

Ngôn ngữ lập trình C

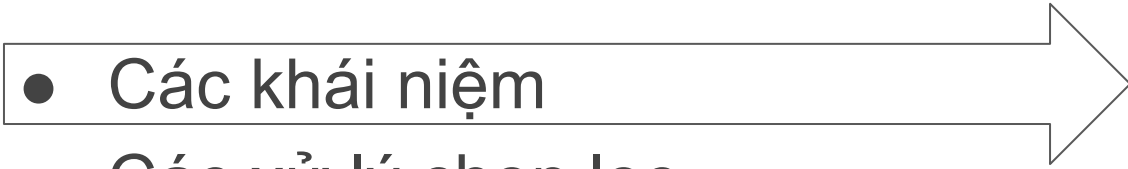
Bài 10. Nhập, xuất dữ liệu với luồng

Soạn bởi: TS. Nguyễn Bá Ngọc

Nội dung

- Các khái niệm
- Các xử lý chọn lọc
- Các ví dụ tổng hợp

Nội dung

- 
- Các khái niệm
 - Các xử lý chọn lọc
 - Các ví dụ tổng hợp

Khái niệm luồng dữ liệu

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

- Luồng có các tính chất đồng nhất hơn so với các phương tiện nhập, xuất cụ thể.
 - Khái quát hơn - Có thể sử dụng cùng 1 tập hàm dựa trên luồng để thao tác với nhiều phương tiện nhập, xuất.
- Bài giảng này tập trung vào đọc và ghi tệp bằng luồng.

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.

Luồng và tệp

- Con trỏ tới luồng có kiểu `FILE *` (`stdio.h`), vì vậy còn được gọi là *con trỏ tệp*.
 - `FILE` là định danh nguyên thủy trong C.
 - Thuật ngữ file/tệp trong quy chuẩn C cũng được sử dụng với nghĩa rộng - bao gồm cả các thiết bị và các tệp trên ổ đĩa.
 - (*Nên đặt tên cấu trúc biểu diễn luồng là `STREAM`?*)
- Theo thiết kế của thư viện chuẩn của C luồng được gắn kết với tệp ngoại bằng cách mở tệp.
- 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*).

Luồng nhị phân vs. luồng văn bản

Quy chuẩn C cung cấp 2 định dạng luồng: Luồng văn bản và luồng nhị phân.

- Luồng nhị phân có biểu diễn cơ bản nhất - là dãy ký tự tuần tự, với cùng 1 triển khai dữ liệu đọc được từ 1 luồng nhị phân giống với dữ liệu đã viết vào đó.
 - Mỗi triển khai có thể thêm số lượng ký tự null khác nhau vào cuối luồng.
- Luồng văn bản có biểu diễn chi tiết hơn luồng nhị phân - dãy ký tự được phân chia thành các dòng.
 - Mỗi dòng được kết thúc bằng 1 ký tự xuống dòng.
 - *(yêu cầu dòng cuối cùng phải có ký tự xuống dòng hay không phụ thuộc vào triển khai cụ thể).*

Luồng nhị phân vs. luồng văn bản₍₂₎

- Bên cạnh đó các ký tự có thể được thêm vào, bị thay đổi, hoặc bị xóa khi đọc, ghi với luồng văn bản để tương thích với các quy tắc hiển thị văn bản trong môi trường thực thi.
 - Tương quan giữa ký tự trong luồng văn bản và biểu diễn bên ngoài không nhất thiết phải là 1-1.
 - Dữ liệu đọc được từ 1 luồng văn bản không bắt buộc phải = dữ liệu đã ghi vào đó.
- Trình biên dịch có thể phân biệt luồng văn bản và luồng nhị phân hoặc không, ví dụ:
 - GNU GCC (trong Linux) không phân biệt.
 - Với luồng văn bản 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.
 - MinGW (trong Windows) có phân biệt.
 - Với luồng văn bản cặp ký tự `"\r\n"` (có thể) được đọc như 1 ký tự `'\n'`, và khi xuất - ký tự `'\n'` (có thể) được chuyển đổi thành 2 ký tự `"\r\n"`.

Các luồng tiêu chuẩn

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.

Tên luồng tiêu chuẩn	Con trỏ tệp	Phương tiện mặc định
Nhập	stdin	Bàn phím
Xuất thường	stdout	Màn hình
Xuất lỗi	stderr	Màn hình

- Thay đổi phương tiện mặc định, ví dụ sử dụng các tệp văn bản, có thể không làm thay đổi các lệnh nhập, xuất
- Các hàm nhập, xuất ngầm định luồng tiêu chuẩn:
 - scanf - ngầm định đọc theo định dạng từ stdin.
 - printf - ngầm định xuất theo định dạng ra stdout.
 - getchar - ngầm định đọc 1 ký tự từ stdin.
 - putchar - ngầm định xuất 1 ký tự ra stdout.

Điều hướng với tệp văn bản

- Có thể điều hướng các luồng tiêu chuẩn bằng câu lệnh chạy chương trình trong môi trường thực thi hoặc bằng lệnh trong chương trình.
- Trong môi trường thực thi:
 - Điều hướng tới luồng nhập (stdin), ví dụ:
 - Gắn stdin với in.txt: `./prog < in.txt`
 - Điều hướng luồng xuất thường (stdout) tới tệp, ví dụ:
 - Gắn stdout với out.txt: `./prog > out.txt`
 - Điều hướng luồng xuất lỗi (stderr) tới tệp, ví dụ:
 - Gắn stderr với err.txt: `./prog 2> err.txt`
 - Điều hướng đồng thời nhiều luồng, ví dụ:
 - `./prog < in.txt > out.txt 2> err.txt`
 - *(Cú pháp có thể thay đổi tùy theo môi trường dòng lệnh)*

Chương trình không thay đổi nhưng sẽ đọc, ghi dữ liệu với tệp văn bản thay vì đọc từ bàn phím và xuất ra màn hình

Điều hướng với tệp văn bản₍₂₎

- Có thể điều hướng các luồng tiêu chuẩn bằng lệnh 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.
- Hàm 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

```
1  #include <stdio.h>
2  int main() {
3      int a, b;
4      scanf("%d%d", &a, &b);
5      printf("%d + %d = %d\n", a, b, a + b);
6      fprintf(stderr, "Test error message\n");
7  }
```

```
>echo 3 5 > inp.txt
>./prog < inp.txt
3 + 5 = 8
Test error message
>./prog < inp.txt > out.txt 2> err.txt
>cat out.txt
3 + 5 = 8
>cat err.txt
Test error message
```

Ví dụ 10.1b. Mở lại luồng tiêu chuẩn

```
1  #include <stdio.h>
2  int main() {
3      int a, b;
4
5      // Mở lại các luồng tiêu chuẩn
6      if (!freopen("inp.txt", "r", stdin) ||
7          !freopen("out.txt", "w", stdout) ||
8          !freopen("err.txt", "w", stderr)) {
9          printf("Lỗi mở lại tệp.\n");
10         return 1;
11     }
12     scanf("%d%d", &a, &b);
13     printf("%d + %d = %d\n", a, b, a + b);
14     fprintf(stderr, "Test error message\n");
15 }
```

```
>echo 20 30 > inp.txt
>./prog
>cat out.txt
20 + 30 = 50
>cat err.txt
Test error message
```


Tệp văn bản và các tệp có định dạng khác

- Các chương trình ứng dụng sử dụng nhiều định dạng tệp khác nhau để lưu trữ dữ liệu lâu dài: Hình ảnh, âm thanh, tài liệu, v.v.. *(Có thể xây dựng các giải thuật đọc, ghi bất kỳ định dạng tệp nào dựa trên luồng nhị phân).*
- Tệp văn bản có định dạng đơn giản và có nhiều chương trình ứng dụng cho phép xử lý tệp văn bản. *(Ví dụ tệp mã nguồn chương trình C là tệp văn bản).*
- Các chương trình soạn thảo văn bản được phát triển độc lập dẫn tới những khác biệt nhất định, cần được lưu ý khi xử lý tệp văn bản.
- Biểu diễn dấu hiệu xuống dòng:
 - Cặp ký tự CR ('\r') và LF ('\n'): Phổ biến trong Windows
 - 1 Ký tự LF: Phổ biến trong Linux và Mac OS (hiện nay)
 - 1 ký tự CR: Phiên bản cũ của Mac OS.

Tệp văn bản và các tệp có định dạng khác₍₂₎

- Biểu diễn dấu hiệu kết thúc tệp (EOF - End Of File):
 - Trong nhiều môi trường trong giai đoạn đầu (ví dụ CP/M), ký tự 0x1A được sử dụng làm dấu hiệu kết thúc tệp.
 - CP/M là 1 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.
- Sử dụng luồng văn bản có thể giúp duy trì tính khả chuyển của các mã nguồn đọc, ghi tệp văn bản đơn giản hơn.
- Có nhiều thư viện chuyên dụng đã được phát triển để xử lý các tệp có định dạng phức tạp hơn như JSON, XML, v.v...

Nội dung

- Các khái niệm
 - Các xử lý chọn lọc
 - Các ví dụ tổng hợp
- 

Mở tệp

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

- Hàm fopen mở tệp ngoại fname trong chế độ được mô tả bằng mode và gắn kết luồng với nó. Hàm trả về con trỏ tới luồng 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 chế độ mở tệp.

Đóng tệp

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

- Đóng tệp ngoại và ngắt kết nối giữa luồng và tệp ngoại. Dữ liệu đệm trong luồng xuất được ghi ra tệp, dữ liệu đệm trong luồng nhập bị hủy bỏ. Các vùng nhớ đệm nếu được cấp phát thì được giải phóng.
- Hàm trả về 0 nếu thành công hoặc EOF nếu phát sinh lỗi.

Nhập, xuất theo định dạng

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

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

- Cặp hàm fscanf và fprintf có thể nhập và xuất dữ liệu với luồng theo định dạng:
 - fprintf(**stdout**, /*...*/); - Tương đương với printf(/*...*/);
 - fscanf(**stdin**, /*...*/); - Tương đương với scanf(/*...*/);
- Có thể thay **stdin**, **stdout** trong lệnh nhập, xuất với fprintf và fscanf bằng con trỏ thích hợp để đọc, ghi với tệp ngoại:
 - FILE *inp = fopen("input.txt", "r");
 - FILE *out = fopen("output.txt", "w"); //...
 - fscanf(inp, "%d", &n);
 - fprintf(out, "%d", total); //...
 - fclose(inp);
 - fclose(out);

Nhập, xuất ký tự

`int getchar(void);` - Tương đương `fgetc(stdin);`

`int fgetc(FILE *stream);`

- Đọc và trả về 1 ký tự như `unsigned char` từ luồng nếu chưa hết dữ liệu, hoặc trả về EOF nếu phát sinh lỗi hoặc đã đọc hết dữ liệu.

Macro trong stdio.h:

`int ungetc(int c, FILE *stream);` `#define EOF (-1)`

- Trả lại ký tự c vào luồng và xóa trạng thái EOF (nếu có), nếu thành công thì 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 `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 EOF và thiết lập cờ lỗi nếu thất bại.

Nhập, xuất chuỗi

`int fputs(const char *s, FILE *stream);`

- Hàm fputs xuất chuỗi s vào luồng được trỏ tới bởi stream. Trả về EOF nếu phát sinh lỗi hoặc 1 giá trị không âm nếu ngược lại. Các lệnh sau đều xuất chuỗi s ra màn hình:
 - `fputs(s, stdin);` Tương đương `puts(s);`
 - `fprintf(stdin, "%s", s);` Tương đương `printf("%s", s);`

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

- Hàm fgets đọc tối đa n - 1 ký tự vào vùng nhớ được trỏ tới bởi s từ luồng được trỏ tới bởi stream. Ký tự null được thêm vào cuối chuỗi s. Hàm trả về s nếu thành công, hoặc NULL nếu không đọc được ký tự nào (*luồng ở trạng thái EOF hoặc phát sinh lỗi đọc dữ liệu*).
 - !Lưu ý: gets không phải trường hợp ngầm định của fgets với stdin, ngừng sử dụng gets trong mã nguồn mới.

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]` - `"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 tương tác với chương trình

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 \leq Bộ nhớ đệm \leq Tệp
 - Khi xuất dữ liệu vào tệp: Dữ liệu \Rightarrow Bộ nhớ đệm \Rightarrow Tệp
 - Bộ nhớ đệm giúp hạn chế số lần truy cập 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 và trong suốt ở mức giao diện đọc, ghi.
- Tùy chỉnh 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 kết hợp (nhập và xuất) với thao tác cuối cùng không phải thao tác đọc, hàm fflush đẩy dữ liệu trong bộ nhớ đệm của luồng vào 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 thỏa mãn yêu cầu và 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.

Lưu ý: Không sử dụng fflush với luồng nhập, áp dụng fflush cho luồng nhập là hành vi bất định

Xóa dữ liệu trong bộ nhớ đệm nhập

Khi đọc 1 giá trị được ngăn cách bởi khoảng trắng, các hàm scanf và fscanf không xóa khoảng trắng đứng sau giá trị.

- Ví dụ, với luồng nhập "123 \nC"
 - `scanf("%d", &n);` - Đọc được `n = 123`.
 - Tiếp theo nếu nhập 1 ký tự và để ký tự nhập được là C chúng ta cần xóa hết các khoảng trắng giữa 123 và C, trong ví dụ này có thể đọc từng ký tự cho tới khi đọc được '\n':
 - `while (fgetc(stdin) != '\n') ;` // Đọc hết dòng hiện hành
- `fflush(stdin)` tuy được định nghĩa trong Windows, nhưng khó hiểu khi kết hợp với điều hướng
 - Nếu "123 \nC" được điều hướng vào `stdin` từ tệp văn bản thì `fflush(stdin)` như trong Windows sẽ xóa hết dữ liệu đệm, bao gồm cả ký tự C.
 - *!!Lưu ý: Không sử dụng `fflush(stdin)`*

Các tệp tạm thời

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

`FILE *tmpfile(void);`

- Hàm `tmpfile` tạo 1 tệp nhị phân tạm thời khác với tất cả các tệp hiện có. Tệp tạm thời được xóa khi đóng tệp 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 như `tmpfile` nhưng trả về tên tệp. Nếu không thể sinh tên tệp thì hàm trả về **NULL**. Nếu `s == NULL` hàm trả về con trỏ tới vùng nhớ chứa tên tệp. Nếu `s != NULL` thì 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_name, const char *new_name);`

- Đổi tên tệp old_name thành new_name. Nếu tệp với tên new_name đã tồn tại thì hành vi phụ thuộc triển khai.
- Hàm trả về 0 nếu thành công, và trả về giá trị != 0 nếu ngược lại (phát sinh lỗi).

Xử lý trạng thái

`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` bằng hàm `fgetc` và lưu theo thứ tự đọc vào 1 mảng `unsigned char *` ở cùng vùng nhớ.
- Hàm trả về số lượng phần tử đọc được, có thể nhỏ hơn nmemb nếu luồng chuyển sang trạng thái EOF hoặc phát sinh lỗi trong quá trình đọc.

Nhập, xuất theo khối₍₂₎

`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 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 đó có thể được sử dụng với fsetpos để khôi phục trạng thái luồng như ở thời điểm gọi fgetpos. Các hàm trả về 0 nếu thành công, hoặc giá trị khác 0 và lưu mã lỗi vào errno nếu thất bại.

```
long ftell(FILE *stream);
```

- Hàm trả về vị trí đọc hiện tại của luồng: Số lượng bytes tính từ đầu tệp đối với luồng nhị phân, hoặc 1 giá trị không xác định nhưng có thể được sử dụng bởi fseek để khôi phục vị trí đọc ở thời điểm gọi ftell đối với luồng văn bản. Hàm trả về -1 nếu phát sinh lỗi.

Định vị trong tệp₍₂₎

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

- Thiết lập vị trí hiện hành cho luồng được trở tới bởi stream.
 - Với luồng nhị phân, vị trí mới được xác định bằng độ lệch offset tính theo bytes từ vị trí được xác định bằng whence: SEEK_SET - Đầu tệp, SEEK_CUR - Hiện tại, SEEK_END - 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);`

- Đưa vị trí hiện hành của luồng về đầu tệp và xóa cò lỗi.

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

Nội dung

- Các khái niệm
- Các xử lý chọn lọc
- Các ví dụ tổng hợp



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

Viết chương trình sao chép tệp với các tên tệp được truyền qua tham số dòng lệnh. Sử dụng cú pháp:

`./copy tệp-nguồn tệp-đích`

Ví dụ 10.2. Đọc, ghi tệp theo từng ký tự₍₂₎

```
1  #include <stdio.h>
2  int main(int argc, char *argv[]) {
3      if (argc != 3) {
4          printf("Usage: ./copy source dest\n");
5          return 1;
6      }
7      FILE *inp = fopen(argv[1], "rb"),
8          *out = fopen(argv[2], "wb");
9      if (!inp || !out) {
10         fprintf(stderr, "Lỗi mở tệp\n");
11         return 1;
12     }
13     int c;
14     long sz = 0;
15     while ((c = fgetc(inp)) != EOF) {
16         fputc(c, out);
17         ++sz;
18     }
19     fclose(inp);
20     fclose(out);
21     printf("Hoàn thành sao chép %ld (bytes)\n", sz);
22 }
```

>./copy

Usage: ./copy source dest

>./copy vd10-2.c copy-of-vd10-2.c

Hoàn thành sao chép 642 (bytes)

>diff vd10-2.c copy-of-vd10-2.c

>

!Lưu ý: Đừng để mất dữ liệu, chương trình có thể ghi đè lên tệp đã có do không kiểm tra tệp đích trước khi mở tệp để ghi.

Ví dụ 10.3. Đọc, ghi dãy số từ tệp văn bản

Viết chương trình đọc 1 dãy n số nguyên từ 1 tệp văn bản rồi xuất ra 1 tệp văn bản khác theo thứ tự ngược lại. Các tên tệp được truyền qua tham số dòng lệnh theo cú pháp:

`./inverse tệp-nguồn tệp-đích`

Lưu dãy số trong tệp theo định dạng dòng đầu tiên là 1 số nguyên dương n - là số lượng phần tử trong dãy, sau đó là n số nguyên được ngăn cách bởi khoảng trắng, ví dụ dãy 3 số nguyên 1 2 3 có thể được lưu theo dạng:

3

1 2 3

Ví dụ 10.3. Đọc, ghi dãy số từ tệp văn bản₍₂₎

```
1  #include <stdio.h>
2  #define MAXN 100
3  int main(int argc, char *argv[]) {
4      FILE *inp = fopen(argv[1], "r"),
5          *out = fopen(argv[2], "w");
6      if (!inp || !out) {
7          fprintf(stderr, "Lỗi mở tệp\n");
8          return 1;
9      }
10     int n;
11     fscanf(inp, "%d", &n);
12     int a[MAXN];
13     for (int i = 0; i < n; ++i) {
14         fscanf(inp, "%d", &a[i]);
15     }
16     fprintf(out, "%d\n", n);
17     for (int i = n - 1; i > 0; --i) {
18         fprintf(out, "%d ", a[i]);
19     }
20     fprintf(out, "%d\n", a[0]);
21     fclose(inp);
22     fclose(out);
23 }
```

```
>cat inp.txt
5
1 3 5 7 11
>./inverse inp.txt out.txt
>cat out.txt
5
11 7 5 3 1
>
```

Ví dụ 10.4. Đảo ngược các dòng trong tệp

Viết chương trình đọc 1 tệp văn bản có chứa không quá 100 dòng và mỗi dòng có không quá 255 ký tự, sau đó xuất các dòng theo thứ tự ngược lại ra 1 tệp khác. Sử dụng cú pháp:
./inverse tệp-nguồn tệp-đích

Ví dụ 10.4. Đảo ngược các dòng trong tệp₍₂₎

```
1  #include <stdio.h>
2  #define MAXN 100
3  #define MAXL 256
4  int main(int argc, char *argv[]) {
5      FILE *inp = fopen(argv[1], "r"),
6          *out = fopen(argv[2], "w");
7      if (!inp || !out) {
8          fprintf(stderr, "Lỗi mở tệp\n");
9          return 1;
10     }
11     int n = 0;
12     char lines[MAXN][MAXL];
13     while (fgets(lines[n], MAXL, inp)) {
14         ++n;
15     }
16     for (int i = n - 1; i >= 0; --i) {
17         fprintf(out, "%s", lines[i]);
18     }
19     fclose(inp);
20     fclose(out);
21 }
```

```
>cat inp.txt
Well
Come
>./inverse inp.txt out.txt
>cat out.txt
Come
Well
>
```

Ví dụ 10.5. Sao chép tệp theo khối

Sử dụng các hàm `fread`, `fwrite` và viết chương trình sao chép tệp với các tên tệp được truyền qua tham số dòng lệnh.

Cú pháp: `./copy tệp-nguồn tệp-đích`

Ví dụ 10.5. Sao chép tệp theo khối₍₂₎

```
1  #include <stdio.h>
2  #define BUFSIZE 1024
3  int main(int argc, char *argv[]) {
4      FILE *inp = fopen(argv[1], "rb"),
5          *out = fopen(argv[2], "wb");
6      if (!inp || !out) {
7          fprintf(stderr,
8              "Lỗi mở tệp\n");
9          return 1;
10     }
11     unsigned char buff[BUFSIZE];
12     long sz = 0;
13     while (!feof(inp)) {
14         int n = fread(buff, 1, BUFSIZE, inp);
15         fwrite(buff, 1, n, out);
16         sz += n;
17     }
18     fclose(inp);
19     fclose(out);
20     printf("Hoàn thành sao chép %ld (bytes)\n", sz);
21 }
```

```
>wc -c vd10-5.c
682 vd10-5.c
>./copy vd10-5.c copy-of-vd10-5.c
Hoàn thành sao chép 682 (bytes)
>diff vd10-5.c copy-of-vd10-5.c
>
```


Ví dụ 10.6. Đo kích thước tệp

Viết chương trình nhận tên tệp qua tham số dòng lệnh và xuất ra kích thước tệp tính theo bytes. Sử dụng cú pháp:

```
./sizeof tên-tệp
```

Ví dụ 10.6. Đo kích thước tệp₍₂₎

```
1  #include <stdio.h>
2  int main(int argc, char *argv[]) {
3      if (argc != 2) {
4          printf("Usage: ./sizeof file.txt\n");
5          return 1;
6      }
7      FILE *inp = fopen(argv[1], "rb");
8      fseek(inp, 0, SEEK_END);
9      long sz = ftell(inp);
10     printf("Size = %ld bytes\n", sz);
11     fclose(inp);
12 }
```

```
>./sizeof vd10-6.c
Size = 522 bytes
>wc -c vd10-6.c
522 vd10-6.c
>
```

