

CONFIDENTIAL

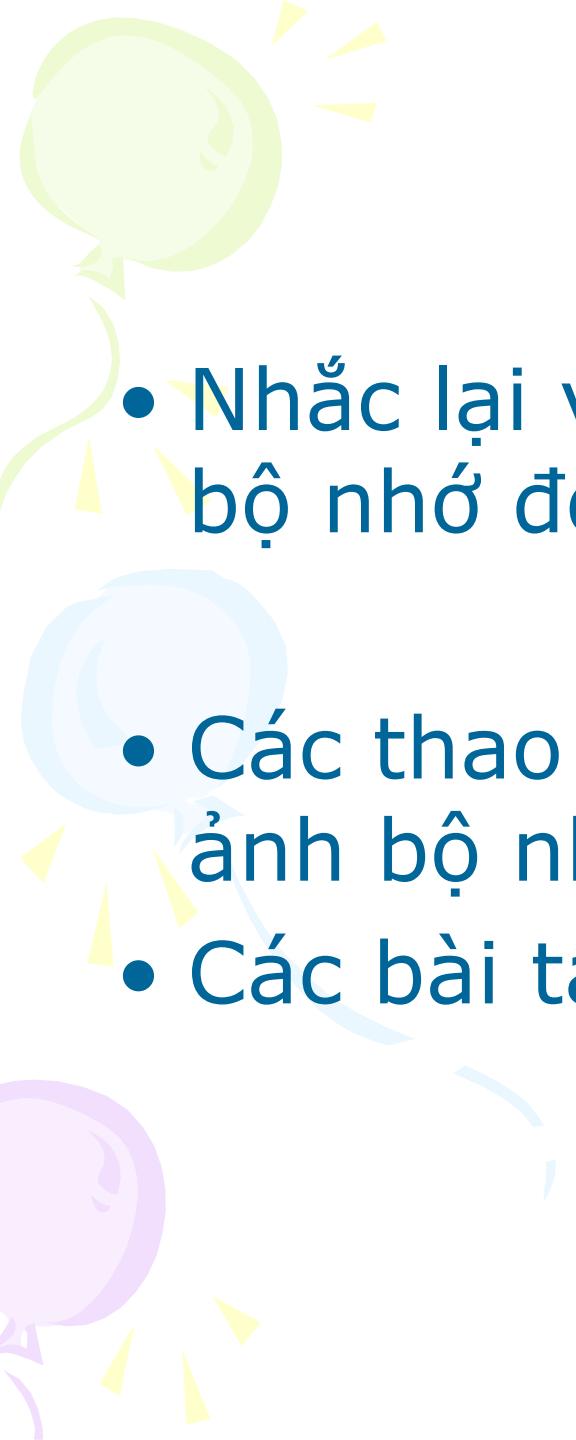
C Programming Basic – week 2

For HEDSPI Project

Lecturer :

Đỗ Quốc Huy

**Dept of Computer Science
Hanoi University of Technology**



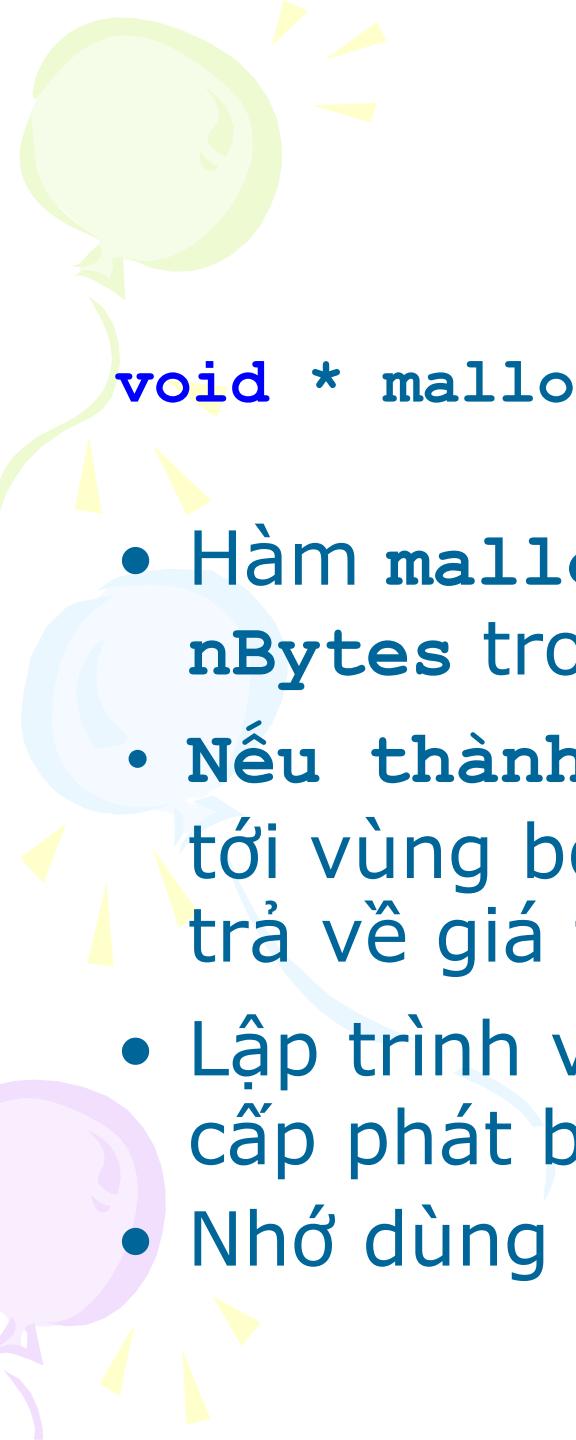
Các chủ đề

- Nhắc lại về các cấu trúc, phân phối bộ nhớ động
- Các thao tác với File dựa trên hình ảnh bộ nhớ
- Các bài tập thực hành



Phân phối bộ nhớ động

- Các biến mảng có kích thước **cố định**, dùng để lưu trữ một số lượng biến cố định đã biết trước –**biết từ thời điểm biên dịch**
- Kích thước này không thể thay đổi sau khi biên dịch
- Tuy nhiên, không phải lúc nào chúng ta cũng có thể biết trước ta cần bao nhiêu bộ nhớ cho một mảng hay biến
- Ta sẽ muốn phân phối bộ nhớ một cách linh động



Hàm malloc

```
void * malloc(unsigned int nbytes);
```

- Hàm `malloc` dùng để phân phối động `nBytes` trong bộ nhớ
- Nếu thành công, `malloc` trả về con trỏ tới vùng bộ nhớ được cấp phát, thất bại sẽ trả về giá trị `NULL`
- Lập trình viên nên luôn luôn kiểm tra việc cấp phát bộ nhớ thành công hay không
- Nhớ dùng lệnh `#include <stdlib.h>`

Example -dynamic_reverse_array

```
int main(void)
{
    int i, n, *p;

    printf("How many numbers do you want to enter?\n");
    scanf("%d", &n);

    /* Phân phôi mảng dữ liệu kiểu int với kích cỡ thích hợp */
    p = (int *)malloc(n * sizeof(int));
    if (p == NULL)
    {
        printf("Memory allocation failed!\n");
        return 1;
    }
    /* Get the numbers from the user */
    ...
    /* Display them in reverse */
    ...
    /* Free the allocated space */
    free(p);
    return 0;
}
```

Example -dynamic_reverse_array

```
int main(void)
{
    . . .
    /* Get the numbers from the user */
    printf("Please enter numbers now:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);

    /* Display them in reverse */
    printf("The numbers in reverse order are - \n");
    for (i = n - 1; i >= 0; --i)
        printf("%d ", p[i]);
    printf("\n");
    free(p);
    return 0;
}
```

Vì sao cần khuôn dạng dữ liệu?

Khuôn dạng dữ liệu trong lệnh

p = (int *)malloc(n*sizeof(int));

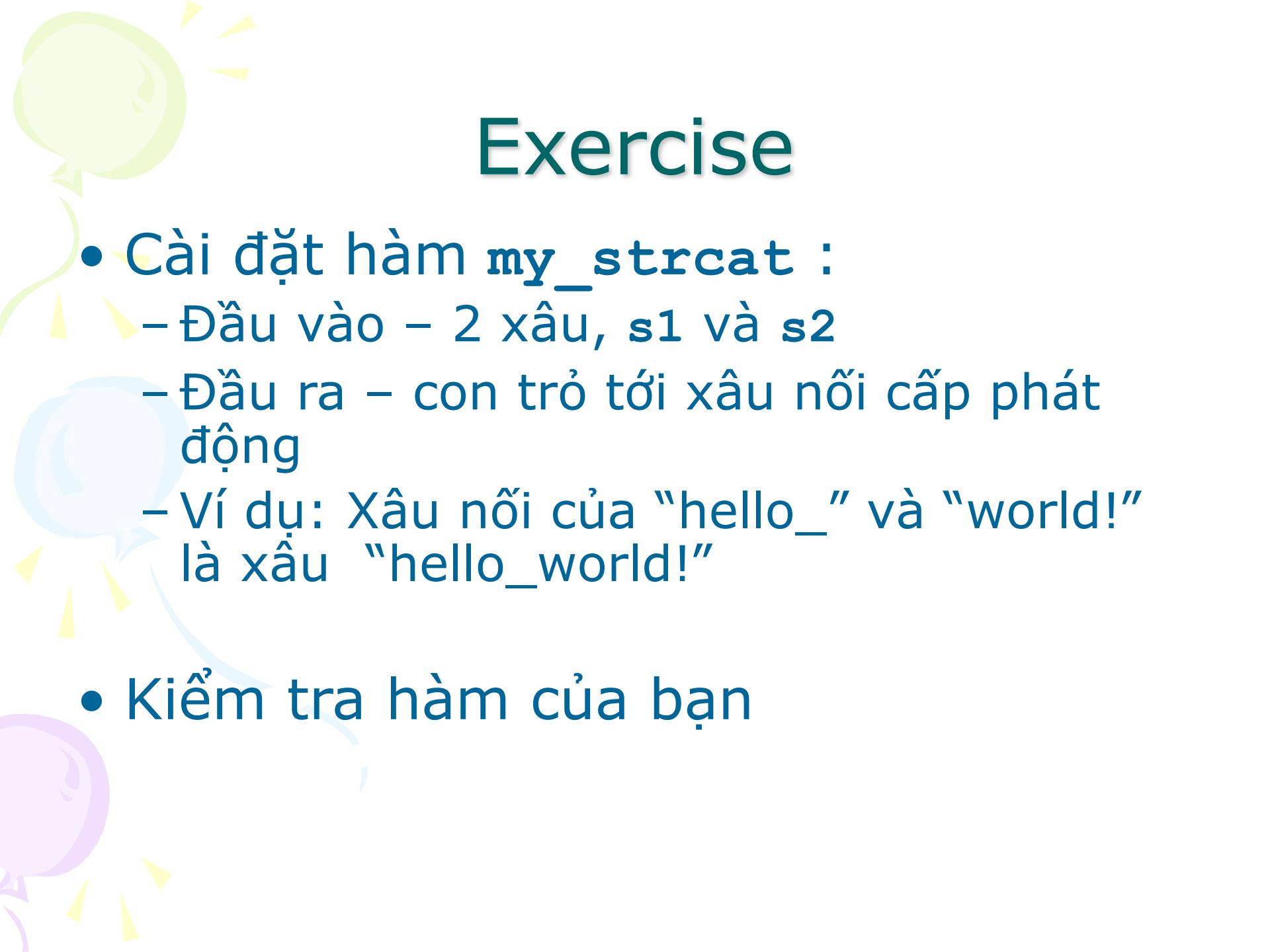
Là cần thiết vì **malloc** trả về kiểu **void *** :
void * malloc(unsigned int nbytes);

Kiểu (**void ***) chỉ ra một con trỏ tổng quát, có thể được chuyển thành bất kỳ kiểu con trỏ dữ liệu khác.

Giải phóng bộ nhớ đã được cấp phát

```
void free(void *ptr);
```

- Ta dùng **free(p)** giải phóng bộ nhớ đã được cấp cho **p**
Nếu **p** không trỏ tới vùng dữ liệu được phân phối bởi **malloc**, một lỗi run-time error sẽ xảy ra
- Luôn **nhớ** phải giải phóng bộ nhớ đã cấp phát khi bạn không dùng đến nó nữa

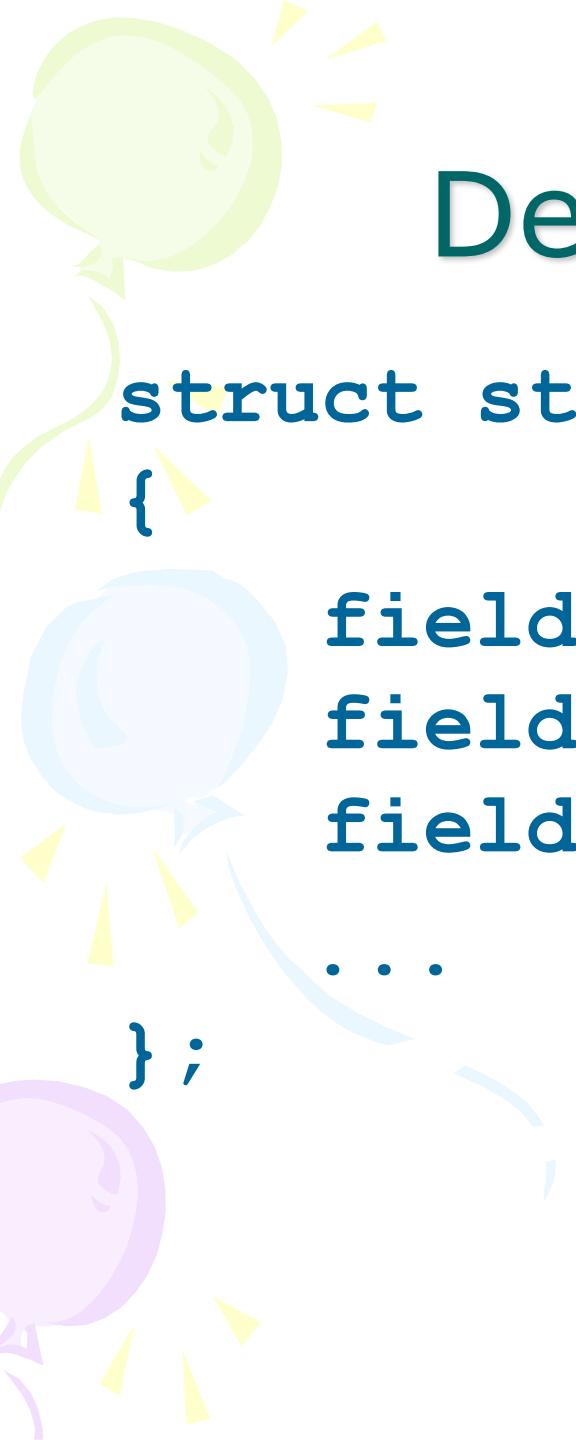


Exercise

- Cài đặt hàm `my_strcat` :
 - Đầu vào – 2 xâu, `s1` và `s2`
 - Đầu ra – con trỏ tới xâu nối cấp phát động
 - Ví dụ: Xâu nối của “hello_” và “world!” là xâu “hello_world!”
- Kiểm tra hàm của bạn

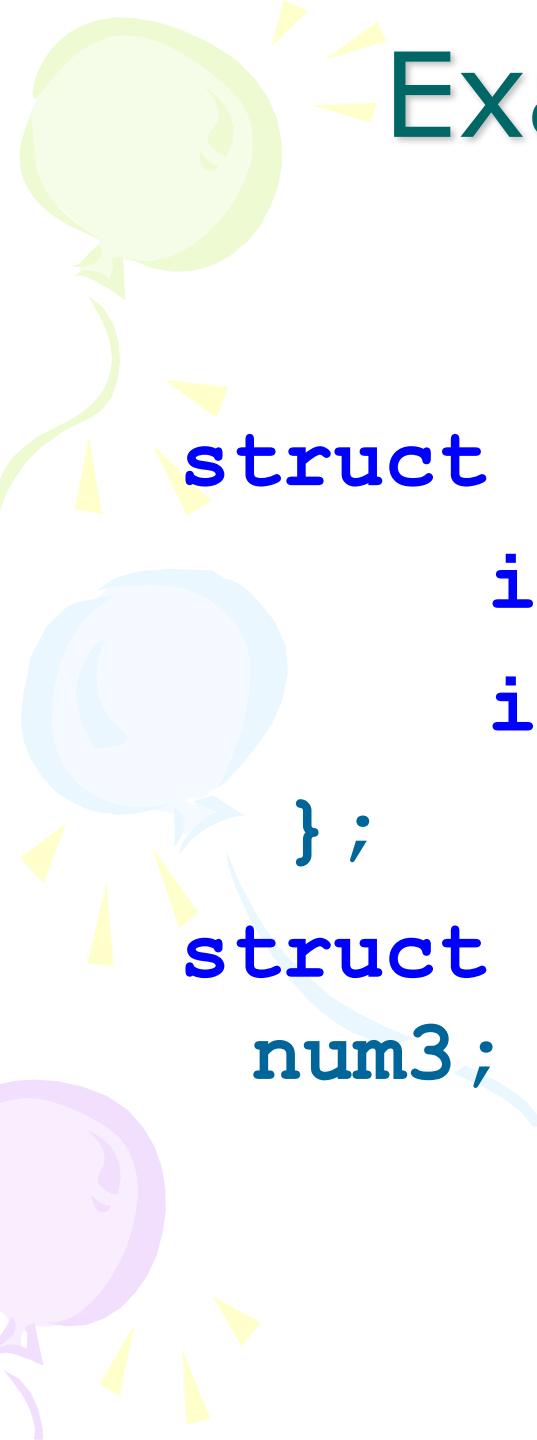
Cấu trúc - Kiểu dữ liệu do người dùng định nghĩa

- Tập các biến trong cùng một tên.
- Cách thuận tiện để nhóm các thông tin liên quan lại với nhau .
- Các biến trong một **struct** (rút gọn của structure) được gọi là các biến thành phần hay các trường.



Defining a struct

```
struct struct-name  
{  
    field-type1 field-name1;  
    field-type2 field-name2;  
    field-type3 field-name3;  
    ...  
};
```



Example - complex numbers

```
struct complex {  
    int real;  
    int img;  
};  
struct complex num1, num2,  
num3;
```

Typedef

- Ta có thể kết hợp `typedef` với định nghĩa cấu trúc:

```
typedef struct complex {  
    int real;  
    int img;  
} complex_t;
```

```
complex_t num1, num2;
```

Exercise

- Given two following structure:

```
typedef struct point
{
    double x;
    double y;
} point_t;
```

```
typedef struct circle
{
    point_t center;
    double radius;
} circle_t;
```

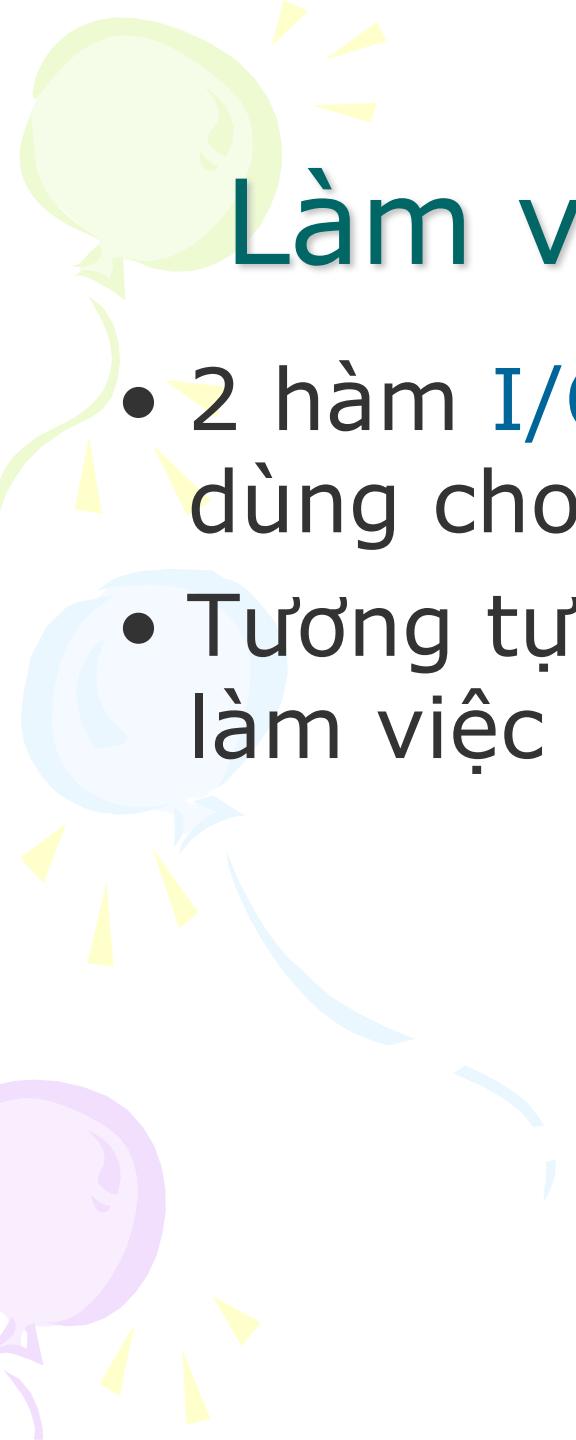
- Viết một hàm `is_in_circle` trả về 1 nếu điểm p bị bao phủ bởi c. Kiểm tra hàm này trong chương trình.

Các con trỏ trong cấu trúc

- Nếu một thành phần của **struct** là một con trỏ, nó sẽ sao chép con trỏ (địa chỉ) bản thân nó
- Exercise: Cho kiểu dữ liệu Student (week 3)

Các chế độ làm việc với file nhị phân

Chế độ	Miêu tả
"rb"	opens an existing binary file for reading.
"wb"	creates a binary file for writing.
"ab"	opens an existing binary file for appending.
"r+b"	opens an existing binary file for reading or writing.
"w+b"	creates a binary file for reading or writing.
"a+b"	opens or create an existing binary file for appending.



File handle: Làm việc với khối dữ liệu

- 2 hàm I/O : `fread()` và `fwrite()`, được dùng cho các thao tác I/O khối.
- Tương tự các file handle khác, chúng làm việc với `file pointer`.

fread()

- Cú pháp hàm fread() là

```
size_t fread(void *ptr, size_t size,  
            size_t n, FILE *stream);
```

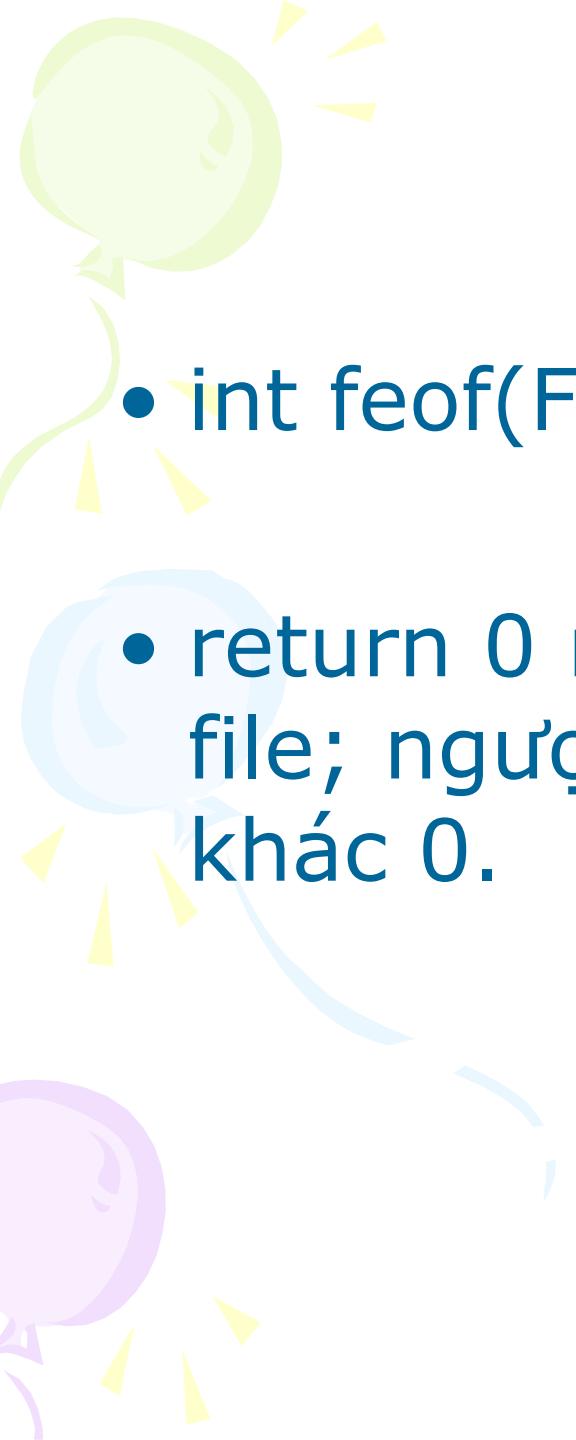
- **Ptr:** là con trỏ trỏ tới mảng nơi dữ liệu được lưu trữ.
- **size:** Kích thước từng phần tử mảng.
- **n:** số phần tử được đọc.
- **stream:** con trỏ file gắn với file đã được mở để đọc.
- Hàm **fread()** trả về số phần tử thực tế được đọc vào.

fwrite()

- Cú pháp hàm fwrite() là

```
size_t fwrite(const void *ptr, size_t  
size, size_t n, FILE *stream);
```

- **Ptr:** là con trỏ trỏ tới mảng nơi chứa dữ liệu để ghi ra file
- **n:** số phần tử để ghi ra.
- **stream:** con trỏ file gắn với file đã được mở để ghi ra.
- Hàm **fwrite()** trả về số phần tử thực tế được ghi ra.



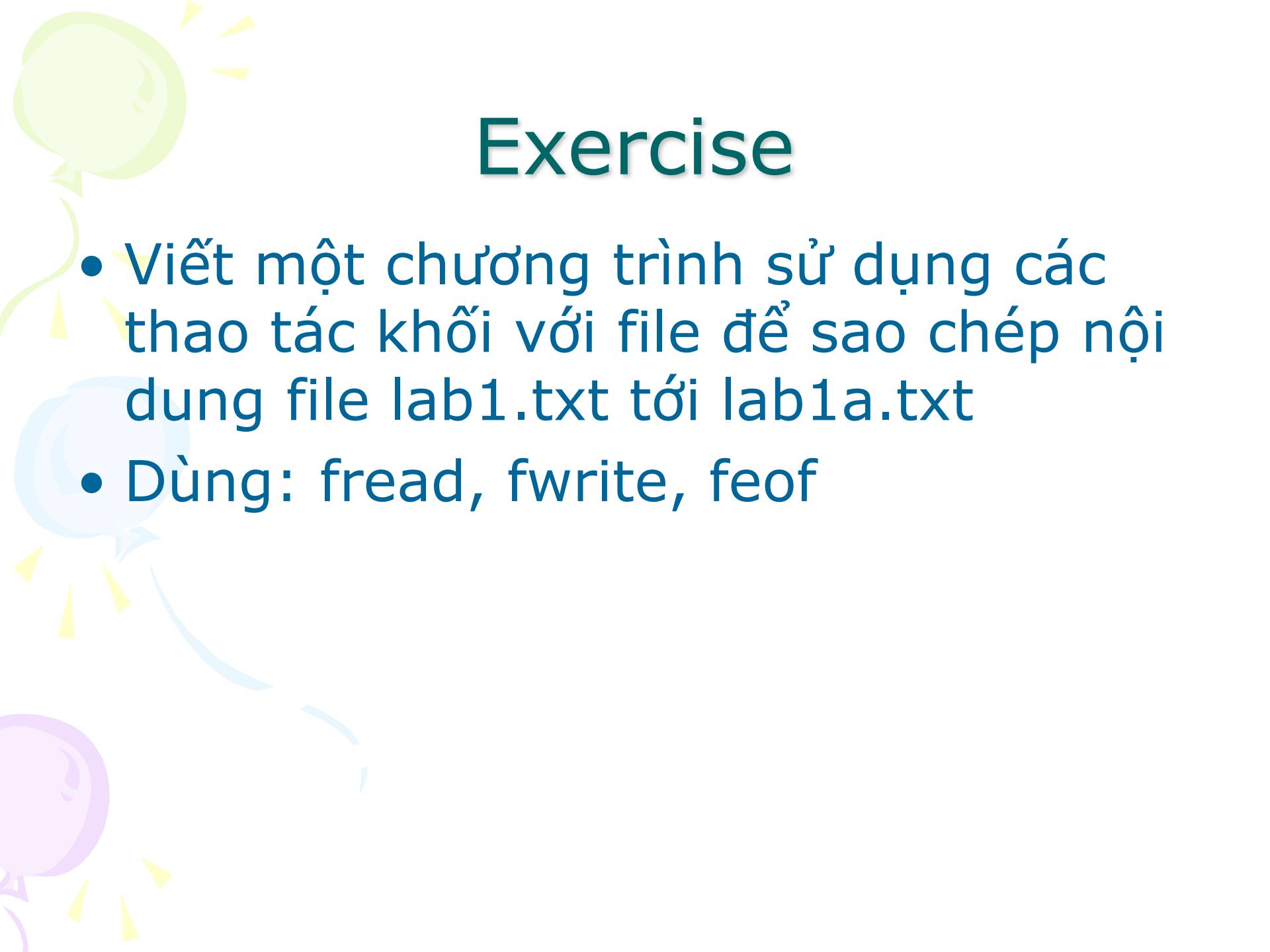
hàm feof

- `int feof(FILE *stream);`
- return 0 nếu chưa tới vị trí kết thúc file; ngược lại, nó trả về số nguyên khác 0.

Examples

- Read 80 bytes from a file.

```
enum {MAX_LEN = 80};  
int num;  
FILE *fptr2;  
char filename2[] = "haiku.txt";  
char buff[MAX_LEN + 1];  
if ((fptr2 = fopen(filename2, "r")) == NULL) {  
    printf("Cannot open %s.\n", filename2);  
    reval = FAIL; exit(1);  
}  
.  
.  
.  
num = fread(buff, sizeof(char), MAX_LEN, fin);  
buff[num * sizeof(char)] = '\0';  
printf("%s", buff);
```



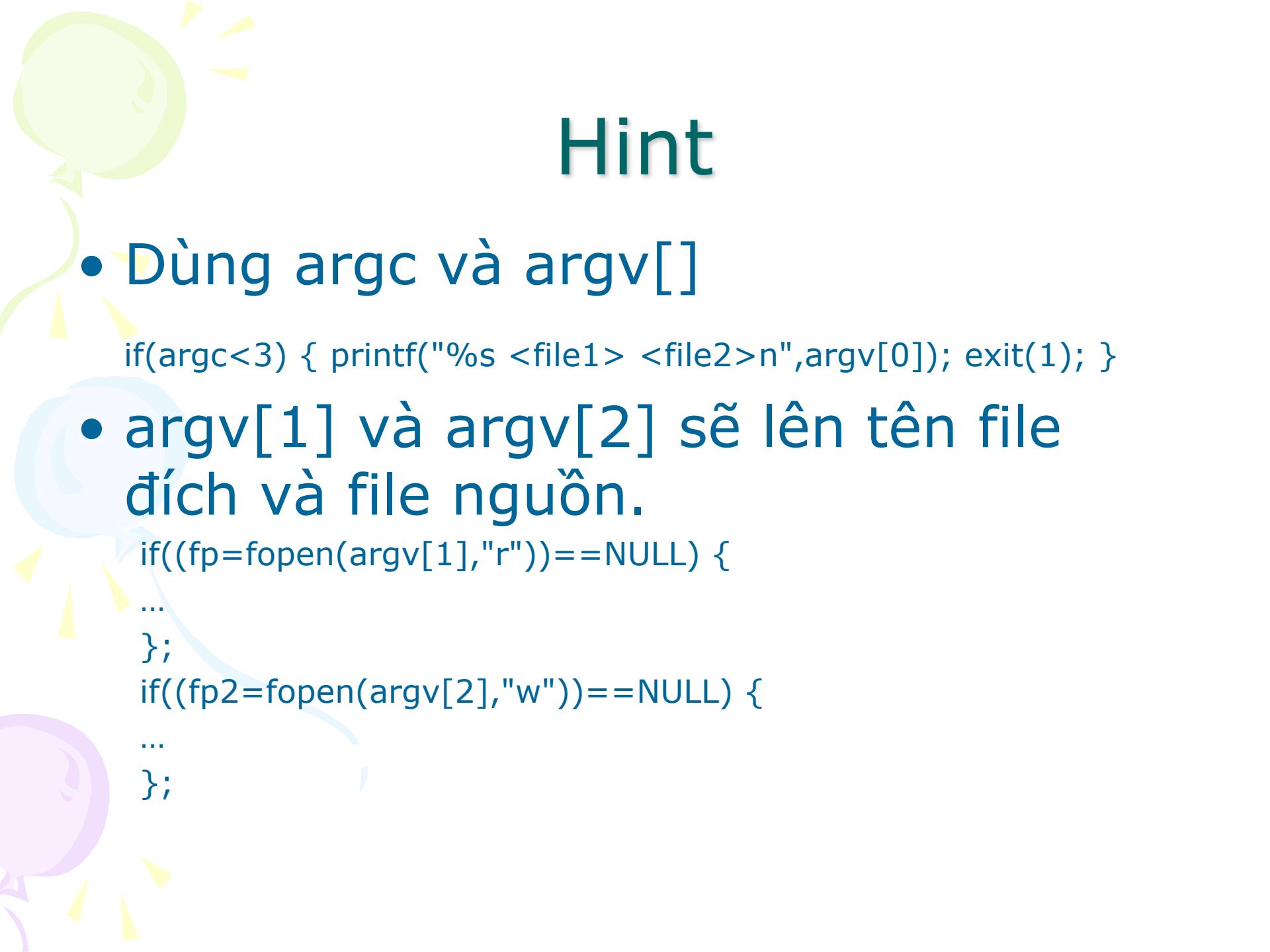
Exercise

- Viết một chương trình sử dụng các thao tác khõi với file để sao chép nội dung file lab1.txt tới lab1a.txt
- Dùng: fread, fwrite, feof



Exercise

- A) Cải thiện chương trình trong bài tập trước để nó nhận vào 2 tên file từ tham số dòng lệnh.
- Ví dụ: nếu chương trình của bạn đặt tên là "filecpy". Có thể dùng cú pháp sau (trên Linux):
 - ./filecpy haiku.txt haiku2.txt
- B. Viết chương trình có cùng tính năng với lệnh cat trong Linux
 - ./cat1 haiku.txt



Hint

- Dùng argc và argv[]

```
if(argc<3) { printf("%s <file1> <file2>\n",argv[0]); exit(1); }
```

- argv[1] và argv[2] sẽ lên tên file đích và file nguồn.

```
if((fp=fopen(argv[1],"r"))==NULL) {  
...  
};  
if((fp2=fopen(argv[2],"w"))==NULL) {  
...  
};
```

Exercice: Input và Output của cấu trúc

- Giả sử ta tạo ra danh bạ điện thoại.
- Định nghĩa một kiểu dữ liệu có thể lưu giữ "name," "telephone number," "e-mail address," và tạo ra mảng các cấu trúc có thể lưu giữ nhiều nhất 100 dữ liệu.
- Nhập 10 dữ liệu vào mảng này.
- Viết một chương trình để ghi nội dung mảng đã nhập liệu vào file, sử dụng fwrite() và dùng hàm fread () để đọc dữ liệu từ mảng.

File truy cập ngẫu nhiên

- Hai hàm: fseek() và ftell()
- fseek(): hàm truyền con trỏ trong file tới vị trí bạn muốn truy cập trong file.
- Cú pháp

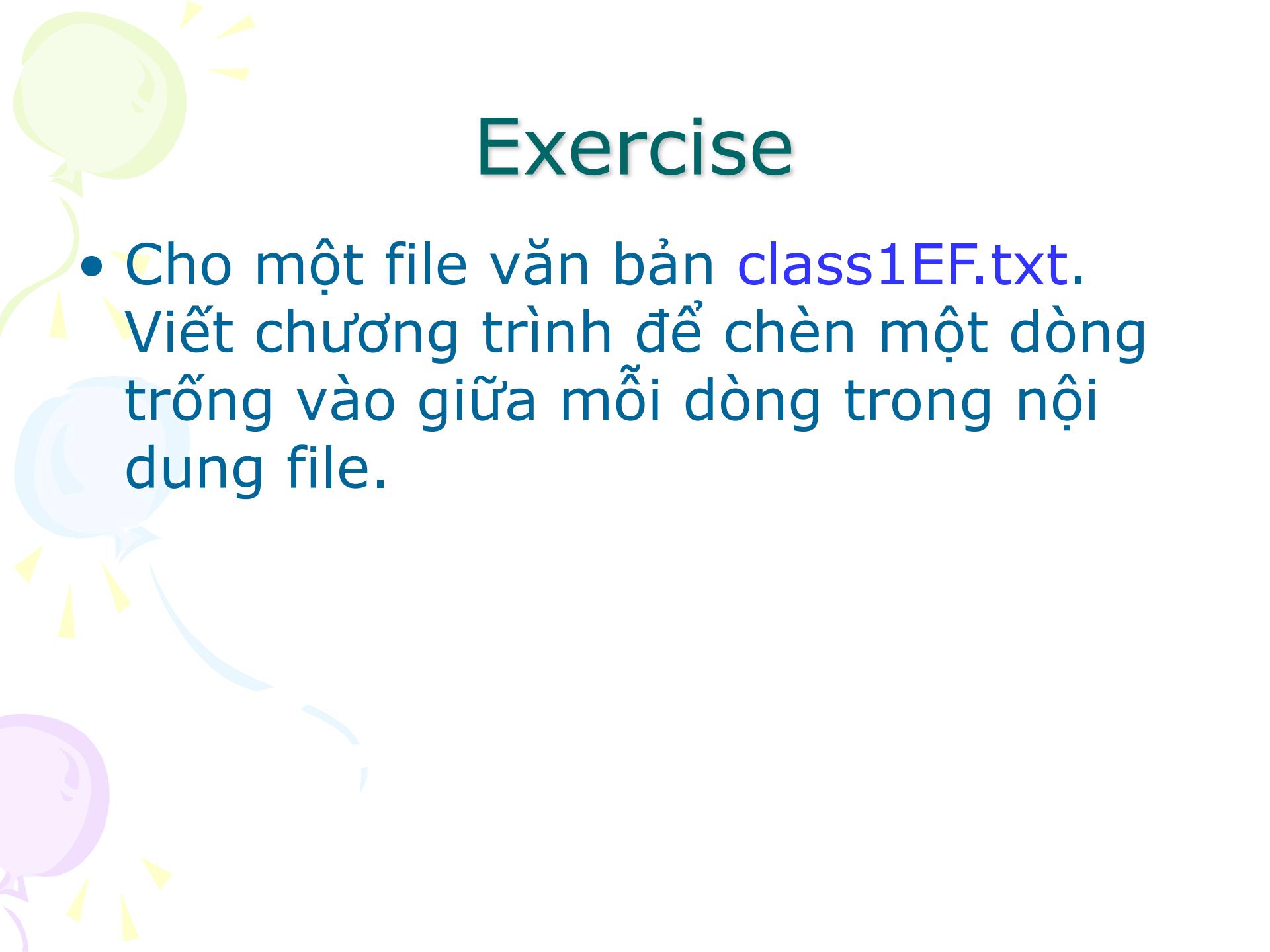
```
fseek(FILE *stream, long offset, int whence);
```
- Stream là con trỏ file gắn với một file đã mở
- Offset chỉ ra số bytes từ một vị trí cụ thể
- Whence: SEEK_SET, SEEK_CUR, và SEEK_END
 - SEEK_SET: từ đầu file
 - SEEK_CUR: từ vị trí hiện tại
 - SEEK_END: từ cuối file

File truy cập ngẫu nhiên (tiếp.)

- **ftell:** nhận giá trị con trỏ vị trí làm việc hiện tại trong file
- Cú pháp:
`long ftell(FILE *stream);`
- **rewind():** reset the file position indicator and put it at the beginning of a file
- Cú pháp:
- `void rewind(FILE *stream);`

Dynamic memory allocation

- Viết 1 chương trình nhập vào một lượng dữ liệu danh bạ từ tệp tin (ví dụ, “dữ liệu thứ 3 tới dữ liệu thứ 6” hoặc “dữ liệu thứ 2 tới dữ liệu thứ 3”), sửa một số dữ liệu và lưu lại vào file.
- Nhưng, bạn phải phân phối lượng bộ nhớ cần thiết tối thiểu để lưu trữ dữ liệu bằng hàm malloc (kích thước cần thiết cho “dữ liệu thứ 3 tới dữ liệu thứ 6” là **4**, còn cho “dữ liệu thứ 1 tới dữ liệu thứ 2” là **2**).



Exercise

- Cho một file văn bản class1EF.txt.
Viết chương trình để chèn một dòng
trống vào giữa mỗi dòng trong nội
dung file.

Input Output với khuôn dạng (format)

- 2 hàm fprintf và fscanf
- Tương tự printf và scanf.

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

```
fprintf(fp, "%d %s", 5, "bear");  
fscanf(fp, "%d %s", &n, s);
```

Homework

- Viết chương trình để nhập vào vài số từ luồng vào tiêu chuẩn và xuất ra file “out.txt” theo thứ tự ngược lại. Thêm vào đó, xuất tổng các số vào cuối file out.txt.
- Khuôn dạng dữ liệu nhập vào từ luồng vào tiêu chuẩn là như sau: số đầu tiên là số lượng dữ liệu, số thứ 2 trở đi là dữ liệu để tính toán. Ví dụ khi nhập vào:
4 12 -45 56 3
- “4” là số con số theo sau nó, và phần còn lại “12 -45 56 3” sẽ là các số được xuất ra file “out.txt”. Dữ liệu xuất ra “out.txt” là:
3 56 -45 12 26
- Con số cuối cùng “26” là tổng của 4 số trước đó.
- Ví số lượng con số nhập vào thay đổi theo mỗi lần chạy chương trình, bạn sẽ phải phân phôi bộ nhớ động cho số lượng dữ liệu: sử dụng hàm malloc().