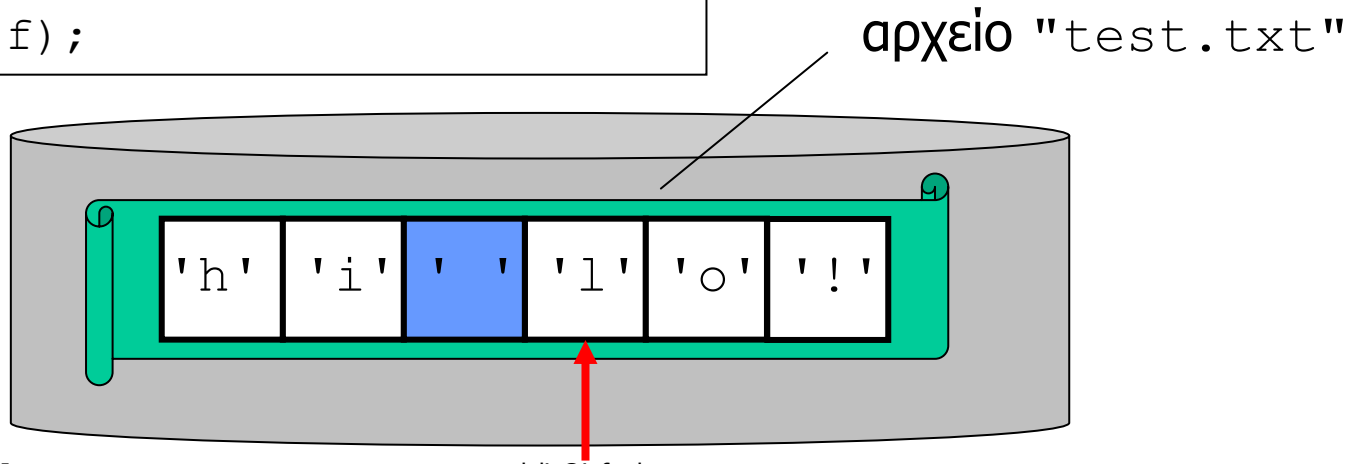
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



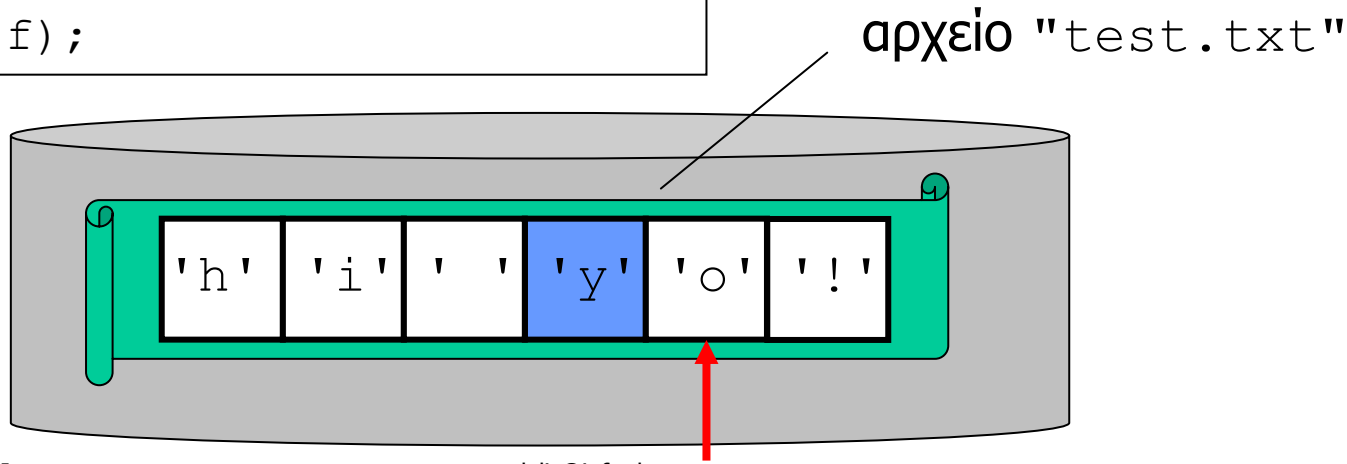
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



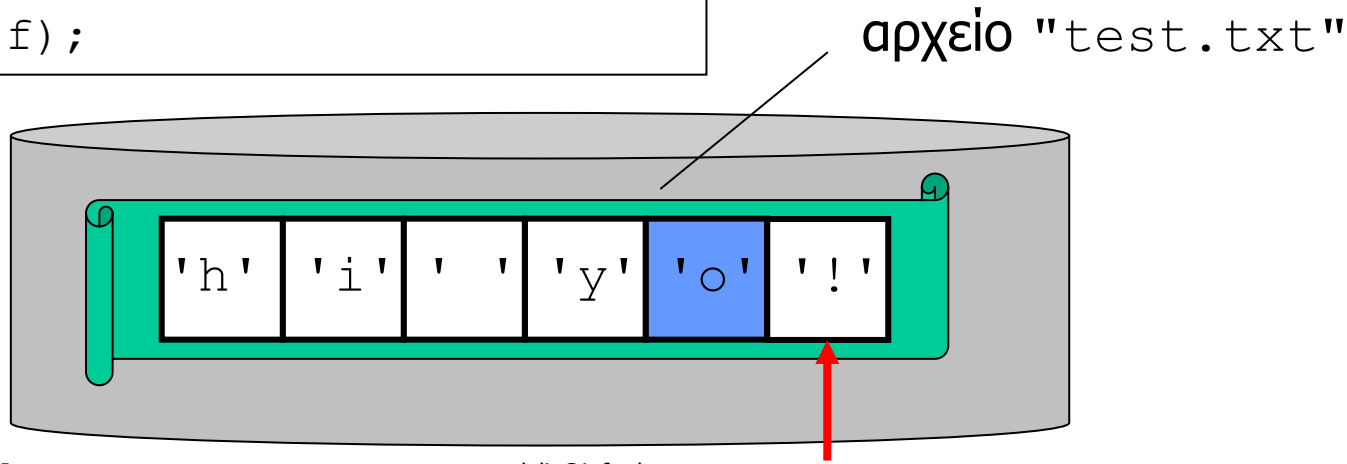
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



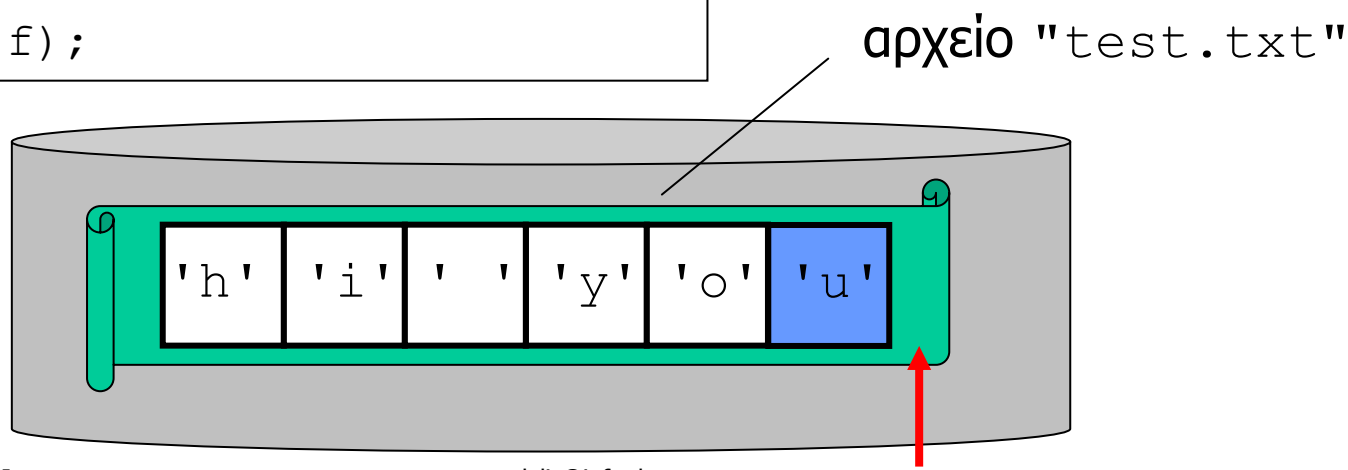
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



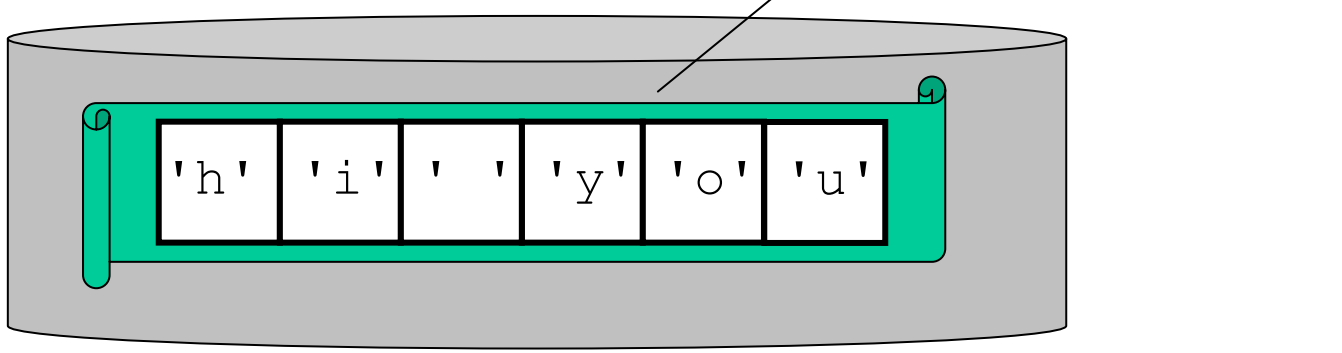
```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



```
FILE *f; char c;

f = fopen("test.txt","r+");

fseek(f,-1,SEEK_END);
c = fgetc(f); putchar(c);

fseek(f,-5,SEEK_CUR);
fputc('i',f); fflush(f);
fputc(' ',f); fflush(f);
fputc('y',f); fflush(f);
fputc('o',f); fflush(f);
fputc('u',f); fflush(f);

fclose(f);
```



```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  FILE *f; char c;

  f = fopen(argv[1],"w");

  do {
    c = getchar();
    fputc(c,f);
  } while (c != '~');
  fflush(f); fclose(f);

  f = fopen(argv[1],"r");

  while (!feof(f)) {
    c = fgetc(f);
    putchar(c);
  }
  fclose(f);

}
```

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  FILE *f; char s[64];

  f = fopen(argv[1],"w");

  do {
    scanf("%63s",s);
    fprintf(f,"%s ",s);
  } while (strcmp(s,"end"));
  fflush(f); fclose(f);

  f = fopen(argv[1],"r");

  while (!feof(f)) {
    fscanf(f,"%63s",s);
    printf("%s ",s);
  }
  fclose(f);

}
```

κενός χαρακτήρας ως **διαχωριστικό** ανάμεσα στα δύο αλφαριθμητικά

Αποθήκευση binary / ASCII

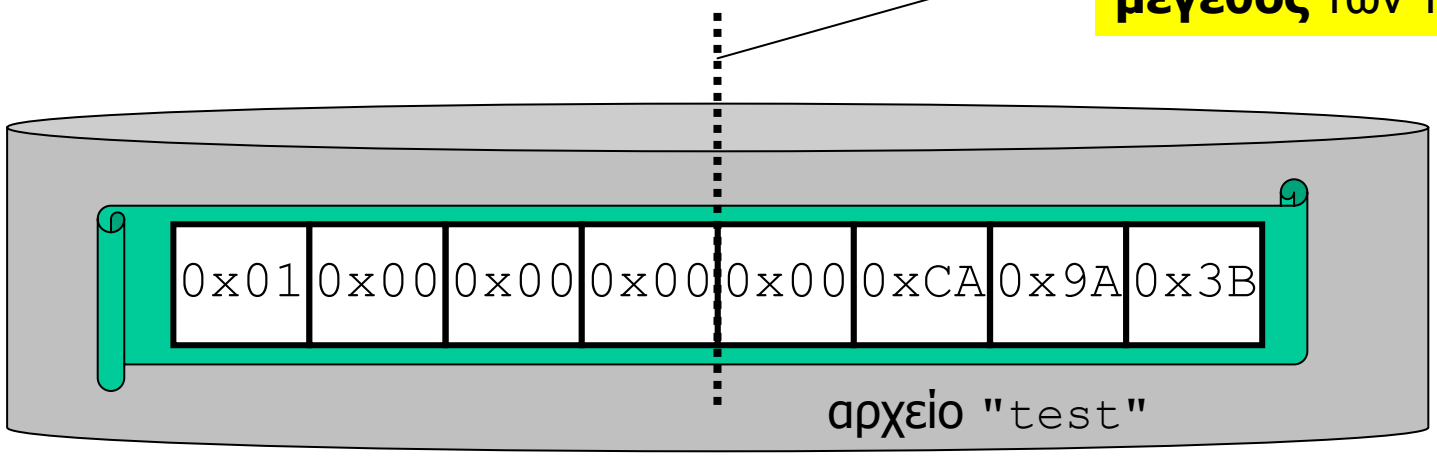
- Μπορούμε να αποθηκεύσουμε τα δεδομένα σε μορφή binary ή σε μορφή χαρακτήρων ASCII (text).
- Για την **ίδια** πληροφορία **διαφέρει** ο αριθμός και φυσικά(!) οι τιμές των bytes που γράφονται.
- Π.χ. `int i=1;`
  - binary: `x00 0x00 0x00 0x01`
  - ascii: `0x31`
- Π.χ. `int i=100000000;`
  - binary: `0x3B 0x9A 0xCA 0x00`
  - ascii: `0x31 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30`

Αποθήκευση binary

- Γίνεται με την συνάρτηση `fwrite`.
- Δεν απαιτούνται (απαραίτητα) διαχωριστικά bytes καθώς για κάθε τύπο δεδομένων γράφεται ένας **συγκεκριμένος** αριθμός bytes – ακριβώς όσα bytes αντιστοιχούν στο **μέγεθος** του τύπου.
- Κατά την ανάγνωση, μέσω `fread`, διαβάζονται όσα bytes αντιστοιχούν στο μέγεθος του τύπου.
- Υπάρχει πρόβλημα **ασυμβατότητας**, αν τα δεδομένα γράφονται / διαβάζονται από προγράμματα που εκτελούνται σε περιβάλλοντα με διαφορετικές συμβάσεις (α) για τα μεγέθη βασικών και σύνθετων τύπων ή/και (β) την αποθήκευση τους στην μνήμη.

```
f=fopen("test","w");

i=1; j=1000000000;

fwrite((void*)&i,sizeof(int),1,f);

fwrite((void*)&j,sizeof(int),1,f);

fflush(f); fclose(f);
```

τα όρια μεταξύ των δεδομένων δίνονται **έμμεσα** από το **μέγεθος** των τύπων



Συμβάσεις κωδικοποίησης δεδομένων

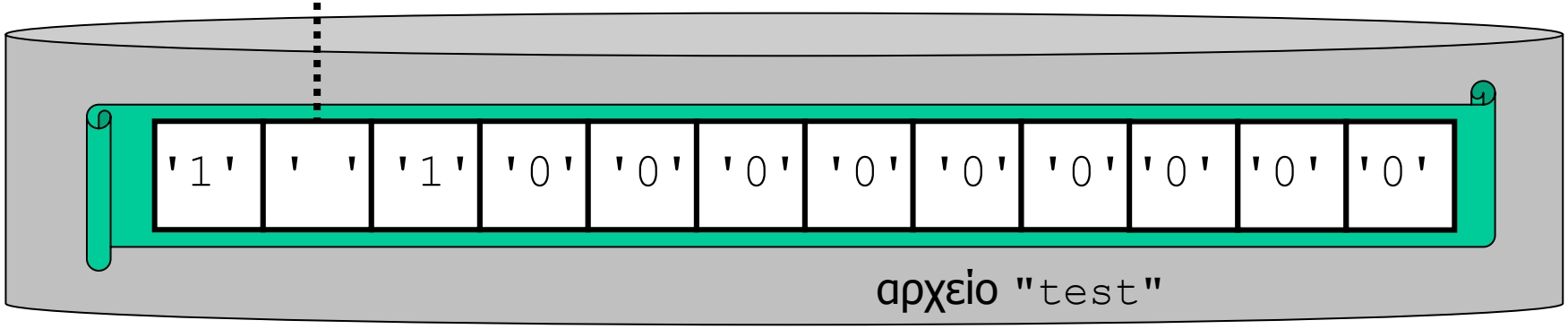
- Για να επιτευχθεί μεταφερσιμότητα των δεδομένων, υιοθετούνται **κοινές συμβάσεις εξωτερικής αναπαράστασης**, για το **πως** αποθηκεύονται τιμές όλων των βασικών τύπων δεδομένων με τρόπο που είναι **ανεξάρτητος** της μεθόδου αποθήκευσης δεδομένων στην μνήμη του εκάστοτε περιβάλλοντος εκτέλεσης (γλώσσα, μεταφραστής, επεξεργαστής).
- Σημείωση: το ίδιο πρόβλημα πρέπει να λυθεί για να επιτευχθεί ανταλλαγή δεδομένων πάνω από δίκτυο.
- Μια κλασική λύση: αποθήκευση σε μορφή ASCII.

Αποθήκευση ASCII

- Γίνεται με την συνάρτηση `fprintf`.
- Απαιτείται η εισαγωγή **επιπλέον χαρακτήρων μορφοποίησης** έτσι ώστε να **οριοθετούνται** τα διάφορα δεδομένα που αποθηκεύονται.
- Ως **διαχωριστικό** χρησιμοποιούνται συνήθως οι χαρακτήρες `' '` ή `'\n'` , όπως και για τις τιμές που εκτυπώνουμε στην οθόνη μέσω `printf`, για να μπορεί ο χρήστης να εντοπίζει (οπτικά) τον τερματισμό μιας τιμής και την αρχή της επόμενης.
- Σημείωση: χρειάζεται προσοχή για τον σωστό χειρισμό των «διαχωριστικών» χαρακτήρων που τυχόν δημιουργούνται (ότι δεν είναι γράμμα/ψηφίο).

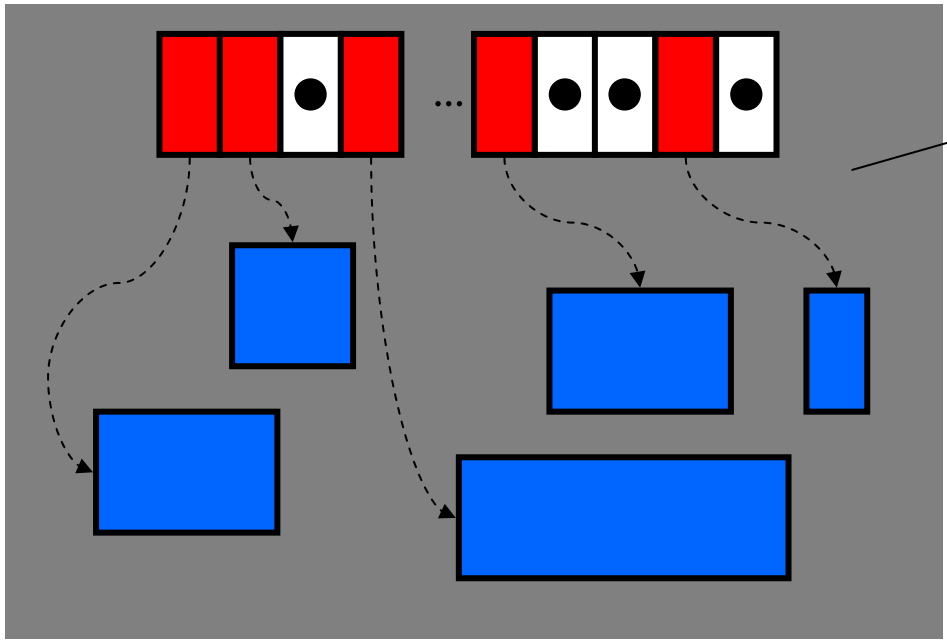
```
f=fopen("test","w");

i=1; j=1000000000;

fprintf(f,"%d %d",i,j);

fflush(f); fclose(f);
```

κενός χαρακτήρας ως **διαχωριστικό** ανάμεσα στα δύο αλφαριθμητικά



Σύμβαση γραψίματος/ανάγνωσης

- Ανεξάρτητα με την μορφή που γίνεται η αποθήκευση, binary ή ASCII, πρέπει να υπάρχει **κοινή σύμβαση γραψίματος και ανάγνωσης** δεδομένων την οποία να ακολουθεί τόσο ο κώδικας γραψίματος όσο και ο κώδικας ανάγνωσης των δεδομένων.
- Πρέπει να καθοριστούν: (α) η **μορφή** αποθήκευσης,, (β) η **κωδικοποίηση** των δεδομένων, καθώς και (γ) η **σειρά** (και η **λογική**) με την οποία γράφονται τα δεδομένα στο αρχείο –και σύμφωνα με την οποία θα διαβαστούν τα δεδομένα από το αρχείο.
- Αυτή η σύμβαση πρέπει να **τεκμηριωθεί** κατάλληλα.
- Η μετατροπή πολύπλοκων δομών σε αποθηκεύσιμη μορφή ονομάζεται συχνά και **«σειριοποίηση»**.



αποθήκευση δεδομένων σε δομές κυρίως μνήμης Η/Υ

σειριοποίηση

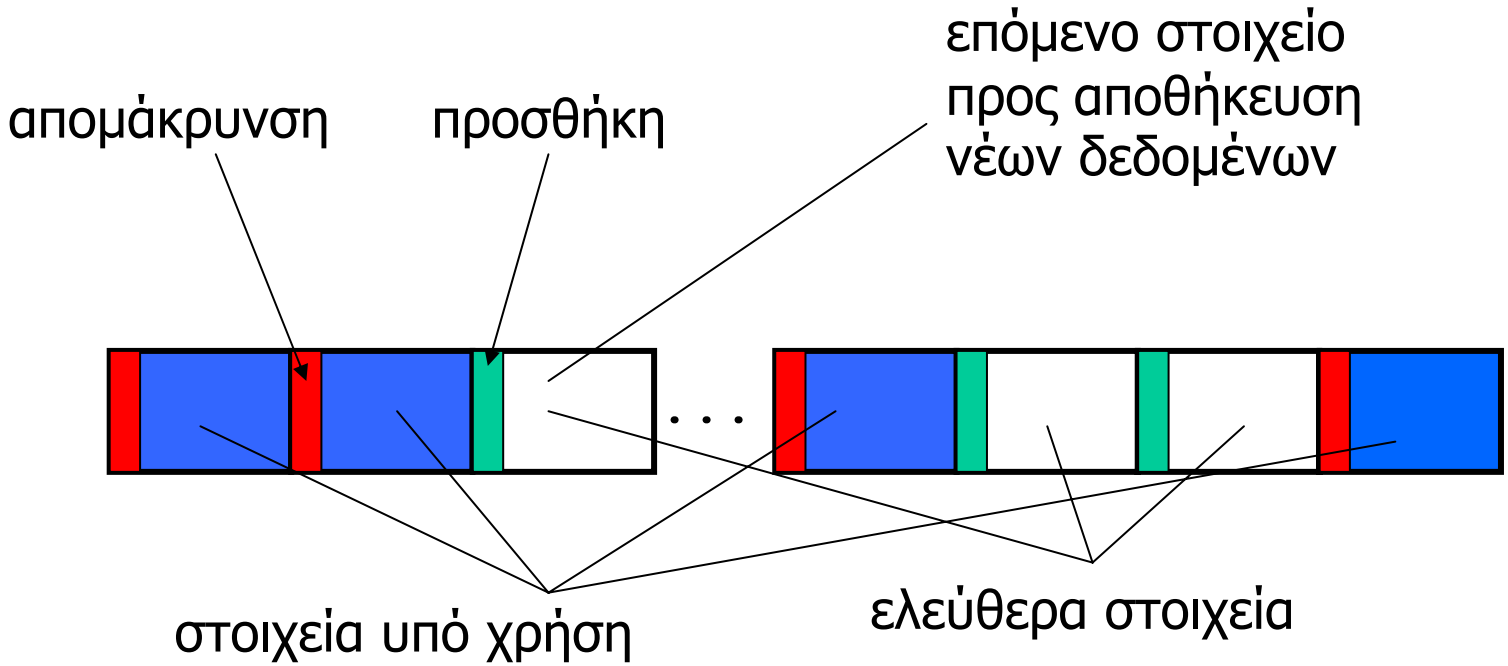
αποσειριοποίηση

αποθήκευση δεδομένων σε «επίπεδη» μορφή

Παρένθεση
(σώσιμο και επαναφορά βάσης δεδομένων σε αρχείο)

Πρόβλημα

- Ζητούμενο: επιθυμούμε να σώσουμε τα δεδομένα που διαχειριζόμαστε μέσω της δομής στην κυρίως μνήμη σε ένα αρχείο, καθώς και να επαναφέρουμε μια «προηγούμενη»κατάσταση ανά πάσα στιγμή διαβάζοντας τα δεδομένα από το αντίστοιχο αρχείο.
- Διαδικασία εγγραφής
  - διασχίσουμε την δομή των δεδομένων
  - για κάθε στοιχείο / κόμβο που είναι υπό χρήση, πραγματοποιούμε αντίστοιχη εγγραφή στο αρχείο
- Διαδικασία ανάγνωσης
  - δημιουργούμε μια «άδεια» δομή
  - για κάθε εγγραφή που διαβάζουμε από το αρχείο, προσθέτουμε ένα αντίστοιχο στοιχείο / κόμβο


επόμενο στοιχείο
προς αποθήκευση
νέων δεδομένων
απομάκρυνση
προσθήκη
στοιχεία υπό χρήση
ελεύθερα στοιχεία

```
/* phone book */

#define N 100

struct entry {
  char used;
  char name[64];
  char phone[64];
};

struct entry entries[N];

void phonebook_init() {
  int i;
  for (i=0; i<N; i++) { entries[i].used=0; }
}
```

```
void phonebook_save_binary(const char *fname) {
  int i; FILE *f;

  f = fopen(fname,"w");
  for (i=0; i<N; i++) {
    if (entries[i].used) {
      fwrite(&entries[i],sizeof(struct entry),1,f);
    }
  }
  fclose(f);
}

void phonebook_load_binary(const char *fname) {
  int i; FILE *f; char tag;

  phonebook_init(); /* clean state */
  f = fopen(fname,"r");
  i=0;
  while (fread(&entries[i],sizeof(struct entry),1,f)) {
    i++;
  }
  fclose(f);
}
```

Σχόλιο

- Οι συναρτήσεις `save` και `restore` αποθήκευουν τα δεδομένα σε μορφή binary, και μάλιστα «ως έχουν» στην μνήμη του περιβάλλοντος εκτέλεσης του Η/Υ.
- Η υλοποίηση δεν επιτυγχάνει μεταφερσιμότητα δεδομένων, π.χ. στην περίπτωση που το πρόγραμμα (ξανα)μεταφραστεί με διαφορετικό μεταφραστή ή/και εκτελεστεί πάνω σε διαφορετική αρχιτεκτονική.
- Για να επιτευχθεί μεταφερσιμότητα πρέπει να ορίσουμε και να υλοποιήσουμε μια κωδικοποίηση που είναι ανεξάρτητη περιβάλλοντος εκτέλεσης.
- Μια «εύκολη» λύση: μετατρέπουμε τα δεδομένα σε/από χαρακτήρες ASCII, μέσω `fprintf`/`fscanf`.

```
void phonebook_save_ascii(const char *fname) {
  int i; FILE *f;

  f = fopen(fname,"w");
  for (i=0; i<N; i++) {
    if (entries[i].used) {
      fprintf(f,"%s %s\n",entries[i].name,entries[i].phone);
    }
  }
  fclose(f);
}

void phonebook_load_ascii(const char *fname) {
  int i,tag; FILE *f;

  phonebook_init(); /* clean state */
  f = fopen(fname,"r");
  i=0;
  while (fscanf(f,"%s %s",entries[i].name,entries[i].phone)==2) {
    entries[i].used = 1; i++;
  }
  fclose(f);
}
```

Σε μια σοβαρή εφαρμογή

- Θα γινόταν έλεγχος για περίπτωση αποτυχίας μετά από κάθε πράξη εγγραφής / ανάγνωσης αρχείου.
- Κατά την ανάγνωση θα γινόταν έλεγχος για το κατά πόσο υπάρχει πρόβλημα ακεραιότητας του αρχείου (για την αποφυγή ανάγνωσης αρχείων που δεν περιέχουν δεδομένα του προγράμματος ή αρχείων που έχουν «καταστραφεί»).
- Η αντικατάσταση των δεδομένων του προγράμματος θα γινόταν **εφόσον** η διαδικασία ανάγνωσης των δεδομένων ολοκληρωνόταν επιτυχώς.

Παρένθεση
(σώσιμο και επαναφορά βάσης δεδομένων σε αρχείο)