



Περιεχόμενα

Κεφάλαιο 9: Είσοδος και έξοδος	1
Είσοδος και έξοδος	1
Αντιγραφή αρχείων	8
Εκτύπωση γραμμών εισόδου με αντίστροφη σειρά	9

Κεφάλαιο 9: Είσοδος και έξοδος

Είσοδος και έξοδος

- Κάθε πρόγραμμα πρέπει να είναι σε θέση να επικοινωνεί με το περιβάλλον του, δηλαδή να διαβάζει δεδομένα και να γράφει αποτελέσματα, όποτε το χρειάζεται.
- Στην C, οι μηχανισμοί εισόδου/εξόδου παρέχονται από μία σειρά συναρτήσεων, οι οποίες περιλαμβάνονται στην πρότυπη βιβλιοθήκη εισόδου/εξόδου της γλώσσας. Όταν χρησιμοποιούμε συναρτήσεις της βιβλιοθήκης αυτής, πρέπει να συμπεριλαμβάνουμε στα προγράμματά μας και το σχετικό αρχείο επικεφαλίδας:

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

- Οι μονάδες εισόδου/εξόδου είναι τα ρεύματα. Μπορούμε να συσχετίσουμε ένα αρχείο, ανοίγοντάς το, με ένα ρεύμα, αλλά υπάρχουν και προκαθορισμένα ρεύματα, αυτό της πρότυπης εισόδου (stdin), της πρότυπης εξόδου (stdout) και της πρότυπης εξόδου για διαγνωστικά μηνύματα (stderr).
- Κάθε πρόγραμμα έχει, κατ' αρχήν, αντιστοιχίσει στο ρεύμα stdin το πληκτρολόγιο και στα ρεύματα stdout και stderr την οθόνη.
- Η διαφορά μεταξύ stdout και stderr συνίσταται στο ότι συνηθίζεται τα προγράμματα να εκτυπώνουν τα αποτελέσματά τους στο stdout και διαγνωστικά μηνύματα στο stderr.

- Σε πολλά περιβάλλοντα (Unix, Command Prompt των Windows, Cygwin, κλπ.) υπάρχει η δυνατότητα ένα πρόγραμμα που διαβάζει τα δεδομένα του από την πρότυπη είσοδο (stdin) να μπορεί να κληθεί με ανακατεύθυνση της πρότυπης εισόδου σε κάποιο αρχείο, οπότε θα διαβάζει πλέον τα δεδομένα του από το αρχείο αυτό. Παράδειγμα:

```
% ./myprog < inp_file
```

Εδώ, το πρόγραμμα ./myprog, αντί να διαβάσει τα δεδομένα του από το πληκτρολόγιο, θα τροφοδοτηθεί με αυτά από το αρχείο inp_file.

- Ομοίως, μπορούμε να έχουμε και ανακατεύθυνση της πρότυπης εξόδου (stdout) σε αρχείο, οπότε, αντί να εκτυπώνονται τα αποτελέσματα ενός προγράμματος στην οθόνη, φυλάσσονται στο αρχείο αυτό. Παράδειγμα:

```
% ./otherprog arg1 > out_file
```

Εδώ, αντί να εκτυπωθούν τα αποτελέσματα του προγράμματος ./otherprog στην οθόνη, θα φυλαχθούν στο αρχείο out_file.

- Επίσης, μπορούμε να δημιουργήσουμε σωληνώσεις από προγράμματα, δηλαδή η έξοδος του πρώτου προγράμματος της σωλήνωσης να είναι είσοδος για το δεύτερο πρόγραμμα, του οποίου η έξοδος να είναι είσοδος για το τρίτο, κ.ο.κ. Παράδειγμα:

```
% ./prog1 22 13 | ./prog2 | ./prog3 27 | ./prog4 10 99
```

Εδώ, τα προγράμματα ./prog1, ./prog2, ./prog3 και ./prog4 συμμετέχουν σε μία σωλήνωση, όπου η έξοδος καθενός είναι είσοδος στο επόμενο πρόγραμμα στη σωλήνωση.

- Επισημαίνεται ότι τις ανακατευθύνσεις (εισόδου και εξόδου) και τις σωληνώσεις τις διαχειρίζεται το περιβάλλον εκτέλεσης των προγραμμάτων (λειτουργικό σύστημα ή οτιδήποτε άλλο). Τα προγράμματα που συμμετέχουν σ' αυτές ποτέ δεν το μαθαίνουν. Με άλλα λόγια, οι χαρακτήρες <, > και |, καθώς και τα ονόματα αρχείων στις ανακατευθύνσεις **δεν είναι ορίσματα στη γραμμή εντολών** και δεν μπορούμε (και δεν χρειάζεται) να τα προσπελάσουμε μέσω argc και argv.
- Στη συνέχεια, περιγράφονται συνοπτικά οι πιο συχνά χρησιμοποιούμενες συναρτήσεις της πρότυπης βιβλιοθήκης εισόδου/εξόδου
- Περί εισόδου/εξόδου στην C, υπάρχει εκτεταμένη αναφορά στο Κεφάλαιο 7 του [KR] (σελ. 211–233).
- `int getchar(void)`
Επιστρέφει τον επόμενο χαρακτήρα από το stdin, η EOF, αν έχει διαβαστεί όλη η είσοδος.
- `int putchar(int ch)`
Γράφει τον χαρακτήρα ch στο stdout. Επιστρέφει τον χαρακτήρα αυτό, ή EOF σε περίπτωση λάθους.
- `int puts(const char *s)`
Γράφει τη συμβολοσειρά s στο stdout και αμέσως μετά μία αλλαγή γραμμής. Επιστρέφει

μη αρνητική τιμή σε επιτυχία, ή EOF σε περίπτωση λάθους.¹

- `int printf(const char *format, <op>1, <op>2, ...)` Γράφει στο `stdout` τα ορίσματα `<op>1, <op>2, ...`, σύμφωνα με την οδηγία φόρμας `format`, και επιστρέφει το πλήθος των χαρακτήρων που γράφτηκαν.
 - Η συμβολοσειρά `format` περιλαμβάνει χαρακτήρες που μεταφέρονται αυτούσιοι στην έξοδο και προδιαγραφές μετατροπής για την εκτύπωση των `<op>1, <op>2, ...`
 - Μία προδιαγραφή μετατροπής αρχίζει με τον χαρακτήρα `%` και τερματίζεται με ένα χαρακτήρα που δείχνει το είδος της μετατροπής που πρέπει να γίνει.
 - Η γενική μορφή μίας προδιαγραφής μετατροπής είναι η εξής: `%[τρ][επ][.ακρ][μμ]χμ` Ό,τι βρίσκεται μέσα σε `[ και ]`, μπορεί και να μην υπάρχει.
 - Μεταξύ του `%` και του χαρακτήρα μετατροπής (`χμ`) μπορεί να υπάρχουν:
 - * Κάποιοι τροποποιητές (`τρ`), με οποιαδήποτε σειρά, δηλαδή ένα `-`, που επιβάλλει αριστερή στοίχιση, ή/και ένα `+`, που καθορίζει ότι ο αριθμός θα εμφανίζεται πάντα με πρόσημο, ή/και ένα `0`, που, για αριθμούς, επιβάλλει τη συμπλήρωση του πλάτους πεδίου με μηδενικά στην αρχή, εφ' όσον έχουμε δεξιά στοίχιση, δηλαδή αν δεν έχει δοθεί και το `-` σαν τροποποιητής.
 - * Το ελάχιστο πλάτος πεδίου (`επ`). Συμπληρώνονται κενά αριστερά ή δεξιά, ανάλογα με τη στοίχιση, ή μηδενικά στην αρχή αν έχουμε δεξιά στοίχιση και έχει δοθεί και η ένδειξη `0` ότι θέλουμε συμπλήρωση με μηδενικά.
 - * Μία τελεία `.` και ένας αριθμός, η ακρίβεια (`ακρ`), που καθορίζει το μέγιστο αριθμό χαρακτήρων που θα εκτυπωθούν για ένα αλφαριθμητικό, τον αριθμό των ψηφίων² για αριθμό κινητής υποδιαστολής³ ή τον ελάχιστο αριθμό ψηφίων για ακέραιο.
 - * Το μέγεθος μεταβλητής (`μμ`), δηλαδή `h`, `l` ή `L`, αν πρόκειται για εκτύπωση `short int`, `long int` ή `long double`, αντίστοιχα.
 - Πιθανοί χαρακτήρες μετατροπής είναι:
 - * `d`: Για αριθμούς στο δεκαδικό σύστημα.
 - * `o`: Για απροσήμεστους οκταδικούς αριθμούς.
 - * `x`, `X`: Για απροσήμεστους δεκαεξαδικούς αριθμούς. Με μετατροπή `x`, τα “ψηφία” μετά το 9 εμφανίζονται πεζά (`a`, `b`, `c`, `d`, `e`, `f`), ενώ με μετατροπή `X`, εμφανίζονται κεφαλαία (`A`, `B`, `C`, `D`, `E`, `F`).
 - * `u`: Για απροσήμεστους αριθμούς στο δεκαδικό σύστημα.
 - * `c`: Για χαρακτήρες.
 - * `s`: Για συμβολοσειρές.
 - * `f`: Για αριθμούς κινητής υποδιαστολής, σε αναπαράσταση χωρίς εκθέτη.
 - * `e`, `E`: Για αριθμούς κινητής υποδιαστολής, σε αναπαράσταση με εκθέτη. Το

¹Υπάρχει και η “αδελφή” συνάρτηση `char *gets(char *s)`, για την ανάγνωση από το `stdin` μίας συμβολοσειράς, αλλά δεν πρέπει να χρησιμοποιείται, για λόγους ασφαλείας. Μάλιστα, ο `gcc` δίνει σχετική προειδοποίηση όταν χρησιμοποιούμε αυτήν τη συνάρτηση σ' ένα πρόγραμμα. Αντ' αυτής, θα δούμε στη συνέχεια την ασφαλή συνάρτηση `fgets`.

²δεκαδικών για μετατροπές `f`, `e` ή `E` ή σημαντικών για μετατροπές `g` ή `G`

³αν δεν δοθεί ακρίβεια, αυτή θεωρείται ίση με 6

σύμβολο της ύψωσης σε δύναμη του 10 εμφανίζεται σαν e ή E, ανάλογα με τον χαρακτήρα μετατροπής που χρησιμοποιήθηκε.

- * g, G: Για αριθμούς κινητής υποδιαστολής, σε αναπαράσταση με ή χωρίς εκθέτη, ανάλογα με το μέγεθος του εκθέτη σε σχέση με την ακρίβεια που έχει ζητηθεί. Αν ο εκθέτης είναι μικρότερος του -4 ή μεγαλύτερος από ή ίσος με την ακρίβεια, εφαρμόζεται μετατροπή e ή E (ανάλογα με το αν έχουμε g ή G), αλλιώς εφαρμόζεται μετατροπή f. Τα μηδενικά και η υποδιαστολή στο τέλος δεν εμφανίζονται.

- * %: Για εκτύπωση του %.

– Παραδείγματα εκτυπώσεων:

Προδιαγραφή	int x = 3456	Προδιαγραφή	double d = -3768.08549
%d	3456	%f	-3768.085490
%7d	3456	%14.4f	-3768.0855
%2d	3456	%-14.3f	-3768.085
%-7d	3456	%e	-3.768085e+03
%+7d	+3456	%14.4E	-3.7681E+03
%07d	0003456	%-14.3e	-3.768e+03
%7.5d	03456	%g	-3768.09
%7o	6600	%14.4g	-3768
%-7x	d80	%-14.3G	-3.77E+03
%7.6X	000D80	%014.4f	-00003768.0855

Προδιαγραφή	char *p = "this is a test phrase"
%s	this is a test phrase
%25s	this is a test phrase
%12s	this is a test phrase
%.12s	this is a te
%-25s	this is a test phrase
%25.12s	this is a te
%-25.12s	this is a te

Το σύμβολο δείχνει ένα κενό διάστημα στην έξοδο.

- `int scanf(const char *format, <op>1, <op>2, ...)` Διαβάζει από το `stdin` δεδομένα, σύμφωνα με την οδηγία φόρμας `format`, και τα αποθηκεύει εκεί που δείχνουν οι **δείκτες** `<op>1`, `<op>2`, Επιστρέφει το πλήθος των δεδομένων που διαβάστηκαν ή EOF, αν έχει διαβαστεί όλη η είσοδος.

– Ένα δεδομένο εισόδου είναι μία ακολουθία από μη λευκούς χαρακτήρες που

- τερματίζει στον επόμενο λευκό χαρακτήρα,⁴ ή στον επόμενο χαρακτήρα που αναμένεται να διαβαστεί με βάση το `format`, ή όταν εξαντληθεί το πλάτος πεδίου.
- Η συμβολοσειρά `format` μπορεί να περιλαμβάνει:
 - * Κενά ή στηλογνώμονες, που επιβάλλουν την κατανάλωση όλων των λευκών χαρακτήρων από την τρέχουσα θέση της εισόδου και μετά.
 - * Άλλους χαρακτήρες, εκτός από τον `%`, που πρέπει να συμφωνούν με τον επόμενο μη λευκό χαρακτήρα στην είσοδο, αλλιώς η `scanf` σταματά το διάβασμα.
 - * Προδιαγραφές μετατροπής, που σχηματίζονται, με αυτή τη σειρά, από:
 - τον χαρακτήρα `%`
 - ένα προαιρετικό χαρακτήρα `*`, που δείχνει ότι το δεδομένο που διαβάστηκε δεν πρέπει να φυλαχθεί
 - ένα προαιρετικό αριθμό που καθορίζει το μέγιστο πλάτος πεδίου
 - ένα προαιρετικό `h`, για την ανάγνωση `short int`, ή ένα `l`, για `long int` ή `double`, ή ένα `L` για `long double`.
 - ένα χαρακτήρα μετατροπής, αντίστοιχα με αυτούς της `printf` (`d`, `o`, `x`, `u`, `c`, `s`, `f`, `e`, `g`, `%`) ή `[...]`, το οποίο προκαλεί την ανάγνωση της μεγαλύτερης ακολουθίας χαρακτήρων στην είσοδο, από αυτούς που περιλαμβάνονται μέσα στα `[ και ]`, ή `[^...]`, το οποίο είναι παρόμοιο με το προηγούμενο, αλλά διαβάζονται χαρακτήρες που δεν περιέχονται μέσα στα `[ και ]`
 - Επισημαίνεται ότι η χρήση της `scanf` για ανάγνωση συμβολοσειρών παρουσιάζει παρόμοιο πρόβλημα ασφάλειας με αυτό της συνάρτησης `gets`, το οποίο όμως αντιμετωπίζεται αν δηλωθεί και μέγιστο πλάτος πεδίου (π.χ. `%20s`) για την ανάγνωση.
- `FILE *fopen(const char *filename, const char *mode)`
 Ανοίγει το αρχείο με όνομα `filename` για να το χρησιμοποιήσουμε με τον τρόπο που περιγράφεται στο `mode`. Επιστρέφει ένα ρεύμα (ή δείκτη σε αρχείο), με βάση το οποίο μπορούμε στη συνέχεια να αναφερόμαστε στο αρχείο, ή `NULL`, αν για κάποιο λόγο δεν ήταν δυνατόν να ανοίξει το αρχείο.
 - Το `FILE` είναι μία δομή (typedef'ed από το `stdio.h`) με κατάλληλα μέλη για να γίνεται ο χειρισμός του αρχείου.
 - Το `mode` μπορεί να είναι:
 - * `"r"`: Για διάβασμα από υπάρχον αρχείο.
 - * `"w"`: Για γράψιμο σε αρχείο. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν υπάρχει, τα προηγούμενα περιεχόμενά του διαγράφονται και το γράψιμο αρχίζει από την αρχή του αρχείου.
 - * `"a"`: Για γράψιμο σε αρχείο με προσάρτηση στο τέλος του των νέων δεδομένων, αν το αρχείο υπάρχει ήδη, χωρίς διαγραφή των προηγούμενων περιεχομένων

⁴κενό, στηλογνώμονας ή αλλαγή γραμμής

- του.
- * "r+": Για διάβασμα και γράψιμο, οπουδήποτε μέσα σε υπάρχον αρχείο, χωρίς διαγραφή των προηγούμενων περιεχομένων του.
 - * "w+": Για διάβασμα και γράψιμο, οπουδήποτε μέσα στο αρχείο, με διαγραφή των προηγούμενων περιεχομένων του, εφ' όσον αυτό υπάρχει ήδη.
 - * "a+": Για διάβασμα από οπουδήποτε μέσα από το αρχείο και γράψιμο μόνο στο τέλος του, χωρίς διαγραφή των προηγούμενων περιεχομένων του.
- Σε ορισμένα συστήματα, γίνεται διάκριση μεταξύ αρχείων κειμένου και δυαδικών αρχείων (στο Unix πάντως, όχι), οπότε εκεί, αν θέλουμε να χειριστούμε ένα δυαδικό αρχείο, πρέπει στο mode να προστεθεί και ο χαρακτήρας b, δηλαδή να έχουμε, ανάλογα με την περίπτωση "rb", "wb", "ab", "r+b" ("rb+"), "w+b" ("wb+"), ή "a+b" ("ab+").
- `int fclose(FILE *fp)`
Κλείνει το ρεύμα `fp`. Επιστρέφει 0 ή EOF, σε περίπτωση επιτυχίας ή αποτυχίας, αντίστοιχα.
 - `char *fgets(char *buf, int max, FILE *fp)`
Διαβάζει το πολύ `max-1` χαρακτήρες από το ρεύμα `fp`, μέχρι την αλλαγή γραμμής (`\n`). Οι χαρακτήρες που διαβάστηκαν (μαζί και με το `\n`) φυλάσσονται στη συμβολοσειρά `buf`, η οποία τερματίζεται κανονικά με `\0`. Επιστρέφει το `buf`, ή NULL αν έχει διαβαστεί όλη η είσοδος.⁵
 - `int feof(FILE *fp)`
Επιστρέφει μη μηδενική τιμή αν έχουν διαβαστεί όλα τα δεδομένα από το ρεύμα `fp`, δηλαδή έχουμε φτάσει στο τέλος του αρχείου που αντιστοιχεί στο ρεύμα, ή, αλλιώς, 0.
 - `int getc(FILE *fp)`
Επιστρέφει τον επόμενο χαρακτήρα από το ρεύμα `fp`, ή EOF, αν έχουμε φτάσει στο τέλος του αντίστοιχου αρχείου.
 - `int putc(int ch, FILE *fp)`
Γράφει τον χαρακτήρα `ch` στο ρεύμα `fp`. Επιστρέφει τον χαρακτήρα αυτό, ή EOF σε περίπτωση λάθους.
 - `int ungetc(int ch, FILE *fp)`
Γυρίζει πίσω στο ρεύμα `fp` τον χαρακτήρα `ch`, έτσι ώστε να διαβαστεί πάλι σε επόμενη ανάγνωση. Επιστρέφει τον χαρακτήρα αυτό, ή EOF σε περίπτωση λάθους.
 - `int fprintf(FILE *fp, const char *format, <op>1, <op>2, ...)`

⁵Επισημώς, υπάρχει και η συνάρτηση `char *gets(char *buf)`, η οποία κάνει περίπου ό,τι και η `fgets`. Διαβάζει από το ρεύμα `stdin`, χωρίς να βάζει κάποιο πάνω όριο στο πλήθος των χαρακτήρων που θα διαβαστούν και, γι' αυτό ακριβώς, για λόγους ασφαλείας, αποθαρρύνεται η χρήση της. Επίσης, η `gets` δεν τοποθετεί τον χαρακτήρα αλλαγής γραμμής (`\n`) που διάβασε μέσα στο `buf`. Αν θέλουμε να διαβάσουμε μία γραμμή χαρακτήρων από την πρότυπη είσοδο, μπορούμε να χρησιμοποιήσουμε την `fgets`, δίνοντας σαν τελευταίο όρισμα το `stdin`, αντί να χρησιμοποιήσουμε την `gets`.

Ίδια με την `printf`, μόνο που γράφει στο ρεύμα `fp`, αντί για το `stdout`.

- `int fscanf(FILE *fp, const char *format, <op>1, <op>2, ...)`

Ίδια με την `scanf`, μόνο που διαβάζει από το ρεύμα `fp`, αντί από το `stdin`.

- `int sprintf(char *str, const char *format, <op>1, <op>2, ...)`

Ίδια με την `printf`, μόνο που γράφει στη συμβολοσειρά `str`, αντί για το ρεύμα `stdout`.

- `int sscanf(char *str, const char *format, <op>1, <op>2, ...)`

Ίδια με την `scanf`, μόνο που διαβάζει από τη συμβολοσειρά `str`, αντί από το ρεύμα `stdin`.

- `size_t fread(void *ptr, size_t size, size_t count, FILE *fp)` ⁶

Διαβάζει από το ρεύμα `fp` το πολύ `count` δεδομένα μεγέθους `size` το καθένα και τα τοποθετεί από τη διεύθυνση `ptr` και μετά. Επιστρέφει το πλήθος των δεδομένων που διαβάστηκαν. Ο έλεγχος τέλους της εισόδου μπορεί να γίνει με τη συνάρτηση `feof`.

- `size_t fwrite(const void *ptr, size_t size, size_t count, FILE *fp)`

Γράφει στο ρεύμα `fp` το πολύ `count` δεδομένα μεγέθους `size` το καθένα, παίρνοντάς τα από τη διεύθυνση `ptr` και μετά. Επιστρέφει το πλήθος των δεδομένων που γράφτηκαν. Αν γραφούν λιγότερα από `count` δεδομένα, αυτό θα οφείλεται σε κάποιο λάθος που συνέβη.

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

Θέτει την τρέχουσα θέση στο ρεύμα `fp` να είναι:

- `offset` (≥ 0) χαρακτήρες από την αρχή, αν το `origin` έχει τεθεί `SEEK_SET`.
- `offset` χαρακτήρες από την τρέχουσα θέση, αν το `origin` έχει τεθεί `SEEK_CUR`.
- `offset` χαρακτήρες από το τέλος, αν το `origin` έχει τεθεί `SEEK_END`.

Επιστρέφει 0 σε περίπτωση επιτυχίας.

- `long ftell(FILE *fp)`

Επιστρέφει την τρέχουσα θέση στο ρεύμα `fp`, ή -1L σε περίπτωση λάθους.

- `int fflush(FILE *fp)`

Γράφει στο ρεύμα εξόδου `fp` ό,τι πιθανώς βρίσκεται στην ενδιάμεση περιοχή αποθήκευσης. Επιστρέφει 0 ή EOF, σε περίπτωση επιτυχίας ή αποτυχίας, αντίστοιχα.

- `void perror(const char *s)`

Εκτυπώνει τη συμβολοσειρά `s` και μία περιγραφή του πιο πρόσφατου λάθους που έχει συμβεί.

⁶Ο τύπος `size_t` είναι ο `unsigned int`, typedef'ed.

- Η συνάρτηση `perror` είναι πολύ κατατοπιστική όταν χρησιμοποιούμε συναρτήσεις της πρότυπης βιβλιοθήκης της C, οι οποίες ενδέχεται να προκαλέσουν κάποιο λάθος, για παράδειγμα η `malloc` ή οι συναρτήσεις διαχείρισης αρχείων, όπως η `fopen`, και μας ενδιαφέρει να μάθουμε ποιο ακριβώς λάθος προκλήθηκε.
- Σχετική με τη συνάρτηση `perror` είναι και μία εξωτερική μεταβλητή `errno`, η οποία έχει σαν τιμή έναν ακέραιο που αντιστοιχεί στο πιο πρόσφατο λάθος που έχει συμβεί.
- Αν θέλουμε να χρησιμοποιήσουμε τη μεταβλητή `errno` μέσα σ' ένα πρόγραμμα, πρέπει να έχουμε συμπεριλάβει και το αρχείο επικεφαλίδας στο οποίο ορίζεται:

```
#include <errno.h>
```

Μία επισημάνση που πρέπει να γίνει είναι ότι δεν μπορούμε να χρησιμοποιήσουμε οποιαδήποτε συνάρτηση σε οποιοδήποτε ρεύμα. Εξαρτάται από το `mode` που δόθηκε όταν άνοιξε το αντίστοιχο αρχείο.

Αντιγραφή αρχείων

```
/* File: filecopy.c */
#include <stdio.h>

int main(int argc, char *argv[])
{ FILE *ifp, *ofp;
  int n;
  char buf[1024];
  if (argc != 3) {
    fprintf(stderr,
              "Usage: %s <source-file> <target-file>\n", argv[0]);
    return 1;
  }
  if ((ifp = fopen(argv[1], "rb")) == NULL) { /* Open source file */
    perror("fopen source-file");
    return 1;
  }
  if ((ofp = fopen(argv[2], "wb")) == NULL) { /* Open target file */
    perror("fopen target-file");
    return 1;
  }
  while (!feof(ifp)) { /* While we don't reach the end of source */
    /* Read characters from source file to fill buffer */
    n = fread(buf, sizeof(char), sizeof(buf), ifp);
    /* Write characters read to target file */
    fwrite(buf, sizeof(char), n, ofp);
  }
}
```

```
    fclose(ifp);
    fclose(ofp);
    return 0;
}

% gcc -o filecopy filecopy.c
% ./filecopy
Usage: ./filecopy <source-file> <target-file>
% ./filecopy bla foo
fopen source-file: No such file or directory
%
% cat helloworld.c
/* File: helloworld.c */
#include <stdio.h>

main()
{ printf("Hello world\n");
}
% ./filecopy helloworld.c newhelloworld.c
% cat newhelloworld.c
/* File: helloworld.c */
#include <stdio.h>

main()
{ printf("Hello world\n");
}
% ls -l helloworld.c newhelloworld.c
-rw----- 1 ip www 81 Dec 18 21:45 newhelloworld.c
-rw-r----- 1 ip www 81 Oct 13 12:42 helloworld.c
% diff helloworld.c newhelloworld.c
%
% ./filecopy filecopy Copy\ of\ filecopy
% ls -l filecopy Copy\ of\ filecopy
-rw----- 1 ip www 6884 Dec 18 21:46 Copy of filecopy
-rwx----- 1 ip www 6884 Dec 18 21:34 filecopy*
% cmp filecopy Copy\ of\ filecopy
%
```

Εκτύπωση γραμμών εισόδου με αντίστροφη σειρά

```
/* File: revinput.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
typedef struct Slistnode *Slistptr;

struct Slistnode {                /* A list node structure */
    char *line;                   /* with a string as member */
    Slistptr next;
};

void Sinsert_at_start(Slistptr *, char *);
void Sprint(Slistptr);

int main(void)
{ char buf[1024];
  Slistptr list;
  list = NULL;    /* Initialize list to store lines to be read */
  while (fgets(buf, sizeof buf, stdin) != NULL) /* Read a line */
      /* and insert it at the start of the list */
      Sinsert_at_start(&list, buf);
  Sprint(list);    /* Print the list, i.e. the input reversed */
  return 0;
}

void Sinsert_at_start(Slistptr *ptraddr, char *buf)
    /* The well-known insert_at_start function */
{ Slistptr templist;
  templist = *ptraddr;
  *ptraddr = malloc(sizeof(struct Slistnode));
  (*ptraddr)->line = malloc((strlen(buf)+1) * sizeof(char));
  strcpy((*ptraddr)->line, buf);
  (*ptraddr)->next = templist;
}

void Sprint(Slistptr list)        /* Just print out the list */
{ while (list != NULL) {
    printf("%s", list->line);
    list = list->next; }
}

% gcc -o revinput revinput.c
% ./revinput
These are some lines
to test program revinput.
Ok, one more line.
These are my last words
```

before the minus signs

^D

before the minus signs

These are my last words

Ok, one more line.

to test program revinput.

These are some lines

%

% cat helloworld.c

/* File: helloworld.c */

#include <stdio.h>

main()

{ printf("Hello world\n");

}

%

% ./revinput < helloworld.c

}

{ printf("Hello world\n");

main()

#include <stdio.h>

/* File: helloworld.c */