Khoa CNTT – ĐH Khoa học Tự nhiên
TP HCM, tháng 02/2022

Kỹ thuật lập trình

Thao tác trên tập tin &

Tham số dòng lệnh

Nguyễn Trọng Việt

Mục tiêu

Sau bài học, sinh viên có thể:

- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Sử dụng tham số dòng lệnh

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Tại sao cần đến file ?

- Khi chương trình kết thúc toàn bộ dữ liệu được tạo ra sẽ mất. Dùng file để lưu giữ dữ liệu khi chương trình kết thúc
- Nhập một lượng lớn dữ liệu đầu vào cho chương trình. Mất thời gian nhập liệu, khi sai hoặc thiếu phải nhập lại từ đầu. Dùng file giúp dễ dàng chỉnh sửa và cập nhật dữ liệu đầu vào
- Dễ dàng mang dữ liệu sang máy khác
- Chẩn đoán lỗi của chương trình (log file)

https://www.programiz.com/c-programming/c-file-input-output (05/2020)

Các loại file

- File văn bản
- File nhị phân

File & bộ nhớ

	File	Bộ nhớ
	Dùng để lưu trữ dữ liệu của 1 chương trình	
Tốc độ	Chậm hơn	Nhanh hơn
Cách truy xuất	Tuần tự	Ngẫu nhiên
Chi phí	Rẻ hơn	Đắt hơn
Dung lượng	Lớn hơn	Nhỏ hơn
Thời gian lưu trữ	Lâu dài	Tạm thời

Con trỏ file

- Là con trỏ dùng để điều khiển và theo vết truy xuất trên file
- Kiểu dữ liệu: `FILE`
  - Được định nghĩa trong `stdio.h`
  - Là kiểu dữ liệu của con trỏ file
  - Là cấu trúc *mờ* (opaque)
- Các hàm mở/đóng file, đọc/ghi nội dung file đều sử dụng con trỏ file để thực hiện

```
FILE* fp = nullptr;
```

https://fresh2refresh.com/c-programming/c-interview-questions-answers/what-is-file-pointer-in-c/ (05/2020)

Con trỏ đến cấu trúc mờ

- Không cần quan tâm vùng nhớ cho cấu trúc mờ được cấp phát động/tĩnh
- Tuyệt đối không thực hiện các tác vụ truyền thống của con trỏ lên đối tượng kiểu cấu trúc mờ
  - Truy xuất nội dung
  - Gán
  - Giải phóng bằng `free/delete`
  - . . .
- Chỉ sử dụng con trỏ đến cấu trúc mờ trong những hàm được thiết kế trước

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Các thao tác trên file

Trong C/C++, đối với cả file văn bản và nhị phân

- Tạo file mới
- Mở file đã có
- Đóng file
- Đọc nội dung file
- Ghi nội dung file

Thứ tự thao tác trên file

1. Mở file
2. Kiểm tra việc mở thành công
3. Đọc/ghi nội dung từ/vào file
4. Đóng file
5. Reset con trỏ file

Mở file

- Thư viện: `stdio.h`
- Hàm mở file

```
FILE* fopen("path", "mode");
```

- Input:
  - `path`: đường dẫn đến file cần mở
  - `mode`: chế độ mở
- Output:
  - Con trỏ trỏ đến file nếu việc mở thành công
- Ví dụ:

```
FILE* fp = fopen("D:/MyData/look.txt", "rt");
```

Mở file

```
FILE* fopen("path", "mode");
```

- Chế độ mở file `mode`:

Chế độ	Ý nghĩa	Ghi chú
`"r", "rt"`	Mở file văn bản để đọc	File phải tồn tại Nếu không trả về NULL
`"rb"`	Mở file nhị phân để đọc	File phải tồn tại Nếu không trả về NULL
`"w", "wt"`	Mở file văn bản để ghi	Nếu file chưa tồn tại, tạo mới Nếu file đã tồn tại, ghi đè
`"wb"`	Mở file nhị phân để ghi	Nếu file chưa tồn tại, tạo mới Nếu file đã tồn tại, ghi đè
`"a", "at"`	Mở file văn bản để thêm vào cuối	Nếu file chưa tồn tại, tạo mới
`"ab"`	Mở file nhị phân để thêm vào cuối	Nếu file chưa tồn tại, tạo mới

http://www.cplusplus.com/reference/cstdio/fopen/ (05/2020)

Mở file

```
FILE* fopen("path", "mode");
```

- Chế độ mở file `mode`:

Chế độ	Ý nghĩa	Ghi chú
"r+"	Mở file văn bản để vừa đọc vừa ghi	File phải tồn tại Nếu không trả về NULL
"rb+"	Mở file nhị phân để vừa đọc vừa ghi	File phải tồn tại Nếu không trả về NULL
"w+"	Mở file văn bản để vừa đọc vừa ghi	Nếu file chưa tồn tại, tạo mới Nếu file đã tồn tại, ghi đè
"wb+"	Mở file nhị phân để vừa đọc vừa ghi	Nếu file chưa tồn tại, tạo mới Nếu file đã tồn tại, ghi đè
"a+"	Mở file văn bản để vừa đọc vừa thêm vào cuối	Nếu file chưa tồn tại, tạo mới
"ab+"	Mở file nhị phân để vừa đọc vừa thêm vào cuối	Nếu file chưa tồn tại, tạo mới

http://www.cplusplus.com/reference/cstdio/fopen/ (05/2020)

Mở file – secure mode

- Thư viện: `stdio.h`
- Hàm mở file

```
errno_t fopen_s(FILE**, “path”, “mode”);
```

- Input:
  - `path`: đường dẫn đến file cần mở
  - `mode`: chế độ mở
- Output:
  - Trả về 0 nếu thành công
  - Con trỏ trỏ đến file nếu mở thành công
- Ví dụ:

```
FILE* fp = nullptr;
fopen_s(&fp, “D:/MyData/look.txt”, “rt”);
```

Đóng file

- Thư viện: `stdio.h`
- Hàm đóng file

```
void fclose(FILE*);
```

- Input:
  - Con trỏ đến file đang mở
- Ví dụ:

```
FILE* fp = fopen("D:/MyData/look.txt", "rt");

...

fclose(fp);
fp = nullptr;
```

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Đọc nội dung file văn bản

- Hàm đọc 1 kí tự từ file văn bản

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

- Chức năng:
  - Đọc 1 kí tự từ file tại vị trí `fp` trỏ tới
- Trả về:
  - Kí tự đang được con trỏ file trỏ tới
  - `EOF` nếu đã ở cuối file
  - `EOF` nếu lỗi trong quá trình đọc, dùng `ferror` để check
- Lưu ý:
  - Sau khi đọc, con trỏ sẽ dịch chuyển đến vị trí kế tiếp trên file

Đọc nội dung file văn bản

- Hàm đọc chuỗi kí tự từ file văn bản

```
char* fgets(char *buffer, int num, FILE *fp);
```

- Chức năng:
  - Đọc các kí tự từ vị trí hiện tại của con trỏ file `fp` và lưu vào `buffer` như 1 chuỗi. Việc đọc kết thúc nếu đủ `num - 1` kí tự hoặc gặp `\n` hoặc hết file
- Trả về:
  - Con trỏ `buffer` nếu thành công
  - `EOF` nếu đã ở cuối file
  - `nullptr` nếu lỗi trong quá trình đọc, dùng `ferror` để check

Đọc nội dung file văn bản

- Hàm đọc dữ liệu đã định dạng từ file

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

- Chức năng:
  - Đọc dữ liệu theo định dạng `format` quy định tại vị trí hiện tại của con trỏ file `fp` và lưu vào `...`
- Trả về:
  - Số lượng item khớp với định dạng format
  - 0 nếu đến cuối file, dùng `feof` để check
  - 0 nếu lỗi trong quá trình đọc file, dùng `ferror` để check

Kiểm tra cuối file

- Hàm kiểm tra cuối file

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

- Chức năng:
  - Kiểm tra con trỏ file `fp` đã đến cuối file chưa
- Trả về:
  - 0 nếu chưa đến cuối file
  - ≠ 0 nếu ngược lại

Kiểm tra lỗi thao tác file

- Hàm kiểm tra lỗi thao tác file

```
int ferror(FILE *fp);
```

- Chức năng:
  - Kiểm tra lỗi có phát sinh trên con trỏ file `fp` hay không
- Trả về:
  - 0 nếu không có lỗi phát sinh trên `fp`
  - ≠ 0 nếu ngược lại

Ví dụ - đọc kí tự

```
#include <stdio.h>

int main ()
{
  FILE* fp = fopen("myfile.txt","rt");
  if (!fp) {
     printf("Error opening file");
     return -1;
  }

  int n = 0;
  while (fgetc(fp) != EOF) {
      ++n;
  }

  if (feof(fp)) {
     printf ("Total number of bytes read: %d.\n", n);
  }
  else {
     printf("End-of-File was not reached.");
  }
  fclose (fp);

  return 0;
}
```

Ví dụ - đọc chuỗi kí tự

```
#define MAX_SIZE 100

int main () {
  FILE* fp = fopen("myfile.txt","rt");
  if (!fp) {
     printf("Error opening file");
     return -1;
  }

  char str[MAX_SIZE] = {'\0'};
  if (fgets(str, MAX_SIZE, fp)) {
     printf("File content: %s.\n", str);
  }
  else {
    if (feof(fp)) {
       printf("End-of-File was reached.");
    }
    else {
       if (ferror(fp)) {
            printf("Error in reading.");
       }
    }
  }
  fclose(fp);

  return 0;
}
```

Ví dụ - đọc chuỗi kí tự

```
#define MAX_SIZE 100

int main (){
  FILE* fp = fopen("myfile.txt","rt");
  if (!fp) {
     printf("Error opening file");
     return -1;
  }

  char str[MAX_SIZE] = {'\0'};
  do {
     if (fgets(str, MAX_SIZE, fp)) {
        printf ("File content: %s.\n", str);
        break;
     }
     if (feof(fp)) {
        printf("End-of-File was reached.");
        break;
    }
    if (ferror(fp)) {
        printf("Error in reading.");
        break;
    }
  } while( 0 );
  fclose (fp);

  return 0;
}
```

Ví dụ - đọc dữ liệu theo format

```
int main ()
{
  FILE* fp = fopen("myfile.txt","rt");
  if (!fp) {
     printf("Error opening file");
     return -1;
  }

  int a = 0;
  int b = 0;
  int n = fscanf(fp, "%d %d", &a, &b);

  printf("There are %d items read", n);
  fclose (fp);

  return 0;
}
```

Ghi nội dung file văn bản

- Hàm ghi 1 kí tự ra file văn bản

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

- Hàm ghi chuỗi kí tự ra file văn bản

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

  - Ghi đến khi gặp kí tự kết thúc chuỗi

- Hàm ghi dữ liệu có định dạng ra file văn bản

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

Đường dẫn file

- Đường dẫn tương đối
  - Chỉ cần tên của file muốn mở
  - Chạy từ IDE: đặt chung thư mục với file chứa hàm main
  - Chạy từ CMD: đặt cạnh file thực thi
- Đường dẫn tuyệt đối
  - Chỉ định cụ thể đường dẫn từ ổ đĩa/thư mục gốc đến file cần mở
- Dấu phân cách thư mục – thư mục/tập tin
  - Dùng forward slash /
  - Dùng double backward slash \\

```
FILE* fp1 = fopen("D:/MyData/look.txt", "rt");
FILE* fp2 = fopen("D:\\MyData\\look.txt", "rt");
```

Tóm tắt đọc/ghi file văn bản

	Đọc	Ghi
Kí tự	`int fgetc(FILE *fp);`	`int fputc(int c, FILE *fp)`
Chuỗi	`char* fgets(char *buffer, int num, FILE *fp);`	`int fputs(const char* s, FILE *fp);`
Format	`int fscan(FILE *fp, const char* format, ...)`	`int fprintf(FILE *fp, const char* format, ...)`

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Đọc nội dung file nhị phân

- Hàm đọc 1 khối dữ liệu từ file nhị phân

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

- Chức năng:
  - Đọc 1 mảng gồm `count` phần tử, mỗi phần tử có kích thước `size`, từ vị trí hiện tại của con trỏ file `fp` vào khối bộ nhớ do `ptr` trỏ đến
- Trả về:
  - Số lượng phần tử đọc được

http://www.cplusplus.com/reference/cstdio/fread/ (05/2020)

Đọc nội dung file nhị phân

- Hàm đọc 1 khối dữ liệu từ file nhị phân

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

- Lưu ý:
  - Lỗi có thể phát sinh khi đọc file hoặc hết file nên số lượng phần tử `fread` trả về có thể khác `count`. Cần dùng `ferror` hoặc `feof` để xác định
  - Sau mỗi lần gọi `fread` dù thành công hay thất bại, `fp` vẫn bị di chuyển so với vị trí ban đầu
  - Nếu `size` hoặc `count` bằng 0, `fread` trả về 0 và `fp` vẫn giữ nguyên vị trí hiện tại

http://www.cplusplus.com/reference/cstdio/fread/ (05/2020)

Ghi nội dung file nhị phân

- Hàm ghi 1 khối dữ liệu ra file

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

- Chức năng:
  - Ghi 1 mảng gồm `count` phần tử, mỗi phần tử có kích thước `size`, do `ptr` trỏ đến ra file từ vị trí hiện tại của con trỏ `fp`
- Trả về:
  - Số lượng phần tử ghi được

http://www.cplusplus.com/reference/cstdio/fwrite/ (05/2020)

Ghi nội dung file nhị phân

- Hàm ghi 1 khối dữ liệu ra file

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

- Lưu ý:
  - Số lượng phần tử ghi thành công có thể khác count, dùng `ferror` để check
  - Sau mỗi lần gọi `fwrite`, dù thành công hay thất bại `fp` vẫn bị di chuyển đi
  - Nếu `size` hoặc `count` bằng 0, `fwrite` trả về 0 và `fp` vẫn giữ nguyên vị trí hiện tại

http://www.cplusplus.com/reference/cstdio/fwrite/ (05/2020)

Ghi kí tự ra file nhị phân

- Hàm ghi 1 kí tự ra file

```
int fputc(const char* s, FILE *fp);
```

- Hàm ghi 1 chuỗi kí tự ra file

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

- Lưu ý:
  - Ghi đến khi gặp kí tự kết thúc chuỗi
  - Nên dùng `fwrite` hơn

Các hàm quan trọng khác

- Hàm kiểm tra cuối file

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

- Hàm kiểm tra lỗi thao tác file

```
int ferror(FILE *fp);
```

Các kiểu số nguyên chuẩn

- Thư viện: `stdint.h`
- Không phụ thuộc hệ máy/platform
- Kiểu số nguyên chuẩn

Kiểu	Kích thước
`int8_t, uint8_t`	1 byte
`int16_t, uint16_t`	2 bytes
`int32_t, uint32_t`	4 bytes
`int64_t, uint64_t`	8 bytes

Các thao tác trên file nhị phân

```
#include <stdint.h>

struct Fraction {
    int16_t num;
    int16_t denom;
};
```

```
int main ()
{
  Fraction a;
  a.num = 10;
  a.denom = 12;

  FILE* fp = fopen("myfile.bin","wb");
  if (!fp) {
     printf("Error opening file");
     return -1;
  }

  size_t n_write = fwrite( &a, sizeof(a), 1, fp );
  if ( n_write > 0 ) {
     printf("Write successfully");
  }
  else {
     printf("Error writing file");
  }

  fclose (fp);
  return 0;
}
```

Các thao tác trên file nhị phân

```
int main ()
{
  Fraction b;

  FILE* fp = fopen(“myfile.bin”,“rb”);
  if (!fp) {
     printf("Error opening file");
     return -1;
  }

  size_t n_read = fread( &b, sizeof(b), 1, fp );
  if ( n_read > 0 ) {
     printf(“Read successfully, num = %d, denom = %d”,
         b.num, b.denom);
  }
  else {
     printf(“Error reading file”);
  }

  fclose (fp);
  return 0;
}
```

Thao tác trên file nhị phân

Giải trí:

- Viết hàm phát sinh ngẫu nhiên `n` phân số và ghi ra file
- Viết hàm đọc `n` phân số từ file và tìm phân số nhỏ nhất

- Viết hàm phát sinh ngẫu nhiên mảng 2 chiều có số dòng `m`, số cột `n` và lưu ra file
- Viết hàm đọc mảng 2 chiều từ file và in ra file khác danh sách tất cả các phần tử nhỏ nhất trong mảng 2 chiều

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Định vị con trỏ file

Thiết lập vị trí con trỏ file

Làm sạch vùng đệm

Các con trỏ file đặc biệt

Tái kết nối con trỏ file

Định vị con trỏ file

Thiết lập vị trí con trỏ file

Làm sạch vùng đệm

Các con trỏ file đặc biệt

Tái kết nối con trỏ file

Định vị con trỏ file

- Hàm trả về vị trí con trỏ file

```
long int ftell( FILE * fp );
```

- Chức năng:
  - Trả về vị trí hiện tại của con trỏ `fp`
  - File nhị phân: số byte tính từ đầu file
  - File văn bản: …
- Trả về:
  - Vị trí hiện tại của con trỏ file
  - `-1L` nếu lỗi, truy xuất biến `errno` để biết mã lỗi

Thiết lập vị trí cho con trỏ file

- Hàm thiết lập vị trí cho con trỏ file

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

- Chức năng:
  - Thiết lập vị trí mới cho con trỏ `fp`
  - File nhị phân: đặt con trỏ `fp` tại vị trí cách `offset` bytes tính từ vị trí tham chiếu do `origin` quy ước
  - File văn bản: …
- Trả về:
  - 0 nếu thành công
  - ≠ 0 nếu ngược lại

http://www.cplusplus.com/reference/cstdio/fseek/ (05/2020)

Thiết lập vị trí cho con trỏ file

- Hàm thiết lập vị trí cho con trỏ file

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

- Tham số `origin`:

Hằng số	Vị trí tham chiếu
SEEK_SET	Tính từ đầu file
SEEK_CUR	Tính từ vị trí hiện tại của con trỏ
SEEK_END	Tính từ cuối file (*)

(*) Thư viện cài đặt có thể không hỗ trợ

http://www.cplusplus.com/reference/cstdio/fseek/ (05/2020)

Thiết lập vị trí cho con trỏ file

```
#include <stdio.h>

int main ()
{
  FILE * fp = nullptr;
  fp = fopen ( "example.txt" , "wb" );

  fputs ( "This is an apple." , fp );
  fseek ( fp , 9 , SEEK_SET );
  fputs ( " sam" , fp );

  fclose ( fp );
  return 0;
}
```

http://www.cplusplus.com/reference/cstdio/fseek/ (05/2020)

Định vị và thiết lập vị trí con trỏ file

- Xác định kích thước file theo bytes

```
#include <stdio.h>

int main ()
{
  FILE * fp = fopen("myfile.txt", "rb");
  if ( !fp ) {
     printf("Error opening file");
     return 1;
  }

  fseek(fp, 0, SEEK_END);
  long size = ftell(fp);

  fclose(fp);
  printf("Size of myfile.txt: %ld bytes.\n", size);

  return 0;
}
```

Thiết lập con trỏ về đầu file

- Hàm thiết lập con trỏ về đầu file

```
void rewind( FILE * fp );
```

- Chức năng:
  - Đưa con trỏ `fp` về đầu file
- Lưu ý:
  - Đối với file được mở ở chế độ read + write, `rewind` có thể được dùng để chuyển đổi qua lại giữa việc đọc và ghi

Thiết lập con trỏ về đầu file

```
#include <stdio.h>

int main ()
{
  FILE * fp = fopen("myfile.txt", "w+");
  for ( char c = 'A' ; c <= 'Z' ; c++) {
    fputc(c, fp);
  }

  rewind(fp);

  char buffer [27];
  fread(buffer, 1, 26, fp);
  fclose(fp);

  buffer[26]='\0';
  puts(buffer);

  return 0;
}
```

Định vị con trỏ file

Thiết lập vị trí con trỏ file

Làm sạch vùng đệm

Các con trỏ file đặc biệt

Tái kết nối con trỏ file

Làm sạch vùng đệm

- Hàm đẩy (tống) dữ liệu từ vùng đệm (buffer) ra file

```
int fflush ( FILE * fp );
```

- Chức năng:
  - Nếu file mở để ghi, hàm `fflush` ghi tất cả dữ liệu trong vùng đệm đang chờ ghi ra file do `fp` trỏ tới
  - Nếu `fp` là `nullptr`, tất cả vùng đệm file của tiến trình sẽ được làm sạch
- Lưu ý:
  - File vẫn mở sau hàm này
  - Nếu file bị đóng (bằng fclose hoặc tiến trình bị terminated), tất cả vùng đệm của file đó sẽ được làm sạch tự động

Làm sạch vùng đệm

- Hàm đẩy (tống) dữ liệu ra file

```
int fflush ( FILE * fp );
```

- Thời điểm làm sạch:
  - Nếu file mở ở chế độ read + write, vùng đệm sẽ được làm sạch tự động sau mỗi lần chuyển từ thao tác write sang read.
  - Các thao tác `fseek`, `rewind`, `fsetpos` sẽ kích hoạt việc làm sạch vùng đệm
  - Chủ động gọi

Làm sạch vùng đệm

```
#include <stdio.h>

int main()
{
   FILE * fp = fopen(“example.txt”, “r+”);
   if (!fp) {
     printf("Error opening file");
     return -1;
   }

   char my_buffer[80];

   fputs(“test”, fp);
   fflush(fp);    // flushing or repositioning required
   fgets(my_buffer, 80, fp);

   puts(my_buffer);
   fclose(fp);

   return 0;
}
```

Định vị con trỏ file

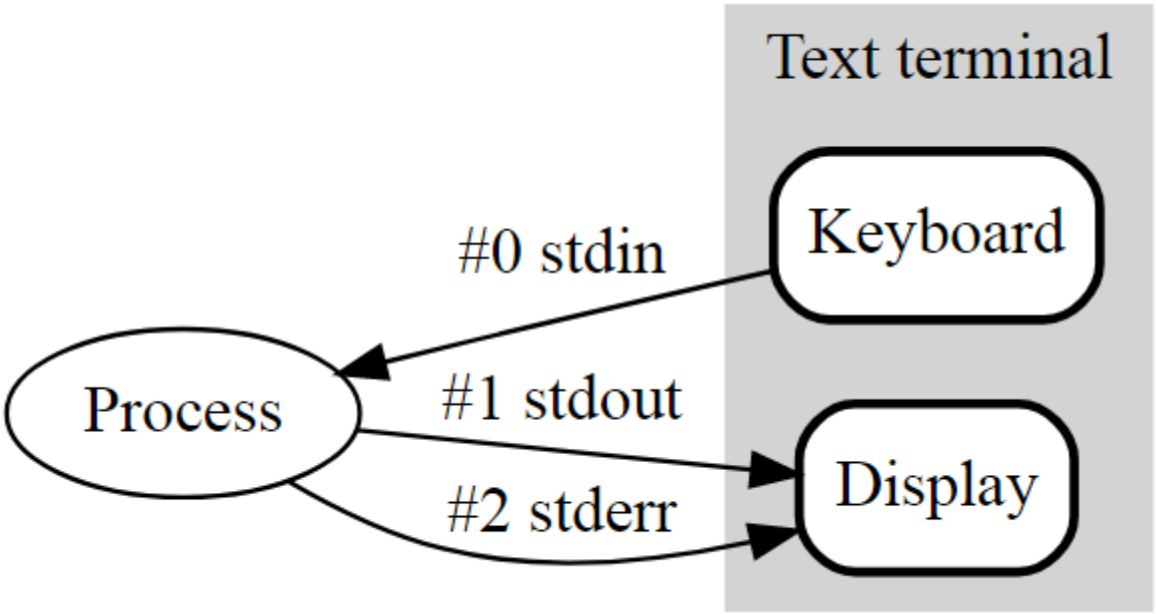
Thiết lập vị trí con trỏ file

Làm sạch vùng đệm

Các con trỏ file đặc biệt

Tái kết nối con trỏ file

Các con trỏ file đặc biệt



https://vi.wikipedia.org/wiki/%C4%90%E1%BA%B7c_t%E1%BA%A3_t%E1%BA%ADp_tin (05/2020)

Các con trỏ file đặc biệt

Con trỏ `stdin`:

- Standard Input Stream
- Được kết nối trực tiếp đến thiết bị nhập chuẩn cung cấp dữ liệu (thường là bàn phím)
- Dùng như con trỏ file đối với tất cả hàm thao tác trên file
- Được khai báo sẵn
- Được mở sẵn, không cần/được đóng
- Có thể kết nối đến thiết bị nguồn khác không phải bàn phím
- Được trang bị vùng nhớ đệm

Các con trỏ file đặc biệt

Con trỏ `stdout`:

- Standard Output Stream
- Được kết nối trực tiếp đến thiết bị xuất chuẩn để kết xuất dữ liệu (thường là màn hình)
- Dùng như con trỏ file đối với tất cả hàm thao tác trên file
- Được khai báo sẵn
- Được mở sẵn, không cần/được đóng
- Có thể kết nối đến thiết bị đích khác không phải màn hình
- Được trang bị vùng nhớ đệm

Các con trỏ file đặc biệt

Con trỏ `stderr`:

- Standard Error Stream
- Được kết nối trực tiếp đến thiết bị xuất chuẩn dùng xử lý thông điệp lỗi hoặc chẩn đoán cảnh báo (thường là màn hình)
- Dùng như con trỏ file đối với tất cả hàm thao tác trên file
- Được khai báo sẵn
- Được mở sẵn, không cần/được đóng
- Có thể kết nối đến thiết bị đích khác không phải màn hình
- **Không** được trang bị vùng nhớ đệm

Các con trỏ file đặc biệt

```
#include <stdio.h>

int main ()
{
  int a = 0;

  fscanf(stdin, "%d", &a);
  fprintf(stdout, "Hello world!");

  return 0;
}
```

Các con trỏ file đặc biệt

```
#include <stdio.h>

int main ()
{
  FILE* fp = fopen ("myfile.txt", "rb");
  if ( !fp ) {
     printf("Error opening file");
     return -1;
  }

  char s[100];
  size_t n_read = fread(s, 1, 100, fp);
  s[n_read] = '\0';
  fclose(fp);

  fwrite(s, 1, n_read, stdout);

  return 0;
}
```

Định vị con trỏ file

Thiết lập vị trí con trỏ file

Làm sạch vùng đệm

Các con trỏ file đặc biệt

Tái kết nối con trỏ file

Tái kết nối con trỏ file

- Hàm tái kết nối con trỏ file

```
FILE * freopen ( const char * filename, const char *
mode, FILE * fp );
```

- Chức năng:
  - Tái sử dụng con trỏ file `fp` để mở file `filename` hoặc thay đổi chế độ mở hiện tại thành `mode`
- Trả về:
  - Con trỏ file `fp` nếu tái kết nối thành công
  - `nullptr` nếu ngược lại, dùng biến `errno` để xác định lỗi

Tái kết nối con trỏ file

```
#include <stdio.h>

int main ()
{
  freopen ("myfile.txt", "wt", stdout);
  printf ("This sentence is redirected to a file.");
  fclose (stdout);

  return 0;
}
```

Đóng tất cả file

- Hàm đóng tất cả con trỏ file đang mở

```
int fcloseall ( void );
```

- Chức năng:
  - Đóng tất cả file đang được process hiện tại mở
  - Windows: không đóng các con trỏ file chuẩn stdin, stdout và stderr
  - Linux: đóng luôn cả những con trỏ file chuẩn
- Trả về:
  - Windows: số file đóng được nếu thành công và EOF nếu ngược lại
  - Linux: 0 nếu thành công và EOF nếu ngược lại

https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/fclose-fcloseall?view=vs-2019 (05/2020)
https://www.man7.org/linux/man-pages/man3/fcloseall.3.html (05/2020)

Đóng tất cả file

- Hàm đóng tất cả con trỏ file đang mở

```
int fcloseall ( void );
```

- Lưu ý:
  - Hàm này hoàn toàn không khả chuyển (portable): mỗi nền tảng định nghĩa behavior và giá trị trả về khác nhau
  - Không nằm trong danh sách hàm được thư viện chuẩn hỗ trợ
  - Windows: đã bỏ hàm này, dùng **`_fcloseall`** để thay thế

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Luồng dữ liệu trong C++

- C++ hỗ trợ các lớp đối tượng tương tác với file
- Thư viện: `fstream`

Lớp	Chức năng
`std::ofstream`	Ghi ra file
`std::ifstream`	Đọc từ file
`std::fstream`	Vừa đọc vừa ghi

Phương thức chung

std::ofstream	std::ifstream	std::fstream
void open(const char* filename, ios_base::openmode mode)		
bool is_open()		
void close()		
bool good() const bool bad() const		
bool eof() const		
...		

Phương thức

std::ofstream	std::ifstream	std::fstream
<<	>>	<<, >>
put(...)	get(...)	put(...), get(...)
write(...)	read(...)	write(...), read(...)
flush()		flush()

Luồng dữ liệu trong C++

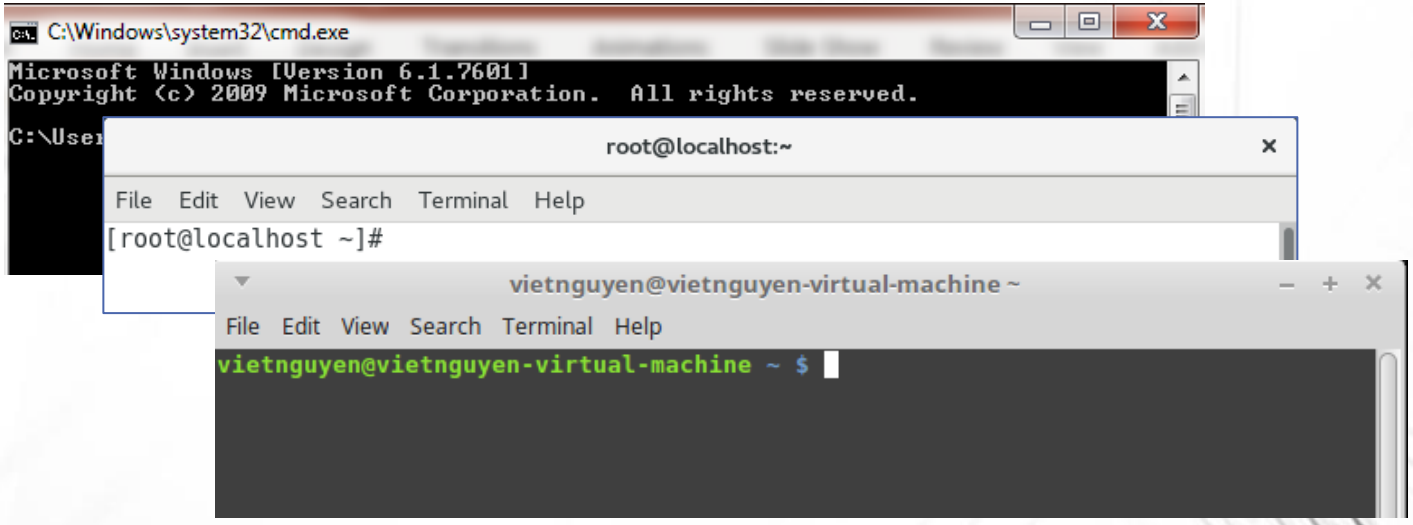
```
#include <iostream>
#include <fstream>

int main ()
{
  std::ofstream my_file(“myfile.txt”, ios::out | ios::app);

  if (my_file.is_open())
  {
    my_file << “Write a line to file.\n";
    my_file.close();
  }
  else
  {
    std::cout << "Unable to open file";
  }

  return 0;
}
```

Nội dung

- File và con trỏ file
- Thao tác cơ bản trên file
- Đọc/ghi file văn bản
- Đọc/ghi file nhị phân
- Nâng cao
- Luồng dữ liệu trong C++
- Tham số dòng lệnh

Tham số dòng lệnh là gì ?

- Là những tham số đi kèm ngay sau tên chương trình trong giao diện dòng lệnh của hệ điều hành
  - CMD trong Windows
  - Terminal trong Linux/macOSX
- Cho phép truyền tham số cho chương trình từ bên ngoài



Cách khai báo

- Sử dụng hàm main có tham số
  - `argc` – argument count: số lượng tham số được truyền, bao gồm cả tên chương trình
  - `argv` – argument value: mảng chuỗi các tham số
  - `envp` – Environment pointer: mảng chuỗi các biến môi trường (chỉ Windows)

```
int main(int argc, char *argv[]) { ... }

int main(int argc, char **argv) { ... }

int main(int argc, char *argv[], char *envp[]) { ... }

int main(int argc, wchar_t *argv[], wchar_t *envp[]) { ... }

int main(int argc, TCHAR_t *argv[]) { ... }
```

Tham số dòng lệnh

```
// main of lab.exe

int main (int argc, char* argv[])
{
    for( int i = 0 ; i < argc ; i++ )
    {
        printf("- %s\n", argv[i]);
    }
    return 0;
}
```

```
D:\MyProject\Debug\lab.exe -fn Viet -ln Nguyen
```

Tham số dòng lệnh

Đặc điểm của tham số dòng lệnh:

- `argv[0]`: tên chương trình /đường dẫn đến file thực thi
- `argv[1]`: tham số đầu tiên của chương trình
- `argv[argc]`: con trỏ `nullptr`
- Các tham số phân biệt nhau bởi dấu `SPACE`/`TAB`
- Nếu tham số có chứa `SPACE`, cần rào bằng “” hoặc ‘’

Nguyên tắc parse tham số dòng lệnh

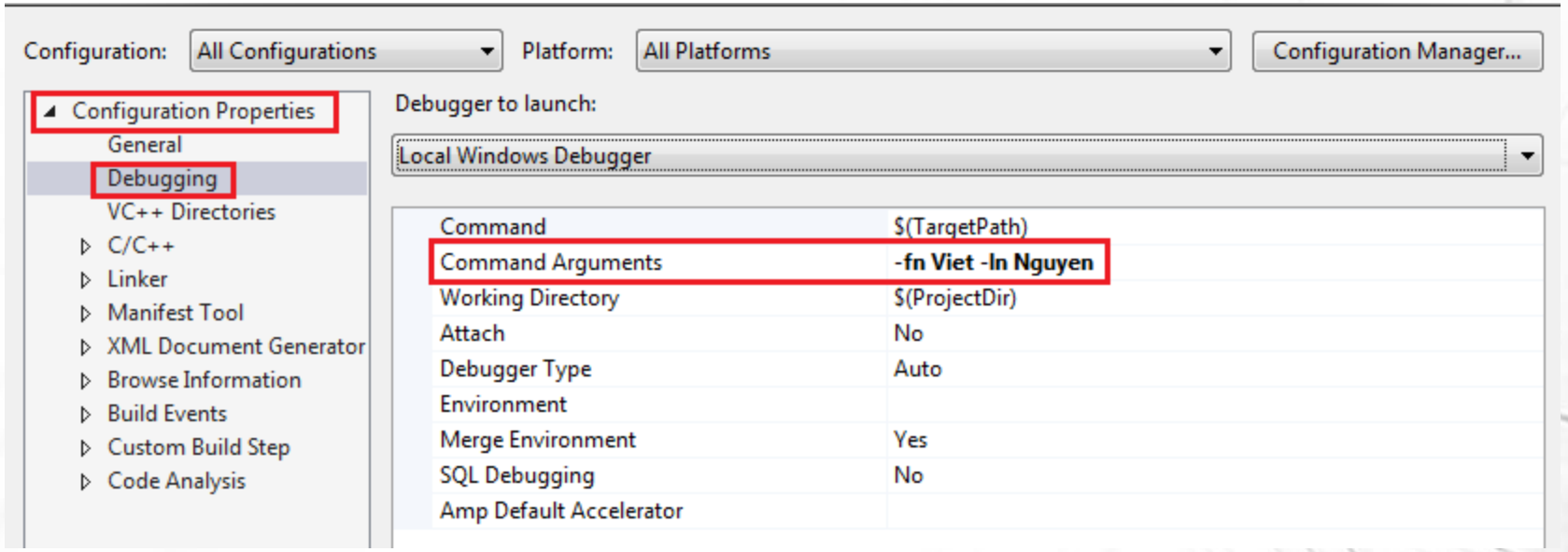
- Dấu caret ^ trong tham số dòng lệnh sẽ bị bỏ qua
- Dấu \ được hiểu là 1 kí tự
- Tổ hợp \" được hiểu là "
- Tổ hợp \\ được hiểu là \

Tham số	argv[0]	argv[1]	argv[2]
ab^c d	abc	d	
"abc" d e	abc	d	e
a\\b d"e f"g h	a\\b	de fg	h
a\\\"b c d	a\"b	c	d
a\\\\"b c" d e	a\\b c	d	e

https://docs.microsoft.com/en-us/cpp/cpp/main-function-command-line-args?view=vs-2019 (05/2020)

Debug mode

- Truyền tham số dòng lệnh mặc định trong chế độ debug
- Vào menu Project → <Project_name> Properties … → Configuration Properties → Debugging → Command Arguments



Đánh giá đạt mục tiêu

Sau bài học, liệu sinh viên có thể:

- Đọc/ghi file văn bản ?
- Đọc/ghi file nhị phân ?
- Sử dụng tham số dòng lệnh ?