

Εισαγωγή στον Προγραμματισμό

Αρχεία

Φροντιστήριο

Αριστείδης Ηλίας

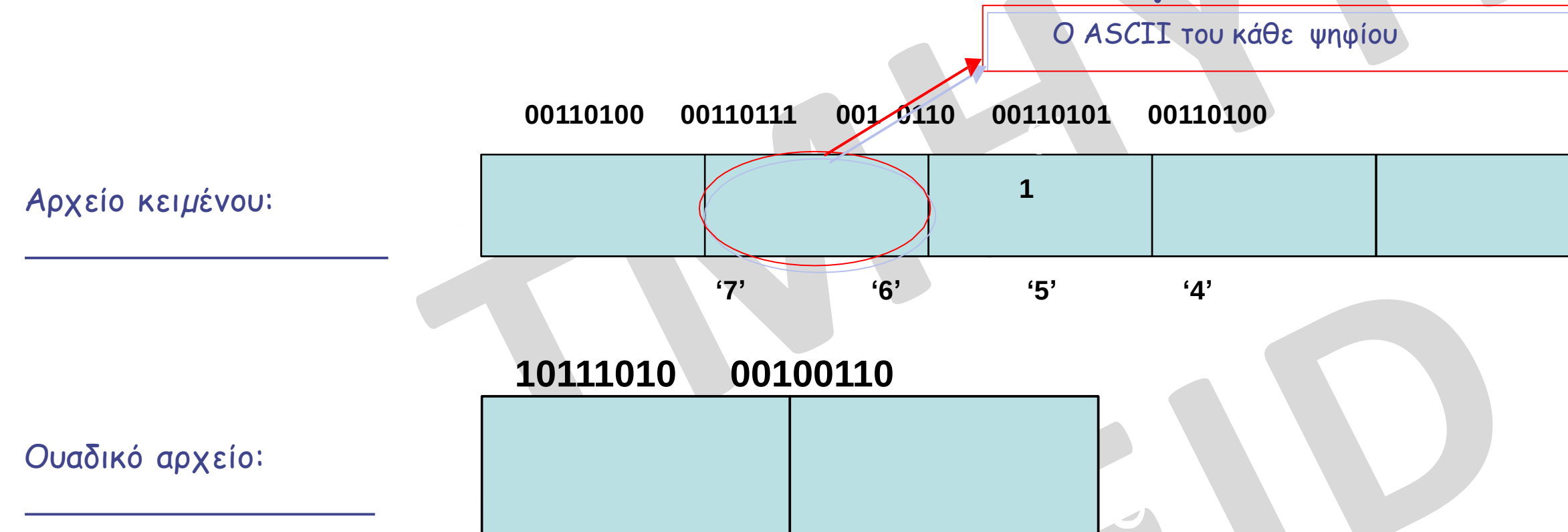
- Μορφές αρχείων
 - αρχεία κειμένου (text files) και
 - δυαδικά αρχεία (binary files)
- Γραμμογράφηση αρχείου κειμένου (π.χ. myfile.txt)

1	1	0000-00-00 00:00:00	Το δικαίωμα στην εργασία	The right to work
3	2	0000-00-00 00:00:00	Το δικαίωμα σε δίκαιες συνθήκες εργασίας	The right to fair working conditions
2	3	0000-00-00 00:00:00	Το δικαίωμα σε συνθήκες ασφάλειας	The right to safety and hygiene at work
3	4	0000-00-00 00:00:00	Το δικαίωμα σε δίκαιη αμοιβή	The right to a fair remuneration
3	5	0000-00-00 00:00:00	Το δικαίωμα στο συνδικαλισμό	The right to trade unionism
3	6	0000-00-00 00:00:00	Το δικαίωμα στις διαπραγματεύσεις	The right to collective bargaining
4	7	0000-00-00 00:00:00	Το δικαίωμα των παιδιών στην προστασία	The right of children and young people to protection
4	8	0000-00-00 00:00:00	Το δικαίωμα των εργαζομένων γυναικών	The right of working women to maternity protection
1	9	0000-00-00 00:00:00	Το δικαίωμα στον προσανατολισμό	The right to vocational guidance
1	10	0000-00-00 00:00:00	Το δικαίωμα στην επαγγελματική κατάρτιση	The right to vocational training
2	11	0000-00-00 00:00:00	Το δικαίωμα στην προστασία της υγείας	The right to health protection
2	12	0000-00-00 00:00:00	Το δικαίωμα στην κοινωνική ασφάλιση	The right to social security
2	13	0000-00-00 00:00:00	Το δικαίωμα στην κοινωνική πρόνοια	The right to social welfare
...				

- Τα αρχεία κειμένου αποτελούνται από μία ή περισσότερες γραμμές, οι οποίες περιέχουν χαρακτήρες σε αναγνώσιμη μορφή, σύμφωνα με κάποια κωδικοποίηση, όπως ο ASCII κώδικας
- Κάθε γραμμή τελειώνει με τον ειδικό χαρακτήρα που χρησιμοποιεί το λειτουργικό σύστημα για να καθορίσει το τέλος της γραμμής
 - Στα Windows για παράδειγμα, το ζευγάρι των χαρακτήρων '\r' (Carriage Return) και '\n' (Line Feed), CR/LF, με ASCII κωδικούς 13 και 10 αντίστοιχα, χαρακτηρίζει το τέλος της γραμμής
 - Άρα, ο χαρακτήρας νέας γραμμής '\n' αντικαθίσταται από αυτό το ζευγάρι, όταν γράφεται στο αρχείο ενώ η αντίστροφη αντικατάσταση συμβαίνει όταν διαβάζουμε αυτό το ζευγάρι χαρακτήρων από το αρχείο
 - Αντίθετα, σε Unix συστήματα, δεν συμβαίνει αυτή η μετατροπή, γιατί ο χαρακτήρας '\n' συμβολίζει το τέλος της γραμμής
 - Στα αρχεία κειμένου το λειτουργικό σύστημα μπορεί να προσθέσει έναν ειδικό χαρακτήρα που να δηλώνει το τέλος του αρχείου κειμένου, Π.χ. στα Windows αυτός ο χαρακτήρας είναι ο Control-Z (CTRL-Z)
 - Αντίθετα, στα δυναμικά αρχεία δεν υπάρχει κανένας χαρακτήρας με ειδική σημασία, είναι όλοι το ίδιο

...Αρχεία κειμένου και Δυαδικά αρχεία

- Το **μέγεθος** των **δυαδικών αρχείων** είναι συνήθως **μικρότερο** από το μέγεθος των **αρχείων κειμένου**, γιατί στα δυαδικά αρχεία κάθε τύπος δεδομένων αποθηκεύεται με τον αντίστοιχο αριθμό bytes που απαιτούνται
 - ◆ Π.χ. το μέγεθος ενός **αρχείου κειμένου** που περιέχει τον αριθμό 47654 ($47654_{10} = 1011101000100110_2$) θα είναι 5 bytes (αφού περιέχει 5 ψηφία), ενώ το μέγεθος ενός αντίστοιχου **δυαδικού αρχείου** θα είναι 2 bytes



$$47654_{10} = 1011101000100110_2$$

- Τέλος, οι διαδικασίες **εγγραφής** και **ανάγνωσης** σε ένα δυαδικό αρχείο **εκτελούνται πιο γρήγορα** από ότι σε ένα αρχείο κειμένου

Άνοιγμα Αρχείου...

- Για το **άνοιγμα ενός αρχείου** χρησιμοποιείται η συνάρτηση `fopen()`
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
FILE *fopen(const char *filename, const char *mode);
```
- Η παράμετρος `filename` δηλώνει το όνομα του αρχείου
- Η παράμετρος `mode` καθορίζει τις ενέργειες που μπορούν να εκτελεστούν στο αρχείο, σύμφωνα με τον πίνακα της επόμενης διαφάνειας
- Αν η συνάρτηση `fopen()` εκτελεστεί **επιτυχημένα**, τότε **επιστρέφει έναν δείκτη** σε μία δομή τύπου `FILE`
- Το πρότυπο της δομής `FILE` δηλώνεται στο αρχείο `stdio.h` και στα πεδία αυτής της δομής αποθηκεύεται πληροφορία σχετικά με το αρχείο
- Όλες οι επόμενες **λειτουργίες** που θα εκτελεστούν στο αρχείο, π.χ. ανάγνωση ή εγγραφή σε αυτό, θα γίνονται **με χρήση αυτού του δείκτη**

...Άνοιγμα Αρχείου...

Επιλογή	Ενέργεια
r	Ανοίγει το αρχείο για ανάγνωση.
w	Ανοίγει το αρχείο για εγγραφή. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν υπάρχει, το περιεχόμενό του διαγράφεται.
a	Ανοίγει το αρχείο για προσάρτηση. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν υπάρχει, τα υπάρχοντα δεδομένα διατηρούνται και τα νέα δεδομένα προστίθενται στο τέλος.
r+	Ανοίγει το αρχείο για ανάγνωση και εγγραφή.
w+	Ανοίγει το αρχείο για ανάγνωση και εγγραφή. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν υπάρχει, το περιεχόμενό του διαγράφεται.
a+	Ανοίγει το αρχείο για ανάγνωση και προσάρτηση. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν υπάρχει, τα υπάρχοντα δεδομένα διατηρούνται και τα νέα δεδομένα προστίθενται στο τέλος.

...Άνοιγμα Αρχείου...

- Για το **άνοιγμα ενός αρχείου** χρησιμοποιείται η συνάρτηση `fopen()`
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
FILE *fopen(const char *filename, const char *mode);
```
- Η παράμετρος `filename` δηλώνει το όνομα του αρχείου
- Η παράμετρος `mode` καθορίζει τις ενέργειες που μπορούν να εκτελεστούν στο αρχείο, σύμφωνα με τον πίνακα της επόμενης διαφάνειας
- Αν η συνάρτηση `fopen()` εκτελεστεί **επιτυχημένα**, τότε **επιστρέφει έναν δείκτη** σε μία δομή τύπου `FILE`
- Το πρότυπο της δομής `FILE` δηλώνεται στο αρχείο `stdio.h` και στα πεδία αυτής της δομής αποθηκεύεται πληροφορία σχετικά με το αρχείο
- Όλες οι επόμενες **λειτουργίες** που θα εκτελεστούν στο αρχείο, π.χ. ανάγνωση ή εγγραφή σε αυτό, θα γίνονται **με χρήση αυτού του δείκτη**

- Για το άνοιγμα ενός αρχείου κειμένου επιλέγουμε κάποιον από τους παραπάνω τρόπους
- Αν **αποτύχει** η εκτέλεση της συνάρτησης `fopen()`, τότε αυτή επιστρέφει την τιμή `NULL`
Παράδειγμα αποτυχίας:
 - ◆ Περίπτωση προσπάθειας ανάγνωσης ενός αρχείου (δηλ. η παράμετρος `mode` είναι `"r"`), ενώ η παράμετρος `filename` περιέχει το όνομα ενός αρχείου που δεν υπάρχει
- Όταν **τελειώσουμε** με την επεξεργασία του αρχείου, το **κλείνουμε** με τη συνάρτηση `fclose()`

- Αν το αρχείο βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο πρόγραμμα, αρκεί να γράψουμε μόνο το όνομα του αρχείου σε διπλά εισαγωγικά

Π.χ.

- ◆ `fopen("test.dat", "r")` ; ανοίγει για ανάγνωση το αρχείο κειμένου `test.dat`, το οποίο βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο πρόγραμμα
 - ◆ `fopen("test.dat", "wb")` ; ανοίγει για εγγραφή το δυαδικό αρχείο `test.dat` (αν δεν υπάρχει, δημιουργείται στον ίδιο φάκελο με το εκτελέσιμο πρόγραμμα, ενώ, αν υπάρχει, τα περιεχόμενά του διαγράφονται)
- Αν το αρχείο βρίσκεται σε διαφορετικό φάκελο από το εκτελέσιμο πρόγραμμα, τότε πρέπει να καθοριστεί η πλήρης διαδρομή

Αν το λειτουργικό σύστημα χρησιμοποιεί τον χαρακτήρα της ανάστροφης κεκλιμένης \ για τον διαχωρισμό των φακέλων (π.χ. Windows), τότε, στη διαδρομή του αρχείου, πρέπει να προστεθεί μία ακόμα \, γιατί, θυμηθείτε, η C χειρίζεται τον χαρακτήρα \ σαν την αρχή μίας ακολουθίας διαφυγής

Π.χ.

αν το πρόγραμμα εκτελείται σε Windows και θέλουμε να ανοίξουμε για διάβασμα το αρχείο `test.txt` που υπάρχει στη διαδρομή `d:\dir1\dir2`, πρέπει να γράψουμε:

```
fopen("d: \\dir1 \\dir2 \\test.txt", "r");
```

Ωστόσο, σε περίπτωση που το όνομα του αρχείου εισάγεται μέσω της γραμμής εντολών, δεν χρειάζεται να προσθέσουμε τη δεύτερη \, δηλαδή θα πρέπει να πληκτρολογήσουμε:

```
d:\dir1\dir2\test.txt (με μία \)
```

```
fopen("test.txt", "r");
```

Ανοίγει για ανάγνωση το αρχείο κειμένου `test.txt`, το οποίο βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο πρόγραμμα

```
fopen("c:\\src\\test.bin", "a+b");
```

Ανοίγει (σε λειτουργικό σύστημα Windows) για ανάγνωση και προσάρτηση το δυαδικό αρχείο `test.bin`, το οποίο βρίσκεται στον φάκελο `c:\\src`.
Αν το αρχείο δεν υπάρχει, δημιουργείται στον φάκελο `c:\\src`, ενώ, αν ήδη υπάρχει, τα υπάρχοντα δεδομένα διατηρούνται και τα νέα δεδομένα προστίθενται στο τέλος του.

```
fopen("test.dat", "wb");
```

Ανοίγει για εγγραφή το δυαδικό αρχείο `test.dat`, που βρίσκεται στον ίδιο φάκελο με το εκτελέσιμο πρόγραμμα

Αν το `test.txt` δεν υπάρχει, τότε δημιουργείται στον ίδιο φάκελο με το εκτελέσιμο πρόγραμμα, ενώ αν υπάρχει το περιεχόμενό του διαγράφεται

...Άνοιγμα Αρχείου...

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fp;
    char fname[100];

    printf("Enter file name: ");
    gets(fname);

    fp = fopen(fname, "r"); /* Άνοιγμα αρχείου για ανάγνωση. */
    if(fp == NULL)
    {
        printf("Error: File can not be opened\n");
        exit(1); /* Το αρχείο δεν άνοιξε, οπότε το πρόγραμμα
τερματίζεται. */
    }
    else
    {
        printf("File is opened successfully\n");
        fclose(fp); /* Κλείσιμο αρχείου. */
    }
    return 0;
}
```

Διαβάζει το όνομα ενός αρχείου και εμφανίζει μήνυμα για το αν αυτό μπορεί να διαβαστεί ή όχι...

Επαναλαμβάνεται, ότι **η τιμή επιστροφής** της συνάρτησης `fopen()` **πρέπει πάντα** να ελέγχεται και **μόνο αν** αυτή **είναι διαφορετική από NULL**, θα πρέπει να επιτρέπεται να γίνουν διάφορες ενέργειες στο αρχείο

- Ο έλεγχος της επιστροφής της `fopen()` θα μπορούσε να γίνει σε μία γραμμή κώδικα ως εξής:

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

Οι εσωτερικές παρενθέσεις μπαίνουν για λόγους προτεραιότητας

- Η συνάρτηση `fclose()` χρησιμοποιείται για το κλείσιμο ενός ανοικτού αρχείου
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
int fclose(FILE *fp);
```

- Η παράμετρος `fp` είναι ο δείκτης σε μία δομή τύπου `FILE`, που έχει επιστραφεί από μία προηγούμενη κλήση της `fopen()`
- Η συνάρτηση `fclose()` επιστρέφει 0 αν το αρχείο έκλεισε επιτυχώς, αλλιώς επιστρέφει την ειδική τιμή `EOF`

- Η ειδική τιμή **EOF** δηλώνεται στο αρχείο **stdio.h** , έχει τιμή **-1** και χρησιμοποιείται για να μας ενημερώσει ότι:
 - ◆ είτε φτάσαμε στο **τέλος του αρχείου**
 - ◆ είτε **συνέβη κάποιο λάθος** στην εκτέλεση μίας λειτουργίας στο αρχείο

Αν και ένα **ανοικτό αρχείο κλείνει αυτόματα όταν το πρόγραμμα τερματίζει**, προτείνεται **να το κλείνετε** όταν τελειώσετε με την επεξεργασία του

Ένας καλός λόγος για να πειστείτε είναι ότι, ακόμα και αν το πρόγραμμά σας τερματιστεί ανώμαλα (π.χ. crash), το αρχείο θα παραμείνει ανέπαφο και τα δεδομένα του δεν θα χαθούν

- Για κάθε ανοικτό αρχείο υπάρχει ένα **πεδίο-δείκτης** στη δομή **FILE** που δείχνει σε **ποια Θέση** του αρχείου θα γίνει η επόμενη λειτουργία εγγραφής ή ανάγνωσης
 - ◆ Π.χ. όταν ανοίγει ένα αρχείο για **ανάγνωση** ή **εγγραφή**, τότε αυτός ο δείκτης **δείχνει στην αρχή** του αρχείου
 - ◆ Αν όμως ανοίγει ένα αρχείο **για προσάρτηση**, τότε αυτός ο δείκτης **δείχνει στο τέλος** του αρχείου
- Κάθε φορά που εκτελείται μία λειτουργία εγγραφής ή ανάγνωσης, **η τιμή** αυτού του **δείκτη Θέσης** ενημερώνεται **αυτόματα**
 - ◆ Π.χ. αν ανοίξει ένα αρχείο για ανάγνωση και με μία εντολή ανάγνωσης διαβαστούν 50 χαρακτήρες, τότε ο δείκτης μετακινείται δεξιά και δείχνει στη θέση του αρχείου που είναι 50 bytes μετά από την αρχή του αρχείου
 - ◆ Αντίστοιχα, σε μία εγγραφή δεδομένων ο δείκτης μετακινείται δεξιά κατά τόσες θέσεις όσες και ο αριθμός των bytes που γράφτηκαν στο αρχείο
- Η συνάρτηση **fseek()** που θα παρουσιαστεί παρακάτω χρησιμοποιείται για να **μετακινήσει** τον δείκτη θέσης του αρχείου **σε οποιοδήποτε σημείο** του αρχείου

- Οι κυριότερες συναρτήσεις που χρησιμοποιούνται για εγγραφή δεδομένων σε ένα αρχείο κειμένου είναι οι:
 - ◆ `fputs()`
 - ◆ `fprintf()`
 - ◆ `putc()`
- Αν και αυτές οι συναρτήσεις εφαρμόζονται κυρίως σε αρχεία κειμένου, μπορούν να χρησιμοποιηθούν και για την εγγραφή δεδομένων σε δυαδικά αρχεία

ΣΗΜΕΙΩΣΗ

Στα παραδείγματα που θα δείτε, για λόγους απλότητας, δεν θα ελέγχουμε την τιμή επιστροφής των συναρτήσεων που χρησιμοποιούμε για εγγραφή δεδομένων στο αρχείο. Θα θεωρούμε, δηλαδή, ότι η εγγραφή τους ήταν επιτυχημένη

Η συνάρτηση fputs()

- Η συνάρτηση `fputs()` χρησιμοποιείται για την **εγγραφή ενός αλφαριθμητικού** σε ένα αρχείο κειμένου
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
int fputs(const char *str, FILE *fp);
```

- Η παράμετρος `str` είναι ένας δείκτης στο αλφαριθμητικό που θα γραφεί στο αρχείο, το οποίο υποδεικνύεται από την παράμετρο `fp`
- Αν η εγγραφή στο αρχείο είναι **επιτυχής**, η συνάρτηση `fputs()` **επιστρέφει μία μη αρνητική τιμή**, αλλιώς επιστρέφει `EOF`

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fp;
    int i;

    fp = fopen("test.txt", "w");
    if(fp == NULL)
    {
        printf("Error: File can not be created\n");
        exit(1);
    }
    for(i = 0; i < 5; i++)
        fputs("Hello\n", fp);    /* Ο χαρακτήρας '\n'
προστίθεται στο τέλος, ώστε κάθε μήνυμα "Hello" να εμφανίζεται
σε ξεχωριστή γραμμή. */

    fclose(fp);
    return 0;
}
```

Γράφει 5 φορές το μήνυμα "Hello" στο
αρχείο test.txt αλλάζοντας και γραμμή
κάθε φορά...

Η συνάρτηση fprintf()

- Η συνάρτηση `fprintf()` είναι πιο γενική από την `fputs()`, γιατί μπορεί να γράψει έναν μεταβλητό αριθμό διαφορετικών δεδομένων σε ένα αρχείο
- Παρόμοια με την `printf()`, η `fprintf()` δέχεται σαν όρισμα ένα αλφαριθμητικό μορφοποίησης, το οποίο καθορίζει τον τρόπο με τον οποίο θα εγγραφούν τα δεδομένα στο αρχείο
- Το πρωτότυπό της δηλώνεται ως εξής:

```
int fprintf(FILE *fp, const char *format, ...);
```

- Η διαφορά μεταξύ των `printf()` και `fprintf()` είναι ότι η `printf()` γράφει πάντα τα δεδομένα στο `stdout` (που εξ' ορισμού είναι συνδεδεμένο με την οθόνη) ενώ η `fprintf()` τα γράφει στο αρχείο που υποδεικνύεται από την παράμετρο `fp`
- Αν η εγγραφή είναι επιτυχής, η `fprintf()` επιστρέφει τον αριθμό των χαρακτήρων που γράφτηκαν στο αρχείο, αλλιώς μία αρνητική τιμή

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fp;
    int i;

    fp = fopen("test.txt","w"); /* Άνοιγμα του αρχείου για
εγγραφή. */
    if(fp == NULL)
    {
        printf("Error: File can not be created\n");
        exit(1);
    }
    for(i = 0; i < 5; i++)
        fprintf(fp,"%d. %s\n",i+1,"Hello");

    fclose(fp);
    return 0;
}
```

Γράφει 5 φορές το μήνυμα "Hello" και τον
αντίστοιχο αύξοντα αριθμό στο αρχείο
test.txt

Παράδειγμα

- Ποιο είναι το μέγεθος του αρχείου που δημιουργείται με το παρακάτω πρόγραμμα σε λειτουργικό σύστημα Windows ???

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fp;

    fp = fopen("test.txt", "w");
    if(fp == NULL)
    {
        printf("Error: File can not be created\n");
        exit(1);
    }

    fprintf(fp, "%d\n", 123);
    fclose(fp);
    return 0;
}
```

Για τον τριψήφιο αριθμό 123 αποθηκεύονται στο αρχείο τρεις χαρακτήρες. Ο χαρακτήρας '1' με ASCII τιμή 49, ο χαρακτήρας '2' με ASCII τιμή 50 και ο χαρακτήρας '3' με ASCII τιμή 51. Επίσης, στα Windows: '\n' € '\r' '\n' (άρα τελικά: 5 bytes)

Η συνάρτηση fprintf()

- Η συνάρτηση `fprintf()` είναι πιο γενική από την `fputs()`, γιατί μπορεί να γράψει έναν μεταβλητό αριθμό διαφορετικών δεδομένων σε ένα αρχείο
- Παρόμοια με την `printf()`, η `fprintf()` δέχεται σαν όρισμα ένα αλφαριθμητικό μορφοποίησης, το οποίο καθορίζει τον τρόπο με τον οποίο θα εγγραφούν τα δεδομένα στο αρχείο
- Το πρωτότυπό της δηλώνεται ως εξής:

```
int fprintf(FILE *fp, const char *format, ...);
```

- Η διαφορά μεταξύ των `printf()` και `fprintf()` είναι ότι η `printf()` γράφει πάντα τα δεδομένα στο `stdout` (που εξ' ορισμού είναι συνδεδεμένο με την οθόνη) ενώ η `fprintf()` τα γράφει στο αρχείο που υποδεικνύεται από την παράμετρο `fp`
- Αν η εγγραφή είναι επιτυχής, η `fprintf()` επιστρέφει τον αριθμό των χαρακτήρων που γράφτηκαν στο αρχείο, αλλιώς μία αρνητική τιμή

Η συνάρτηση fprintf()

- Σημειώστε ότι, αν στο τελευταίο παράδειγμα αλλάζαμε το προσδιοριστικό μετατροπής στην `fprintf()` και αντί για `%d` χρησιμοποιούσαμε `%c`, δηλαδή:

```
fprintf(fp, "%c\n", 123);
```

τότε θα αποθηκευόταν στο αρχείο μόνο ένας χαρακτήρας, αυτός που έχει ASCII τιμή 123 (δηλ. ο '{')

Άρα, σε αυτή την περίπτωση το μέγεθος του αρχείου είναι 3 bytes

- ♦ ένα byte για τον χαρακτήρα με ASCII τιμή 123 (τον '{')
- ♦ ένα byte για το **Carriage Return** ('\r') και
- ♦ ένα byte για το **Line Feed** ('\n')

- Στη γενική περίπτωση, όταν αποθηκεύεται στο αρχείο ένας ακέραιος αριθμός με το προσδιοριστικό `%d`, τότε αποθηκεύονται τόσοι χαρακτήρες, όσα και τα ψηφία του αριθμού

Η συνάρτηση `putc()`

- Η συνάρτηση `putc()` χρησιμοποιείται για την **εγγραφή ενός χαρακτήρα** σε ένα αρχείο
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
int putc(int ch, FILE *fp);
```
- Η παράμετρος `ch` δηλώνει τον χαρακτήρα που θα αποθηκευτεί στο αρχείο, που υποδεικνύεται από την παράμετρο `fp`
- Αν η **εγγραφή** στο αρχείο ήταν **επιτυχής** η συνάρτηση `putc()` **επιστρέφει τον χαρακτήρα** που γράφτηκε στο αρχείο, αλλιώς **EOF**

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char str[] = "This text will be saved in the file";
    int i;

    fp = fopen("test.txt", "w");
    if(fp == NULL)
    {
        printf("Error: File can't be created\n");
        exit(1);
    }
    for(i = 0; str[i] != '\0'; i++)
        putc(str[i], fp);

    fclose(fp);
    return 0;
}
```

Γράφει έναν-έναν τους χαρακτήρες του αλφαριθμητικού "This text will be saved in the file" στο αρχείο κειμένου test.txt

- Αν στα προηγούμενα παραδείγματα η παράμετρος `fp` των συναρτήσεων `fputs()`, `fprintf()` και `putc()` αντικατασταθεί με το `stdout`, τότε τα δεδομένα θα εμφανίζονται στην οθόνη
- Μία συνάρτηση που εκτελεί την ίδια λειτουργία με την `putc()` είναι η `fputc()`
- Η διαφορά τους είναι ότι η `putc()` υλοποιείται σαν **μακροεντολή**, ενώ η `fputc()` σαν **συνάρτηση**
- Αφού οι μακροεντολές τείνουν να **εκτελούνται γρηγορότερα** (περισσότερα για τις μακροεντολές στο επόμενο κεφάλαιο) είναι προτιμότερο να χρησιμοποιείτε την `putc()` έναντι της `fputc()`

- Οι κυριότερες συναρτήσεις που χρησιμοποιούνται για διάβασμα δεδομένων από ένα αρχείο κειμένου είναι οι:
 - ◆ `fscanf()`
 - ◆ `fgets()`
 - ◆ `getc()`
- Αν και αυτές οι συναρτήσεις εφαρμόζονται κυρίως σε αρχεία κειμένου, μπορούν να χρησιμοποιηθούν και για διάβασμα δεδομένων από δυαδικά αρχεία

Η συνάρτηση fscanf()

- Η συνάρτηση `fscanf()` χρησιμοποιείται για να διαβάσουμε έναν μεταβλητό αριθμό διαφορετικών δεδομένων από ένα αρχείο κειμένου
- Παρόμοια με τη `scanf()`, η `fscanf()` δέχεται σαν όρισμα ένα αλφαριθμητικό μορφοποίησης, το οποίο καθορίζει τον τρόπο με τον οποίο θα διαβαστούν τα δεδομένα από το αρχείο
- Το πρωτότυπό της δηλώνεται ως εξής:

```
int fscanf(FILE *fp, const char *format, ...);
```
- Η διαφορά μεταξύ των `scanf()` και `fscanf()` είναι ότι η `scanf()` διαβάζει πάντα τα δεδομένα από το `stdin`, ενώ η `fscanf()` τα διαβάζει από το αρχείο που υποδεικνύεται από την παράμετρο `fp`
- Η `fscanf()` επιστρέφει τον αριθμό των στοιχείων που διαβάστηκαν επιτυχημένα από το αρχείο και εκχωρήθηκαν σε μεταβλητές του προγράμματος
- Αν φτάσουμε στο τέλος του αρχείου ή αν συμβεί κάποιο άλλο λάθος, επιστρέφει **EOF** (θα δούμε στη συνέχεια ότι μπορούμε να χρησιμοποιήσουμε τη συνάρτηση `feof()`, για να διαπιστώσουμε ποιο από τα δύο συνέβη)

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fp;
    int i,ret;

    fp = fopen("test.txt","r");
    if(fp == NULL)
    {
        printf("Error: File can not be loaded\n");
        exit(1);
    }
    while(1)
    {
        ret = fscanf(fp,"%d",&i); /* Η συνάρτηση fscanf()
επιστρέφει το πλήθος των στοιχείων που διάβασε επιτυχημένα από
το αρχείο. Στη συγκεκριμένη περίπτωση αυτή η τιμή πρέπει να
είναι 1, αφού κάθε φορά διαβάζουμε έναν ακέραιο. Όταν φτάσουμε
στο τέλος του αρχείου η τιμή επιστροφής είναι EOF. */

        if(ret != 1) /* Θα μπορούσαμε να μην
χρησιμοποιήσουμε τη μεταβλητή ret και να γράψουμε κατευθείαν
if(fscanf(fp,"%d",&i) != 1) */
            break;

        printf("Val = %d\n",i);
    }

    fclose(fp);
    return 0;
}
```

Διαβάζει τους ακέραιους που περιέχονται
στο αρχείο test.txt και τους εμφανίζει
στην οθόνη

- Προτείνουμε να συγκρίνετε την τιμή επιστροφής της `fscanf()` με τον αριθμό των στοιχείων που διαβάστηκαν και όχι με την τιμή `EOF`

Π.χ. υποθέτοντας ότι ο παρακάτω κώδικας διαβάζει τους πραγματικούς αριθμούς που περιέχονται σε ένα αρχείο κειμένου:

```
float i;  
while (fscanf(fp, "%d", &i) != EOF)  
{  
    /* Χρήση της μεταβλητής i */  
}
```

παρόλο που στη μεταβλητή `i` δεν έχει εκχωρηθεί σωστά η τιμή εξαιτίας του λάθους προσδιοριστικού `%d` (αντί για `%f`) η `fscanf()` δεν επιστρέφει `EOF` και η εκτέλεση του βρόχου συνεχίζεται

- Η χρήση της `fscanf()` προϋποθέτει ότι είναι γνωστός ο τύπος και ο τρόπος που τα δεδομένα έχουν αποθηκευτεί στο αρχείο
- Π.χ. στο πρόγραμμα που ακολουθεί ο προγραμματιστής πρέπει να γνωρίζει ότι κάθε γραμμή του αρχείου `test.txt` περιέχει **ένα αλφαριθμητικό μέχρι 100 χαρακτήρες, έναν ακέραιο και έναν αριθμό υψηλής ακρίβειας**, ώστε να μεταβιβάσει τις κατάλληλες παραμέτρους στην `fscanf()`
- Όταν διαβάζονται διαφορετικά δεδομένα από το αρχείο η `fscanf()`, όπως και η `scanf()`, χρησιμοποιεί το κενό για να ξεχωρίσουν οι τιμές μεταξύ των

Προσέξτε όμως, **αν το αλφαριθμητικό περιέχει κενά**, η `fscanf()` **δεν θα λειτουργήσει σωστά**, γιατί μετά τον πρώτο κενό χαρακτήρα θεωρεί ότι ακολουθεί ο επόμενος τύπος δεδομένων και όχι το υπόλοιπο τμήμα του αλφαριθμητικού

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char str[100];
    int i;
    double d;

    fp = fopen("test.txt", "r");
    if(fp == NULL)
    {
        printf("Error: File can't be loaded\n");
        exit(1);
    }
    while(1)
    {
        if(fscanf(fp, "%s%d%lf", &str, &i, &d) != 3)
            break;
        printf("%s %d %f\n", str, i, d);
    }
    fclose(fp);
    return 0;
}
```

Η συνάρτηση fgets()

- Η συνάρτηση `fgets()` χρησιμοποιείται για το **διάβασμα χαρακτήρων** από ένα αρχείο
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
char *fgets(char *str, int size, FILE *fp);
```
- Η παράμετρος `str` είναι ένας δείκτης στη μνήμη, στην οποία θα αποθηκευτούν οι χαρακτήρες
- Η παράμετρος `size` δηλώνει τον αριθμό των χαρακτήρων που θα διαβαστούν από το αρχείο (το οποίο υποδεικνύεται από την παράμετρο `fp`)
- Ο αριθμός αυτός **δεν πρέπει** να είναι μεγαλύτερος από το μέγεθος της δεσμευμένης μνήμης, για να μην συμβεί **υπερχείλιση** (Σημ. η `fgets()` προσθέτει στο τέλος τον τερματικό χαρακτήρα `'\0'`)
- Η συνάρτηση `fgets()` σταματάει να διαβάζει χαρακτήρες από το αρχείο, όταν διαβάσει τον χαρακτήρα νέας γραμμής ή όταν διαβάσει `size-1` χαρακτήρες
- Αν η `fgets()` εκτελεστεί επιτυχημένα, οι χαρακτήρες αποθηκεύονται στη μνήμη που δείχνει ο δείκτης `str` και αυτός ο δείκτης επιστρέφεται
- **Αν φτάσουμε στο τέλος του αρχείου ή αν συμβεί κάποιο άλλο λάθος**, τότε η `fgets()` επιστρέφει την τιμή `NULL`

- Θυμηθείτε τη συνάρτηση `gets()` από τα Αλφαριθμητικά και τον κίνδυνο κατά τη χρήση της, αν το πρόγραμμα διαβάσει περισσότερους χαρακτήρες από το μέγεθος της δεσμευμένης μνήμης, το πιθανότερο είναι να δημιουργηθεί πρόβλημα κατά την εκτέλεση του προγράμματος
- Αφού με την `fgets()` μπορούμε να καθορίσουμε το μέγιστο πλήθος των χαρακτήρων που θα διαβαστούν **είναι ασφαλέστερο** να γράψουμε:

```
fgets(str, sizeof(str), stdin);
```

αντί για:

```
gets(str);
```

Την `gets()` να τη χρησιμοποιείτε μόνο αν είστε βέβαιοι ότι ο χρήστης δεν θα εισάγει περισσότερους χαρακτήρες από αυτούς που μπορούν να αποθηκευτούν

Παράδειγμα...

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char ch, str[100];
    int i, times;

    fp = fopen("test.txt", "w+");

    if(fp == NULL)
    {
        printf("Error: File can't be loaded\n");
        exit(1);
    }
    for(i = 0; i < 50; i++)
    {
        printf("Enter name: ");
        gets(str);
        fprintf(fp, "%s\n", str);
    }
}
```

Ως εδώ, διαβάζει τα ονόματα 50 φοιτητών (μέχρι 100 χαρακτήρες) και τα αποθηκεύει σε διαφορετικές γραμμές ενός αρχείου κειμένου test.txt

...Παράδειγμα

```
printf("Enter char: ");
ch = getchar();

fseek(fp, 0, SEEK_SET);
times = 0;
while(1)
{
    if(fgets(str, sizeof(str), fp) == NULL)
        break;
    if(str[0] == ch)
    {
        printf("Name: %s\n", str);
        times++;
    }
}
printf("Total occurrences = %d\n", times);
fclose(fp);
return 0;
}
```

Στη συνέχεια, διαβάζει έναν χαρακτήρα, μετά διαβάζει τα ονόματα από το αρχείο και εμφανίζει αυτά που αρχίζουν από τον συγκεκριμένο χαρακτήρα. Πριν τερματίσει, το πρόγραμμα εμφανίζει το πλήθος των ονομάτων που βρέθηκαν

- Η συνάρτηση `getc()` χρησιμοποιείται για το διάβασμα ενός χαρακτήρα από ένα αρχείο
- Το πρωτότυπο της συνάρτησης δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
int getc(FILE *fp);
```

- Αν η συνάρτηση `getc()` εκτελεστεί επιτυχημένα, επιστρέφει τον χαρακτήρα που διαβάστηκε
- Αν φτάσουμε στο τέλος του αρχείου ή αν συμβεί κάποιο άλλο λάθος, τότε η `getc()` επιστρέφει την τιμή `EOF`
- Όπως με τις προηγούμενες συναρτήσεις διαβάματος, μπορούμε να χρησιμοποιήσουμε την `feof()` για να διαπιστώσουμε την αιτία του λάθους

- Μία συνάρτηση που εκτελεί την ίδια λειτουργία με την `getc()` είναι η `fgetc()`
- Όπως και με τις `putc()` και `fputc()`, η διαφορά μεταξύ `getc()` και `fgetc()` είναι ότι η `getc()` υλοποιείται σαν **μακροεντολή**, επομένως **εκτελείται γρηγορότερα** από την `fgetc()`

Η τιμή επιστροφής της `getc()`, όπως και των `fgetc()` και `getchar()`, πρέπει να αποθηκεύεται σε μεταβλητή τύπου **`int`** και όχι τύπου **`char`**

Π.χ. ας υποθέσουμε ότι χρησιμοποιούμε την `getc()` για να διαβάσουμε τους χαρακτήρες από ένα δυαδικό αρχείο, το οποίο περιέχει την τιμή 255. Όταν διαβαστεί το 255, η τιμή που θα αποθηκευτεί σε μία προσημασμένη **`char`** μεταβλητή θα είναι το -1, λόγω υπερχείλισης. Έτσι, η συνθήκη που κάνει σύγκριση της τιμής της με την τιμή **EOF** θα γίνει αληθής και το πρόγραμμα θα σταματήσει να διαβάζει άλλους χαρακτήρες

Η συνάρτηση fseek()...

- Η συνάρτηση **fseek()** χρησιμοποιείται για να μετακινήσει τον δείκτη θέσης του αρχείου, σε κάποιο σημείο του αρχείου
- Το πρωτότυπό της δηλώνεται στο αρχείο **stdio.h** και είναι το ακόλουθο:

```
int fseek(FILE *fp, long int offset, int origin);
```

- Η **fseek()** **μετακινεί** τον δείκτη θέσης του αρχείου που υποδεικνύεται από την παράμετρο **fp** σε μία **νέα θέση** που **απέχει** **offset** bytes από το πεδίο **origin**
- Η τιμή του **offset** **μπορεί να είναι και αρνητική**, γεγονός που σημαίνει ότι η μετατόπιση του δείκτη γίνεται **"προς τα πίσω"** μέσα στο αρχείο)

```
int fseek(FILE *fp, long int offset, int origin);
```

- Η παράμετρος **origin** καθορίζει το σημείο εκκίνησης από το οποίο θα μετακινηθεί ο δείκτης και οι επιτρεπτές τιμές φαίνονται στον πίνακα

Σταθερά	Τιμή	Περιγραφή
SEEK_SET	0	ο δείκτης θέσης του αρχείου μετατοπίζεται offset bytes από την αρχή του αρχείου
SEEK_CUR	1	ο δείκτης θέσης του αρχείου μετατοπίζεται offset bytes από την τρέχουσα θέση του
SEEK_END	2	ο δείκτης θέσης του αρχείου μετατοπίζεται offset bytes από το τέλος του αρχείου

- Η συνάρτηση **fseek()** επιστρέφει την τιμή 0, αν ο δείκτης μετακινήθηκε επιτυχώς, αλλιώς επιστρέφει μία μη μηδενική τιμή

Παραδείγματα

για να μετακινηθούμε στο τέλος του αρχείου γράφουμε:

```
fseek(fp, 0, SEEK_END) ;
```

για να μετακινηθούμε 20 bytes από την αρχή του αρχείου γράφουμε:

```
fseek(fp, 20, SEEK_SET) ;
```

για να μετακινηθούμε 10 bytes πίσω από την τρέχουσα θέση του αρχείου γράφουμε:

```
fseek(fp, -10, SEEK_CUR) ;
```

- Η `fseek()` χρησιμοποιείται με ασφάλεια σε δυαδικά αρχεία

Όταν η `fseek()` χρησιμοποιείται σε αρχεία κειμένου, **χρειάζεται προσοχή με τους χαρακτήρες νέας γραμμής**

Π.χ., ας υποθέσουμε ότι ο παρακάτω κώδικας γράφει κάποιο κείμενο στις δύο πρώτες γραμμές ενός αρχείου κειμένου και μετά επιχειρεί να μετακινήσει τον δείκτη θέσης στην αρχή της δεύτερης γραμμής

```
fputs("text\n", fp); fputs("another text\n", fp);  
/* Μετακίνηση στην αρχή της δεύτερης γραμμής. */ fseek(fp,  
6, SEEK_SET);
```

Αν το λειτουργικό σύστημα μετατρέπει το `'\n'` σε `'\r'` και `'\n'`, η τιμή του `offset` πρέπει να είναι 6 (και όχι 5)

- Η συνάρτηση `feof()` χρησιμοποιείται για να επιβεβαιώσουμε ότι φτάσαμε στο τέλος του αρχείου
- Το πρωτότυπό της δηλώνεται στο αρχείο `stdio.h` και είναι το ακόλουθο:

```
int feof(FILE *fp);
```

- Αν μία συνάρτηση ανάγνωσης επιχειρήσει να διαβάσει δεδομένα μετά το τέλος του αρχείου, η `feof()` επιστρέφει μη μηδενική τιμή, αλλιώς επιστρέφει την τιμή 0
- Είπαμε νωρίτερα ότι για να διαπιστώσουμε αν συνέβη κάποιο λάθος σε μία λειτουργία ανάγνωσης ελέγχουμε την τιμή επιστροφής της συνάρτησης που χρησιμοποιήσαμε
- Σε αυτή την περίπτωση (που έχουμε δηλαδή κάποια ένδειξη λάθους) μπορούμε να χρησιμοποιήσουμε την `feof()`, για να διαπιστώσουμε αν η λειτουργία ανάγνωσης απέτυχε επειδή φτάσαμε στο τέλος του αρχείου ή για κάποιον άλλον λόγο

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char str[100];

    fp = fopen("test.txt", "r");
    if(fp == NULL)
    {
        printf("Error: File can't be loaded\n");
        exit(1);
    }
    while(1)
    {
        if(fgets(str, sizeof(str), fp) != NULL)
            printf("%s", str);
        else
        {
            if(feof(fp))
                printf("End of file\n");
            else
                printf("Failed for another reason\n");
            break; /* Τερματισμός του βρόχου. */
        }
    }
    fclose(fp);
    return 0;
}
```

Το πρόγραμμα διαβάζει τα περιεχόμενα ενός αρχείου κειμένου (θεωρούμε ότι κάθε γραμμή του έχει λιγότερους από 100 χαρακτήρες) και όταν η fgets() αποτύχει χρησιμοποιούμε την feof() για να διαπιστώσουμε την αιτία του λάθους

Ευχαριστώ

Ερωτήσεις?