Xuất nhập (input/output)

1

Khái niệm

- Người lập trình thường xuyên phải làm việc với một số thiết bị vào ra như màn hình, bàn phím, file, máy in,…
- Với mỗi chương trình, có:
  - Đầu ra chuẩn stdout: mặc định là màn hình console, nhưng có thể được coi như một file ảo chỉ ghi, và có thể định nghĩa lại là một file trên đĩa hoặc máy in
  - Đầu ra chuẩn cho lỗi stderr: tương tự stdout, nhưng thường dùng để ghi các dòng lỗi gặp phải trong chương trình
  - Đầu vào chuẩn stdin: mặc định là bàn phím, nhưng có thể được coi như một file ảo chỉ đọc, và có thể định nghĩa lại là một file trên đĩa

2

Mở đầu

- Xuất ra stdout
  - Xuất một ký tự:
    ```
    int putchar(int c);
    ```
  - Xuất một dòng ký tự:
    ```
    int puts(const char* s);
    ```
  - Xuất một chuỗi theo định dạng:
    ```
    int printf(const char* format, ...);
    ```
- Nhập từ stdin
  - Đọc một ký tự:
    ```
    int getchar();
    ```
  - Đọc một dòng ký tự:
    ```
    char* gets(char* s);
    ```
  - Đọc một chuỗi theo định dạng:
    ```
    int scanf(const char* format, ...);
    ```


3

Xuất nhập từ file

- Kiểu file:
  - `typedef struct { … } FILE;`
- Trình tự thao tác với file: Mở/tạo file → Đọc/ghi dữ liệu → Đóng
  - Trong kiểu FILE có trường lưu thông tin vị trí đang đọc/ghi của file, gọi là con trỏ file
- Mở file:
  - `FILE* fopen(const char* fname, const char* mode);`

mode	Ý nghĩa	mode	Ý nghĩa
"r"	Chỉ cho phép đọc	"r+"	Cho phép đọc và ghi
"w"	Chỉ cho phép ghi, xoá nội dung file cũ nếu có hoặc tạo file mới nếu chưa có	"w+"	Cho phép đọc và ghi, xoá nội dung file cũ nếu có hoặc tạo file mới nếu chưa có
"a"	Chỉ cho phép ghi, trỏ con trỏ đến cuối file để ghi tiếp hoặc tạo file mới nếu chưa có	"a+"	Cho phép đọc và ghi, trỏ con trỏ tới cuối file để ghi tiếp hoặc tạo file mới nếu chưa có
"t"	Đọc/ghi dạng văn bản (text)	"b"	Đọc/ghi dạng nhị phân (binary)

4

Chú ý với việc mở file

- Việc mở file có thể không thành công và trả về NULL → cần kiểm tra giá trị trả về của fopen() để biết đã mở file thành công không
- Các lý do có thể khiến mở file không thành công:
  - Mở file để đọc mà file đó không tồn tại
  - Người dùng hiện tại không có quyền
  - File đang được mở với chế độ hạn chế bởi một chương trình nào đó
  - Có quá nhiều file đang mở (hệ điều hành có giới hạn số file được mở đồng thời)
- Các file được mở với hàm fopen() không hạn chế được mở lại

5

Mở file và hạn chế mở lại

- Đôi khi ta không muốn chương trình khác can thiệp vào một file ta đang mở để đọc/ghi
  - `FILE* _fsopen(const char* fname, const char* mode, int shflag);`
    - shflag: cờ cho phép file được mở lại hay không
    - `#include <share.h>`

shflag	Ý nghĩa
_SH_DENYNO	Không hạn chế
_SH_DENYRD	Hạn chế được mở lại với chế độ đọc
_SH_DENYWR	Hạn chế được mở lại với chế độ ghi
_SH_DENYRW	Hạn chế được mở lại với cả chế độ đọc và ghi

- Lưu ý: Hàm này chỉ có trong MS Visual C

6

Ghi vào file

- File văn bản (text) và nhị phân (binary)
  - File văn bản: một số ký tự đặc biệt như chuyển đổi giữa '\n' và "\r\n", xử lý ký tự hết file → thích hợp file dạng văn bản
  - File nhị phân: không thay đổi dữ liệu ghi vào → thích hợp với việc lưu dữ liệu dạng nhị phân
- Ghi chuỗi (file văn bản):
  - `int fputc(int c, FILE* file);`
  - `int fputs(const char* s, FILE* file);`
  - `int fprintf(FILE* file, const char* format, ...);`
  - Dùng tương tự các hàm putchar(), puts(), printf()
- Ghi dữ liệu (file nhị phân):
  - `int fwrite(const void* buf, int size, int count, FILE* file);`
  - Ghi mảng count phần tử, kích thước mỗi phần tử là size, địa chỉ mảng là buf

7

Đọc từ file

- Đọc hàm fgets() có kiểm tra giới hạn dữ liệu. Hàm fgetc() trchuỗi:
  - `int fgetc(FILE* file);`
  - `int fgets(char* s, int n, FILE* file);`
  - `int fscanf(FILE* file, const char* format, ...);`
  - Dùng tương tự các hàm getchar(), gets(), scanf() nhưng trả về EOF nếu đã kết thúc file.
- Đọc dữ liệu:
  - `int fread(void* buf, int size, int count, FILE* file);`
  - Đọc một mảng với count phần tử, kích thước mỗi phần tử là size, địa chỉ mảng là buf
- Kiểm tra kết thúc file hay chưa:
  - `int feof(FILE* file);`
- Vì việc đọc/ghi file có sử dụng bộ đệm, nên thường phải dùng hàm fflush() để làm sạch bộ đệm trước khi chuyển từ ghi sang đọc, hoặc từ đọc sang ghi nếu mở file ở chế độ đọc và ghi đồng thời
  - `int fflush(FILE* file);`

8

Các hàm khác về đọc/ghi file

- Đóng file:
  - `int fclose(FILE* file);`
- Chuyển con trỏ file:
  - `void rewind(FILE* file);`
  - `int fseek(FILE* file, long offs, int org);`
    - org = SEEK_CUR: tính từ vị trí hiện tại
    - org = SEEK_END: tính từ cuối file
    - org = SEEK_SET: giá trị tuyệt đối (tính từ đầu file)
- Ví trí hiện tại của con trỏ:
  - `long ftell(FILE* file);`
- Xoá file:
  - `int remove(const char* path);`
- Đổi tên và chuyển file:
  - `int rename(const char* old, const char* new);`

9

Ví dụ: hàm copy file

```
int copy_file(const char* src, const char* dst) {
    FILE *fs = NULL, *fd = NULL;
    char buf[1024];
    int num;

    if ((fs = fopen(src,"rb")) == NULL) return -1;
    if ((fd = fopen(dst,"wb")) == NULL) { fclose(fs); return -1; }

    while(!feof(fs)) {
        num = fread(buf, 1, sizeof(buf), fs);
        fwrite(buf, 1, num, fd);
    }

    fclose(fs); fclose(fd);
    return 0;
}
```


10

stdin, stdout, stderr

- Đầu vào/ra chuẩn thực chất là các biến kiểu FILE* được định nghĩa sẵn, nên việc đọc/ghi với các hàm printf(…), scanf(…) tương đương với việc dùng fprintf(stdout,…) và fscanf(stdin,…)
- Tương tự với các hàm putchar(), puts(), getchar(), gets() cũng thực hiện việc đọc/ghi trên stdin và stdout
- Định hướng lại đầu vào/ra chuẩn:

Ký hiệu	Ý nghĩa
`command > file` `command 1> file`	Đổi stdout ra file
`command 2> file`	Đổi stderr ra file
`command >> file` `command 1>> file`	Đổi stdout ra file và nối tiếp vào file đó
`command 2>> file`	Đổi stderr ra file và nối tiếp vào file đó
`command < file`	Đổi stdin từ file
`command1 | command2`	Đổi stdout của command1 thành stdin của command2

11

stdin, stdout, stderr *(tiếp)*

- Một số file đặc biệt

Tên file	Ý nghĩa	Tên file	Ý nghĩa
&0	stdin	nul	Bỏ qua
&1	stdout	prn, lpt1	Máy in
&2	stderr	con	Màn hình

- Ví dụ:
  - Dẫn hướng cả stdout và stderr vào file result.txt
    `C:\>dir *.dat >result.txt 2>&1`
  - Dẫn hướng cả stdout ra máy in và stderr vào file error.log
    `C:\>stuff >prn 2>error.log`
  - Dẫn hướng đầu vào từ file input.txt và đầu ra là file output.txt
    `C:\>process <input.txt >output.txt`
  - Tạo pipe (output của lệnh nọ là input của lệnh kia)
    `C:\>type source.c | more`

12

Đọc/ghi trên bộ nhớ

- Ghi:
  - `sprintf(char* buffer, const char* format, ...);`
- Đọc:
  - `sscanf(const char* buffer, const char* format, ...);`
- Dùng tương tự như fprintf() và fscanf() nhưng dữ liệu được lưu vào một vùng nhớ xác định trong tham số buffer
- Ví dụ:
  - `char s[50];`
  - `sprintf(s, "sin(pi/3) = %.3f", sin(3.14/3));`
  - Kết quả: s sẽ chứa chuỗi "sin(pi/3) = 0.866"

13

Đọc/ghi an toàn

14

Lỗi tràn bộ đệm

- Xảy ra khi chương trình ghi dữ liệu vào một biến nhiều hơn kích thước của nó
  - Ví dụ: copy một chuỗi 10 ký tự vào biến chỉ dài 5 ký tự

```
char s[5];
strcpy(s, "0123456789");
```

- Lỗi tràn bộ đệm rất nguy hiểm vì gây ra những lỗi không dự đoán trước, đặc biệt có thể khiến người sử dụng kiểm soát máy tính và làm bất cứ gì
- Cần kiểm soát chiều dài của dữ liệu nhập so với vùng nhớ được cấp phát cho các biến
- Các hàm chuẩn của C không kiểm tra lỗi tràn bộ đệm
  → sử dụng các hàm mở rộng trong Visual C từ 2005

15

Các hàm về chuỗi và bộ nhớ

- `memcpy_s(void* dest, int size, const void* src, int count);`
- `memmove_s(void* dest, int size, const void* src, int count);`
- `strcpy_s(char* dest, int size, const char* src);`
- `strcat_s(char* dest, int size, const char* src);`
- `_strlwr_s(char* str, int size);`
- `_strupr_s(char* str, int size);`

16

Các hàm đọc dữ liệu

- `gets_s(char* str, int size);`
- `scanf_s(const char* format, …);`
  - Thêm các tham số kiểm tra kích thước biến với chuỗi và ký tự

```
int i;
float f;
char c;
char s[10];
scanf_s("%d %f %c %s", &i, &f, &c, 1, s, 10);
```

  - Tương tự với các hàm fscanf_s(), sscanf_s()

17

Bài tập

1. Viết chương trình nối một file vào một file khác
2. Viết chương trình in ra màn hình dòng thứ 10 của một file
3. Viết một hàm trả về kích thước của một file
4. Viết chương trình đổi các ký tự trong một file sang chữ hoa (tên file từ tham số dòng lệnh)
5. Viết chương trình đếm số dòng của một file
6. Viết chương trình đếm số từ và số dòng trong một file (quy ước từ cách nhau bởi một trong các ký tự: cách, tab, xuống dòng)
7. Viết chương trình chèn một dòng vào dòng thứ 10 của một file
8. Viết chương trình nhập dữ liệu cho cấu trúc SinhVien từ bàn phím, sau đó thay đổi stdin và stdout sang file và xem kết quả

18

Sửa lỗi

- Sử dụng fgets với kiểm lỗi:

```
char *result = fgets(str, sizeof(str), stdin);
char len = strlen(str);
if(result != NULL && str[len - 1] == '\n')
{
  str[len - 1] = '\0';
}
else
{
  // handle error
}
```


19

Kiến thức cần biết trước khi học bài này

- Lý thuyết
  - Khái niệm cơ bản về file và các thao tác trên file
  - Khái niệm dòng lệnh, tham số dòng lệnh
- Kỹ năng, thực hành
  - Biết viết chương trình C trong Visual Studio
  - Nắm các lệnh vào ra cơ bản: printf, putchar, puts, scanf, getchar, gets

20

Kiến thức cần nắm

- Lý thuyết
  - Phân biệt được file nhị phân, file văn bản,
  - Khi nào thì dùng file nhị phân
  - Nắm vững và phân biệt các chế độ mở file
  - Nắm vững khái niệm và cách sử dụng con trỏ file
  - Khái niệm, mục đích, vài ứng dụng của việc đọc ghi trên bộ nhớ
- Thực hành
  - Sử dụng được các câu lệnh cơ bản: mở file, đọc từ file, ghi ra file, đóng file ứng với cả 2 chế độ file văn bản và file nhị phân
  - Sử dụng được câu lệnh di chuyển con trỏ file
  - Viết được chương trình sử dụng hàm copy file trong slide
  - Đọc ghi trên bộ nhớ để tạo và tách các khung truyền tin giữa 2 máy tính

21

Kiến thức cần nắm (nâng cao)

- Lý thuyết
  - Nắm được khái niệm stdin, stdout, stderr và cách sử dụng
  - Thay đổi dẫn hướng vào/ra của chương trình
  - Tại sao cần đọc/ghi an toàn
- Thực hành nâng cao
  - Sử dụng fgets
  - Xử lí fscanf khi nhập kí tự Enter
  - Xử lí các lỗi đọc file
  - Sử dụng được các hàm đọc/ghi an toàn

22