

Κεφάλαιο

14

# Χειρισμός αρχείων



## Χειρισμός αρχείων

Η C είναι μια γλώσσα που διακρίνεται για την "ειλικρίνειά" της. Είναι μια γλώσσα που δεν προσπαθεί να κρύψει από τον προγραμματιστή την πραγματικότητα.

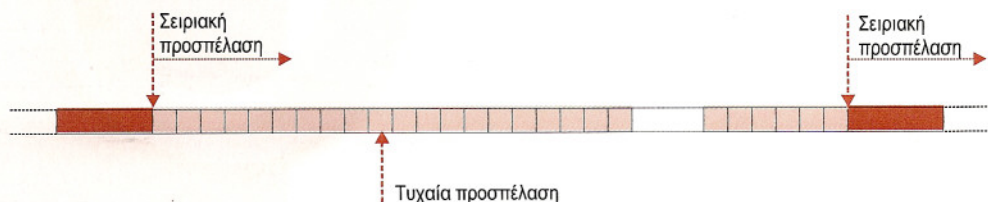
Στη C δεν υπάρχουν δύο ειδών αρχεία όπως σε άλλες γλώσσες (σειριακά και τυχαίας προσπέλασης), αλλά μόνο ένας τύπος αρχείων —όπως είναι στην πραγματικότητα. Αυτό που αλλάζει μεταξύ της σειριακής και της τυχαίας προσπέλασης σε ένα αρχείο δεν είναι ο τύπος του αρχείου αλλά ο διαφορετικός τρόπος προσπέλασης στο αρχείο.

**Σειριακή προσπέλαση** αρχείου σημαίνει ότι διαβάζουμε (ή γράφουμε) πληροφορίες στο αρχείο με τη σειρά, ξεκινώντας από την αρχή του αρχείου (ή και από το τέλος σε περίπτωση εγγραφής). Για να διαβάσουμε π.χ. το 100ό δεδομένο, πρέπει πρώτα να διαβάσουμε με τη σειρά τα προηγούμενα 99.

Στη σειριακή προσπέλαση μπορούμε:

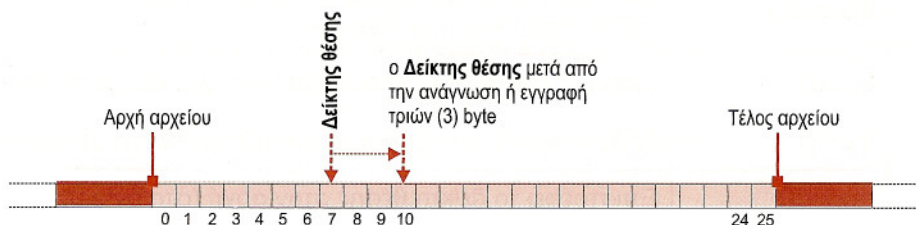
- Να διαβάσουμε με τη σειρά ξεκινώντας από την αρχή του αρχείου.
- Να γράψουμε ξεκινώντας από την αρχή του αρχείου διαγράφοντας τα υπάρχοντα δεδομένα.
- Να γράψουμε ξεκινώντας από το τέλος του αρχείου και προσθέτοντας τα νέα δεδομένα στα υπάρχοντα.

Στην **τυχαία προσπέλαση** μπορούμε να μεταφερθούμε κατευθείαν σε ένα συγκεκριμένο σημείο του αρχείου και να διαβάσουμε ή να γράψουμε δεδομένα ξεκινώντας από αυτό το σημείο.



Για τη C, ένα αρχείο είναι μια σειρά από byte όπου το πρώτο byte έχει αύξοντα αριθμό 0, το δεύτερο 1, κ.ο.κ.

Ο δείκτης θέσης καθορίζει ποιο θα είναι το επόμενο byte που θα διαβαστεί ή θα γραφεί. Έτσι, στο επόμενο παράδειγμα, αν διαβαστούν 3 byte, αυτά θα είναι το 7, το 8, και το 9. Επίσης, αν γραφούν 3 byte, θα γραφούν στη θέση του 7, του 8, και του 9 αντικαθιστώντας τα παλιά τους περιεχόμενα. Μετά την ανάγνωση των τριών byte, ο δείκτης θα μετακινηθεί κατά τρεις θέσεις ώστε να δείχνει το δέκατο byte του αρχείου.



👉 Κάθε φορά που διαβάζεται ή γράφεται ένα byte σε ένα αρχείο, ο δείκτης θέσης μετακινείται μία θέση δεξιά ώστε να δείχνει το αμέσως επόμενο byte.

Η C διαθέτει μία συνάρτηση, την `fseek()`, με την οποία μπορούμε να τοποθετήσουμε το δείκτη θέσης σε όποιο σημείο του αρχείου επιθυμούμε. Με την `fseek()` επιτυγχάνεται ουσιαστικά η τυχαία προσπέλαση του αρχείου.

Ο επόμενος πίνακας παρουσιάζει τις συναρτήσεις χειρισμού αρχείων στη C:

Συνάρτηση	Λειτουργία
<code>fopen()</code>	Ανοίγει ένα αρχείο
<code>fclose()</code>	Κλείνει ένα αρχείο
<code>fputc()</code>	Γράφει ένα χαρακτήρα σε αρχείο
<code>fgetc()</code>	Διαβάζει ένα χαρακτήρα από αρχείο
<code>fprintf()</code>	Γράφει όπως η <code>printf</code> , αλλά σε αρχείο
<code>fscanf()</code>	Διαβάζει όπως η <code>scanf</code> , αλλά από αρχείο




Συνάρτηση	Λειτουργία
feof()	Επιστρέφει τιμή 'αλήθεια' (1) όταν έχουμε φτάσει στο τέλος του αρχείου
ferror()	Επιστρέφει τιμή 'αλήθεια' (1) αν έχει συμβεί κάποιο λάθος στο χειρισμό του αρχείου
fflush()	Αδειάζει την περιοχή προσωρινής αποθήκευσης εισόδου/εξόδου (I/O buffer) ενός αρχείου
fflushall()	Αδειάζει την περιοχή προσωρινής αποθήκευσης εισόδου/εξόδου (I/O buffer) όλων των αρχείων
fgets()	Διαβάζει ένα σύνολο χαρακτήρων (αλφαριθμητικό) από αρχείο
fputs()	Γράφει ένα σύνολο χαρακτήρων (αλφαριθμητικό) στο αρχείο
rewind()	Επιστρέφει το δείκτη θέσης στην αρχή (στο byte 0)
fseek()	Τοποθετεί το δείκτη θέσης σε συγκεκριμένη θέση μέσα στο αρχείο (π.χ. στο 125ο byte)
fread()	Διαβάζει από αρχείο έναν αριθμό τμημάτων από byte (π.χ. 10 τμήματα των 25 byte)
fwrite()	Γράφει στο αρχείο έναν αριθμό τμημάτων από byte (π.χ. 10 τμήματα των 25 byte)

Οι παραπάνω συναρτήσεις αναλύονται στη συνέχεια του κεφαλαίου.

Η αναφορά σε ένα αρχείο που έχει ήδη ανοιχτεί γίνεται μέσω ενός δείκτη τύπου **FILE**. Ο τύπος **FILE** ορίζεται στο **stdio.h** και είναι μια εσωτερική δομή την οποία χρησιμοποιεί η C για το χειρισμό των αρχείων της. Ένας δείκτης σε μια τέτοια δομή ορίζεται με τον επόμενο τρόπο:

```
FILE *fp;
```

Στην ανάλυση της **fopen()** που ακολουθεί γίνεται κατανοητό το νόημα και η χρήση ενός τέτοιου δείκτη.

 Δεν πρέπει να ξεχνάμε να συμπεριλαμβάνουμε (με **include**) το **stdio.h** σε όποιο πρόγραμμα χρησιμοποιεί αρχεία.

## Ανοιγμα/κλείσιμο αρχείου

### fopen()

Η **fopen()** εξυπηρετεί δύο σκοπούς: Αφενός ανοίγει ένα αρχείο και αφετέρου επιστρέφει ένα δείκτη τύπου FILE, ο οποίος προσδιορίζει το συγκεκριμένο αρχείο.

Το συντακτικό της είναι το εξής:

**FILE \*fopen(char \*όνομα\_αρχείου, char \*τρόπος)**

Η **fopen()** επιστρέφει ένα δείκτη τύπου FILE (ο τύπος αυτός ορίζεται στο **stdio.h**).

Ο γενικός τρόπος χρήσης της **fopen()** είναι ο ακόλουθος:

```
FILE *fp;
```

```
fp = fopen(όνομα.αρχείου, τρόπος);
```

όπου **όνομα.αρχείου** είναι ένα σύνολο χαρακτήρων με το όνομα του αρχείου που σκοπεύουμε να ανοίξουμε και **τρόπος** ένα σύνολο χαρακτήρων που καθορίζει τον τρόπο με τον οποίο θα ανοιχτεί το συγκεκριμένο αρχείο. Οι διάφοροι τρόποι αναφέρονται στον επόμενο πίνακα.

Τρόπος	Λειτουργία	Φορά	Θέση
"r"	Ανοίγει ένα αρχείο κειμένου για ανάγνωση	↔	↗📄
"w"	Δημιουργεί ένα αρχείο κειμένου για εγγραφή	⇒	↗📄
"a"	Προσθέτει στο τέλος ενός αρχείου κειμένου	⇒⇒	↘📄
"rb"	Ανοίγει ένα δυαδικό αρχείο για ανάγνωση	↔	↗📄
"wb"	Δημιουργεί ένα δυαδικό αρχείο για εγγραφή	⇒	↗📄
"ab"	Προσθέτει στο τέλος ενός δυαδικού αρχείου	⇒⇒	↘📄
"r+"	Ανοίγει ένα αρχείο κειμένου για ανάγνωση/εγγραφή	↔	↗📄

Τρόπος	Λειτουργία	Φορά	Θέση
"w+"	Δημιουργεί ένα αρχείο κειμένου για ανάγνωση/εγγραφή	↔	↗ □
"a+"	Ανοίγει ένα αρχείο κειμένου για ανάγνωση/εγγραφή	↔	↘ □
"rb+"	Ανοίγει ένα δυαδικό αρχείο για ανάγνωση/εγγραφή	↔	↗ □
"wb+"	Δημιουργεί ένα δυαδικό αρχείο για ανάγνωση/εγγραφή	↔	↗ □
"ab+"	Ανοίγει ένα δυαδικό αρχείο για ανάγνωση/εγγραφή	↔	↘ □

👉 Η στήλη **φορά** δείχνει την κατεύθυνση των δεδομένων (από ή προς το αρχείο ή και τα δύο). Τα δύο βέλη δείχνουν προσθήκη στο τέλος του αρχείου.

👉 Η στήλη **θέση** δείχνει την αρχική θέση του δείκτη θέσης του αρχείου —στην αρχή (↗ □) ή στο τέλος του (↘ □).

Η `fopen()` επιστρέφει ένα δείκτη τύπου `FILE` ο οποίος πρέπει να αποδοθεί σε μια αντίστοιχη μεταβλητή του ίδιου τύπου. Αν για οποιονδήποτε λόγο η `fopen()` δεν μπορέσει να ανοίξει το αρχείο, θα επιστρέψει ένα δείκτη `NULL`.

```
FILE *fp;
```

```
fp=fopen("data", "r");
```

Η προηγούμενη πρόταση ανοίγει ένα αρχείο κειμένου με όνομα `data` για ανάγνωση δεδομένων.

Ο κλασικός τρόπος με τον οποίο συνηθίζεται να ανοίγουμε ένα αρχείο φαίνεται στον επόμενο κώδικα:


```
FILE *fp;
```

```
.....  
.....
```

```
if ((fp=fopen("data", "r")) == NULL)
```





```
{
    puts("Πρόβλημα στο άνοιγμα του αρχείου");
    exit(2);
}
```

 Όταν η C διαβάζει από ένα αρχείο κειμένου (text) και εντοπίζει τους δύο χαρακτήρες αλλαγής γραμμής (το χαρακτήρα ASCII 10 και το χαρακτήρα ASCII 13), τους θεωρεί ως ένα χαρακτήρα, το χαρακτήρα "new-line" (\n) της C.

Το αντίθετο γίνεται στην περίπτωση που η C γράφει σε ένα αρχείο κειμένου. Τότε ο χαρακτήρας "new-line" (\n) μετατρέπεται στους δύο χαρακτήρες αλλαγής γραμμής (το χαρακτήρα ASCII 10 και το χαρακτήρα ASCII 13).

Στην περίπτωση που χειρίζεται ένα δυαδικό (binary) αρχείο τότε δεν γίνεται καμιά τέτοια μετατροπή.

 Από τη στιγμή που θα ανοίξουμε ένα αρχείο με την **fopen()**, όλες οι υπόλοιπες συναρτήσεις αναφέρονται στο αρχείο χρησιμοποιώντας τον δείκτη που επέστρεψε η **fopen()**.

 Μπορούμε να έχουμε ταυτόχρονα ανοιχτά περισσότερα από ένα αρχεία. Κάθε αρχείο προσδιορίζεται από το δικό του δείκτη.

```
FILE *fp1, *fp2;
fp1 = fopen("DATA-IN", "r");
fp2 = fopen("APOT", "w");
```

## fclose()


Η **fclose()** κλείνει ένα αρχείο το οποίο έχει ανοιχτεί με την **fopen()**. Το συντακτικό της είναι:

**int fclose(FILE \*fp)**

Η **fclose()** γράφει στο αρχείο και όλα τα δεδομένα που βρίσκονται σε περιοχές προσωρινής αποθήκευσης (buffer) πριν κλείσει το αρχείο. Ο γενικός τρόπος χρήσης της **fclose()** είναι:

```
fclose(fp);
```

όπου **fp** ο δείκτης τύπου **FILE** του αρχείου που θέλουμε να κλείσουμε.

 Η συνάρτηση επιστρέφει 0 όταν είναι επιτυχής. Διαφορετικά, επιστρέφει μη μηδενική τιμή.

## ferror()

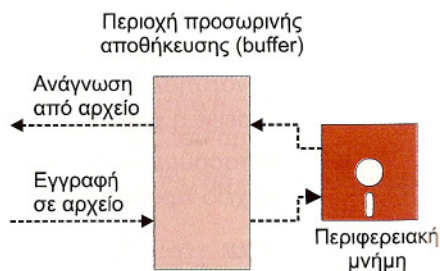
Η συνάρτηση αυτή χρησιμοποιείται για να ελέγξουμε αν μια λειτουργία αρχείου (άλλη συνάρτηση με πρόσβαση στο αρχείο) παρήγαγε κάποιο λάθος. Το συντακτικό της είναι το εξής:

**int ferror(FILE \*fp)**

Επιστρέφει τιμή "αλήθεια" (1) αν έχει προηγηθεί κάποιο λάθος ή "ψέμα" (0) όταν δεν υπάρχει πρόβλημα.

## fflush()

Όταν γίνεται το διάβασμα ή η εγγραφή πληροφοριών από ή μέσα σε ένα αρχείο, χρησιμοποιείται πάντοτε μία περιοχή προσωρινής αποθήκευσης (buffer). Όταν π.χ. γράφουμε πληροφορίες μέσα σε ένα αρχείο, αυτές καταχωρίζονται πρώτα στην περιοχή προσωρινής αποθήκευσης και, όταν αυτή γεμίσει, τότε εγγράφονται πραγματικά στο αρχείο.




Μια ξαφνική διακοπή ρεύματος ή μια δυσλειτουργία του προγράμματος θα έχει αποτέλεσμα την απώλεια των πληροφοριών που υπάρχουν στην περιοχή προσωρινής αποθήκευσης και δεν έχουν ακόμα εγγραφεί στην περιφερειακή μνήμη. Για κάθε ανοιχτό αρχείο χρησιμοποιείται διαφορετικός χώρος για την προσωρινή αποθήκευση.

Η **fflush()** αναγκάζει τη C να αδειάσει την περιοχή προσωρινής αποθήκευσης για κάθε συγκεκριμένο αρχείο και να γράψει τις πληροφορίες στο συγκεκριμένο αρχείο. Το συντακτικό της είναι:

**int fflush(FILE \*fp)**

όπου **fp** ο δείκτης τύπου FILE του αρχείου, την περιοχή προσωρινής αποθήκευσης του οποίου θέλουμε να αδειάσουμε.

 Η συνάρτηση επιστρέφει 0 όταν είναι επιτυχής, διαφορετικά επιστρέφει μη μηδενική τιμή.



## **fflush()**

Η **fflush()** αδειάζει τις περιοχές προσωρινής αποθήκευσης σε όλα τα ανοιχτά αρχεία. Το συντακτικό της είναι:

**int fflush()**

## **Σειριακή προσπάθεια αρχείων**

### **fputc()**

Η συνάρτηση **fputc()** γράφει ένα χαρακτήρα μέσα σε ένα αρχείο. Το συντακτικό της είναι:

**int fputc(int ch, FILE \*fp)**

όπου **ch** ο χαρακτήρας που θα γραφεί στο αρχείο.

### **fgetc()**

Η συνάρτηση **fgetc()** διαβάζει ένα χαρακτήρα από ένα αρχείο. Το συντακτικό της είναι:

**int fgetc(FILE \*fp)**

Η συνάρτηση επιστρέφει ένα χαρακτήρα "end-of-file" ("τέλος αρχείου") όταν φτάσει στο τέλος του αρχείου. Στο **stdio.h** έχει ανατεθεί η τιμή του χαρακτήρα "end-of-file" στο σύμβολο EOF. Επομένως, για να διαβάσουμε ένα αρχείο κειμένου χαρακτήρα-χαρακτήρα μέχρι το τέλος του πρέπει να χρησιμοποιήσουμε τον επόμενο κώδικα:

```
ch=fgetc (fp) ;
```

```
while (ch!=EOF)
```

```
{
```

```
    ch=fgetc (fp) ;
```

```
}
```

Αν το αρχείο είναι δυαδικό (binary), ο χαρακτήρας EOF μπορεί να παρουσιαστεί χωρίς να φτάσουμε στο τέλος του αρχείου. Τότε, το ανωτέρω πρόγραμμα

θα σταματήσει να διαβάζει πριν φτάσουμε στο φυσικό τέλος του αρχείου. Σε αυτή την περίπτωση θα πρέπει να χρησιμοποιήσουμε τη συνάρτηση `feof()`.

## fprintf()

Η συνάρτηση `fprintf()` συμπεριφέρεται όπως η `printf()`, με τη διαφορά ότι αντί να εμφανίσει τα αποτελέσματά της στην οθόνη, τα γράφει σε αρχείο. Το συντακτικό της είναι:

**int fprintf(FILE \*fp, char \*αλφ\_μορφ, παρ1, παρ2, ....)**

Τα αποτελέσματα της `fprintf()` μεταβιβάζονται στο αρχείο που καθορίζεται από το δείκτη `fp`. Η `fprintf()` επιστρέφει τον αριθμό των χαρακτήρων που έγραψε στο αρχείο.

Το επόμενο τμήμα κώδικα ανοίγει ένα αρχείο κειμένου με όνομα `data` για εγγραφή δεδομένων και στη συνέχεια καταχωρίζει μέσα στο αρχείο 100 τυχαίους αριθμούς.

```
FILE *fp;
fp=fopen("data", "w");
for (i=1; i<=100; i++)
    fprintf(fp, "%d\n", rand());
fclose(fp);
```

## fscanf()

Η συνάρτηση `fscanf()` συμπεριφέρεται όπως η `scanf()`, με τη διαφορά ότι διαβάζει μορφοποιημένα δεδομένα, όχι από το πληκτρολόγιο αλλά από αρχείο. Το συντακτικό της είναι:

**int fscanf(FILE \*fp, char \*αλφ\_μορφ, δνση1, δνση2, ....)**

όπου `δνση1`, `δνση2`,... οι διευθύνσεις μεταβλητών στις οποίες θα γίνει η καταχώριση. Η `fscanf()` επιστρέφει τον αριθμό των δεδομένων που διάβασε από το αρχείο.

Ο επόμενος κώδικας διαβάζει από το αρχείο `data` 100 αριθμούς και τους εμφανίζει στην οθόνη:

```

FILE *fp;
int ar;
fp=fopen("data", "r");
for(i=1;i<=100;i++)
{
    fscanf(fp, "%d", &ar);
    printf("%d\n", a);
}
fclose(fp);

```

Στην περίπτωση που το αρχείο περιέχει λιγότερους αριθμούς, θα πάρουμε κάποιο μήνυμα λάθους. Αυτό το πρόβλημα μπορεί να λυθεί με κατάλληλη χρήση της συνάρτησης `feof()`.

## feof()

Η συνάρτηση **feof()** επιστρέφει τιμή "αλήθεια" (1) όταν έχουμε φτάσει στο τέλος ενός αρχείου και "ψέμα" (0) σε κάθε άλλη περίπτωση. Το συντακτικό της είναι:

**int feof(FILE \*fp)**

Το επόμενο τμήμα προγράμματος υπολογίζει τον αριθμό των χαρακτήρων ενός αρχείου.

```

int cnt=0;
char ch;
while(!feof(fp))
{
    ch=fgetc(fp);
    cnt++;
}
printf("Το αρχείο έχει %d χαρακτήρες", cnt);

```

Ο επόμενος κώδικας διαβάζει από το αρχείο `data` το πολύ 100 αριθμούς και τους εμφανίζει στην οθόνη. Στην περίπτωση που το αρχείο περιέχει λιγότερους αριθμούς διαβάζει μόνον όσους περιέχει, χωρίς κανένα πρόβλημα.



```
FILE *fp;
int ar;
fp=fopen("data","r");
for(i=1;i<=100;i++)
{
    fscanf(fp,"%d",&ar);
    printf("%d\n",a);
    if (feof(fp)==1) break;
}
fclose(fp);
```

Η feof() επιστρέφει 1 όταν έχουμε φτάσει στο τέλος του αρχείου. Στην περίπτωση αυτή η επαναληπτική διαδικασία τερματίζεται.

## fgets()

Η **fgets()** λειτουργεί σχεδόν όπως η **gets()**, με τη διαφορά ότι διαβάζει ένα σύνολο χαρακτήρων από ένα αρχείο και όχι από το πληκτρολόγιο. Το συντακτικό της είναι:

**fgets(char \*str,int mikos,FILE \*fp)**

όπου ο δείκτης **str** δείχνει την πρώτη από τις θέσεις μνήμης όπου θα αποθηκευτούν οι χαρακτήρες (και είναι συνήθως ένας πίνακας **char**) και ο δείκτης **fp** καθορίζει το αρχείο από το οποίο θα διαβάσει τους χαρακτήρες. Η συνάρτηση διαβάζει χαρακτήρες μέχρι να διαβάσει είτε ένα χαρακτήρα αλλαγής γραμμής είτε πλήθος χαρακτήρων όσο η τιμή της παράστασης **mikos**.

Ο ακόλουθος κώδικας διαβάζει από ένα αρχείο κειμένου μία-μία γραμμή και την εμφανίζει στην οθόνη. Στο τέλος εμφανίζει το πλήθος των γραμμών που διάβασε.

```
FILE *fp;
int ar=0;
char grammi[80];
fp=fopen("data","r");
while(!feof(fp))
{
    fgets(grammi,80,fp);
```

Ενόςω δεν έχουμε φτάσει στο τέλος του αρχείου, διαβάζει μια-μια γραμμή και την εμφανίζει στην οθόνη.

```

    puts(grammi);
    ar++;
}
printf("Το αρχείο περιέχει %d γραμμές\n", ar);
fclose(fp);

```

Η μεταβλητή `ar` μετράει τις γραμμές που διαβάστηκαν.

## fputs()

Η `fputs()` λειτουργεί όπως η `puts()`, με τη διαφορά ότι γράφει ένα σύνολο χαρακτήρων σε ένα αρχείο. Το συντακτικό της είναι:

**fputs(char \*str, FILE \*fp)**

όπου ο δείκτης `str` δείχνει την πρώτη από τις θέσεις του συνόλου χαρακτήρων που θα γραφεί (και είναι συνήθως ένας πίνακας `char`) και ο δείκτης `fp` καθορίζει το αρχείο στο οποίο θα γραφούν οι χαρακτήρες.

Ο επόμενος κώδικας διαβάζει από ένα αρχείο κειμένου `in` μία-μία γραμμή και αν αυτή η γραμμή αρχίζει από 'A', τη γράφει σε ένα άλλο αρχείο `out`. Τέλος εμφανίζει το πλήθος των γραμμών που έγραψε μέσα στο αρχείο `out`.

```

FILE *fp1, *fp2;
char grammi[80];
int ar=0;
fp1=fopen("in", "r");
fp2=fopen("out", "w");
while(!feof(fp1))
{
    fgets(grammi, 80, fp1);
    if(grammi[0]=='A')
    {
        fputs(grammi, fp2);
        ar++;
    }
}
printf("Στο αρχείο out γράφηκαν %d γραμμές\n", ar);
fclose(fp1);
fclose(fp2);

```

Διαβάζει 80 χαρακτήρες από το αρχείο και τους αποθηκεύει στον πίνακα `grammi`.

Ελέγχει αν ο πρώτος χαρακτήρας της γραμμής είναι 'A'.

Αν η γραμμή αρχίζει από 'A' τη γράφει στο αρχείο `out`.

Η μεταβλητή `ar` μετράει τις γραμμές που γράφονται στο αρχείο `out`.

## rewind()

Η `rewind()` τοποθετεί το δείκτη θέσης αρχείου στην αρχή του αρχείου:

`viod rewind(FILE *fp)`

## Τυχαία προσπέλαση αρχείων

### fseek()

Η `fseek()` τοποθετεί το δείκτη του αρχείου σε κάποιο σημείο του αρχείου. Το συντακτικό της είναι:

`int fseek(FILE *fp, long int apostasi, int thesi)`

όπου ο `fp` καθορίζει το αρχείο και η παράμετρος `apostasi` δηλώνει την απόσταση (σε byte) από τη θέση που καθορίζει η παράμετρος `thesi`. Η παράμετρος `thesi` μπορεί να πάρει τις παρακάτω τιμές:

	Τιμή παραμέτρου	Θέση
0	SEEK_SET	αρχή αρχείου
1	SEEK_CUR	τρέχουσα θέση
2	SEEK_END	τέλος αρχείου

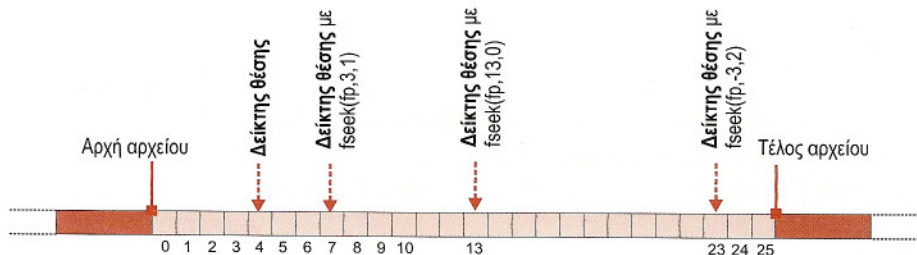
Οι `SEEK_SET`, `SEEK_CUR`, και `SEEK_END` ορίζονται στο `stdio.h` αλλά μπορούν να χρησιμοποιηθούν και οι τιμές 0, 1, και 2 αντίστοιχα.

☞ Όταν ως θέση αναφοράς ορίζεται το τέλος του αρχείου, τότε στην παράμετρο `apostasi` έχουν νόημα μόνο αρνητικές τιμές, αφού η παράμετρος δηλώνει πλέον την απόσταση σε byte από το τέλος του αρχείου.

☞ Η `fseek()` σε συνδυασμό με τις συναρτήσεις `fread()` και `fwrite()` δίνουν στη C τη δυνατότητα να επιτυγχάνει την τυχαία προσπέλαση σε ένα αρχείο.

Στο επόμενο παράδειγμα ο δείκτης θέσης δείχνει το τέταρτο byte του αρχείου. Παρατηρούμε τον τρόπο με τον οποίο μετακινείται ο δείκτης θέσης ανάλογα με την κλήση της `fseek()` με διαφορετικές κάθε φορά παραμέτρους.





## fread()

Η **fread()** διαβάζει από κάποιο αρχείο έναν αριθμό από τμήματα συγκεκριμένου αριθμού byte και τα βάζει σε κάποια περιοχή προσωρινής αποθήκευσης (buffer).

**int fread(void \*buffer, int arbytes, int fores, FILE \*fp)**

Ο **buffer** είναι ένας δείκτης που δείχνει την περιοχή της μνήμης στην οποία θα αποθηκευτούν τα byte που θα διαβαστούν. Η δεύτερη παράμετρος (**arbytes**) δείχνει τον αριθμό των byte κάθε τμήματος και η παράμετρος **fores** το πλήθος των τμημάτων που θα διαβαστούν. Ο δείκτης **fp** καθορίζει το αρχείο.

👉 Η ανάγνωση των byte ξεκινάει από το σημείο του δείκτη θέσης του αρχείου.

Στο επόμενο παράδειγμα η **fread()** θα διαβάσει μία φορά ένα τμήμα από 1000 byte, ξεκινώντας από το 100ό byte του αρχείου, και θα τα αποθηκεύσει στον πίνακα **grammata**.

```
main()
{
    FILE *fp;
    char grammata[1000];
    fp=fopen("TEST", "rb");
    fseek(fp, 100, 0);
    fread(grammata, 1000, 1, fp);
    fclose(fp);
}
```

Η **fseek** τοποθετεί το δείκτη θέσης του αρχείου στο 100ό του byte.

Το αποτέλεσμα θα ήταν ίδιο αν η **fread()** δινόταν με άλλες παραμέτρους:

`fread(grammata, 1, 1000, fp);`     ή

```
fread(grammata, 500, 2, fp); ή  
fread(grammata, 20, 50, fp);
```


Και στις τρεις προηγούμενες περιπτώσεις θα διαβαστούν 1000 byte και θα καταχωριστούν στον πίνακα **grammata**.

## fwrite()

Η **fwrite()** γράφει έναν αριθμό από τμήματα συγκεκριμένου αριθμού byte από κάποια περιοχή προσωρινής αποθήκευσης (buffer) και τα βάζει σε ένα αρχείο.

**int fwrite(void \*buffer, int arbytes, int fores, FILE \*fp)**

Ο **buffer** είναι ένας δείκτης που δείχνει την περιοχή της μνήμης από την οποία θα διαβαστούν τα bytes που θα γραφούν στο αρχείο. Η δεύτερη παράμετρος (**arbytes**) δείχνει τον αριθμό των byte του κάθε τμήματος και η παράμετρος **fores** το πλήθος των τμημάτων που θα διαβαστούν. Ο δείκτης **fp** καθορίζει το αρχείο.

 Η εγγραφή των byte ξεκινάει από το σημείο του δείκτη θέσης του αρχείου.

Στο επόμενο παράδειγμα η **fwrite()** θα γράψει 100 τμήματα του ενός byte, ξεκινώντας από το 50ό byte του αρχείου. Τα byte θα προέρχονται από τον πίνακα χαρακτήρων **grammata**. Θα γραφούν τα πρώτα 100 byte του πίνακα.

```
main()  
{  
    FILE *fp;  
    char grammata[1000];  
    fp=fopen("TEST", "wb");  
    fseek(fp, 50, 0);  
    fwrite(grammata, 1, 100, fp);  
    fclose(fp);  
}
```

Το αποτέλεσμα θα ήταν ίδιο αν η **fwrite()** δινόταν με τις εξής παραμέτρους:



```
fwrite(grammata, 100, 1, fp);  
ή
```

```
fwrite(grammata, 50, 2, fp);
```

ή

```
fwrite(grammata, 20, 5, fp);
```

Και στις τρεις προηγούμενες περιπτώσεις θα γραφούν τα πρώτα 100 byte από τον πίνακα **grammata**.

-  Όταν χρησιμοποιούμε την **fseek()**, την **fread()**, και την **fwrite()** για τυχαία προσπέλαση, πρέπει να ανοίγουμε το αρχείο ως δυαδικό (binary).
-  Συνήθως η **fread()** και η **fwrite()** χρησιμοποιούνται με δομές ή πίνακες δομών. Στα παραδείγματα που ακολουθούν φαίνεται η χρήση των **fread()** και **fwrite()** σε συνδυασμό με τις δομές.

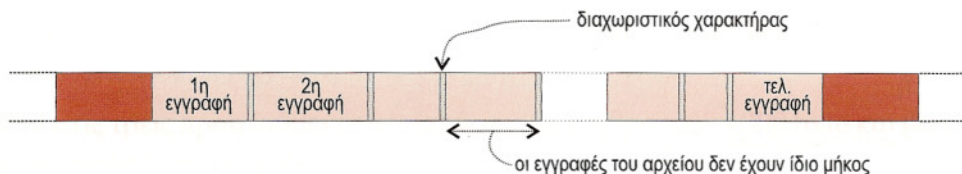
## Η λογική της τυχαίας προσπέλασης

Συνήθως σε ένα αρχείο καταχωρίζουμε σύνολα ομοειδών πληροφοριών τα οποία καλούνται **εγγραφές**. Μια εγγραφή θα μπορούσε π.χ. να αποτελείται από τα στοιχεία ενός ατόμου στον τηλεφωνικό κατάλογο (όνομα, επώνυμο, τηλέφωνο κ.λπ.). Ένα αρχείο μπορεί να περιέχει χιλιάδες τέτοιες εγγραφές. Κάθε εγγραφή περιέχει επιμέρους πληροφορίες όπως το όνομα, το τηλέφωνο, κ.λπ. Οι πληροφορίες αυτές καλούνται **πεδία**.

Ένα αρχείο μπορεί να περιέχει πολλές εγγραφές και κάθε εγγραφή μπορεί να περιέχει πολλά πεδία.

Όταν ένα αρχείο περιέχει εγγραφές διαφορετικού μήκους (μέγεθος σε byte), δεν είναι δυνατό να εντοπίσουμε άμεσα μια εγγραφή επειδή δεν γνωρίζουμε την ακριβή θέση της μέσα στο αρχείο. Ο μόνος τρόπος εντοπισμού μιας τέτοιας εγγραφής είναι ο σειριακός. Δηλαδή, να διαβάζουμε μία-μία τις εγγραφές μέχρι να φτάσουμε στην εγγραφή που θέλουμε. Στην περίπτωση της σειριακής προσπέλασης οι εγγραφές (αλλά και τα πεδία μεταξύ τους), χωρίζονται με συγκεκριμένους χαρακτήρες όπως ο χαρακτήρας του διαστήματος και ο χαρακτήρας αλλαγής γραμμής.

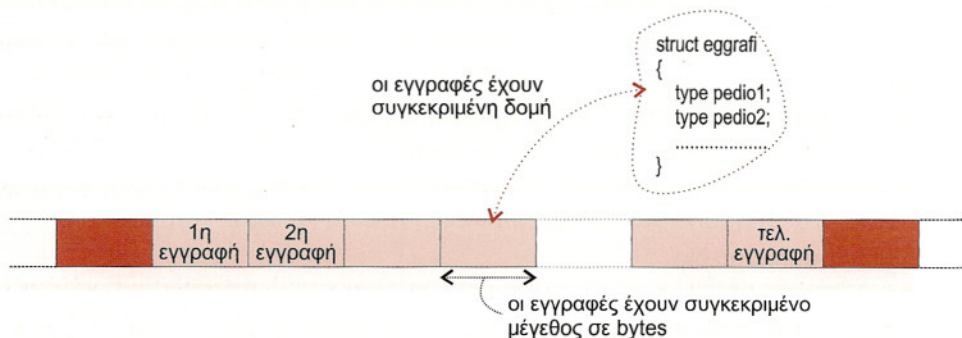




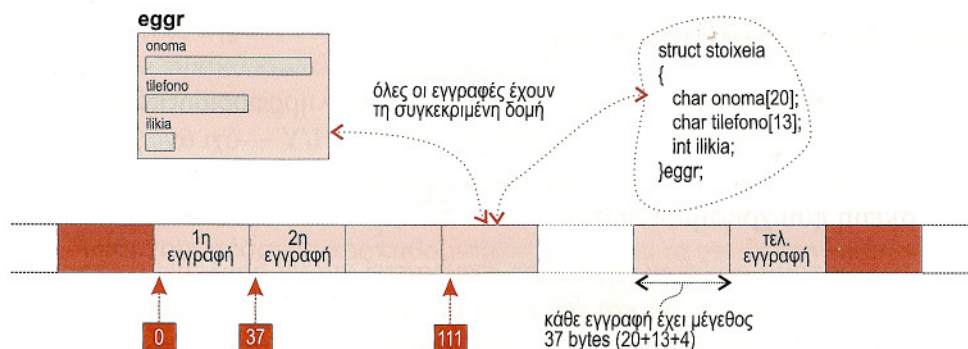
Όταν το αρχείο περιέχει εγγραφές ίδιου μήκους (μέγεθος σε byte), είναι δυνατό να εντοπίζουμε άμεσα μια εγγραφή μέσα στο αρχείο επειδή γνωρίζουμε την ακριβή της θέση. Στην περίπτωση αυτή οι εγγραφές (αλλά και τα πεδία μεταξύ τους) ακολουθούν η μία την άλλη χωρίς να μεσολαβούν διαχωριστικοί χαρακτήρες.

Τα αρχεία που αποτελούνται από εγγραφές ίδιου μήκους επιτρέπουν τυχαία προσπέλαση στα δεδομένα που περιέχουν.

Σε αρχεία αυτού του τύπου, οι εγγραφές έχουν συγκεκριμένη δομή η οποία μπορεί να αναπαρασταθεί με έναν τύπο δεδομένων **struct**:



Όλες οι εγγραφές έχουν το ίδιο συγκεκριμένο μήκος. Στο επόμενο παράδειγμα, κάθε εγγραφή έχει μέγεθος 37 byte. Γνωρίζοντας το μέγεθος της εγγραφής μπορούμε να υπολογίσουμε πού ακριβώς θα βρίσκεται η κάθε μία. Αν γνωρίζουμε π.χ. ότι η τέταρτη εγγραφή ξεκινάει από το byte 111 ( $3 \times 37$ ), μπορούμε με την `fseek()` να μετακινήσουμε τον δείκτη θέσης του αρχείου στην αρχή οποιασδήποτε εγγραφής.



Στην πράξη συνηθίζεται η χρήση μιας μεταβλητής ή ενός πίνακα, του ίδιου τύπου δομής με τη δομή των εγγραφών του αρχείου, για την ανταλλαγή δεδομένων από και προς το αρχείο.

Αν υποθέσουμε το παράδειγμα του προηγούμενου σχήματος, ο επόμενος κώδικας διαβάζει την τέταρτη εγγραφή από το αρχείο, την καταχωρίζει στη μεταβλητή δομής `eggr`, και εμφανίζει τα περιεχόμενά της.

```
fseek(fp, 111, 0);
fread(&eggr, 37, 1, fp);
/* Εμφάνισε τα πεδία της eggr */
printf("Όνομα: %s\n", eggr.onoma);
printf("Τηλέφωνο: %s\n", eggr.telefono);
printf("Ηλικία: %d\n", eggr.ilikia);
```

Η `fseek()` τοποθετεί το δείκτη θέσης στην αρχή της τέταρτης εγγραφής

Η `fread()` διαβάζει ένα τμήμα μήκους 37 byte και το καταχωρίζει στη μνήμη ξεκινώντας από τη διεύθυνση `&eggr`, δηλαδή το καταχωρίζει στη μεταβλητή `eggr`.

Ο επόμενος κώδικας ζητάει να πληκτρολογήσουμε στοιχεία για Όνομα, Τηλέφωνο, Ηλικία, τα καταχωρίζει στην μεταβλητή δομής `eggr` και τα γράφει στην τέταρτη εγγραφή του αρχείου, διαγράφοντας τα παλαιά της περιεχόμενα.

```
printf("Δώσε όνομα:");
gets(eggr.onoma);
printf("Δώσε τηλέφωνο:");
gets(eggr.telefono);
printf("Δώσε ηλικία:");
scanf("%d", &eggr.ilikia);
fseek(fp, 111, 0);
fwrite(&eggr, 37, 1, fp);
```

Στα πεδία της μεταβλητής `eggr` καταχωρίζονται τα στοιχεία που πληκτρολογούνται.

Η `fseek()` τοποθετεί το δείκτη θέσης στην αρχή της τέταρτης εγγραφής

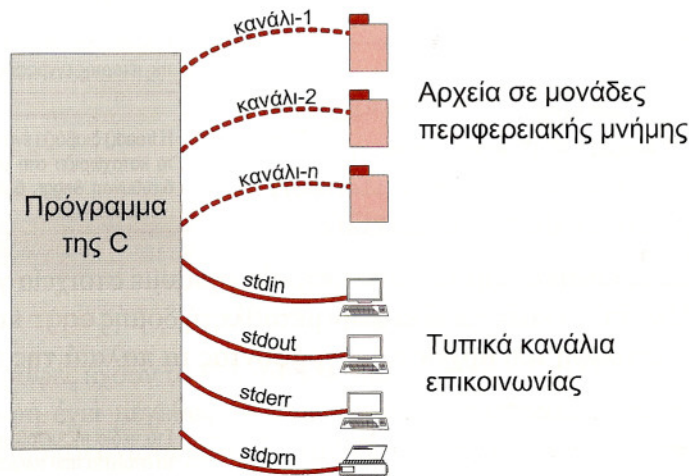
Η `fwrite()` γράφει στο αρχείο ένα τμήμα από 37 byte, αρχίζοντας να το διαβάζει από τη διεύθυνση `&eggr`, δηλαδή από τη μεταβλητή `eggr`.

## Κανάλια επικοινωνίας

Ένα κανάλι (*ρεύμα* – *stream*) είναι μια σειρά πληροφοριών που προέρχονται ή καταλήγουν σε μια περιφερειακή μονάδα του Η/Υ —όχι αναγκαστικά αρχείο. Η C παρέχει στους προγραμματιστές μια διασύνδεση ανεξάρτητη από τη συσκευή που χρησιμοποιείται.

Το σύστημα αρχείων της C έχει σχεδιαστεί για να δουλεύει με τη γενικότερη έννοια του καναλιού και όχι μόνο με την περιορισμένη έννοια του αρχείου. Έτσι οι ίδιες συναρτήσεις της C που χρησιμοποιούνται για τον χειρισμό αρχείων, χρησιμοποιούνται και για τον χειρισμό καναλιών που συνδέουν άλλες περιφερειακές συσκευές (οθόνη, πληκτρολόγιο, εκτυπωτής, ...) και όχι αναγκαστικά περιφερειακές μονάδες μνήμης (δίσκους και δισκέτες).

Το σχήμα που ακολουθεί δίνει μια εποπτική εικόνα του τρόπου με τον οποίο ένα πρόγραμμα της C επικοινωνεί με τις περιφερειακές μονάδες του Η/Υ.



Για να "συνδέσουμε" ένα κανάλι με ένα αρχείο χρησιμοποιούμε την συνάρτηση `fopen()`. Η `fopen()` επιστρέφει ένα δείκτη τύπου `FILE`, ο οποίος προσδιορίζει το συγκεκριμένο κανάλι επικοινωνίας.



Με την έναρξη της εκτέλεσης ενός προγράμματος της C "ανοίγουν" αυτόματα τα λεγόμενα τυπικά κανάλια επικοινωνίας, τα οποία συνδέουν τις συνήθεις περιφερειακές συσκευές.

## Τυπικά κανάλια επικοινωνίας

Στη C υπάρχουν τέσσερα προκαθορισμένα κανάλια τα οποία συνδέουν τις τυπικές (*standard*) συσκευές ενός H/Y. Τα κανάλια αυτά ορίζονται στο `stdio.h` και είναι:

**stdin** Τυπική έξοδος (οθόνη)

**stdout** Τυπική είσοδος (πληκτρολόγιο)

**stderr** Τυπική έξοδος για μηνύματα λάθους (οθόνη)


**stdprn** Παράλληλη θύρα (πρώτος εκτυπωτής)

Τα παραπάνω κανάλια είναι ουσιαστικά δείκτες τύπου `FILE` και μπορούν να χρησιμοποιηθούν από τις συναρτήσεις χειρισμού αρχείων σαν κανονικοί δείκτες προς αρχεία.

Τα επόμενα κανάλια χρησιμοποιούν ως τυπική και προεπιλεγμένη (*default*) συσκευή την κονσόλα (οθόνη για έξοδο και πληκτρολόγιο για είσοδο).

**stdin** Τυπική έξοδος (οθόνη)

**stdout** Τυπική είσοδος (πληκτρολόγιο)

 Πολλά λειτουργικά συστήματα επιτρέπουν την ανακατεύθυνση των `stdin` και `stdout` σε άλλες περιφερειακές συσκευές.

## Παράμετροι γραμμής εντολής

Μέχρι τώρα γνωρίζαμε ότι οι συναρτήσεις της C μπορούν να έχουν παραμέτρους μέσω των οποίων η συνάρτηση που τις καλεί μπορεί να μεταβιβάσει κάποιες πληροφορίες.

Τι γίνεται με τη συνάρτηση `main()`;

Μπορεί να έχει παραμέτρους;

## Πώς θα μεταβιβάσουμε τιμές σε αυτές τις παραμέτρους;

Η `main()` μπορεί να έχει δύο (και μόνο δύο) **συγκεκριμένες** παραμέτρους, οι οποίες παίρνουν τιμές από τη γραμμή εκτέλεσης του προγράμματος στο περιβάλλον του λειτουργικού συστήματος.

Οι παράμετροι αυτές είναι οι `argc` και `argv[]`:

**`main(int argc, char *argv[])`**

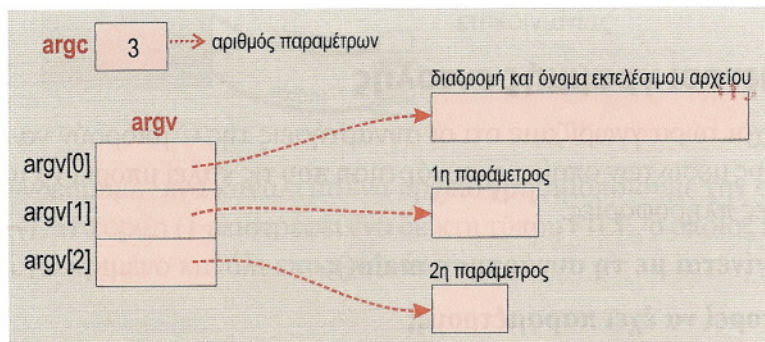
Η `argc` μεταβιβάζει στη `main()` τον αριθμό των παραμέτρων με τις οποίες καλείται το πρόγραμμα. Η τιμή της `argc` είναι τουλάχιστον 1 δεδομένου ότι στις παραμέτρους υπολογίζεται και το όνομα του προγράμματος.

Η `argv[]` είναι ένας δείκτης σε πίνακα που περιέχει δείκτες σε `char`. Η πρώτη θέση μνήμης του πίνακα `argv[]` δείχνει σε ένα σύνολο χαρακτήρων το οποίο περιέχει την απόλυτη διαδρομή του εκτελέσιμου αρχείου του προγράμματος (π.χ. `c:\dev-cpp\test.exe`). Οι υπόλοιπες θέσεις μνήμης του `argv[]` δείχνουν σε σύνολα χαρακτήρων που περιέχουν τις παραμέτρους με τις οποίες έχει κληθεί από τη γραμμή εντολών (*command line*) το πρόγραμμα.

Έστω ότι γράφουμε ένα πρόγραμμα σε ένα αρχείο `test.c`, το οποίο το μεταγλωττίζουμε και προκύπτει ένα εκτελέσιμο αρχείο `test.exe`. Αν εκτελέσουμε το `test.exe` με δύο παραμέτρους, π.χ.

```
test arxeio 1234
```

τότε οι παράμετροι `argc` και `argv[]` της `main()` θα έχουν ως εξής:



Το επόμενο πρόγραμμα δείχνει με σαφήνεια τη χρήση των `argc` και `argv[]`. Εμφανίζει το πλήθος των παραμέτρων καθώς και τις παραμέτρους μία-μία.

```
main(int argc, char *argv[])
{
    int i;
    printf("έδωσες %d παραμέτρους\n", argc);
    for(i=0; i<argc; i++)
    {
        puts(argv[i]);
    }
}
```

## Παραδείγματα

- Π.1** Το επόμενο πρόγραμμα ζητάει λέξεις και τις καταχωρίζει σε ένα αρχείο κειμένου με όνομα `lexeis`. Το πρόγραμμα σταματάει όταν δοθεί ως λέξη το κενό.

```
main()
{
    char lex[40];
    FILE *fp;
    fp=fopen("lexeis", "w");
    while(1)
    {
        printf("Δώσε λέξη:");
        gets(lex);
        if(strcmp(lex, "")==0)
            break;
        fputs(lex, fp);
    }
    fclose(fp);
}
```

Αν αντί για λέξη δώσουμε κενό (πατήσουμε απλώς <ENTER>), η επαναληπτική διαδικασία διακόπτεται.



**Π.2** Υποθέτουμε ότι το αρχείο **DATA** περιέχει έναν άγνωστο αριθμό από ακέραιους αριθμούς. Το παρακάτω πρόγραμμα βρίσκει το μικρότερο και το μεγαλύτερο αριθμό.

```
main()
{
    int ar,min,max;
    FILE *fp;
    fp=fopen("DATA","r");
    fscanf(fp,"%d",&ar);
    min=max=ar;
    while(!feof(fp))
    {
        fscanf(fp,"%d",&ar);
        if(ar>max) max=ar;
        if(ar<min) min=ar;
    }
    fclose(fp);
    printf("min=%d max=%d\n",min,max);
}
```

Διαβάζουμε ξεχωριστά τον πρώτο αριθμό του αρχείου για να τον δώσουμε ως αρχική τιμή στις μεταβλητές min και max.

Διαβάζουμε έναν-έναν όλους τους υπόλοιπους αριθμούς

Αν ο αριθμός είναι μεγαλύτερος από το max, τότε καταχωρίζουμε στο max αυτόν τον αριθμό. Αν είναι μικρότερος από το min, τότε τον καταχωρίζουμε στο min.

**Π.3** Υποθέτουμε ότι έχουμε έναν πίνακα δομών με τη διπλανή γραμμογράφηση. Στον πίνακα αυτό είναι καταχωρισμένα τα στοιχεία των 100 μαθητών ενός σχολείου. Το επόμενο τμήμα κώδικα, αποθηκεύει τα στοιχεία του πίνακα σε ένα αρχείο με όνομα **sxoleio**.

```
struct stoixeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];
```

```
.....
FILE *fp;
fp=fopen("sxoleio","wb");
fwrite(mathites,sizeof(struct stoixeia),100,fp);
fclose(fp);
```

Ας εξετάσουμε προσεκτικά την κλήση της `fwrite`:

```
fwrite(mathites, sizeof(struct stoixeia), 100, fp);
```

- Η παράμετρος **mathites** είναι ένας δείκτης στην αρχή του πίνακα δομών που περιέχει τα δεδομένα μας. Από αυτή τη θέση μνήμης θα αρχίσει η ανάγνωση των προς εγγραφή δεδομένων.
- Η παράσταση **sizeof(struct stoixeia)** αποδίδει το μέγεθος σε byte του τύπου δεδομένων **stoixeia**. Στο συγκεκριμένο παράδειγμα, το μέγεθος σε byte του τύπου **stoixeia** είναι 43 byte (30+5+4+4).
- Ο αριθμός **100** υποδεικνύει το πλήθος των τμημάτων μνήμης μεγέθους 43 byte που θα εγγραφούν στο αρχείο.
- Η παράμετρος **fp** προσδιορίζει το αρχείο.

**Π.4** Υποθέτουμε ότι έχουμε το αρχείο **sxoleio** που δημιουργήθηκε από το προηγούμενο παράδειγμα και το οποίο περιέχει τα στοιχεία των 100 μαθητών. Το επόμενο πρόγραμμα διαβάζει από το αρχείο τα στοιχεία των μαθητών και τα καταχωρίζει στον πίνακα **mathites**, τύπου δομής **stoixeia**.

```
struct stoixeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} mathites[100];

main()
{
    FILE *fp;
    fp=fopen("sxoleio", "rb");
    fread(mathites, sizeof(struct stoixeia), 100, fp);
    fclose(fp);
}
```

Η συνάρτηση `fread()` διαβάζει 100 τμήματα των 43 byte (`sizeof(struct stoixeia)`) και τα καταχωρίζει στο χώρο προσωρινής αποθήκευσης (buffer) που ξεκινάει από τον δείκτη `mathites`, πρακτικά δηλαδή στον πίνακα `mathites`.

- Π.5** Υποθέτουμε ότι έχουμε το αρχείο `sxoleio` που δημιουργήθηκε στο παράδειγμα Π3, το οποίο περιέχει τα στοιχεία μαθητών. Το επόμενο πρόγραμμα ζητάει από το χρήστη να πληκτρολογήσει το επώνυμο ενός μαθητή, εντοπίζει την εγγραφή του μαθητή μέσα στο αρχείο, και εμφανίζει τα υπόλοιπα στοιχεία του. Αν δεν εντοπίσει μαθητή με το επώνυμο που δόθηκε, εμφανίζει σχετικό μήνυμα.

```
struct stoixeia
{
    char eponymo[30];
    char taxi[5];
    float mesos_oros;
    int ilikia;
} m;

main()
{
    FILE *fp;
    char ep[30];
    int found=0;
    printf("Δώσε όνομα:");
    gets(ep);
    fp=fopen("sxoleio","rb");
    while(!feof(fp))
    {
        fread(&m,sizeof(struct stoixeia),1,fp);
        if(strcmp(m.eponymo,ep)==0)
        {
            found++;
            printf("Επώνυμο:%s\n",m.eponymo);
        }
    }
}
```

Δηλώνεται μια μεταβλητή, τύπου `stoixeia`, στην οποία θα καταχωρίζεται μια εγγραφή που θα διαβάζεται από το αρχείο.

Η `fread()` διαβάζει ένα τμήμα (μια εγγραφή) από 43 byte και το καταχωρίζει στη μεταβλητή `m`. Η `fread()` χρειάζεται τη διεύθυνση της `m`, η οποία αποδίδεται από το `&m`.

Ελέγχει αν το επώνυμο της εγγραφής που διαβάστηκε είναι το επώνυμο που αναζητούμε.



```

        printf("Τάξη:%s\n",m.taxi);
        printf("ΜΟ:%f\n",m.mesos_oros);
        printf("Ηλικία:%d\n",m.ilikia);

    }
}

fclose(fp);
if (found==0)
    printf("Δε βρέθηκαν μαθητές με αυτό το επώνυμο\n");
else
    printf("Βρέθηκαν %d μαθητές\n",found);
}

```

**Π.6** Το επόμενο πρόγραμμα χρησιμοποιείται για την κρυπτογράφηση και αποκρυπτογράφηση ενός αρχείου οποιουδήποτε τύπου.

Η κρυπτογράφηση βασίζεται στον τελεστή bitwise "συμπλήρωμα" (~). Ο τελεστής αυτός αντιστρέφει τα bit ενός byte. Για παράδειγμα, αν ένα byte έχει τιμή 10110100, το συμπλήρωμά του είναι 01001011. Εάν τώρα στο αποτέλεσμα εφαρμόσουμε πάλι τον τελεστή του συμπληρώματος έχουμε σαν αποτέλεσμα το αρχικό byte.

$$\begin{aligned} \sim 10110100 &= 01001011 \\ \sim 01001011 &= 10110100 \end{aligned}$$

Στον επόμενο κώδικα, διαβάζουμε ένα-ένα byte από το αρχείο εισόδου υπολογίζουμε το συμπλήρωμά του, και το γράφουμε στο αρχείο εξόδου. Και τα δύο αρχεία τα ανοίγουμε ως δυαδικά.

```

main()
{
    FILE *fp1,*fp2;
    char file_in[30],file_out[30],ch;
    printf("Δώσε όνομα αρχείου εισόδου:");
    gets(file_in);
    printf("Δώσε όνομα αρχείου εξόδου:");
    gets(file_out);
    fp1=fopen(file_in,"rb");

```

Άνοιγμα του αρχείου εισόδου σε δυαδική μορφή για ανάγνωση δεδομένων.

```

if (fp1==NULL)
{
    printf("Πρόβλημα στο άνοιγμα αρχείου εισόδου");
    exit(2);
}
fp2=fopen(file_out,"wb");
if (fp2==NULL)
{
    printf("Πρόβλημα στο άνοιγμα αρχείου εξόδου");
    exit(2);
}
while(!feof(fp1))
{
    fread(&ch,1,1,fp1);
    ch=~ch;
    fwrite(&ch,1,1,fp2);
}
fclose(fp1);
fclose(fp2);
}
    
```

Άνοιγμα του αρχείου εισόδου σε δυαδική μορφή για ανάγνωση δεδομένων.

Διαβάζει ένα byte και το καταχωρίζει μέσα στη ch (η fread() χρειάζεται την διεύθυνση της ch).

Το byte που διαβάστηκε (ch) αντικαθίσταται από το συμπλήρωμά του.

Εγγραφή ενός byte το οποίο διαβάζεται από την ch στο αρχείο εξόδου.

## Ανασκόπηση Κεφαλαίου 14

- Ο χειρισμός ενός αρχείου στη C γίνεται μέσω ενός δείκτη τύπου FILE. Η συνάρτηση **fopen()**, η οποία χρησιμοποιείται για το άνοιγμα ενός αρχείου, επιστρέφει ένα δείκτη αυτού του τύπου.
- Για τη C, ένα αρχείο είναι απλώς μια σειρά από byte.
- Οι συναρτήσεις που χρησιμοποιεί η C για την ανάγνωση και την εγγραφή δεδομένων από και προς ένα αρχείο, είναι παρόμοιες με τις συναρτήσεις που χρησιμοποιεί για ανάγνωση και εμφάνιση δεδομένων στην κονσόλα.
- Οι συναρτήσεις που χρησιμοποιούμε για την τυχαία προσπέλαση σε ένα αρχείο είναι η **fseek()** μαζί με τις **fread()** και **fwrite()**.

## Ασκήσεις Κεφαλαίου 14

- 14.1** Να γραφεί πρόγραμμα το οποίο να δημιουργεί ένα αρχείο κειμένου με όνομα `arithmoi` και να καταχωρίζει στο αρχείο τους αριθμούς από το 1 μέχρι το 100. ★
- 14.2** Να γραφεί πρόγραμμα το οποίο να εμφανίζει το συνολικό άθροισμα και το μέσο όρο των αριθμών που είναι καταχωρισμένοι σε ένα αρχείο κειμένου με όνομα `input`. ★★
- 14.3** Να γραφεί συνάρτηση η οποία να **προσθέτει** στο τέλος ενός αρχείου με όνομα `output` τους αριθμούς που βρίσκονται μέσα σε έναν πίνακα `int a[100]`. Ο πίνακας θα μεταβιβάζεται στη συνάρτηση ως παράμετρος. ★★
- 14.4** Να γραφεί συνάρτηση η οποία να **προσθέτει** στο τέλος ενός αρχείου με όνομα `output` τους αριθμούς που βρίσκονται μέσα σε έναν πίνακα `int a[100]`. Ο πίνακας θα μεταβιβάζεται στη συνάρτηση ως παράμετρος. ★★
- 14.5** Να γραφεί πρόγραμμα το οποίο να διαβάζει τους αριθμούς που βρίσκονται καταχωρισμένοι κατά τριάδες στο αρχείο `data` και να εμφανίζει το μέσο όρο κάθε τριάδας. Δε γνωρίζουμε πόσες τριάδες αριθμών υπάρχουν μέσα στο αρχείο. ★★
- 14.6** Υποθέτουμε ότι έχουμε το αρχείο `sxoleio` που δημιουργήθηκε στο παράδειγμα Π4 και το οποίο περιέχει τα στοιχεία μαθητών. Να γραφεί πρόγραμμα το οποίο να διαβάζει τα στοιχεία από το αρχείο και να εμφανίζει τα στοιχεία του μαθητή με το μεγαλύτερο μέσο όρο. Υποθέστε ότι δεν γνωρίζουμε το πλήθος των εγγραφών του αρχείου. ★★ ★
- 14.7** Υποθέτουμε ότι έχουμε το αρχείο `sxoleio` που δημιουργήθηκε στο παράδειγμα Π4. Να γραφεί πρόγραμμα το οποίο να διαβάζει και να εμφανίζει τα στοιχεία της 15ης εγγραφής. ★

data
10 19 20
11 19 14
8 12 11
19 18 15
.....



- 14.8** Υποθέτουμε ότι έχουμε το αρχείο `sxoleio` που δημιουργήθηκε στο παράδειγμα Π4. Να γραφεί πρόγραμμα το οποίο να αλλάζει τα στοιχεία της 15ης εγγραφής. Τα νέα στοιχεία θα δίνονται από τον χρήστη και θα αντικαθιστούν τα υπάρχοντα στοιχεία της εγγραφής. ★ ★
- 14.9** Υποθέτουμε ότι έχουμε το αρχείο `sxoleio` που δημιουργήθηκε στο παράδειγμα Π4. Να γραφεί πρόγραμμα το οποίο θα ζητάει από το χρήστη ένα χαρακτήρα και θα εμφανίζει στην οθόνη μια λίστα με τα ονόματα και τις ηλικίες των μαθητών που το επώνυμό τους ξεκινάει από το χαρακτήρα που δόθηκε. ★ ★ ★
- 14.10** Γνωρίζουμε ότι ένα αρχείο κειμένου έχει κρυπτογραφηθεί με τον εξής τρόπο: Στο αρχείο γράφεται ο αμέσως επόμενος χαρακτήρας από τον κανονικό. Δηλαδή αντί να γραφεί η λέξη "ΝΙΚΟΣ" γράφεται τη λέξη "ΞΚΛΠΤ". Να γραφεί πρόγραμμα το οποίο θα μας ζητάει το όνομα ενός αρχείου, θα το αποκρυπτογραφεί, και θα εμφανίζει τα περιεχόμενά του στην οθόνη. ★ ★ ★
- 14.11** Ποια από τα επόμενα αληθεύουν: ★
- ☐ Ο χειρισμός ενός αρχείου στη C γίνεται μέσω ενός δείκτη τύπου FILE.
  - ☐ Ένα αρχείο είναι μια σειρά από bit.
  - ☐ Όταν ζητήσουμε να ανοίξουμε ένα αρχείο για εγγραφή και το αρχείο δεν υπάρχει, τότε δημιουργείται.
  - ☐ Δεν μπορούμε να έχουμε περισσότερα από δύο αρχεία ταυτόχρονα ανοιχτά.
  - ☐ Η `fseek()` είναι η συνάρτηση με την οποία επιτυγχάνουμε τυχαία προσπέλαση σε ένα αρχείο.