Ngôn ngữ lập trình C

Bài 10. Nhập, xuất dữ liệu với tệp

Soạn bởi: TS. Nguyễn Bá Ngọc

2021

Nội dung

Các khái niệm Luồng & Tệp

Một số hàm thao tác với tệp

Một số ví dụ đọc/ghi tệp

Nội dung

Các khái niệm Luồng & Tệp

Một số hàm thao tác với tệp

Một số ví dụ đọc/ghi tệp

Khái niệm luồng

Luồng (Stream) là biểu diễn lô-gic của phương tiện nhập, xuất dữ liệu. Luồng có thể được gắn kết với nhiều nguồn nhập và đích xuất khác nhau, ví dụ: Bàn phím, màn hình, tệp v.v..

- Khái niệm luồng giúp trừu tượng hóa các phương tiện nhập, xuất. Người lập trình thực hiện các thao tác nhập, xuất với luồng, và có thể sử dụng cùng 1 tập hàm để thao tác với các phương tiện nhập, xuất khác nhau.
- Trong bài giảng này chúng ta sẽ học cách làm việc với tệp.

Tin mừng: Nhập, xuất dữ liệu với tệp văn bản có nhiều điểm giống với cách nhập từ bàn phím và xuất ra màn hình đã học.

Con trỏ tệp

- Theo thiết kế của thư viện chuẩn của C luồng được được gắn kết với tệp ngoại bằng cách mở tệp.
- Con trỏ tới luồng có kiểu FILE * (stdio.h), vì vậy còn được gọi là *con trỏ tệp*.
  - Định danh FILE được sử dụng từ khi C mới được sáng tạo.
  - *(Có thể gây nhập nhằng giữa khái niệm tệp và luồng?)*
- Các thao tác đọc, ghi với ổ đĩa cứng được quản lý bởi HĐH, số lượng tệp có thể được mở đồng thời phụ thuộc vào môi trường cụ thể và thường ít hơn rất nhiều so với số lượng con trỏ tệp có thể được khai báo trong chương trình.
  - Lệnh mở tệp có thể thất bại.

Các luồng tiêu chuẩn & Điều hướng

Các luồng tiêu chuẩn được mở từ lúc chạy chương trình và được quản lý tự động, mặc định người lập trình không cần viết lệnh mở, hoặc đóng các luồng tiêu chuẩn.

Luồng tiêu chuẩn	**Con trỏ tệp**	**Phương tiện mặc định**
Nhập	stdin	Bàn phím
Xuất (thông tin thường)	stdout	Màn hình
Xuất thông báo lỗi	stderr	Màn hình

- Các hàm nhập, xuất đã học: scanf - đọc từ stdin, printf - xuất ra stdout. *Thực chất chúng ta đã thao tác với luồng từ đầu.*
- Có thể xuất vào stderr bằng các hàm thao tác với tệp.
- Mặc định stdin được gắn kết với bàn phím, stdout và stderr được gắn kết với màn hình dòng lệnh. Tuy nhiên chúng ta có thể điều hướng, gắn kết các luồng tiêu chuẩn với các phương tiện khác (thường là các tệp văn bản).

Điều hướng với tệp văn bản

Có thể gắn kết luồng tiêu chuẩn với tệp văn bản bằng cách bổ xung các tham số vào câu lệnh chạy chương trình, hoặc gọi hàm mở lại tệp trong chương trình.

- Trong môi trường dòng lệnh:
  - Điều hướng luồng nhập (stdin), ví dụ:
    - ./prog < in.txt
    - Gắn kết stdin với tệp in.txt.
  - Điều hướng luồng xuất (stdout), ví dụ:
    - ./prog > out.txt
    - Gắn kết stdout với out.txt
  - Điều hướng luồng xuất thông tin lỗi (stderr), ví dụ:
    - ./prog 2> err.txt
    - Gắn kết stderr với err.txt
  - Điều hướng đồng thời các luồng, ví dụ:
    - ./prog < in.txt > out.txt 2> err.txt
  - *(Cú pháp có thể thay đổi tùy theo môi trường dòng lệnh)*

Điều hướng với tệp văn bản(2)

- Có thể chủ động điều hướng các luồng tiêu chuẩn trong chương trình với hàm freopen:
  FILE *freopen (const char *filename, const char *modes,
  FILE *stream)
  Mở tệp với đường dẫn filename ở chế modes và gắn kết luồng được trỏ tới bởi con trỏ stream với nó. Nếu filename == NULL thì freopen thay đổi chế độ đọc/ghi của luồng.
  Trả về con trỏ stream trong trường hợp thành công, hoặc NULL trong trường hợp ngược lại (thất bại).

- Ví dụ:
  - freopen("in.txt", "r", stdin);
  - freopen("out.txt", "w", stdout);
  - freopen("err.txt", "w", stderr);

Ví dụ 10.1a. Điều hướng trong dòng lệnh

vd10-1.c

```c
#include <stdio.h>

int main() {
  int a, b;

  // Đọc từ stdin
  scanf("%d%d", &a, &b);

  // Xuất ra stdout
  printf("%d + %d = %d\n", a, b, a + b);

  // Xuất ra stderr
  fprintf(stderr, " Test error message\n");
  return 0;
}
```

```
bangoc:$gcc -o prog vd10-1.c
bangoc:$echo 3 5 > inp.txt
bangoc:$./prog < inp.txt > out.txt 2> err.txt
bangoc:$cat out.txt
3 + 5 = 8
bangoc:$cat err.txt
 Test error message
bangoc:$
```

Ví dụ 10.1b. Mở lại luồng tiêu chuẩn

vd10-1b.c

```c
#include <stdio.h>

int main() {
  int a, b;

  // Mở lại các luồng tiêu chuẩn
  if (!freopen("inp.txt", "r", stdin) ||
      !freopen("out.txt", "w", stdout) ||
      !freopen("err.txt", "w", stderr)) {
    printf("Lỗi mở lại tệp.\n");
    return 1;
  }

  scanf("%d%d", &a, &b);  // inp.txt
  printf("%d + %d = %d\n", a, b, a + b); // out.txt
  fprintf(stderr, " Test error message\n");  // err.txt
  return 0;
}
```

```
bangoc:$cat inp.txt
20 50
bangoc:$gcc -o prog vd10-1b.c
bangoc:$./prog
bangoc:$cat out.txt
20 + 50 = 70
bangoc:$cat err.txt
 Test error message
bangoc:$
```

Tệp nhị phân vs. tệp văn bản

Quy chuẩn C mô tả 2 định dạng luồng: Luồng văn bản và luồng nhị phân tương ứng với 2 định dạng tệp cơ bản thường gặp trong thực tế: Tệp văn bản và tệp nhị phân.

- Tệp văn bản có 1 số khác biệt cơ bản so với tệp nhị phân:
  - Tệp văn bản là định dạng cụ thể hơn, chứa nội dung văn bản, tệp nhị phân có thể chứa nội dung bất kỳ.
  - Tệp văn bản được chia thành các dòng, các byte biểu diễn các ký tự của văn bản.
- Phân loại tệp văn bản/nhị phân được thực hiện theo biểu diễn lô-gic, *về bản chất lưu trữ tất cả dữ liệu đều là dãy bits.*
- Triển khai trình biên dịch có thể phân biệt tệp văn bản và tệp nhị phân hoặc không, ví dụ:
  - GNU GCC (trong Linux) không phân biệt.
  - MinGW (trong Windows) có phân biệt.

Tệp nhị phân vs. Tệp văn bản(2)

Tệp văn bản có định dạng đơn giản và có nhiều chương trình ứng dụng cho phép soạn thảo tệp văn bản. Tuy nhiên các chương trình được phát triển độc lập dẫn tới một số khác biệt cần lưu ý khi xử lý tệp văn bản:

- Một số cách biểu diễn dấu hiệu xuống dòng:
  - Cặp ký tự CR ('\r') và LF ('\n'): Phổ biến trong Windows
    - Với MinGW (trong Windows), khi đọc từ luồng văn bản cặp ký tự "\r\n" (có thể) được đọc như 1 ký tự '\n', và khi xuất vào luồng văn bản ký tự '\n' (có thể) được chuyển đổi thành 2 ký tự "\r\n"
    - Với GNU GCC (trong Linux), cặp ký tự "\r\n" luôn được đọc như 2 ký tự '\r' và '\n', ký tự '\n' luôn được giữ nguyên khi xuất ra.
  - 1 Ký tự LF: Phổ biến trong Linux và Mac OS (hiện nay)
    - Phiên bản cũ của Mac OS sử dụng 1 ký tự CR.

Tệp nhị phân vs. Tệp văn bản(3)

- (Dấu hiệu) Kết thúc tệp (EOF - End Of File):
  - Trong một số môi trường (ví dụ CP/M), ký tự 0x1A được sử dụng làm dấu hiệu kết thúc tệp.
    - CP/M là một HĐH ở thời kỳ đầu, hệ thống quản lý tệp không lưu kích thước chính xác của tệp.
  - Trong các HĐH ngày nay, các hệ thống quản lý tệp đều lưu kích thước chính xác của tệp vì vậy không cần lưu ký tự kết thúc tệp trong tệp nữa.
    - Với các triển khai ngày nay kết thúc tệp là một trạng thái của luồng (ví dụ, sau khi đọc hết dữ liệu trong tệp).

Do có nhiều quy ước khác nhau cho định dạng tệp văn bản trong các môi trường khác nhau, sử dụng luồng văn bản có thể giúp duy trì tính khả chuyển của các mã nguồn nhập/xuất với tệp văn bản đơn giản hơn.

Nội dung

Các khái niệm Luồng & Tệp

Một số hàm thao tác với tệp

Một số ví dụ đọc/ghi tệp

Mở tệp

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

- Mở tệp được xác định bằng fname ở chế độ được xác định bằng mode. Hàm trả về con trỏ tới luồng được gắn kết với tệp nếu thành công, hoặc NULL nếu ngược lại.
- Các chế độ mở tệp văn bản:

Chuỗi	**Ý nghĩa**	**Chuỗi**	**Ý nghĩa**
"r"	Đọc	"r+"	Đọc&Ghi từ đầu
"w"	Ghi	"w+"	Đọc&Ghi, xóa nội dung nếu đã có
"a"	Thêm vào cuối	"a+"	Đọc&Ghi (thêm vào cuối nếu đã có)

- Các chế độ mở tệp nhị phân tương tự chế độ mở tệp văn bản được bổ xung ký tự b: "rb", "wb", "ab", "rb+", "wb+", "ab+" (hoặc "r+b", "w+b", "a+b")

Với GNU GCC, ký tự b không làm thay đổi luồng được tạo (không phân biệt luồng nhị phân & luồng văn bản).

Đóng tệp

int fclose(FILE *stream);

Đóng luồng và ngắt kết nối với tệp ngoại. Bộ nhớ đệm xuất được ghi ra tệp, bộ nhớ đệm đọc được hủy bỏ.
Hàm trả về 0 nếu thành công hoặc EOF nếu phát sinh lỗi.

Nhận tên tệp qua tham số dòng lệnh

- Nguyên mẫu hàm main với tham số dòng lệnh:

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

- Các tham số dòng lệnh được truyền cho hàm main như các chuỗi ký tự:
- Ví dụ
  - chạy chương trình với lệnh: ./prog inp.txt out.txt "one two"
  - argv[0] có thể là "./prog" hoặc ""
  - argv[1] là chuỗi "inp.txt"
  - argv[2] - "out.txt"
  - argv[3] có thể là "one two" (tham số chứa dấu cách thường được đặt trong cặp dấu "")

Tham số dòng lệnh là 1 cơ chế hữu ích và phổ biến để truyền tên tệp cho chương trình

Nhập, xuất theo định dạng

Các hàm đã học:
printf(/*.*/) - tương đương với fprintf(stdin, /*.*/)
scanf(/*.*/) - tương đương với fscanf(stdin, /*.*/)

int fprintf(FILE *stream, const char *format, …);
int fscanf(FILE *stream, const char *format, …);
Cặp hàm fprintf và fscanf theo thứ tự có thể xuất và nhập dữ liệu với luồng bất kỳ theo cú pháp tương tự printf và scanf.
Hàm fprintf trả về số lượng ký tự đã truyền cho luồng (có thể khác số lượng bytes thực tế được ghi lên đĩa với luồng văn bản, ví dụ trong môi trường Windows), hoặc giá trị âm nếu phát sinh lỗi. (tương tự printf)
Hàm fscanf trả về số lượng tham số được gán giá trị, có thể ít hơn số lượng tham số thực tế nếu phát sinh lỗi chuyển đổi. Hàm trả về EOF nếu phát sinh lỗi khi chưa đọc được bất kỳ dữ liệu nào. (tương tự scanf)

Bộ nhớ đệm của luồng

- Bộ nhớ đệm được sử dụng để tăng hiệu quả đọc/ghi tệp:
  - Đọc/ghi với ổ đĩa cứng được thực hiện theo khối.
  - Khi đọc dữ liệu vào biến: Biến <= Bộ nhớ đệm <= Tệp
  - Khi xuất dữ liệu vào luồng: Dữ liệu => Bộ nhớ đệm => Tệp
  - Bộ nhớ đệm giúp hạn chế số lần truy cập tới ổ đĩa khi đọc và ghi nhiều lần với lượng dữ liệu nhỏ.
  - Bộ nhớ đệm được quản lý bởi triển khai của thư viện với cấu hình mặc định.
- Các hàm thiết lập vùng nhớ đệm:
  - void setbuf(FILE *stream, char *buf);
  - int setvbuf(FILE *stream, char *buf, int mode, size_t size);
  - *Người lập trình thường không truy cập trực tiếp tới bộ nhớ đệm hay tự thiết lập bộ nhớ đệm, ngoại trừ 1 số trường hợp đặc biệt nếu cấu hình mặc định không đủ hiệu quả.*

Đẩy vùng nhớ đệm của luồng xuất

int fflush(FILE *stream);

Khi áp dụng cho luồng xuất hoặc luồng cập nhật (đọc và ghi) với thao tác cuối cùng không phải thao tác đọc, hàm fflush đẩy dữ liệu trong bộ nhớ đệm của luồng tới phương tiện được gắn kết. Nếu luồng được gắn với tệp thì dữ liệu trong bộ nhớ đệm được ghi ra tệp.

fflush(NULL) - Đẩy tất cả bộ nhớ đệm của các luồng thỏa mãn yêu cầu đã nêu, tương đương với gọi nhiều lệnh fflush với các luồng đang mở.

Nếu thành công fflush trả về 0, nếu ngược lại (phát sinh lỗi) hàm thiết lập cờ lỗi cho luồng và trả về EOF.

Áp dụng fflush *cho luồng nhập như* stdin *là hành vi bất định*

Xóa bộ nhớ đệm của stdin

Trong 1 số trường hợp nhập từ stdin chúng ta cần xóa bộ nhớ đệm, điển hình như nhập 1 ký tự sau khi nhập 1 số với scanf

- Khi nhập 1 số hàm scanf không sử dụng ký tự '\n' (và các ký tự khoảng trắng khác) đứng sau ký tự cuối cùng của số,
- Sau đó để nhập 1 ký tự bất kỳ chúng ta cần xóa bộ nhớ đệm nếu trong thao tác nhập có chứa *khoảng trắng*, ví dụ:
  - scanf("%d", &n);
    - Người dùng nhập 123(Enter), hoặc 123 (Enter), v.v..
    - Bộ nhớ đệm: "123\n"
    - n = 123, ký tự tiếp theo đọc được là '\n'
  - Trong trường hợp này có thể xóa bộ nhớ đệm bằng cách đọc từng ký tự cho tới khi đọc được '\n':
    - while (fgetc(stdin) != '\n') ;
- *!Lưu ý:* fflush(stdin) *tuy được định nghĩa trong* Windows*, nhưng vẫn có thể khó hiểu khi kết hợp với điều hướng.*

Nhập, xuất ký tự

int getchar(void); - Tương đương với fgetc(stdin);
int fgetc(FILE *stream);
Đọc và trả về 1 ký tự như unsigned char từ luồng nếu chưa đọc hết nội dung tệp, hoặc trả về EOF nếu phát sinh lỗi hoặc đã đọc hết nội dung tệp.

int ungetc(int c, FILE *stream);
Trả lại ký tự c vào luồng và xóa trạng thái EOF (nếu có), nếu thành công c sẽ là ký tự được đọc tiếp theo từ luồng và hàm trả về c, nếu ngược lại hàm trả về EOF.

int putchar(int c); - Tương đương với fputc(stdin);
int fputc(int c, FILE *stream);
Viết ký tự c như unsigned char vào luồng. Hàm trả về c nếu thành công, hoặc trả về EOF và thiết lập cờ lỗi nếu thất bại.

EOF *là macro (Được định nghĩa trong* stdio.h: #define EOF (-1)*)*

Nhập, xuất theo dòng

int puts(const char *s); Có hiệu ứng tương tự printf("%s\n", s);
Viết chuỗi s vào luồng được trỏ tới bởi stdout, và thêm dấu hiệu xuống dòng vào luồng. Ký tự null được bỏ qua. Hàm trả về EOF nếu phát sinh lỗi, hoặc giá trị dương nếu ngược lại.
int fputs(const char *s, FILE *stream); - fprintf(stream, "%s", s);
Xuất chuỗi s vào luồng được trỏ tới bởi stream. Hàm trả về EOF nếu phát sinh lỗi hoặc giá trị không âm nếu ngược lại.

char *fgets(char *s, int n, FILE *stream);
Đọc tối đa n - 1 ký tự vào vùng nhớ được trỏ tới bởi s. Dừng sau khi đọc được '\n' (lưu trong s) hoặc luồng chuyển sang trạng thái EOF. Ký tự null được thêm vào s sau ký tự đọc được cuối cùng. Hàm trả về s nếu thành công, hoặc NULL nếu ngược lại *(không đọc được ký tự nào trước khi luồng chuyển sang EOF hoặc phát sinh lỗi)*.

char *gets(char *s); Tương tự fscanf(stdin, "%[^\n]%*c", s));
!Lưu ý: Hàm gets có thể được loại bỏ khỏi thư viện chuẩn.

Các tệp tạm thời

Tệp tạm thời thường được sử dụng để lưu dữ liệu nháp (tạm thời) trong quá trình tính toán, và thường được xóa đi sau khi hoàn thành tính toán.

FILE *tmpfile(void);
Hàm tmpfile tạo một tệp nhị phân tạm thời khác với tất cả các tệp hiện có. Tệp tạm thời được xóa khi đóng hoặc kết thúc chương trình. Hàm trả về con trỏ tệp nếu thành công hoặc NULL nếu ngược lại.

char *tmpnam(char *s);
Tạo tệp tạm thời tương tự tmpfile nhưng trả về tên tệp. Nếu không thể sinh được tên tệp thì hàm trả về NULL. Nếu s == NULL hàm trả về vùng nhớ tĩnh chứa tên tệp. Nếu s != NULL hàm sao chép tên tệp vào vùng nhớ được trỏ tới bởi s.

Các hàm hỗ trợ quản lý tệp

int remove(const char *fname);

Xóa tệp fname. Nếu tệp đang được mở thì hành vi được định nghĩa bởi triển khai cụ thể. Hàm trả về 0 nếu thành công và trả về giá trị != 0 nếu ngược lại (phát sinh lỗi).

int rename(const char *old, const char *new);

Đổi tên tệp old thành new. Nếu tệp với tên new đã tồn tại thì hành vi phụ thuộc triển khai.

Các hàm trả về 0 nếu thành công, và trả về giá trị != 0 nếu ngược lại (phát sinh lỗi).

Xử lý trạng thái kết thúc và trạng thái lỗi

int feof(FILE *stream);
Chỉ trả về giá trị != 0 nếu cờ EOF của luồng được thiết lập, trả về 0 nếu ngược lại (không có cờ EOF).

int ferror(FILE *stream);
Chỉ trả về giá trị != 0 nếu cờ lỗi được thiết lập, trả về 0 nếu ngược lại (cờ lỗi không được thiết lập).

void clearerr(FILE *stream);
Xóa cờ EOF và cờ lỗi của luồng, không trả về giá trị.

Nhập, xuất theo khối

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
Đọc nmemb phần tử có kích thước size vào mảng được trỏ tới bởi ptr. Tương đương với đọc size * nmemb giá trị unsigned char với hàm fgetc và lưu theo thứ tự đọc vào 1 mảng unsigned char * ở cùng vùng nhớ.
Hàm trả về số phần tử đọc được, có thể nhỏ hơn nmemb nếu luồng chuyển sang EOF hoặc phát sinh lỗi trong quá trình đọc.

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
Xuất nmemb phần tử có kích thước size của mảng được trỏ tới bởi ptr vào luồng stream. Tương đương với với xuất mảng nmemb * size giá trị unsigned char ở cùng vùng nhớ theo thứ tự bằng fputc.
Hàm trả về số lượng phần tử được xuất thành công, có thể < nmemb nếu phát sinh lỗi. Nếu size == 0 hoặc nmemb == 0 hàm trả về 0 và trạng thái luồng không thay đổi.

Định vị trong tệp

int fgetpos(FILE *stream, fpos_t *pos);
int fsetpos(FILE *stream, const fpos_t *pos);
Hàm fgetpos lưu trạng thái của luồng và vị trí đọc hiện tại vào đối tượng được trỏ đến bởi pos. Đối tượng này có thể được sử dụng với fsetpos để khôi phục trạng thái luồng ở thời điểm gọi fgetpos. Các hàm trả về 0 nếu thao tác thành công, hoặc giá trị khác 0 và lưu mã lỗi vào errno nếu ngược lại (thất bại).

long ftell(FILE *stream);
Trả về vị trí đọc hiện tại của luồng: Đối với luồng nhị phân là khoảng cách tính bằng bytes từ đầu tệp, đối với luồng văn bản là giá trị không xác định có thể được sử dụng bởi fseek để khôi phục vị trí đọc ở thời điểm gọi ftell. Hàm trả về -1L nếu phát sinh lỗi.

Định vị trong tệp(2)

int fseek(FILE *stream, long offset, int whence);
Thiết lập vị trí hiện hành mới cho luồng được trỏ tới bởi stream. Với luồng nhị phân, vị trí mới được xác định bằng tổng của offset và vị trí được xác định theo whence: SEEK_SET - Vị trí đầu tệp, SEEK_CUR - vị trí hiện tại, SEEK_END - vị trí kết thúc tệp (Quy chuẩn không bắt buộc hỗ trợ SEEK_END). Với luồng văn bản, offset phải = 0 hoặc = giá trị trả về trong lần gọi ftell thành công trên luồng được gắn kết với cùng tệp ngoài, và whence phải là SEEK_SET. Hàm chỉ trả về giá trị != 0 trong trường hợp thất bại.

void rewind(FILE *stream); - Như fseek(stream, 0L, SEEK_SET);
Đưa vị trí đọc hiện hành của luồng về đầu tệp.

Định vị là 1 thao tác chậm và phức tạp, cần hạn chế số lần gọi fseek trong giải thuật đọc/ghi với tệp

Nội dung

Các khái niệm Luồng & Tệp

Một số hàm thao tác với tệp

Một số ví dụ đọc/ghi tệp

Ví dụ 10.2. Đọc/ghi tệp theo từng ký tự

Chương trình sao chép tệp, các tên tệp được truyền qua tham số dòng lệnh. !Lưu ý: Có thể mất dữ liệu do chương trình không kiểm tra tệp đích trước khi mở tệp để ghi.

vd10-2.c

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: ./prog source dest\n");
    return 1;
  }
  char *iname = argv[1], *oname = argv[2];
  FILE *inp = fopen(iname, "rb"),
       *out = fopen(oname, "wb");
  if (!inp || !out) {
    printf("Lỗi mở tệp\n");
    return 1;
  }
  int c;
  while ((c = fgetc(inp)) != EOF) {
    fputc(c, out);
  }
  fclose(inp);
  fclose(out);
  return 0;
}
```

```
bangoc:$gcc -o prog vd10-2.c
bangoc:$./prog
Usage: ./prog source dest
bangoc:$./prog vd10-2.c vd10-2-copy.c
bangoc:$
```

Ví dụ 10.3. Đọc/ghi mảng số nguyên từ tệp

vd10-3.c

```c
#include <stdio.h>

#define MAXN 100

int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: ./prog inp.txt out.txt\n");
    return 1;
  }
  char *iname = argv[1], *oname = argv[2];
  FILE *inp = fopen(iname, "r"),
       *out = fopen(oname, "w");
  if (!inp || !out) {
    printf("Lỗi mở tệp\n");
    return 1;
  }
  int n;
  fscanf(inp, "%d", &n);
  int a[MAXN];
  for (int i = 0; i < n; ++i) {
    fscanf(inp, "%d", &a[i]);
  }
  fprintf(out, "%d\n", n);
  for (int i = n - 1; i >= 0; --i) {
    fprintf(out, " %d", a[i]);
  }
  fprintf(out, "\n");
  fclose(inp);
  fclose(out);
  return 0;
}
```

Chương trình đọc n số nguyên từ tệp và xuất ra tệp khác theo thứ tự ngược lại. Các tên tệp được truyền qua tham số dòng lệnh. !Lưu ý: Tệp đích có thể bị ghi đè.

```
bangoc:$gcc -o prog vd10-3.c
bangoc:$./prog
Usage: ./prog inp.txt out.txt
bangoc:$cat inp.txt
3
1 2 3
bangoc:$./prog inp.txt out.txt
bangoc:$cat out.txt
3
 3 2 1
bangoc:$
```

Ví dụ 10.4. Đảo ngược các dòng trong tệp

Chương trình đọc các dòng trong tệp văn bản và xuất ra tệp khác theo thứ tự ngược lại.

vd10-4.c

```c
#include <stdio.h>

#define MAXN 100
#define MAXL 256

int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: ./prog inp.txt out.txt\n");
    return 1;
  }
  char *iname = argv[1], *oname = argv[2];
  FILE *inp = fopen(iname, "r"),
       *out = fopen(oname, "w");
  if (!inp || !out) {
    printf("Lỗi mở tệp\n");
    return 1;
  }
  int n = 0;
  char lines[MAXN][MAXL];
  while (fgets(lines[n], MAXL, inp)) {
    ++n;
  }
  for (int i = n - 1; i >= 0; --i) {
    fprintf(out, "%s", lines[i]);
  }
  fclose(inp);
  fclose(out);
  return 0;
}
```

```
bangoc:$gcc -o prog vd10-4.c
bangoc:$./prog
Usage: ./prog inp.txt out.txt
bangoc:$cat inp.txt
Well
Come
bangoc:$./prog inp.txt out.txt
bangoc:$cat out.txt
Come
Well
bangoc:$
```

Ví dụ 10.5. Sao chép tệp theo khối

vd10-5.c

```c
#include <stdio.h>

#define MAXSIZE 1024

int main(int argc, char *argv[]) {
  if (argc != 3) {
    printf("Usage: ./prog source dest\n");
    return 1;
  }
  char *iname = argv[1], *oname = argv[2];
  FILE *inp = fopen(iname, "rb"),
       *out = fopen(oname, "wb");
  if (!inp || !out) {
    printf("Lỗi mở tệp\n");
    return 1;
  }
  unsigned char buff[MAXSIZE];
  while (!feof(inp)) {
    int n = fread(buff, 1, MAXSIZE, inp);
    fwrite(buff, 1, n, out);
  }
  fclose(inp);
  fclose(out);
  return 0;
}
```

Tương tự ví dụ 10.2 nhưng sử dụng fread và fwrite.

```
bangoc:$gcc -o prog vd10-5.c
bangoc:$./prog
Usage: ./prog source dest
bangoc:$cat inp.txt
Hello world
bangoc:$./prog inp.txt out.txt
bangoc:$cat out.txt
Hello world
bangoc:$
```

Ví dụ 10.6. Đo kích thước tệp

Chương trình nhận tên tệp qua tham số dòng lệnh và xuất ra kích thước tệp tính theo bytes.

vd10-6.c

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Usage: ./prog inp.txt\n");
    return 1;
  }
  FILE *inp = fopen(argv[1], "rb");
  fseek(inp, 0, SEEK_END);
  long sz = ftell(inp);
  printf("Size = %ld bytes\n", sz);
  fclose(inp);
  return 0;
}
```

```
bangoc:$gcc -o prog vd10-6.c
bangoc:$./prog
Usage: ./prog inp.txt
bangoc:$./prog vd10-6.c
Size = 522 bytes
bangoc:$du -b vd10-6.c
522     vd10-6.c
bangoc:$
```