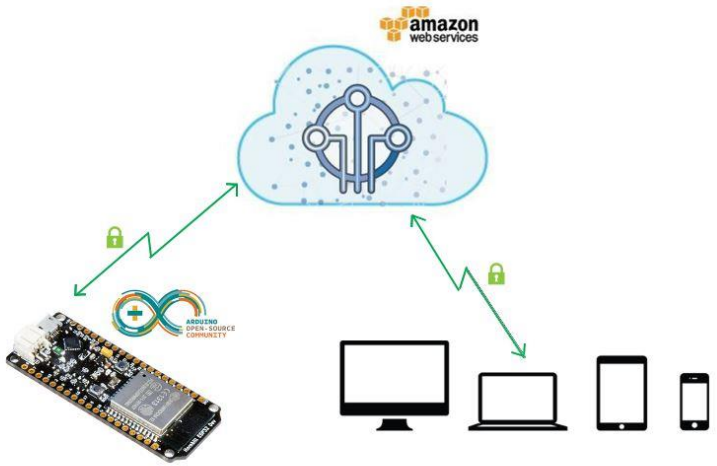
Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Xuất/Nhập File

Tại Sao Cần Xuất/Nhập File?

- Lượng lớn dữ liệu đầu vào.
- Lượng lớn dữ liệu đầu ra.
- Khả năng lưu trữ dữ liệu lâu dài
- Truyền dữ liệu cho các chương trình khác.
- Các dòng xuất/nhập diễn ra đồng thời.



File

- *File* là một tập hợp của dữ liệu ghi trên đĩa
  - Được quản lý bởi người dùng và hệ điều hành
  - Có thể lưu giữ dữ liệu lâu dài
- *Tên file* là phương tiện để người dùng và hệ điều hành nhận biết file
  - Tuân theo quy tắc đặt tên 8.3 trong DOS
- Chúng ta sẽ ôn lại các file dùng trong quá trình biên dịch; ôn lại việc xuất/nhập qua bàn phím, màn hình; và xem xét việc dùng các file text trong chương trình C.
- Nhưng trước hết, chúng ta tìm hiểu các file dữ liệu.

File IO

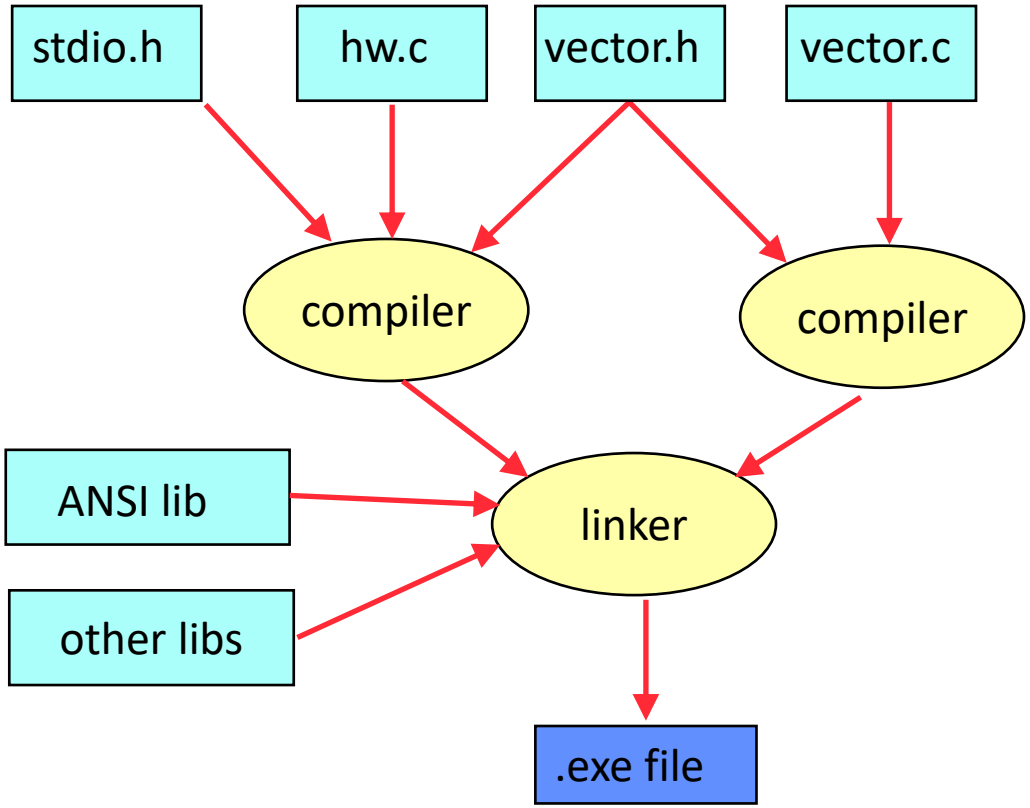
- Ngôn ngữ C có mối quan hệ mật thiết với hệ điều hành UNIX
- Phần lớn thư việc C đều theo mô hình vào ra UNIX coi tất cả đều là các file.
- Bàn phím, màn hình, cổng nối tiếp, GPIO đều đươc truy cập thông qua đọc và viết file
- Cung cấp một giao diện chung cho mọi IO

Ôn Lại: File Dùng Khi Biên Dịch

- File nguồn
  - .c file: chương trình và hàm viết trên C
  - .h file: khai báo
  - Các dự án trong thực tế có thể có hàng trăm file .c và .h
- File được dịch ra (tên tùy thuộc vào từng hệ thống)
  - File đối tượng (object): file đã được dịch và chờ để liên kết
  - File thư viện (library): tập hợp các hàm đã được dịch sẵn
  - File thi hành (executable): file mã máy đã được liên kết, sẵn sàng để chạy trong bộ nhớ máy tính

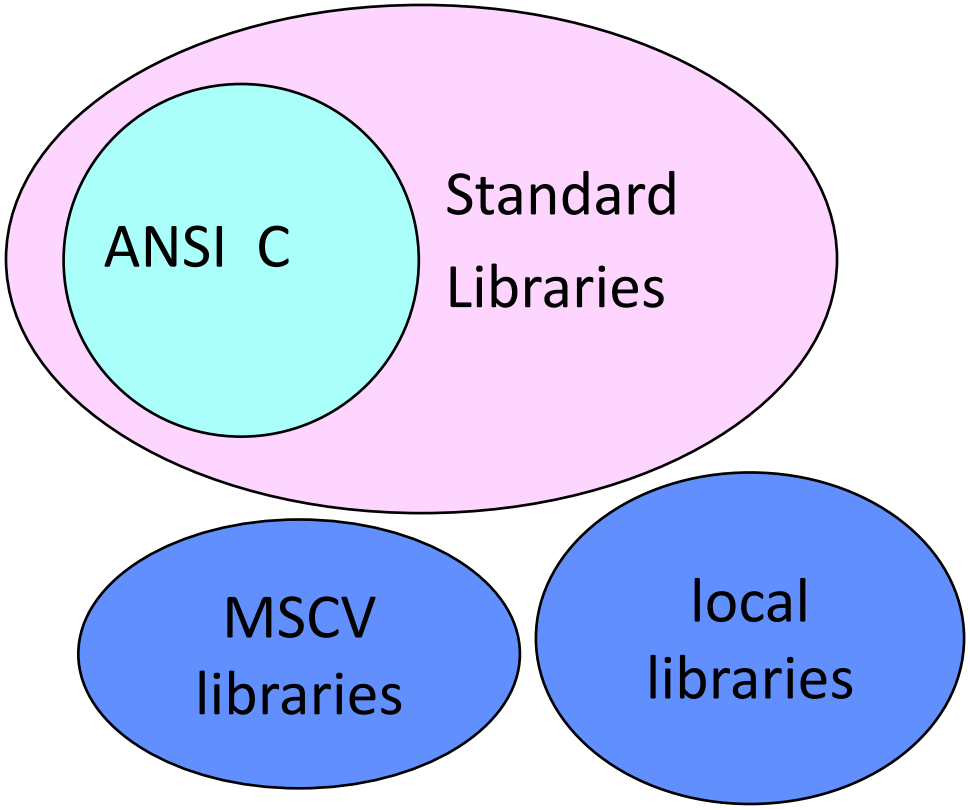
File Tiêu Đề (.h)

- Tệp tiêu đề thường được sử dụng để chứa
  - Định nghĩa kiểu (sử dụng **typedef**)
  - Nguyên mẫu hàm
  - Hằng số tượng trưng
  - Khai báo biến toàn cục



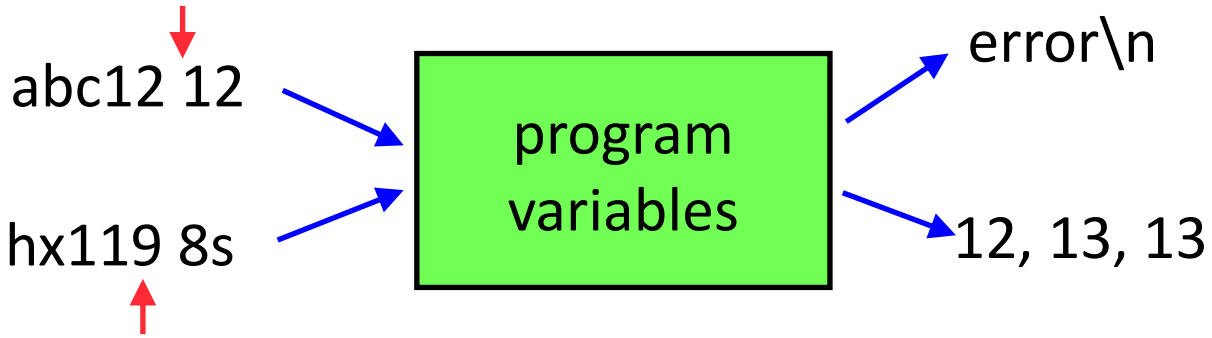
Thư Viện

- Là các file chứa các hàm được viết và dịch sẵn
  - Giảm phụ thuộc vào hệ thống
  - Sử dụng lại mã đã có
  - Tăng tính khả chuyển



Xuất/Nhập Ký Tự Với File

- Bàn phím / màn hình là những trường hợp đặc biệt của các dòng xuất/nhập các ký tự



- Nhiều dòng xuất/nhập có thể tồn tại cùng lúc.
- Trong thực tế, các dòng xuất nhập làm việc với một bộ đệm thay vì trực tiếp với các biến.

Có Gì Trong File **stdio.h**

- Nguyên mẫu của các hàm xuất/nhập
- Định nghĩa các hằng số hữu dụng
  - Ví dụ **EOF**
- Định nghĩa cấu trúc **FILE** để biểu diễn thông tin các file sử dụng trong chương trình C
  - Biến file trong C là các con trỏ tới cấu trúc **FILE**

    FILE *myfile;

Mở File

- Mở file: tạo ra một mối liên kết giữa hệ điều hành (tên file) và chương trình C (biến file)
  - Hàm thư viện **fopen**
  - Xác định tham số **“r”** để đọc file và **“w”** để ghi file
    - Chú ý **“r”** chứ không phải **‘r’**
- File phải được mở trước khi chúng được sử dụng
- File **stdin/stdout** (sử dụng bởi **scanf/printf**) sẽ tự động mở và kết nối tới bàn phím và màn hình

Ví Dụ Về Mở File

```
/*usually done only once in a program*/
/*usually done near beginning of program*/

FILE *infilep, *outfilep; /*file variables*/
char ch;

/* Open input and output files */
infilep  = fopen("Student_Data", "r");
outfilep = fopen("New_Student_Data", "w");
```

Ví Dụ Về Đóng File

- Thường chỉ được làm một lần trong chương trình.
- Thường được làm cuối chương trình.
- Bạn cần phải đóng các file dữ liệu nếu không dữ liệu có thể sẽ bị mất.

```
FILE *infilep;  /*file variable*/

...
infilep = fopen("Student_Data", "r");
.../*process the file */
.../*when completely done with the file:*/
fclose(infilep);
```

Kết Thúc File (EOF)

- Được định nghĩa trong file **stdio.h**
- **#define EOF (một giá trị âm nào đó)**
  - Thông thường là -1
  - Các hàm thư viện xuất/nhập dùng **EOF** để biểu thị kết thúc file
  - Chương trình của bạn cùng có thể dùng **EOF**
- Chú ý: **EOF** là trạng thái, không phải là giá trị đầu vào.

Bốn Hàm Xuất/Nhập File Cơ Bản

- **fopen** và **fclose**: đã nêu ở các trang trước
- **fscanf**: giống như hàm **scanf**, tuy nhiên tham số thứ nhất là một biến file

```
status = fscanf(filepi, "%...", &var,...);

/* fscanf returns EOF on end of file */
```

- **fprintf**: giống như hàm **printf**, tuy nhiên tham số thứ nhất là một biến file

```
fprintf(filepo, "%...", var,...);
```

- File phải được mở trước khi bạn gọi hàm **fscanf** hoặc **fprintf**

Các thao tác với file

- Khi file được mở, thì các hoạt động cho file sẽ diễn ra từ đầu đến hết file
- 4 kiểu cơ bản khi làm việc với file
  - Từng ký tự (Character by character).
  - Từng dòng (Line by line).
  - Định dạng vào ra (Formatted IO).
  - Vào ra nhị phân (Binary IO).

Gửi ký tự ra

- Các hàm gửi ký tự ra:

  int fputc(int c, FILE *fp);

  int putc(int c, FILE *fp);

  int putchar(int c);

- **putchar(c)** tương đương **putc(c, stdout)**.
- **putc()**và **fputc()** là hoàn toàn giống nhau.
- Giá trị trả về:
  - Nếu thành công: ký tự được viết vào
  - Nếu lỗi: **EOF**.

Đọc ký tự vào

- Hàm đọc ký tự vào:

  int fgetc(FILE *fp);

  int getc(FILE *fp);

  int getchar(void);

- **getchar()** tương đương với **getc(stdin)**.
- **getc()** và **fgetc()** giống nhau.
- Giá trị trả về:
  - Thành công: ký tự tiếp theo trong luồng dữ liệu
  - Nếu lỗi: **EOF**.
  - Nếu hết file: **EOF**.
- **Phân biệt EOF**, bằng lời gọi hàm **feof()** or **ferror()**.
- Bạn có thể đẩy ký tự trở lại luồng dữ liệu vào thông qua hàm **ungetc()**.

  int ungetc(int c, FILE *fp);

Định dạng vào ra IO

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

- Tương tự như hàm **printf()** và **scanf()**
- Thực chất **printf()** và **scanf()** có thể viết

  fprintf(stdout, format, arg1, arg2,);

  fscanf(stdin, format, arg1, arg2,);

Nhập vào dòng

- Đọc cả dòng vào như sau:

  char *fgets(char *buf, int max, FILE *fp);

- Giá trị trả về
  - Đọc vào nhiều nhất là **max-1** ký tự từ file.
  - Đọc cả ký tự a **\n**.
  - Kiểm tra cả hết file và lỗi
- Giá trị trả về:
  - Nếu thành công: con trỏ tới **buf**. Lưu ý **fgets()**tự động thêm **\0** vào cuối xâu.
  - Nếu dòng cuối file: **NULL**.
  - Gặp lỗi: **NULL**.
- Sử dụng **feof()** và **ferror()** để xác định nếu lỗi

Gửi dòng ký tự ra file

- Chuỗi ký tự có thể gửi ra file như sau

  int fputs(const char *str, FILE *fp);

- Việc thêm ký tự **\n** phụ thuộc vào lập trình viên
- Giá trị trả về
  - Thành công: zero.
  - Nếu không: **EOF**.

Vào ra nhị phân

- Khi đọc và ghi file nhị phân, chương trình làm việc trực tiếp với đối tượng mà không chuyển đổi đối tượng này thành xâu ký tự
- Vào ra nhị phân gồm:

```
size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE
   *fp);
```

- Dùng để ghi các cấu trúc ra file

```
struct Astruct mystruct[10];
fwrite(&mystruct, sizeof(Astruct), 10, fp);
```

Các thao tác khác với file

- C cung cấp các hàm khác để có thể làm việc với file không thông qua tuần tự.
- 3 hàm cơ bản:

```
long ftell(FILE *fp);
int fseek(FILE *fp, long offset, int from);
void rewind(FILE *fp);
```

Xây Dựng Ứng Dụng Với File

- Với các hàm **fopen**, **fclose**, **fscanf** và **fprintf** bạn có thể viết rất nhiều ứng dụng liên quan đến file
- Bạn có thể gặp nhiều lỗi và các trường hợp ngoại lệ khi dùng file
  - Một chương trình bền vững phải kiểm soát được lỗi
  - Bạn sẽ dần học được kỹ năng này

Đối số theo dòng lệnh

- C cho phép nhập lệnh ngay từ lời gọi chương trình command-line arguments
- Cấu trúc hàm hàm main có thể **main()**:

  int main(void)

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

- argc *argument count* và argv *argument vector*.
- **argc** là số lượng đối số trong dòng lệnh (bao gồm cả tên chương trình). Các lệnh cách nhau bởi dấu cách
- **Argv** là con trỏ tới mảng của các xâu, với mỗi xâu là một lệnh. Độ dài của mảng là **argc + 1** vì **argv[argc]= NULL**.

Ví Dụ Sao Chép File

```
/* Problem: copy an input file to an output file */

/* Technique: loop, copying one char at a time until
EOF, files must already be open before this */

status = fscanf(infilep, "%c", &ch);

while (status != EOF) {

   fprintf(outfilep, "%c", ch);

   status = fscanf(infilep, "%c", &ch);

}

printf("File copied.\n");

fclose(infilep);

fclose(outfilep);
```

Ví Dụ Sao Chép File

```
/* Many C programmers use this style */
...
while (fscanf(infilep, "%c", &ch) != EOF)
   fprintf(outfilep, "%c", ch);

printf("File copied.\n");
fclose(infilep);
fclose(outfilep);
```

Ví Dụ Truy Vấn Cơ Sở Dữ Liệu

```
#include <stdio.h>
int main(void) {
   FILE *inp, *outp;
   int age, j;
   char name[20], ssn[9], ch;
   inp  = fopen("db_file", "r" );
   outp = fopen("result_file", "w");
   /* loop till the end-of-file */
   while (fscanf(inp, "%c", &name[0]) != EOF) {
      /* read name, ssn, age */
      for (j = 1; j < 20; j++)
         fscanf(inp, "%c", &name[j]);
      for (j = 0; j < 9; j++)
         fscanf(inp, "%c",&ssn[j]);
      fscanf(inp, "%d", &age);
      /* read line feed character */
      fscanf(inp, "%c", &ch);
      /* copy name, ssn to output if age > 20 */
      if (age > 20) {
         for (j = 0; j < 20; j++)
            fprintf(outp, "%c", name[j]);
         for (j = 0; j < 9; j++)
            fprintf(outp, "%c, ssn[j]);
         fprintf(outp, "\n");
      }
   }
   fclose(inp);  fclose(outp);
   return (0);
}
```

Equivalent query in SQL database language:

```
SELECT NAME, SSN
FROM DB_FILE
WHERE AGE > 20;
```

Ví Dụ Mở Rộng Tab

```
#include <stdio.h>
int main(void) {
   FILE *infilep, *outfilep;
   char ch;
   int column = 0;
   /* Open input and output files */
   infilep  = fopen("prog.c", "r");
   outfilep = fopen("tabless-prog.c", "w");
   /* process each input character */
   while (fscanf(infilep, "%c", &ch) != EOF){
      if (ch == '\n' || ch == '\r') {
         /* end of line: reset column counter */
         column = 0;
         fprintf(outfilep, "%c", ch);
      } else if (ch == '\t') {
         /* tab: output one or more spaces, */
         /* to reach the next multiple of 8.    */
         do {
            fprintf(outfilep, "%c", ' ') ;
            column++;
         } while ((column % 8) != 0);
      } else {
         /* all others: count it, and copy it out */
         column ++;
         fprintf(outfilep, "%c", ch);
      }
   }
   fclose(infilep); fclose(outfilep);
   return 0;
}
```

```
Input:    a b \t c
          d \t e f

Output:   a b      c
          d   e f
```

Ví Dụ Nối Hai File Có Thứ Tự

```
#include <stdio.h>
#define MAXLINE 10000 /*ASSUMES no line longer*/
int main(void) {
   FILE *in1p, * in2p, *outp;
   char buffer1[MAXLINE], buffer2[MAXLINE];
   char *stat1, *stat2;
   in1p  = fopen("sorted-file1", "r");
   in2p  = fopen("sorted-file2", "r");
   outp  = fopen("merged-file", "w");
   stat1 = fgets(buffer1, MAXLINE, in1p);
   stat2 = fgets(buffer2, MAXLINE, in2p);
   while (stat1 != NULL && stat2 != NULL) {
      if (strcmp(buffer1, buffer2) < 0) {
         fprintf(outp, "%s", buffer1);
         stat1 = fgets(buffer1, MAXLINE, in1p);
      } else {
         fprintf(outp, "%s", buffer2);
         stat2 = fgets(buffer2, MAXLINE, in2p);
      }
   }
   while (stat1 != NULL) {
      fprintf(outp, "%s", buffer1);
      stat1 = fgets(buffer1, MAXLINE, in1p);
   }
   while (stat2 != NULL) {
      fprintf(outp, "%s", buffer2);
      stat2 = fgets(buffer2, MAXLINE, in2p);
   }
   fclose(in1p);  fclose(in2p);  fclose(outp);
   return 0;
}
```