

# Διάλεξη 16 - Δεδομένα Εισόδου #2 - Αρχεία

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

## Ανακοινώσεις / Διευκρινήσεις

- Η Εργασία #1 έχει προθεσμία την ερχόμενη Τετάρτη
- Η Εργασία #2 θα βγει προσεχώς

## Την προηγούμενη φορά

- Εμβέλεια Μεταβλητών
- Παγκόσμια / Στατική Μνήμη
- Συμβολοσειρές (strings)

# Σήμερα

- Δεδομένα Εισόδου #2
  - Συναρτήσεις χειρισμού stdin (π.χ., scanf)
  - Αρχεία (Files) και ρεύματα δεδομένων (data streams)
  - Συναρτήσεις χειρισμού αρχείων
  - Παραδείγματα

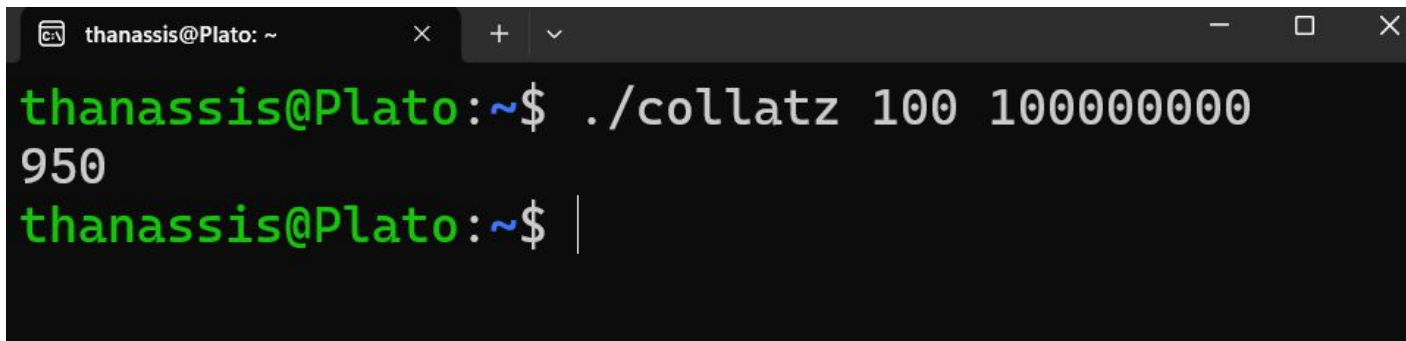
# Δεδομένα Εισόδου και Εξόδου (Input and Output Data)



## Δεδομένα Εισόδου (Input Data)

Τα **δεδομένα εισόδου** (**input data**) είναι μια σειρά από χαρακτήρες (bytes) τα οποία ο χρήστης δίνει στο πρόγραμμα. Υπάρχουν 4 μέθοδοι να εισάγουμε δεδομένα:

1. **Ορίσματα** στην γραμμή εντολών

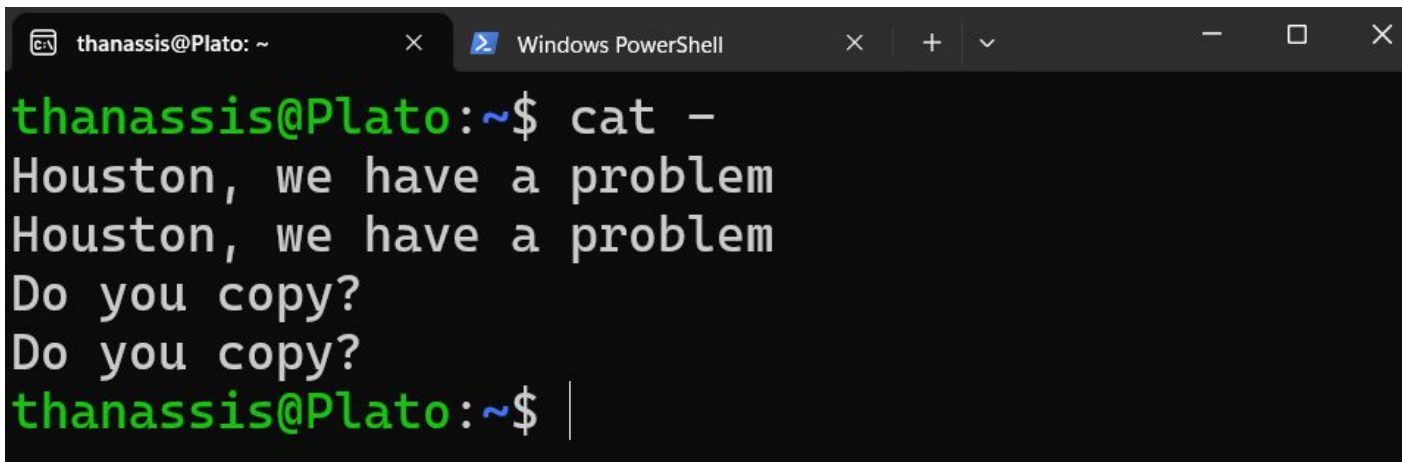


```
thanassis@Plato: ~  
thanassis@Plato:~$ ./collatz 100 100000000  
950  
thanassis@Plato:~$ |
```

## Δεδομένα Εισόδου (Input Data) - 2/4

Τα **δεδομένα εισόδου** (**input data**) είναι μια σειρά από χαρακτήρες (bytes) τα οποία ο χρήστης δίνει στο πρόγραμμα. Υπάρχουν 4 μέθοδοι να εισάγουμε δεδομένα:

2. Γράφοντας κείμενο στην **πρότυπη είσοδο** (**standard input** ή **stdin**) συνήθως με το πληκτρολόγιο

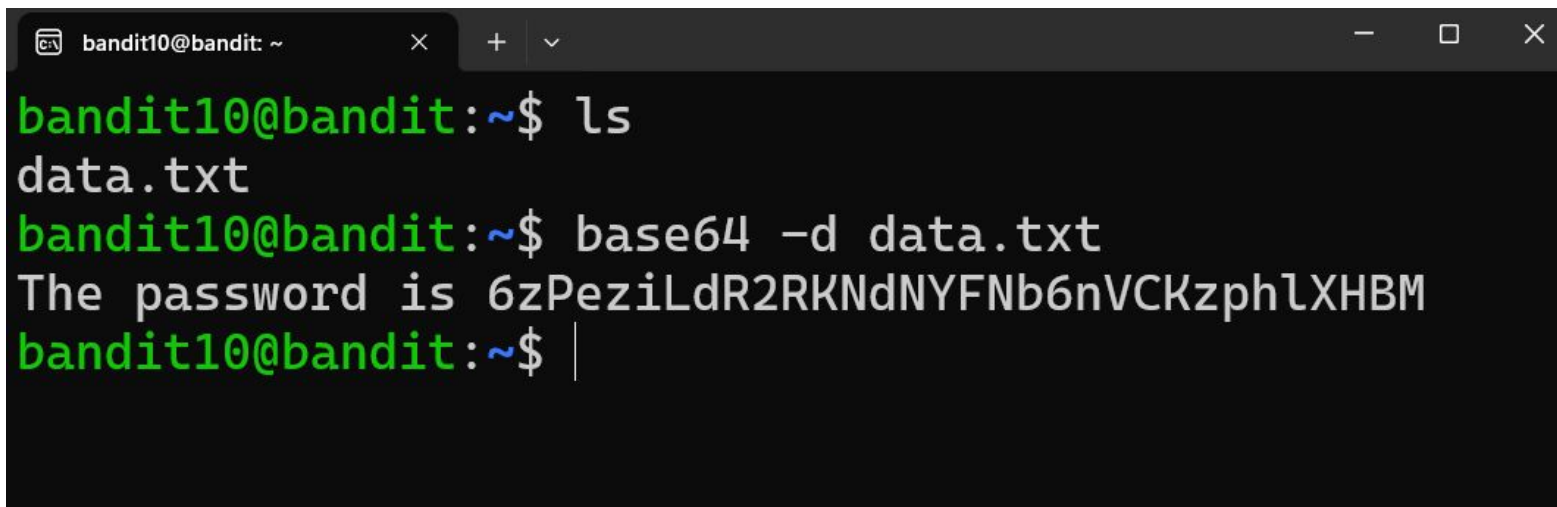


```
thanassis@Plato: ~  
Windows PowerShell  
thanassis@Plato:~$ cat -  
Houston, we have a problem  
Houston, we have a problem  
Do you copy?  
Do you copy?  
thanassis@Plato:~$ |
```

## Δεδομένα Εισόδου (Input Data) - 3/4

Τα **δεδομένα εισόδου** (input data) είναι μια σειρά από χαρακτήρες (bytes) τα οποία ο χρήστης δίνει στο πρόγραμμα. Υπάρχουν 4 μέθοδοι να εισάγουμε δεδομένα:

3. Διαβάζοντας **αρχεία** από το σύστημα αρχείων (σήμερα!)



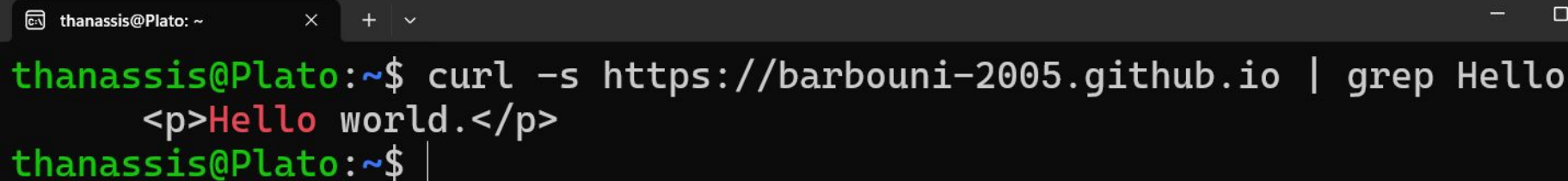
```
bandit10@bandit: ~  
bandit10@bandit:~$ ls  
data.txt  
bandit10@bandit:~$ base64 -d data.txt  
The password is 6zPezilDR2RKNdNYFNb6nVCKzphlXHBM  
bandit10@bandit:~$ |
```



## Δεδομένα Εισόδου (Input Data) - 4/4

Τα **δεδομένα εισόδου** (**input data**) είναι μια σειρά από χαρακτήρες (bytes) τα οποία ο χρήστης δίνει στο πρόγραμμα. Υπάρχουν 4 μέθοδοι να εισάγουμε δεδομένα:

4. Διαβάζοντας από το **δίκτυο** ή άλλες πηγές - π.χ., User Interface (σε επόμενα εξάμηνα)



```
thanassis@Plato: ~  
thanassis@Plato:~$ curl -s https://barbouni-2005.github.io | grep Hello  
<p>Hello world.</p>  
thanassis@Plato:~$
```

## Δεδομένα Εισόδου (Input Data)

Τα **δεδομένα εισόδου** (**input data**) είναι μια σειρά από χαρακτήρες (bytes) τα οποία ο χρήστης δίνει στο πρόγραμμα. Υπάρχουν 4 μέθοδοι να εισάγουμε δεδομένα:

1. **Ορίσματα** στην γραμμή εντολών ✓
2. Γράφοντας κείμενο στην **πρότυπη είσοδο** (**standard input** ή **stdin**) ✓ ←
3. Διαβάζοντας **αρχεία** από το σύστημα αρχείων (σήμερα) ←
4. Διαβάζοντας από το **δίκτυο** ή άλλες πηγές (άλλα εξάμηνα)

50%



Θέλω να διαβάσω δύο αριθμούς από την πρότυπη είσοδο και να τους προσθέσω. Πως;

```
$ ./addnums
```

```
Give me a number: 40
```

```
Give me another number: 2
```

```
Total: 42
```

Θέλω να διαβάσω δύο αριθμούς από την πρότυπη είσοδο και να τους προσθέσω. Πως;

```
$ ./addnums
```

```
Give me a number: 40
```

```
Give me another number: 2
```

```
Total: 42
```

Κάνοντας χρήση της `getchar` και φτιάχνοντας μια συνάρτηση `getinteger` προκειμένου να διαβάσουμε τους χαρακτήρες έναν-έναν και να τους μετατρέψουμε σε αριθμό. Υπάρχει άλλος τρόπος να επιτύχουμε το ίδιο αποτέλεσμα;

```
#define ERROR -1

int getinteger(int base) {

    char ch;

    int val = 0;

    while ((ch = getchar()) != '\n')

        if (ch >= '0' && ch <= '0' + base - 1)

            val = base * val + (ch - '0');

        else

            return ERROR;

    return val;

}
```

# Η συνάρτηση `scanf`

Η συνάρτηση `scanf` ορίζεται στο header file `stdio.h` και χρησιμοποιείται για να διαβάσει δεδομένα εισόδου πολλών τύπων από το `stdin` του προγράμματος και να αποθηκεύσει τις τιμές τους σε μεταβλητές. Αν επιτύχει, επιστρέφει πόσα δεδομένα εισόδου διάβασε. Αν αποτύχει, επιστρέφει την τιμή End-Of-File / EOF (-1).

Πως μπορώ να βρω πως συμπεριφέρεται;

Ανοίγω ένα τερματικό και τρέχω  
`man scanf!`

# Η συνάρτηση `scanf`

Η συνάρτηση `scanf` ορίζεται στο header file `stdio.h` και χρησιμοποιείται για να διαβάσει δεδομένα εισόδου πολλών τύπων από το `stdin` του προγράμματος και να αποθηκεύσει τις τιμές τους σε μεταβλητές. Αν επιτύχει, επιστρέφει *πόσα δεδομένα εισόδου* διάβασε. Αν αποτύχει, επιστρέφει την τιμή `End-Of-File / EOF (-1)`. Η συνάρτηση έχει την ακόλουθη μορφή:

```
int scanf(const char *restrict format, ...);
```

Έχει μια συμβολοσειρά μορφοποίησης  
(format string)

Δέχεται όσα ορίσματα περάσουμε  
(άλλο μάθημα)

# Η συνάρτηση scanf

Η συνάρτηση **scanf** ορίζεται στο header file `stdio.h` και χρησιμοποιείται για να διαβάσει δεδομένα εισόδου πολλών τύπων από το **stdin** του προγράμματος και να αποθηκεύσει τις τιμές τους σε μεταβλητές. Αν επιτύχει, επιστρέφει πόσα δεδομένα εισόδου διάβασε. Αν αποτύχει, επιστρέφει την τιμή End-Of-File / EOF (-1). Η συνάρτηση έχει την ακόλουθη μορφή:

```
int scanf(const char *restrict format, ...);
```

```
int printf(const char *restrict format, ...);
```

Είναι η συμμετρική της `printf` για διάβασμα αντί για εκτύπωση

# Χρήση της συνάρτησης scanf

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n;
    printf("Gimme a number: ");
    scanf("%d", &n);
    printf("Square: %d\n", n * n);
    return 0;
}
```



# Χρήση της συνάρτησης scanf

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n;
    printf("Gimme a number: ");
    scanf("%d", &n);
    printf("Square: %d\n", n * n);
    return 0;
}
```

Περνάμε την διεύθυνση της μεταβλητής `n` ώστε η `scanf` να μπορέσει να αναθέσει την τιμή που διάβασε

Τυπώνει στο `stdout` το τετράγωνο του αριθμού που γράψαμε στο `stdin`

# Χρήση της συνάρτησης scanf

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {

    int n;

    printf("Gimme a number: ");
    scanf("%d", &n);
    printf("Square: %d\n", n * n);
    return 0;
}
```

Τι θα γινόταν αν γράφαμε `scanf("%d", n);`; Γιατί;  
\$ ./scanf  
Gimme a number: 3  
Segmentation fault

Περνάμε την διεύθυνση της μεταβλητής `n` ώστε η `scanf` να μπορέσει να αναθέσει την τιμή που διάβασε

Τυπώνει στο `stdout` το τετράγωνο του αριθμού που γράψαμε στο `stdin`

# Χρήση της συνάρτησης scanf

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n;
    printf("Gimme a number: ");
    scanf("%d", &n);
    printf("Square: %d\n", n * n);
    return 0;
}
```

Είναι σωστό αυτό το πρόγραμμα;

# Χρήση της συνάρτησης scanf

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n;
    printf("Gimme a number: ");
    scanf("%d", &n);
    printf("Square: %d\n", n * n);
    return 0;
}
```

Είναι σωστό αυτό το πρόγραμμα;

Όχι καθώς δεν ελέγχουμε την τιμή επιστροφής της scanf (EOF ή ίσως 0!)

# Χρήση της συνάρτησης scanf

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n;
    printf("Gimme a number: ");
    scanf("%d", &n);
    printf("Square: %d\n", n * n);
    return 0;
}
```

```
$ ./scanf
Gimme a number: 16
Square: 256
$ ./scanf
Gimme a number: Square:
1068701481
$ ./scanf
Gimme a number: hello
Square: 1072038564
```

# Χρήση της συνάρτησης scanf - Πολλά ορίσματα

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n1, n2;
    printf("Gimme two numbers: ");
    scanf("%d %d", &n1, &n2);
    printf("Result: %d\n", n1 * n2);
    return 0;
}
```

# Χρήση της συνάρτησης scanf - Πολλά ορίσματα

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int n1, n2;
    printf("Gimme two numbers: ");
    scanf("%d %d", &n1, &n2);
    printf("Result: %d\n", n1 * n2);
    return 0;
}
```

```
$ ./scanf2
Gimme two numbers: 2 4
Result: 8
```

Καθώς ψάχνει για δεκαδικό ψηφίο, η scanf αγνοεί τους κενούς χαρακτήρες ή αλλαγές γραμμής

# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>
#include <math.h>
int main() {
    double d1, d2;
    printf("Gimme two doubles: ");
    scanf("%lf %lf", &d1, &d2);
    printf("Hypotenuse: %.1lf\n", sqrt(d1 * d1 + d2 * d2));
    return 0;
}
```



# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>
```

```
#include <math.h>
```

```
int main() {
```

```
    double d1, d2;
```

```
    printf("Gimme two doubles: ");
```

```
    scanf("%lf %lf", &d1, &d2);
```

```
    printf("Hypotenuse: %.1lf\n", sqrt(d1 * d1 + d2 * d2));
```

```
    return 0;
```

```
}
```

```
$ ./hypotenuse
```

```
Gimme two numbers: 3.0 4.0
```

```
Result: 5.0
```

# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    char message[7];
    printf("Say something: ");
    scanf("%s", message);
    printf("%s\n", message);
    return 0;
}
```

# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    char message[7];
    printf("Say something: ");
    scanf("%s", message);
    printf("%s\n", message);
    return 0;
}
```

```
$ ./message
Say something: hello!
hello!
```

Δεν βάλαμε & πριν την μεταβλητή message. Πως και λειτουργεί;

# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    char message[7];
    printf("Say something: ");
    scanf("%s", message);
    printf("%s\n", message);
    return 0;
}
```

```
$ ./message
Say something: hello!
hello!
```

Μπορεί να πάει κάτι στραβά εδώ;

# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    char message[7];
    printf("Say something: ");
    scanf("%s", message);
    printf("%s\n", message);
    return 0;
}
```

```
$ ./message
Say something: Houston,
we've had a problem here.
Houston,
Segmentation fault
```

Προσοχή! Δεν υπάρχει κανένας έλεγχος ότι δεν θα διαβαστούν περισσότεροι χαρακτήρες από 6

# Χρήση της συνάρτησης scanf - Άλλοι τύποι ορισμάτων

Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    char message[7];
    printf("Say something: ");
    scanf("%6s", message);
    printf("%s\n", message);
    return 0;
}
```

```
$ ./message
Say something: Houston,
we've had a problem here.
Houston,
Segmentation fault
```

Μπορούμε να θέσουμε περιορισμό στους πόσους χαρακτήρες θα διαβαστούν

# Η συνάρτηση gets

Η συνάρτηση gets συμπεριφέρεται παρόμοια με την scanf( "%s" ) και γενικά αποφεύγεται για λόγους ασφαλείας.

```
ethan@pegasus: ~
GETS(3) Linux Programmer's Manual GETS(3)

NAME
  gets - get a string from standard input (DEPRECATED)

SYNOPSIS
  #include <stdio.h>

  char *gets(char *s);

DESCRIPTION
  Never use this function.

  gets() reads a line from stdin into the buffer pointed to by s until
  either a terminating newline or EOF, which it replaces with a null byte
```

# Χειρισμός Αρχείων (File Handling)



# Filesystem (Σύστημα Αρχείων)

[Αρχείο \(File\)](#) είναι ένας πόρος για να καταγράψουμε δεδομένα σε έναν υπολογιστή. Συνήθως αποθηκεύεται στην δευτερεύουσα μνήμη (π.χ., σκληρός δίσκος).

Στο Linux σχεδόν τα [πάντα είναι ένα αρχείο](#).

Κάθε αρχείο:

Το περιεχόμενο του αρχείου είναι ένας πίνακας από χαρακτήρες `char bytes[ ]`

- Έχει ένα όνομα (**filename/basename**)
- Βρίσκεται μέσα σε ένα συγκεκριμένο φάκελο (**directory/folder**)
- Έχει ένα πλήρες μονοπάτι (**filepath**) που καθορίζει που βρίσκεται το αρχείο.

Παράδειγμα: το filepath ενός αρχείου είναι `/home/users/thanassis/documents/students.txt`, ο φάκελος μέσα στον οποίο βρίσκεται αυτό το αρχείο είναι ο `/home/users/thanassis/documents` ενώ το όνομα του αρχείου είναι `students.txt`. Το `.txt` στο τέλος του ονόματος λέγεται επέκταση (**extension**) και συνήθως να περιγράφει τον τύπο του αρχείου.

# Ο τύπος FILE

Ο τύπος **FILE** ορίζεται στην κεφαλίδα `stdio.h` και μας επιτρέπει να αναφερόμαστε σε αρχεία που άνοιξε το πρόγραμμά μας.

```
#include <stdio.h>

int main() {
    printf("Size of FILE type: %zu\n", sizeof(FILE));
    return 0;
}
```

```
$ ./filedef
216
```

216 bytes σε ένα σύστημα Debian! Τι περιέχει εντός;  
Πολλά και διάφορα ίσως το συζητήσουμε άλλη φορά

# Η συνάρτηση fopen

Η συνάρτηση **fopen** επιτρέπει στο πρόγραμμά μας να ανοίξει ένα αρχείο και να επιστρέψει έναν δείκτη σε FILE. Επιστρέφει NULL αν αποτύχει να ανοίξει το αρχείο για οποιοδήποτε λόγο. Η δήλωση της συνάρτησης είναι

```
FILE *fopen(const char *restrict pathname, const char *restrict mode);
```

**pathname:** το μονοπάτι που αντιστοιχεί στο αρχείο

**mode:** με ποιο τρόπο να ανοίξουμε το αρχείο (διάβασμα ή γράψιμο;)

**DESCRIPTION**

The **fopen()** function opens the file whose name is the string pointed to by pathname and associates a stream with it.

The argument mode points to a string beginning with one of the following sequences (possibly followed by additional characters, as described below):

- r** Open text file for reading. The stream is positioned at the beginning of the file.
- r+** Open for reading and writing. The stream is positioned at the beginning of the file.
- w** Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
- w+** Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
- a** Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
- a+** Open for reading and appending (writing at end of file). The file is created if it does not exist. Output is always appended to the end of the file. POSIX is silent on what the initial read position is when using this mode. For glibc, the initial file position for reading is at the beginning of the file, but for Android/BSD/MacOS, the initial file position for reading is at the end of the file.

# Παράδειγμα: Άνοιγμα Αρχείων

```
#include <stdio.h>

int main() {
    FILE *fileToRead, *fileToWrite;
    fileToRead = fopen("input.txt", "r");
    if (!fileToRead) {
        return 1;
    }
    fileToWrite = fopen("output.txt", "w");
    if (!fileToWrite) {
        return 1;
    }
    return 0;
}
```

Ελέγχουμε πάντα το αποτέλεσμα της fopen αν είναι NULL. Για ποιο λόγο μπορεί να αποτύχει;

# Παράδειγμα: Άνοιγμα Αρχείων

```
#include <stdio.h>

int main() {
    FILE *fileToRead, *fileToWrite;
    fileToRead = fopen("input.txt", "r");
    if (!fileToRead) {
        return 1;
    }
    fileToWrite = fopen("output.txt", "w");
    if (!fileToWrite) {
        return 1;
    }
    return 0;
}
```

Ελέγχουμε πάντα το αποτέλεσμα της fopen αν είναι NULL. Για ποιο λόγο μπορεί να αποτύχει;

1. Το αρχείο δεν υπάρχει
2. Ο φάκελος δεν υπάρχει
3. Δεν έχουμε ελεύθερο δίσκο για να γράψουμε αρχείο!
4. Δεν έχουμε δικαιώματα να φτιάξουμε αρχείο
5. Έχουμε ανοίξει τον μέγιστο επιτρεπτό αριθμό αρχείων
6. ...

## Η συνάρτηση `fclose`

Η συνάρτηση `fclose` μας επιτρέπει να κλείσουμε ένα αρχείο. Επιστρέφει 0 αν επιτύχει ή EOF αν αποτύχει. Η δήλωση της συνάρτησης είναι

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

**Πάντα** κλείνουμε τα αρχεία που άνοιξε το πρόγραμμά μας αφού τελειώσουμε με την χρήση τους.

# Παράδειγμα: Κλείσιμο Αρχείων

```
#include <stdio.h>

int main() {
    FILE *fileToRead, *fileToWrite;

    fileToRead = fopen("input.txt", "r");
    ...

    fileToWrite = fopen("output.txt", "w");
    ...

    fclose(fileToRead);
    fclose(fileToWrite);

    return 0;
}
```

Σε κάθε fopen πρέπει να αντιστοιχεί ένα fclose



## Η συνάρτηση `fread`

Η συνάρτηση `fread` μας επιτρέπει να διαβάσουμε bytes από ένα ανοιχτό αρχείο. Επιστρέφει το πλήθος των δεδομένων που διαβάστηκαν . Η δήλωση της συνάρτησης είναι

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *restrict stream);
```

**ptr**: δείκτης στην μνήμη όπου θα αποθηκευτούν τα δεδομένα εισόδου

**size**: ο αριθμός bytes κάθε δεδομένου που θα διαβαστεί

**nmemb**: πόσα δεδομένα να προσπαθήσει να διαβάσει

**stream**: δείκτης στο ανοιχτό αρχείο

# Παράδειγμα: Διάβασμα Αρχείων

```
#include <stdio.h>

int main() {

    FILE *fileToRead, *fileToWrite;

    fileToRead = fopen("input.txt", "r");

    if (!fileToRead) return 1;

    char buffer[1024];

    size_t bytesRead = fread(buffer, sizeof(char), 1023, fileToRead);

    buffer[bytesRead] = '\0';

    printf("# of bytes read: %d\n", bytesRead);

    printf("String read: %s\n", buffer);

    fclose(fileToRead);

    return 0;

}
```

# Παράδειγμα: Διάβασμα Αρχείων

```
#include <stdio.h>

int main() {

    FILE *fileToRead, *fileToWrite;

    fileToRead = fopen("input.txt", "r");

    if (!fileToRead) return 1;

    char buffer[1024];

    size_t bytesRead = fread(buffer, sizeof(char), 1023, fileToRead);

    buffer[bytesRead] = '\0';

    printf("# of bytes read: %d\n", bytesRead);

    printf("String read: %s\n", buffer);

    fclose(fileToRead);

    return 0;

}
```

```
$ ./fread
$ echo hello > input.txt
$ ./fread
# of bytes read: 6
String read: hello
```

## Παράδειγμα #2: Διάβασμα Αρχείων

```
#include <stdio.h>

int main() {

    FILE *fileToRead, *fileToWrite;

    fileToRead = fopen("input.txt", "r");

    if (!fileToRead) return 1;

    int buffer[1024];

    size_t integersRead = fread(buffer, sizeof(int), 1023, fileToRead);

    printf("# of integers read: %d\n", integersRead);

    printf("Integer read: %d %08x\n", buffer[0], buffer[0]);

    fclose(fileToRead);

    return 0;

}
```

## Παράδειγμα #2: Διάβασμα Αρχείων

```
#include <stdio.h>

int main() {
    FILE *fileToRead, *fileToWrite;
    fileToRead = fopen("input.txt", "r");
    if (!fileToRead) return 1;
    int buffer[1024];
    size_t integersRead = fread(buffer, sizeof(int), 1023, fileToRead);
    printf("# of integers read: %d\n", integersRead);
    printf("Integer read: %d %08x\n", buffer[0], buffer[0]);
    fclose(fileToRead);
    return 0;
}
```

```
$ ./freadint
# of integers read: 1
Integer read: 1819043176 6c6c6568
```

What?!

```
$ hexdump -C input.txt
00000000 68 65 6c 6c 6f 0a |hello.|
```

# Endianness

**Endianness** λέμε τον τρόπο με τον οποίο οι ακέραιοι αποθηκεύονται στην μνήμη. Οι ακέραιοι αποτελούνται από πολλά bytes και επομένως πρέπει να αποφασίσουμε αν τους αποθηκεύουμε από το μικρότερο στο μεγαλύτερο (little endian) ή από το μεγαλύτερο στο μικρότερο (big endian).



Το πρώτο byte στην μνήμη είναι το μικρότερο byte του αριθμού

**6c6c6568**

# Παράδειγμα Endianness

Τι θα τυπώσει το παρακάτω:

```
#include <stdio.h>

int main() {
    int x = 42;
    char * bytes = (char*)&x;
    int i;
    for(i = 0; i < sizeof(int) / sizeof(char); i++)
        printf("%02x\n", bytes[i]);
    return 0;
}
```

# Παράδειγμα Endianness

Τι θα τυπώσει το παρακάτω:

```
#include <stdio.h>

int main() {
    int x = 42;
    char * bytes = (char*)&x;
    int i;
    for(i = 0; i < sizeof(int) / sizeof(char); i++)
        printf("%02x\n", bytes[i]);
    return 0;
}
```

```
$ ./int
2a
00
00
00
```



# Η συνάρτηση `fwrite`

Η συνάρτηση `fwrite` μας επιτρέπει να γράψουμε έναν αριθμό δεδομένων σε ένα αρχείο. Επιστρέφει τον αριθμό των στοιχείων που γράφτηκαν. Η δήλωση της συνάρτησης είναι

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *restrict stream);
```

**ptr:** δείκτης στην μνήμη απ'όπου θα διαβάσουμε τα δεδομένα εξόδου

**size:** ο αριθμός bytes κάθε δεδομένου που θα γραφτεί

**nmemb:** πόσα δεδομένα να προσπαθήσει να γράψει

**stream:** δείκτης στο ανοιχτό αρχείο

# Παράδειγμα: Γράψιμο Αρχείων

```
#include <stdio.h>

int main() {

    FILE *fileToWrite;

    fileToWrite = fopen("output.txt", "w");

    if (!fileToWrite) return 1;

    int numbers[4] = {0x42, 0x43, 0x44, 0x45};

    size_t numsWritten = fwrite(numbers, sizeof(int), 4, fileToWrite);

    printf("Wrote: %zu numbers\n", numsWritten);

    fclose(fileToWrite);

    return 0;

}
```

# Παράδειγμα: Γράψιμο Αρχείων

```
#include <stdio.h>

int main() {

    FILE *fileToWrite;

    fileToWrite = fopen("output.txt", "w");

    if (!fileToWrite) return 1;

    int numbers[4] = {0x42, 0x43, 0x44, 0x45};

    size_t numsWritten = fwrite(numbers, sizeof(int), 4, fileToWrite);

    printf("Wrote: %zu numbers\n", numsWritten);

    fclose(fileToWrite);

    return 0;
}
```

```
$ ./fwrite
```

```
Wrote: 4 numbers
```

```
$ hexdump -C output.txt
```

```
00000000  42 00 00 00 43 00 00 00  44 00 00 00 45 00 00 00
|B...C...D...E...|
```

# File Descriptor (FD)

Κάθε ανοιχτό αρχείο για ένα πρόγραμμα έχει έναν μοναδικό αριθμό που λέγεται file descriptor. Μπορούμε να βρούμε αυτόν τον αριθμό με την συνάρτηση `fileno`.

```
FILE *fileToRead, *fileToWrite;
fileToRead = fopen("input.txt", "r");
fileToWrite = fopen("output.txt", "w");
// ...
printf("fileToRead: %d\n", fileno(fileToRead));
printf("fileToWrite: %d\n", fileno(fileToWrite));
printf("stdin: %d\n", fileno(stdin));
printf("stdout: %d\n", fileno(stdout));
printf("stderr: %d\n", fileno(stderr));
```

```
$ ./fileno
fileToRead: 3
fileToWrite: 4
stdin: 0
stdout: 1
stderr: 2
```

## stdin, stdout και stderr

Τα **stdin**, **stdout** και **stderr** είναι δείκτες σε ανοικτά αρχεία που αρχικοποιούνται όταν το πρόγραμμα ξεκινά την εκτέλεσή του και κλείνουν όταν τερματίζει.

**stdin**: αντιστοιχεί στην πρότυπη είσοδο του προγράμματος και συνήθως έχει file descriptor 0.

**stdout**: αντιστοιχεί στην πρότυπη έξοδο του προγράμματος και συνήθως έχει file descriptor 1.

**stderr**: αντιστοιχεί στην έξοδο σφάλματος του προγράμματος και συνήθως έχει file descriptor 2.

Σύγκρινε το `find / -name foo` με το `find / -name foo 2> error.txt`

## Η συνάρτηση `fscanf`

Η συνάρτηση `fscanf` είναι όμοια με την `scanf`, απλά αντί να διαβάζει από το `stdin`, μπορεί να διαβάσει από οποιοδήποτε αρχείο. Η συνάρτηση έχει την ακόλουθη μορφή:

```
int fscanf(FILE *stream, const char *format, ...);
```

Η κλήση `scanf(x, y, z)` είναι ουσιαστικά ισοδύναμη με την `fscanf(stdin, x, y, z)`

## Η συνάρτηση `fprintf`

Η συνάρτηση `fprintf` είναι όμοια με την `printf`, απλά αντί να γράφει στο `stdout`, μπορεί να γράψει σε οποιοδήποτε αρχείο. Η συνάρτηση έχει την ακόλουθη μορφή:

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

Η κλήση `printf(x, y, z)` είναι ουσιαστικά ισοδύναμη με την `fprintf(stdout, x, y, z)`

# Παράδειγμα χρήσης fprintf/fscanf

```
#include <stdio.h>

int main() {

    FILE *fileToRead, *fileToWrite;

    fileToRead = fopen("input.txt", "r");

    fileToWrite = fopen("output.txt", "w");

    // ...

    int num;

    fscanf(fileToRead, "%d", &num);

    fprintf(fileToWrite, "Number: %d\n", num);

    // ...

    return 0;

}
```



# Παράδειγμα χρήσης fprintf/fscanf

```
#include <stdio.h>

int main() {
    FILE *fileToRead, *fileToWrite;

    fileToRead = fopen("input.txt", "r");

    fileToWrite = fopen("output.txt", "w");

    // ...

    int num;

    fscanf(fileToRead, "%d", &num);

    fprintf(fileToWrite, "Number: %d\n", num);

    // ...

    return 0;
}
```

```
$ echo "    42" > input.txt
$ ./stream
$ cat output.txt
Number: 42
```

## Άλλες χρήσιμες συναρτήσεις

```
char *fgets(char *s, int size, FILE *stream);
```

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

```
int fseek(FILE *stream, long offset, int whence);
```

```
int fgetc(FILE *stream);
```

```
int fputc(int c, FILE *stream);
```

## Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου καλύψαμε τις σελίδες 136-151.

- [Scanf specifiers](#) και [άλλα](#)
- [Fread](#)
- [File descriptor](#)
- [Endianness](#)

Ευχαριστώ και καλό σαββατοκύριακο εύχομαι!  
Keep Coding ;)