

Kỹ Thuật Lập Trình

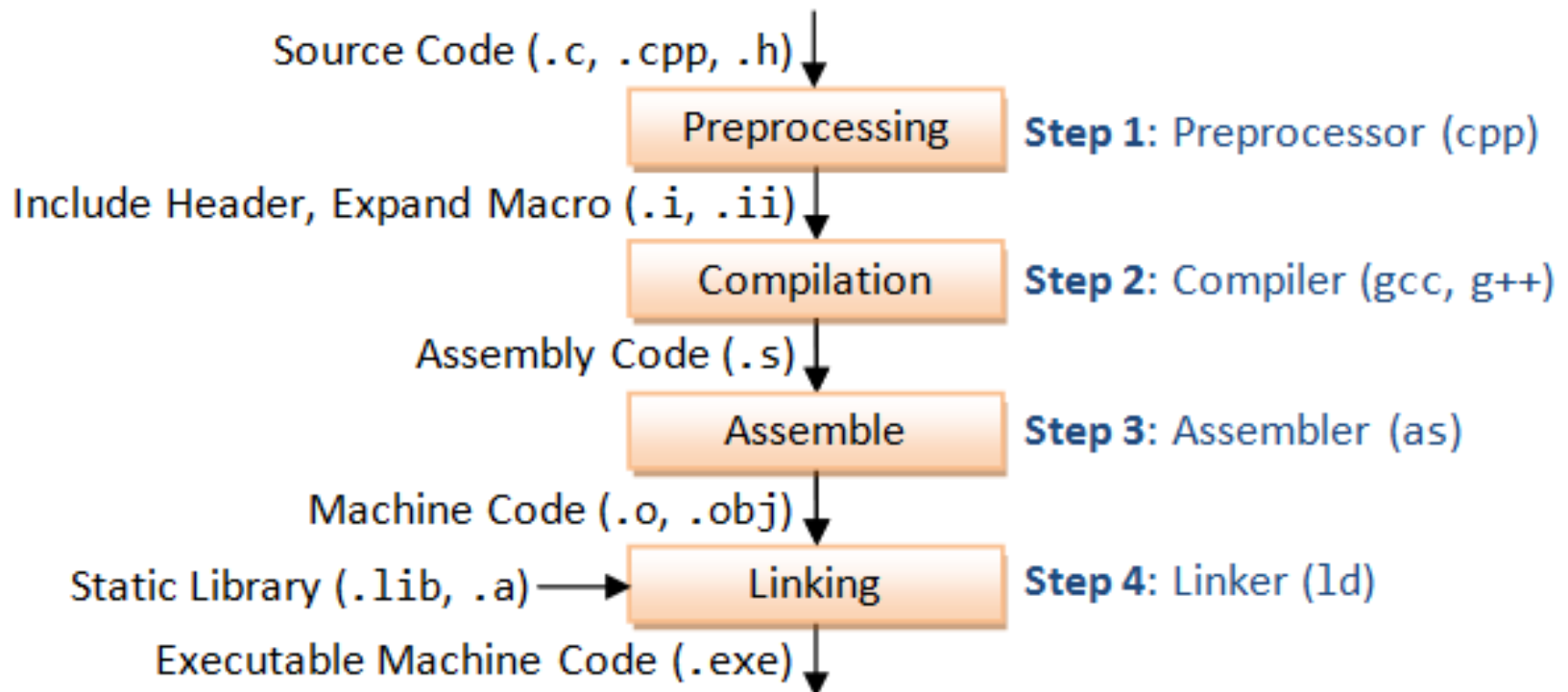
(Ngôn Ngữ Lập Trình C)

Bài giảng số 2

Các khái niệm cơ bản trong C

Nguyễn Trí Cường

Quy trình biên dịch (compiler)




Biến Trong C


```
#include <stdio.h>
int main(void)
{
```

```
int firstOperand;
int secondOperand;
int thirdOperand;
```

```
firstOperand = 1;
secondOperand = 2;
thirdOperand = firstOperand + secondOperand;
printf("%d", thirdOperand);
```

```
return 0;
}
```

 *Thứ bạn cần trong mọi chương trình*

 *Khai báo biến
(xác định ô nhớ cho các biến)*

 *Các chỉ dẫn cho CPU* (các câu lệnh)

Vài Chú Ý

- Mỗi biến trong C được dành riêng một vùng nhớ.
- Tên biến cần được chọn phù hợp nhằm giúp người đọc chương trình hiểu được chức năng của biến trong chương trình.
- Khi tất cả các biến đã có ô nhớ cụ thể, chương trình bắt đầu được thực hiện.
- Tại một thời điểm chỉ có một lệnh được thực hiện, trình tự thực hiện các lệnh phụ thuộc vào vị trí của lệnh trong chương trình.
- Các biến cần được *khởi tạo* trước khi dùng.

Tên Biến

- Trong C, *tên* (từ định danh, identifier) cần tuân theo các nguyên tắc sau
 - Dùng kí tự, số và dấu gạch dưới (_)
 - Không bắt đầu với một số
 - Không trùng với *từ dành riêng* (reserved words, từ khóa)
 - Có phân biệt giữa chữ hoa và chữ thường
 - Có thể có độ dài bất kỳ
- *Việc lựa chọn tên rất quan trọng trong việc đọc hiểu chương trình*
 - Thông thường danh từ hoặc chuỗi danh từ mô tả nội dung biến được chọn cho tên biến

Ví Dụ Tên Biến

Đúng	Không đúng	Đúng nhưng...
<code>rectangleWidth</code>	<code>10TimesLength</code>	<code>a1</code>
<code>rectangle_Width</code>	<code>My Variable</code>	<code>1</code>
<code>rectangle_width</code>	<code>int</code>	<code>0</code>

Khai Báo Biến

- **int months;**

- Các biến kiểu nguyên đại diện cho các số nguyên

- 1, 17, -32, 0

Không phải 1.5, 2.0, 'A'

- **double pi;**

- Các biến dấu phẩy động đại diện cho các số thực

- 3.14, -27.5, 6.02e23, 5.0

Không phải 3

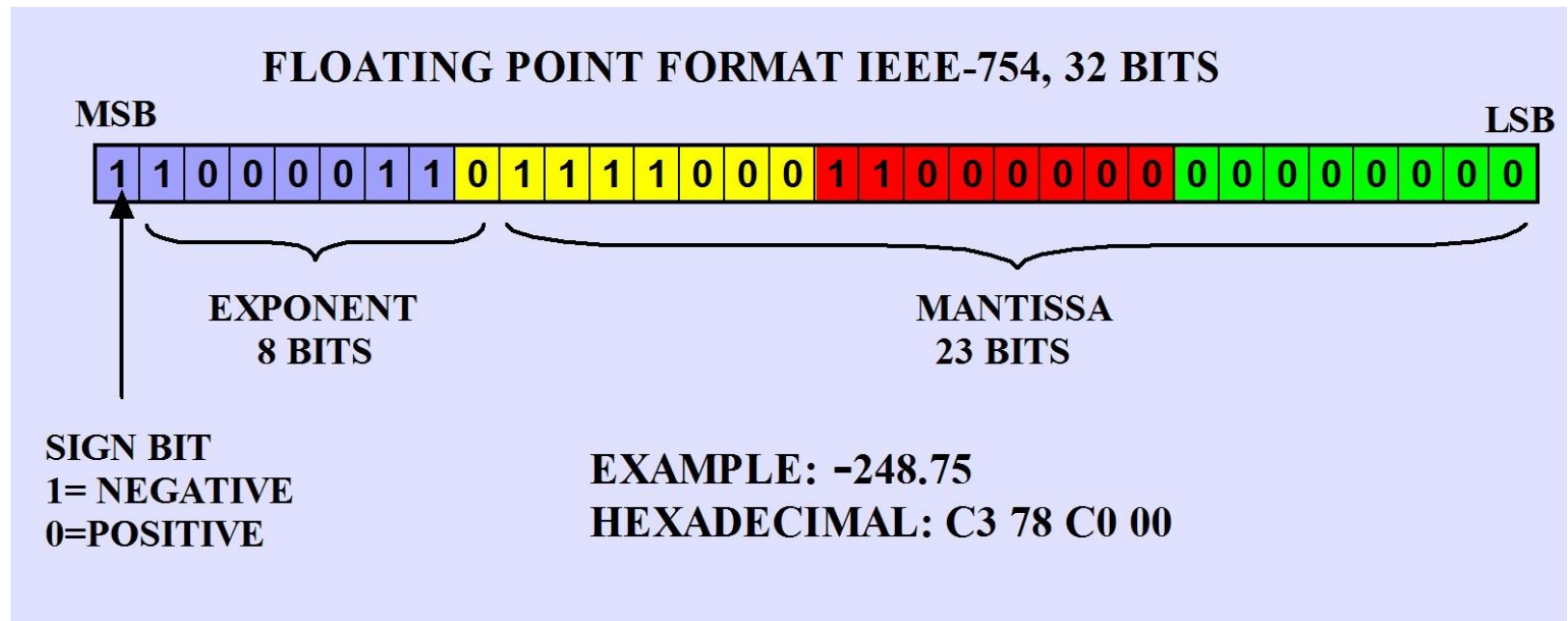
- **char first_initial, marital_status;**

- Các biến kiểu ký tự đại diện cho các ký tự của bàn phím

- 'a', 'b', 'M', '0', '9', '#', ' '

Không phải "Bill"

Ví dụ về kiểu float trong C



BT: Tìm hiểu quy đổi Floating point từ Nhị phân sang Thập phân và ngược lại theo tiêu chuẩn IEEE-754 + viết chương trình

Ví Dụ $F \rightarrow C$ (chương trình 1)

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    double fahrenheit, celsius;
```

```
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
```

```
    return(0) ;
```

```
}
```

Ví Dụ F→C (chương trình 2)

```
#include <stdio.h>

int main(void)
{
    double fahrenheit, celsius;

    printf("Enter a Fahrenheit temperature: ");
    scanf("%lf", &fahrenheit);

    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;

    printf("That equals %f degrees Celsius.",
        celsius);

    return(0);
}
```

Ví Dụ $F \rightarrow C$ (chương trình 3)

```
#include <stdio.h>

int main(void)
{
    double fahrenheit, celsius;

    printf("Enter a Fahrenheit temperature: ");
    scanf("%lf", &fahrenheit);

    celsius = fahrenheit - 32.0;
    celsius = celsius * 5.0 / 9.0;

    printf("That equals %f degrees Celsius.",
        celsius);

    return(0);
}
```

Cần Quan Tâm Đến Biểu Thức?

- Chúng ta cần các quy tắc chính xác giúp cho việc hiểu ý nghĩa của các biểu thức

$4 - 4 * 4 + 4$ có giá trị bằng bao nhiêu?

- Phép tính số học trong máy tính không phải lúc nào cũng chính xác

$(1.0 / 9.0) * 9.0$ có thể có kết quả
 0.99999998213

- Phép chia hai số có kiểu `int` có thể cho kết quả hoàn toàn khác với những gì ta mong đợi

$2 / 3$ cho kết quả bằng `0` trong C

Tại Sao Cần Dùng **int**

- Đôi khi chỉ số có kiểu **int** là phù hợp
 - “ô số 15” thay vì “ô số 14.9999999”
- Số có kiểu **double** đôi khi không chính xác trong việc biểu diễn số nguyên
 - Trong toán học: $3 * 15 * (1/3) = 15$
 - Nhưng trong máy tính: $3.0 * 15.0 * (1.0/3.0)$ có thể bằng **14.9999999997**
- Và...
 - Các phép tính với số **double** thường chậm hơn với số **int**
 - Số có kiểu **double** cần nhiều bộ nhớ hơn số có kiểu **int**

Ví Dụ Cụ Thể

- Công thức toán học

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

- Biểu thức trong C

`(- b + sqrt (b*b - 4.0*a*c)) / (2.0*a)`

Biểu Thức Chứa Nhiều Kiểu

- Giá trị của **2 * 3.14** là bao nhiêu?
- Trình biên dịch sẽ *tự động* (implicitly) chuyển kiểu **int** sang kiểu **double** mỗi khi gặp biểu thức dạng này

int + double

double + double

2 * 3 * 3.14

(2*3) * 3.14

6 * 3.14

6.0 * 3.14

18.84

- Bạn nên đặc biệt tránh sử dụng các biểu thức chứa nhiều kiểu dữ liệu
 - Nên dùng **2.0 / 3.0 * 3.14**

Chuyển Kiểu Trong Phép Gán

```
int total, count;
```

```
double avg;
```

```
total = 97;
```

```
count = 10;
```

```
avg = total / count;    /*avg is 9.0*/
```

```
total = avg;            /*BAD*/
```

chuyển kiểu tự động
trong phép gán



Phép Toán Chuyển Kiểu

- Bạn có thể sử dụng *phép toán chuyển kiểu* (casting) trong chương trình để chuyển kiểu cho một giá trị
 - Chuyển giá trị của một biểu thức sang một kiểu khác
- Dạng lệnh: **(type) expression**
- Ví dụ
(double) myage
(int) (balance + deposit)
- Chú ý: Phép toán chuyển kiểu không thay đổi cách thức tính toán biểu thức.

Sử Dụng Phép Toán Chuyển Kiểu

```
int total, count ;
```

```
double avg;
```

```
total = 97 ;
```

```
count = 10 ;
```

```
/*avg is 9.0*/
```

```
avg = total / count;
```

```
/*avg is 9.7*/
```

```
avg = (double) total / (double) count;
```

```
/*avg is 9.0*/
```

```
avg = (double) (total / count);
```

chuyển kiểu tự động
trong phép gán



phép toán
chuyển kiểu



Kiểu Dữ Liệu Trong C

- Tất cả các biến, giá trị, biểu thức đều có kiểu dữ liệu.
- C quan tâm đến kiểu dữ liệu của từng đại lượng.
- Từ bây giờ: *luôn nhớ kiểu dữ liệu của tất cả các đại lượng trong chương trình của bạn.*

Bài Học Cơ Bản

- Chương trình cần được viết sáng của nhất có thể.
- Chương trình cần đơn giản, những biểu thức phức tạp cần được cài đặt bằng những câu lệnh đơn giản.
- Sử dụng dấu ngoặc đơn để chỉ rõ thứ tự ưu tiên của các toán tử trong những trường hợp có thể gây hiểu nhầm.
- Quá trình chuyển đổi kiểu cần được thể hiện rõ ràng bởi người lập trình nhằm tránh việc tự động chuyển kiểu trong các biểu thức hay các lệnh gán.
- Cần chú ý đến kiểu dữ liệu trong việc viết các biểu thức.

Hiển Thị Dữ Liệu Xuất/Nhập

- Các hàm **printf** và **scanf** là các hàm cung cấp các chức năng xuất, nhập cơ bản

printf("control string", list of expressions);

scanf("control string", list of &variables);

- *Control string*: chuỗi xác định *định dạng* (format) của chuỗi xuất/nhập.
- *Expressions*: chứa dữ liệu cần đưa ra.
- *Variables*: các biến lưu trữ dữ liệu đầu vào.
- **&**: ký tự bắt buộc.

Ví Dụ Định Dạng Đầu Ra

%10.2f 1 2 3 . 5 5 **double**

%10.4f 1 2 3 . 5 5 0 0

%.2f 1 2 3 . 5 5

%10d 4 7 5 **int**

%-10d 4 7 5

%10c a **char**

scanf()

```
scanf("control string", &input list);
```

```
int numPushups;
```

```
printf("Hello. Do how many pushups? ");
```

```
scanf("%d", &numPushups);
```

```
printf("Do %d pushups.\n", numPushups);
```

output:

Hello. Do how many pushups? 5
Do 5 pushups.

- *Các biến nằm trong danh sách đầu vào bắt buộc phải có ký tự **&** ở phía trước.*

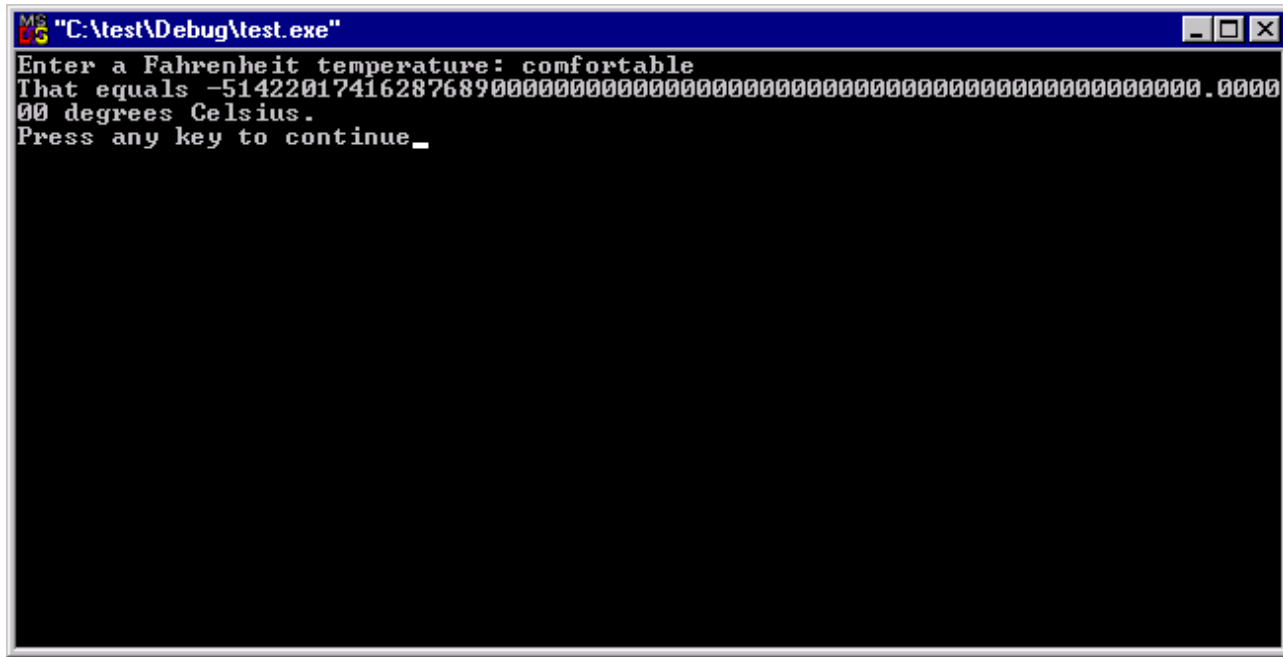
Nếu Bạn Quên &

- Chương trình của bạn vẫn có thể dịch thành công nhưng...



Lỗi Thường Gặp Với Xuất/Nhập

- Giả sử chương trình của bạn có câu lệnh sau
`scanf("%lf", &fahrenheit);`
- Nhưng tại con trỏ lệnh bạn gõ chuỗi “comfortable” như là đầu vào
 - Dữ liệu không được đọc và do vậy **fahrenheit** không được khởi tạo
 - Chương trình có thể gặp nhiều lỗi tiếp theo vì vấn đề này
- *Đây là lỗi của người lập trình, không phải của người dùng.*



Bạn Có Thể Làm Gì?

- Bản thân lệnh **scanf()** trả về số biến đầu vào đã được đọc thành công

```
int scanfCount;
```

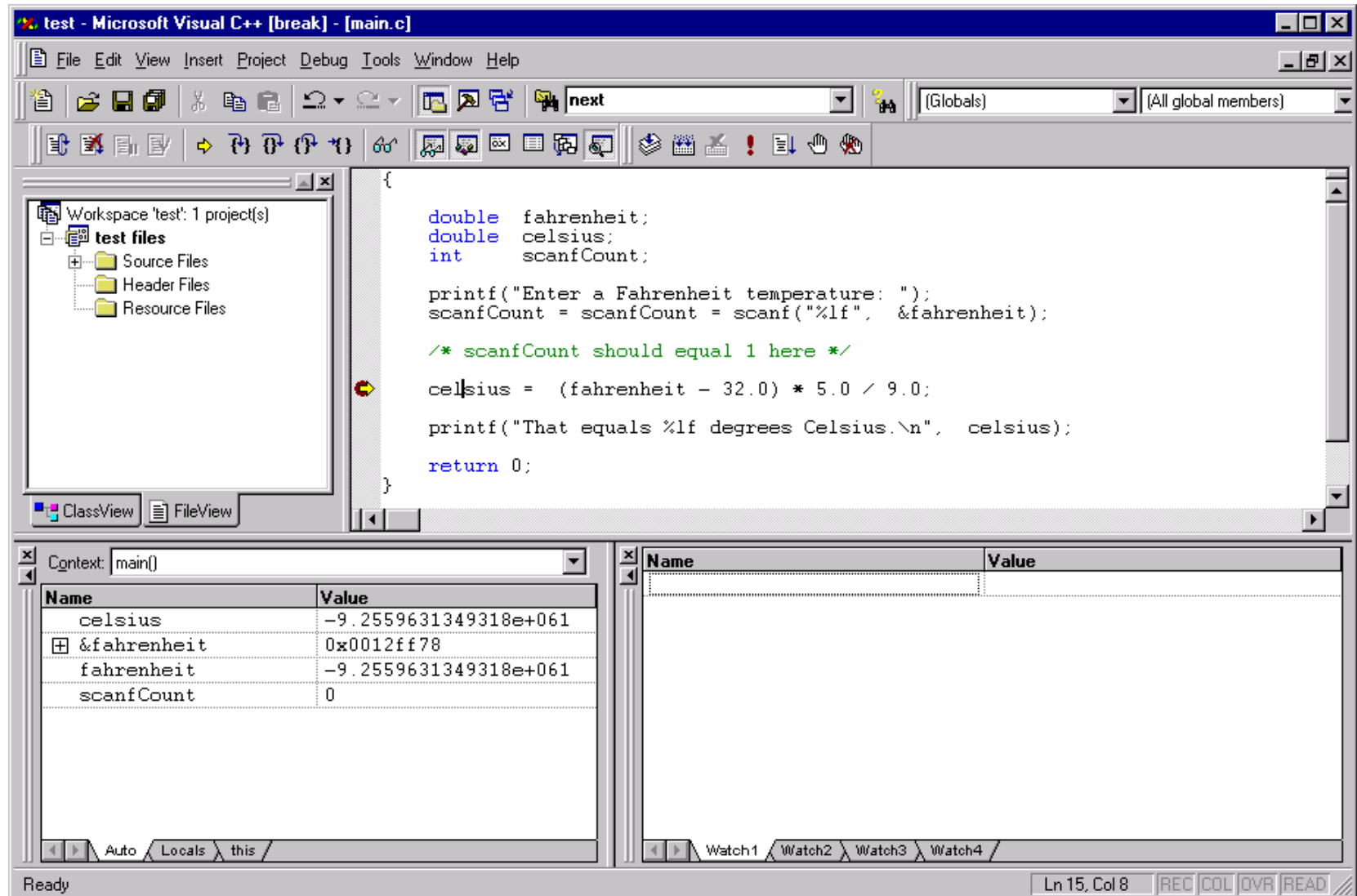
```
scanfCount = scanf("%d", &studentID) ;
```

```
/* if scanfCount is not equal to 1 at this point,
```

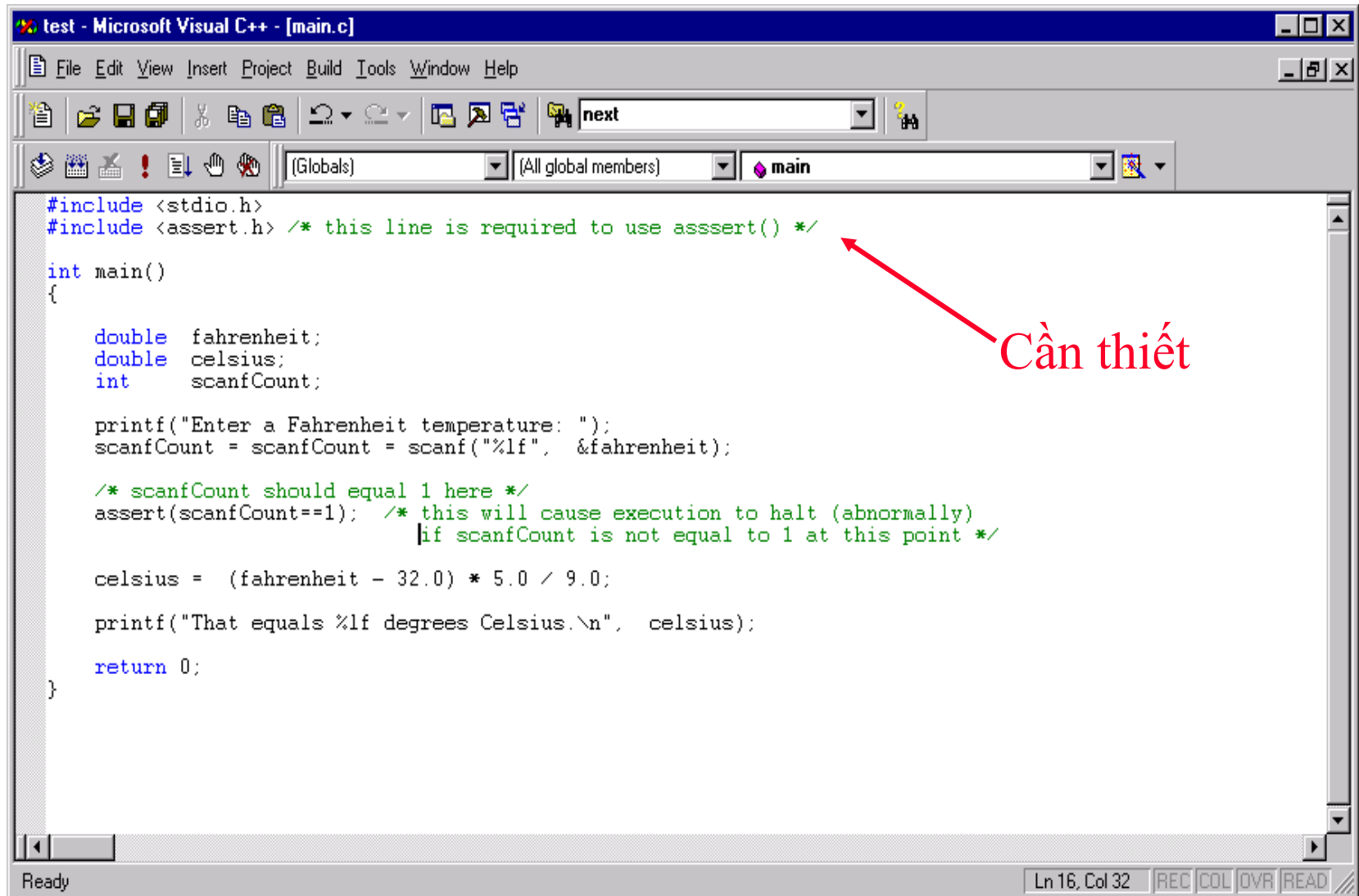
```
the user has made some kind of mistake.
```

```
Handle it. */
```

Chương Trình Dừng Tại Breakpoint



Dùng Câu Lệnh `assert()`



The screenshot shows the Microsoft Visual C++ IDE with a file named `main.c`. The code is as follows:

```
#include <stdio.h>
#include <assert.h> /* this line is required to use assert() */

int main()
{
    double fahrenheit;
    double celsius;
    int scanfCount;

    printf("Enter a Fahrenheit temperature: ");
    scanfCount = scanf("%lf", &fahrenheit);

    /* scanfCount should equal 1 here */
    assert(scanfCount==1); /* this will cause execution to halt (abnormally)
                           |if scanfCount is not equal to 1 at this point */

    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %lf degrees Celsius.\n", celsius);
    return 0;
}
```

A red arrow points from the text "Cần thiết" (Necessary) to the `#include <assert.h>` line, indicating that this line is required for the `assert()` function to work.

Tóm Lược Về Xuất/Nhập

printf(“control string”, output list);

- **control string** : kiểu dữ liệu và định dạng mong muốn
- **output list** : các biểu thức, giá trị cần xuất ra

scanf(“control string”, &input list);

- **control string**: các biến, giá trị cần đọc vào
 - **input list**: các kiểu dữ liệu và định dạng mong muốn
 - Có thể dùng để khởi tạo các biến
- Chú ý: không dùng **&** với **printf()**, dùng **&** với **scanf()**
 - Với cả hai câu lệnh trên: *điều khiển giữ chỗ trong chuỗi định dạng cần phù hợp với các biểu thức xuất/nhập về số lượng, thứ tự và kiểu dữ liệu.*

Ý Niệm Về Hàm

- Xác định một “*vấn đề nhỏ*” cần được giải quyết trong chương trình của bạn.
- Viết một đoạn mã giải quyết vấn đề.
- Đặt tên cho đoạn mã.
- Mỗi khi bạn cần giải quyết “*vấn đề nhỏ*” đó, sử dụng tên đã đặt cho đoạn mã để nói rằng “*chạy đến đoạn mã để giải quyết vấn đề này và không quay lại cho đến khi vấn đề được giải quyết*”.

Các Hàm Viết Sẵn

- Các hàm viết sẵn thông thường được đóng gói trong “*thư viện*”.
- Mọi trình biên dịch C chuẩn đều có một tập hợp các thư viện chuẩn.
- Nhớ lại **#include <stdio.h>**?
 - Thông báo cho trình biên dịch biết bạn dự định dùng các hàm trong “*thư viện xuất/nhập chuẩn*”
 - **printf()** và **scanf()** thuộc thư viện xuất/nhập chuẩn
 - Và còn rất nhiều hàm liên quan đến xuất/nhập khác nữa
- Có rất nhiều hàm hữu dụng khác trong nhiều thư viện khác nhau.

void

- Từ khóa void có hai chức năng khác nhau trong định nghĩa hàm này:

Cho biết hàm sẽ không
trả về giá trị

```
/* write separator line on output*/
```

```
void PrintBannerLines (void)
```

```
{
```

```
    printf("*****");
```

```
    printf("*****\n");
```

```
}
```

Cho biết hàm sẽ
không có tham số

Kiểu Và Giá Trị Hàm

- Một hàm có thể trả về một giá trị.
- Giống như tất cả các giá trị trong C, giá trị trả về của hàm có kiểu. Hàm được gọi là có kiểu của biến trả về.

```
/* return a "random" number */  
double GenRandom (void)  
{  
    double result;  
    result = ...  
    return result;  
}
```

Kiểu hàm (kiểu của giá trị trả về). Ta nói **GenRandom()** là hàm có kiểu **double** hoặc **GenRandom()** trả về một giá trị **double**

Biến cục bộ, tồn tại chỉ khi hàm đang được thực hiện

Câu lệnh trả về

Giá trị trả về

Các Biến Cục Bộ

- Một hàm có thể định nghĩa các *biến cục bộ* (local variable) dùng riêng.
- Các biến cục bộ này *chỉ* có nghĩa trong phạm vi hàm
 - Các biến cục bộ được cấp phát bộ nhớ khi hàm được gọi
 - Được giải phóng khỏi bộ nhớ khi thoát khỏi hàm
- *Các tham số cũng là các biến cục bộ.*

```
/* Yield area of circle with radius r */  
double circle_area (double r) {  
    double x, area1;  
    x = r * r ;  
    area1 = 3.14 * x ;  
    return( area1 );  
}
```

Tham số

Các biến cục bộ

Thứ Tự Các Hàm Trong .c File

- Tên hàm cũng phải tuân theo nguyên tắc: cần phải được khai báo *trước* khi dùng.

```
#include <stdio.h>

void    fun2 (void) { ... }

void    fun1 (void) { ...; fun2(); ... }

int     main (void) { ...; fun1(); ... return 0; }
```

- Hàm **fun1** gọi hàm **fun2** do vậy hàm **fun2** cần phải được khai báo trước hàm **fun1**, etc.

Ví dụ về lỗi biến cục bộ

- Hàm có thể trả về con trỏ.

```
int* func_returns_pointer(void) ;
```

- Tuy nhiên sẽ là lỗi nếu trả về con trỏ của biến cục bộ.

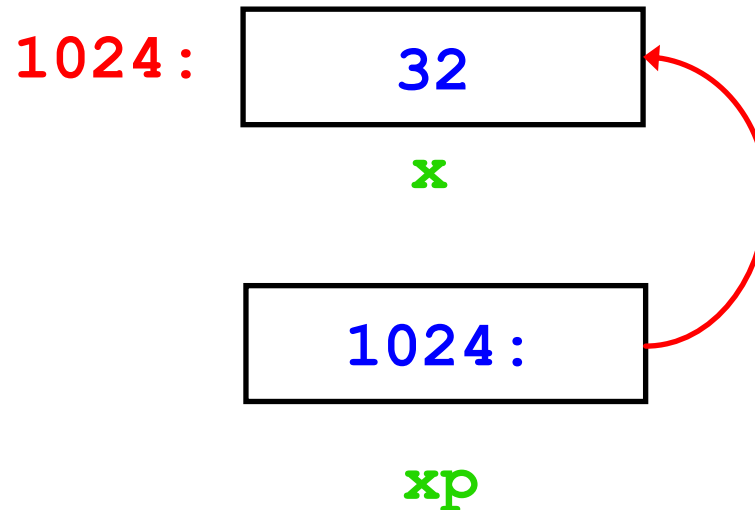
```
int* misguided(void)
{
    int array[10], i;
    for (i = 0; i < 10; ++i)
        array[i] = i;
    return array;
}
```

Cảnh Báo

- C cho phép bạn định nghĩa biến không nằm trong bất cứ hàm nào, còn gọi là *biến toàn cục*.
- Chú ý: ký hiệu **#define** không tạo ra biến.
- Biến toàn cục có thể được dùng thay cho việc truyền tham số cho hàm nhưng đây là một cách tồi, bạn nên có thói quen sử dụng biến cục bộ.

Kiểu Dữ Liệu Mới: Con Trỏ

- Con trỏ chứa tham chiếu đến một biến khác – có nghĩa là con trỏ có giá trị là địa chỉ của một biến khác.



- xp** là con trỏ tới một biến kiểu **int**.

Khai Báo Và Sử Dụng Con Trỏ

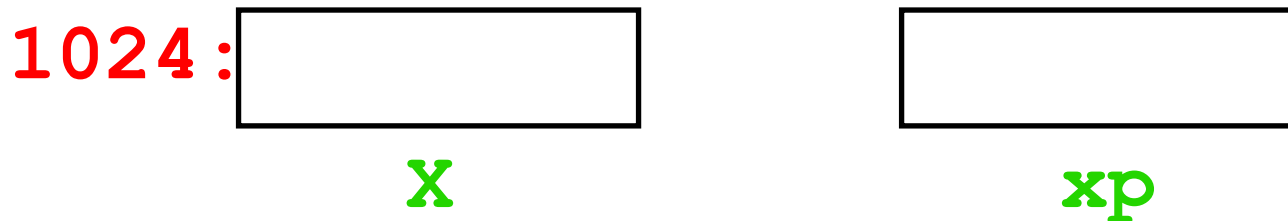
```
int x;           /* declares an int variable */
```

```
int *xp;         /* declares a pointer to int */
```

Giả sử **xp** lưu trữ địa chỉ của **x**, khi đó:

```
*xp = 0;         /* Assign integer 0 to x */
```

```
*xp = *xp + 1;   /* Add 1 to x */
```



Giải Pháp Dùng Con Trỏ Với `move_one`

```
void move_one (int *x_ptr, int *y_ptr) {  
    *x_ptr = *x_ptr - 1;  
    *y_ptr = *y_ptr + 1;  
}
```

```
int main (void) {  
    int a, b;  
    a = 4;  
    b = 7;  
    move_one (&a , &b) ;  
    printf ("%d %d", a, b) ;  
    return (0) ;  
}
```

Địa Chỉ Và Con Trỏ

- Ba kiểu mới dữ liệu mới
 - **int *** “con trỏ tới kiểu **int**”
 - **double *** “con trỏ tới kiểu **double**”
 - **char *** “con trỏ tới kiểu **char**”
- Hai toán tử (một ngôi) mới
 - **&**: toán tử lấy địa chỉ của biến
 - Có thể được dùng với mọi loại biến (hoặc tham biến)
 - *****: toán tử lấy giá trị lưu trong ô nhớ con trỏ trỏ tới
 - Chỉ được sử dụng bởi con trỏ

Quay Lại Hàm **scanf**

```
int x,y,z;
```

```
printf("%d %d %d", x, y, x+y);
```

```
scanf("%d %d %d", x, y, x+y);
```

NO!

```
scanf("%d %d", &x, &y);
```

YES! ?

Tại Sao Cần Dùng Con Trỏ?

- Các hàm cần thay đổi tham số thực trong lời gọi hàm
 - Ví dụ như hàm **move_one**
- Có thể trả về nhiều giá trị
 - Ví dụ như hàm **scanf**
- Trong lập trình cấp cao, con trỏ được dùng để tạo ra các cấu trúc dữ liệu động.

Sắp Xếp Hai Số Nguyên

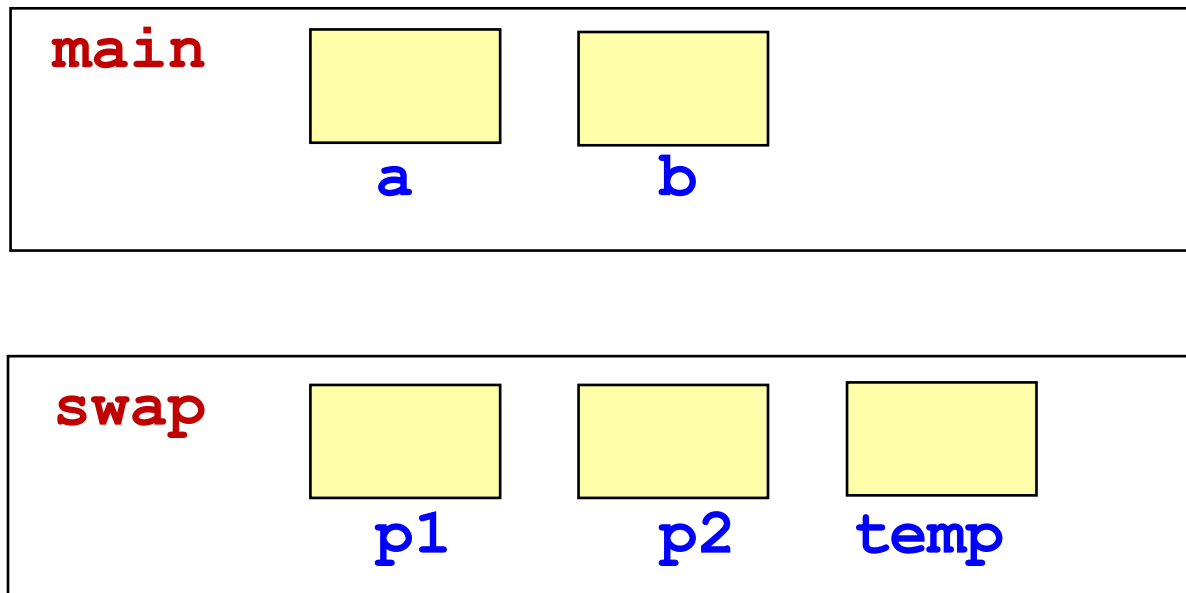
```
/* read in and sort 2 integers */
int c1, c2, temp;
printf("Enter 2 integers: ");
scanf("%d%d", &c1, &c2);
/* the 2 values may be in either order */
if (c2 < c1) {    /* swap if out of order */
    temp = c1;
    c1    = c2;
    c2    = temp;
}
/* at this point c1 <= c2 (guaranteed) */
```

Hàm **swap** Dùng Con Trỏ

```
void swap(int *p1, int *p2) {  
    int temp;  
    temp = *p1;  
    *p1  = *p2;  
    *p2  = temp;  
}
```

```
int a, b ;  
a = 4; b = 7;  
...  
swap(&a, &b) ;
```

Hàm **swap** Dùng Con Trỏ



Ngôn Ngữ C Định Kiểu Mạnh

```
int i;   int * ip;
double x; double * xp;
...
x = i;           /* no problem */
i = x;           /* not recommended */

ip = 30;          /* No way */
ip = i;           /* Nope */
ip = &i;          /* just fine */
ip = &x;          /* forget it! */
xp = ip;          /* bad */
&i = ip;         /* meaningless */
```


Mảng

- Định nghĩa: mảng là một tập hợp có thứ tự các giá trị có cùng kiểu dữ liệu được đặt tên.
- Ví dụ: điểm của 7 sinh viên
 - Đặt tên tập hợp: **grade**
 - Đánh số các phần tử: **0** đến **6**

double grade[7];	0	3.0
	1	3.8
		1.7
	.	2.0
	.	2.5
		2.1
	6	3.2

Biểu thức trong C:

grade[0] = 3.0;

grade[6] = 3.2;

2.0*grade[3] = 4.0;

...

Thuật Ngữ Mảng

```
type name[size];
```

Khai báo mảng

Kích thước mảng phải là một số nguyên

```
double grade[7];
```

- **grade** là *mảng số thực* với **7** thành phần (kích thước = 7)
- **grade[0]**, **grade[1]**, ... , **grade[6]** là các *thành phần* (element) của mảng **grade**. Từng thành phần đều có kiểu **double**.
- **0**, **1**, ... , **6** là các *chỉ số* của mảng
- Giá trị lớn nhất và nhỏ nhất của chỉ số là *ranh giới* (bound) của mảng (ở đây là 6 và 0)

Quy Tắc Với Chỉ Số Mảng

- Quy tắc: *Chỉ số mảng phải có giá trị là một số nguyên trong khoảng từ 0 đến $n-1$, trong đó n là số thành phần của mảng (không có ngoại lệ).*

- Ví dụ

grade[i+3+k] /* OK as long as $0 \leq i+3+k \leq 6$ */

- Chỉ số có thể đơn giản chỉ là một số nguyên

grade[0]

- Hoặc có thể là một biểu thức phức tạp

grade[(int) (3.1 * fabs(sin (2.0*PI*sqrt(29.067))))]

Kiểm Tra Ranh Giới Mảng

```
#define CLASS_SIZE 7
```

```
double grade[CLASS_SIZE];
```

```
int index;
```

```
index = 9;
```

```
...
```

```
grade[index] = 3.5; /* Out of range?? */
```

```
if (0<=index && index<CLASS_SIZE) {
```

```
    grade[index] = 3.5;
```

```
} else {
```

```
    printf("Index %d out of range.\n", index);
```

```
}
```

Ví Dụ Sử Dụng Thành Phần Mảng

```
double grade[7];   int i=3;   /*declarations*/

printf("Last two are %f, %f", grade[5],
grade[6]);

grade[5] = 0.0;

grade[i] = 2.0 * grade[i+1];

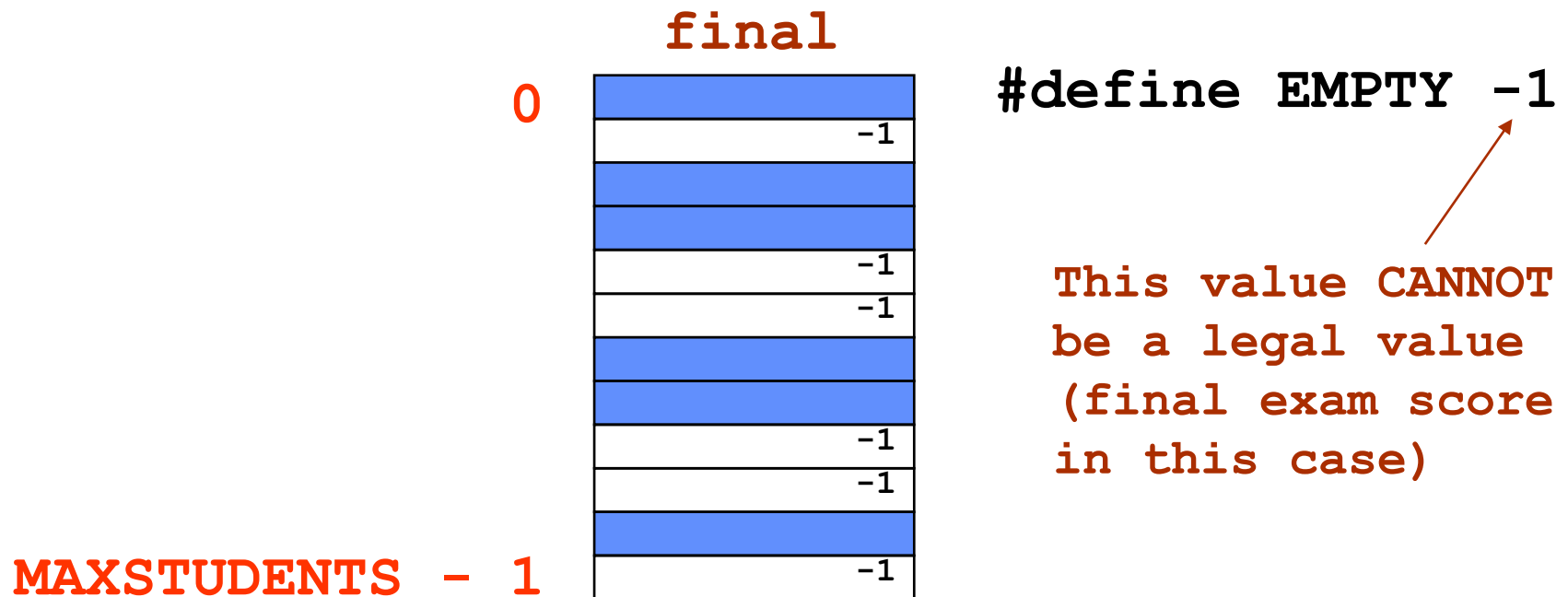
scanf("%lf", &grade[0]);

swap(&grade[i], &grade[i+1]);
```

Có Thể & Không Thể

- Bạn **không thể**
 - Dùng toán tử **=** để gán giá trị của một mảng này cho một mảng khác
 - Dùng toán tử **==** để so sánh trực tiếp hai mảng
 - Dùng câu lệnh **scanf** và **printf** trực tiếp với cả một mảng
- Nhưng bạn **có thể**
 - Thực hiện các thao tác trên với các thành phần mảng
 - Hoặc viết các hàm thực hiện các thao tác trên với mảng

Dùng Giá Trị Đặc Biệt



!!!!!!

```
for (student=0; student < MAXSTUDENTS; student++) {  
    if (final[student] != EMPTY) {  
        ...  
    }  
}
```

Dịch Chuyển Thành Phần Mảng

```
/* Shift x[0], x[1], ..., x[n-1] one  
   position upwards to make space for a new  
   element at x[0]. Insert the value new at  
   x[0]. Update the value of n. */
```

```
for (k = n; k >= 1; k = k - 1)
```

```
    x[k] = x[k-1];
```

```
x[0] = new;
```

```
n = n + 1;
```



Khởi Tạo Mảng

```
int w[4] = {1, 2, 30, -4};
```

```
/* w has size 4, all 4 are initialized */
```

```
char vowels[6] = {'a', 'e', 'i', 'o', 'u'};
```

```
/* vow has size 6, only 5 have initializers */
```

```
/* vowels[5] is uninitialized */
```

- Bạn không thể dùng cách trên trong câu lệnh gán

```
w = {1, 2, 30, -4}; /* SYNTAX ERROR */
```

Khi Kích Thước Mảng Không Xác Định

```
double x[] = {1.0, 3.0, -15.0, 7.0, 9.0};  
/* x has size 5, all 5 are initialized */
```

- Tuy nhiên

```
double x[];      /* ILLEGAL */
```

Cả Mảng Làm Tham Số

```
#define ARRAY_SIZE 200

double average (int a[ARRAY_SIZE]) {
    int i, total = 0;
    for (i = 0; i < ARRAY_SIZE; i = i + 1)
        total = total + a[i];
    return ((double) total / ARRAY_SIZE);
}

int x[ARRAY_SIZE];

...

x_avg = average(x);
```

Mảng Làm Tham Số Trả Về

```
/* Sets vsum to sum of vectors a and b. */  
void VectorSum(int a[3], int b[3], int vsum[3]) {  
    int i;  
    for (i = 0; i < 3; i = i + 1)  
        vsum[i] = a[i] + b[i];  
}
```

Không có toán
tử * và &

```
int main(void) {  
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];  
    VectorSum(x,y,z);  
    printf("%d %d %d", z[0], z[1], z[2]);  
    reutrn 0;  
}
```

Mảng 2 Chiều

- Một tập hợp có trật tự các giá có cùng kiểu.
 - Đặt tên tập hợp, đánh số các thành phần trong tập hợp
- Ví dụ: điểm số của 7 sinh viên trong 4 bài kiểm tra

score	hw	0	1	2	3
student	0	22	15	25	25
student	1	12	12	25	20
student	2	5	17	25	24
student	3	15	19	25	13
student	4	2	0	25	25
student	5	25	22	24	21
student	6	8	4	25	12

Biểu thức trong C:

`score[0][0] = 22;`

`score[6][3] = 12;`

`2*score[3][0] = 30;`

Khai Báo Mảng 2 Chiều

```
#define MAX_STUDENTS 80
```

```
#define MAX_HWS 6
```

```
...
```

```
int score[MAX_STUDENTS][MAX_HWS];
```

```
int nstudents, nhws, i, j;
```

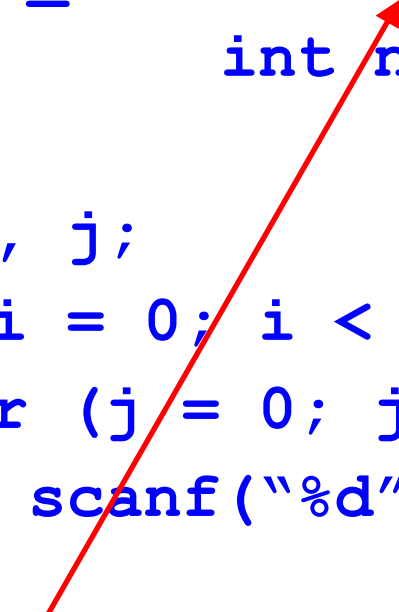
Nhập Mảng 2 Chiều

```
scanf("%d %d", &nstudents, &nhws);  
  
if (nstudents <= MAX_STUDENTS &&  
    nhws <= MAX_HWS) {  
    for (i = 0; i < nstudents; i = i + 1)  
        for (j = 0; j < nhws; j = j + 1)  
            scanf("%d", &score[i][j]);  
}
```

- Một phần mảng không được dùng đến.

Mảng 2 Chiều Làm Tham Số Hình Thức

```
void read_2D (int a[MAX_STUDENTS][MAX_HWS],  
              int nstudents, int nhws)  
{  
    int i, j;  
    for (i = 0; i < nstudents; i = i + 1)  
        for (j = 0; j < nhws; j = j + 1)  
            scanf("%d", &a[i][j]);  
}
```



`int a[][MAX_HWS]`

`int (*a)[MAX_HWS]` : a pointer to an array of 13 integers

`int *a[MAX_HWS]` : an array of 13 pointers to integers

Mảng 2 Chiều Làm Tham Số Thực

```
int main(void)
{
    int score[MAX_STUDENTS][MAX_HWS];
    int nstudents, nhws;

    scanf("%d %d", &nstudents, &nhws);
    if (nstudents <= MAX_STUDENTS &&
        nhws <= MAX_HWS)
        read_2D (score, nstudents, nhws);
    ...
}
```



no &

Tóm Tắt Về Mảng

- Thành phần mảng
 - Giống như một biến có cùng kiểu với mảng, có thể được dùng làm tham số xuất/nhập
- Toàn bộ mảng
 - Không được truyền tham số cho mảng bằng cách copy
 - Tham số hình thức: *không có toán tử **
type array_name [SIZE]
type array_name []
 - Tham số thực: **array_name** , *không có toán tử [] và &*

Ký Tự Và Chuỗi Ký Tự

- Hằng ký tự: sử dụng `' '`

`'a', 'A', '0', '\n', ' ', 'i', 'l', '\0'`

- Hằng chuỗi ký tự: sử dụng `" "`

`"Bill"`

`"Mary had a little %c%c%c%c. \n"`

the null character



- Biến ký tự

```
char va = 'l', vb = 'a', vc = 'm', vd = 'b';
```

```
printf("Mary had a little  
      %c%c%c%c.\n", va, vb, vc, vd);
```

Chuỗi Ký Tự

- Chuỗi ký tự: mảng các ký tự

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' };  
printf("Mary had a little %s.\n", pet);
```

- Định nghĩa chính xác hơn: mảng các ký tự với ký tự kết thúc là ký tự null

pet:

'l'	'a'	'm'	'b'	'\0'
-----	-----	-----	-----	------

pet[0] **pet[4]**

- Chuỗi ký tự không phải là một kiểu dữ liệu trong C.
- Lập trình viên cần phải chắc chắn ký tự kết thúc là ký tự null **'\0'**.

Khởi Tạo Chuỗi

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' };
```

```
char pet[5];
```

```
pet[0] = 'l'; pet[1] = 'a'; pet[2] = 'm';
```

```
pet[3] = 'b'; pet[4] = '\0';
```

```
char pet[5] = "lamb";
```

```
char pet[ ] = "lamb";
```

tương
đương

- Chú ý không được dùng

```
char pet[5];
```

```
pet = "lamb"; /* No array assignment in C */
```

- Không được dùng câu lệnh gán để khởi tạo mảng trong ngôn ngữ C.

Có Thể Và Không Thể

- Bạn không thể
 - Dùng toán tử **=** để gán giá trị của một chuỗi ký tự này cho một chuỗi ký tự khác (hãy dùng hàm **strcpy** trong thư viện)
 - Dùng toán tử **==** để so sánh trực tiếp hai chuỗi ký tự (hãy dùng hàm **strcmp** trong thư viện)
 - Định nghĩa một hàm có kiểu trả về là chuỗi ký tự
- Bạn có thể
 - Trực tiếp xuất/nhập chuỗi ký tự sử dụng hàm **printf** và **scanf** (sử dụng điều khiển giữ chỗ là **%s**)

Tự Gán Chuỗi Ký Tự

```
char str1[10], str2[ ] = "Saturday";
int i;
/* can't do: str1 = str2; */
/* can do: */
i = 0;
while (str2[i] != '\0') {
    str1[i] = str2[i];
    i = i + 1;
}
str1[i] = '\0';
```

Sử Dụng Hàm **strcpy**

```
/* strcpy is defined in string.h:
   copy source string into dest, stop with \0 */
void strcpy(char dest[ ], char source[ ])
{
    int i = 0;
    while (source[i] != '\0') {
        dest[i] = source[i];
        i = i + 1;
    }
    dest[i] = '\0' ;
}
```


Nguy Hiêm Tiềm Ân

```
#include <string.h>

...

char medium[ ] = "Four score and seven";
char big[1000];
char small[5];
strcpy(big, medium);
strcpy(big, "Bob");
strcpy(small, big);
strcpy(small, medium);
/* looks like trouble... */
```

Kết Quả Sử Dụng Hàm **strcpy**

medium: Four score and seven\0

big: Four score and seven\0?????...

big: Bob\0 score and seven\0?????...

small: Bob\0?

small: Four score and seven\0

Tính Độ Dài Chuỗi Ký Tự: **strlen**

```
/* return the length of string s, i.e.,  
   number of characters before terminating '\0',  
   or equivalently, index of first '\0'.  
*/  
int strlen(char s[ ])  
{  
    int n = 0;  
    while (s[n] != '\0')  
        n = n + 1 ;  
    return (n) ;  
}
```

Ví Dụ Về Độ Dài Chuỗi Ký Tự

```
#include <string.h> /* defn of strlen, strcpy */
...
char pet[ ] = "lamb";
int len1, len2, len3, len4, len5;

len1 = strlen(pet);
len2 = strlen("wolf");
len3 = strlen("");
len4 = strlen("Help\n");
strcpy(pet, "cat");
len5 = strlen(pet);
```

	0	1	2	3	4	5	6
lamb	l	a	m	b		\0	
wolf	w	o	l	f		\0	
						\0	
Help\n	H	e	l	p	\n		\0
cat	c	a	t		\0		\0

Nối Chuỗi Ký Tự

```
#include <string.h>
```

```
...
```

```
char str1[ ] = "lamb", str2[ ] = "chop";
```

```
char str3[11];
```

```
strcpy(str3, str1);
```

```
strcat(str3, str2);
```

```
/* strcat(s1, s2)
```

```
    make a copy of s2 at the end of s1. */
```

Kết Quả Sử Dụng Hàm **strcat**

str1

l a m b \0

str2

c h o p \0

str3

? ? ? ? ? ? ? ? ? ?

str3

l a m b \0 ? ? ? ? ? ?

str3

l a m b c h o p \0 ? ?

So Sánh Chuỗi Ký Tự

- Chuỗi **str_1** được coi là nhỏ hơn chuỗi **str_2** nếu
 - **j** là vị trí đầu tiên hai chuỗi khác nhau
 - Và **str_1[j] < str_2[j]**

“**lamb**” is less than “**wolf**” **j = 0, ‘l’ < ‘w’**

“**lamb**” is less than “**lamp**” **j = 3, ‘b’ < ‘p’**

“**lamb**” is less than “**lambch**” **j = 4, ‘\0’ < ‘c’**

Lỗi So Sánh Chuỗi Ký Tự

`str1 = str2;` **Syntax "error"**

`if (str1 == str2) ...` **No syntax error (but
almost surely a logic error)**

`if (str1 < str2) ...` **Likewise**

Hàm So Sánh Chuỗi Ký Tự

```
/* function strcmp in <string.h> */  
int strcmp(char str_1[ ], char str_2[ ]);
```

- Giá trị nguyên trả về có giá trị
 - Âm nếu **str_1** nhỏ hơn **str_2**
 - Bằng không nếu **str_1** bằng **str_2**
 - Dương nếu **str_1** lớn hơn **str_2**
- Lỗi thường gặp

```
if (!strcmp(str1, str2)) ...
```

means "if they ARE equal"

Xuất/Nhập Chuỗi Ký Tự

- **scanf** với điều khiển giữ chỗ “%s”
 - Bỏ qua ký tự trắng ở đầu
 - Chèn ký tự null ‘\0’ vào vị trí của ký tự trắng kế tiếp
 - Nguy hiểm tiềm ẩn: không kiểm tra độ dài chuỗi ký tự

char in_string[10];

scanStatus = scanf(“%s”, in_string);

- **printf** với điều khiển giữ chỗ “%s”



không sử dụng &

Tự Nhập Cả Dòng Ký Tự

```
char line[LENGTH + 1];
int i, scanStatus;

/* read input characters into line until end of
   input line reached or all available space in
   line used */
i = 0;
scanStatus = scanf("%c", &line[i]);
while (1 == scanStatus && i < LENGTH &&
       line[i-1] != '\n') {
    i++;
    scanStatus = scanf("%c", &line[i]);
}
line[i] = '\0'; /* is this a bug? */
```


Mảng Chuỗi Ký Tự

```
char month[12][10] = {  
    "January",  
    "February",  
    ...  
    "September", /* longest month: 9 letters */  
    ...  
    "December" };  
...  
printf("%s is hot\n", month[7]); /* August */
```

Ví Dụ Nhập/Xuất Chuỗi Ký Tự

```
char name[NUM_NAMES][MAX_NAME + 1];  
  
int age[NUM_NAMES], i;  
  
for (i = 0; i < NUM_NAMES; i = i + 1)  
{  
    scanf("%s %d", name[i], &age[i]);  
    printf("%s %d \n", name[i], age[i]);  
}
```

không sử dụng &



Rất Nhiều Hàm Trong **<string.h>**

strcat, strncat

nối

strcmp, strncmp

so sánh

strtod, strtol, strtoul

chuyển

- Các hàm hữu dụng liên quan trong **<ctype.h>**
 - Hoạt động với từng ký tự đơn lẻ
 - Chuyển dạng ký tự, kiểm tra loại ký tự...

Sử Dụng Thư Viện Các Hàm

- Bạn nên sử dụng các hàm xử lý chuỗi ký tự có sẵn trong thư viện **string.h**
- Sử dụng các hàm có sẵn trong thư viện là một đặc trưng của lập trình với ngôn ngữ C
 - Các thư viện chuẩn ANSI C như **stdio.h**, **string.h**, **ctype.h**
 - Còn rất nhiều thư viện mở khác
 - Bạn thậm chí có thể tự tạo thư viện cho riêng mình
- Bạn khó có thể trở thành một lập trình viên tài ba nếu không có khả năng học nhanh cách sử dụng các thư viện mới

Ôn Lại: Cấu Trúc Dữ Liệu

- Hàm được dùng để tổ chức chương trình.
- Các cấu trúc dữ liệu được sử dụng để tổ chức dữ liệu, đặc biệt khi
 - Lượng dữ liệu lớn
 - Lượng dữ liệu thay đổi
 - Các tập dữ liệu có liên hệ với nhau
- Mảng có thể được dùng với trường hợp thứ nhất và thứ hai, không thể dùng trong trường hợp thứ ba
 - Dữ liệu mô tả một chiếc xe tăng: kiểu, màu sắc, vị trí
 - Thông tin về một sinh viên: tên, điểm, mã số sinh viên,...

Định Nghĩa **struct**

```
#define MAX_NAME 40
typedef struct{
/* typedefs go at the top of the program */
    char    name [MAX_NAME + 1];
    int     id;
    int     hw, exams;
    double  grade;
} student_record;
```

- Định nghĩa một kiểu dữ liệu mới **student_record**, không phải là khai báo hay tạo ra một biến mới, không cấp phát bộ nhớ.

Thuật Ngữ

- Một cấu trúc (*struct*) đôi khi được gọi là một bản ghi (*record*).
- Các thành phần (*component*) của cấu trúc còn được gọi là các trường (*field*) hay các thành viên (*member*) của cấu trúc.
- Cấu trúc là cơ sở của lớp (*class*) trong C++ và Java.

Các Kiểu Dữ Liệu Tự Định Nghĩa

- Cung cấp một tập hợp có giới hạn các kiểu dữ liệu có sẵn: **int**, **char**, **double** (và các biến thể).
- Bạn có thể bổ xung thêm thông qua việc dùng con trỏ và mảng.
- Tuy nhiên các đối tượng trong thế giới thực và trong các chương trình máy tính thường phức tạp và không thể được biểu diễn bằng các kiểu dữ liệu trên.
- Với **struct**, bạn có thể tự định nghĩa thêm các kiểu dữ liệu khi cần.

Khai Báo Biến Kiểu **struct**

```
/* typedef students_record goes at top of
program */

...

int    i1;                /* int decls. */
int    count = 0;        /* nothing new */
char   c1[5];            /* array decls. */

student_record s1;
student_record harvey;

/* student_record is a type; s1 and harvey are
variables. */
```

Có Thể Và Không Thể

- Bạn có thể
 - Dùng toán tử **=** để gán giá trị của hai biến có cùng kiểu cấu trúc
 - Định nghĩa một hàm có kiểu trả về là một cấu trúc
- Bạn không thể
 - Dùng toán tử **==** để so sánh trực tiếp hai biến có cùng kiểu cấu trúc (tuy nhiên bạn có thể so sánh các trường)
 - Trực tiếp xuất/nhập chuỗi ký tự sử dụng hàm **printf** và **scanf** (tuy nhiên có thể xuất/nhập từng trường)

Khởi Tạo Biến Kiểu **struct**

```
/*typedef structs go at top*/
```

```
int  i1;                /* int decls. */
int  count = 0;         /* nothing new */
char c1[5];             /* array decls. */
char pet[5] = "lamb";   /* string initializer */

student_record harvey = {"Harvey S.",
                          9501234, 87, 74, 3.1};
```

So Sánh **struct**

```
if (pt1 == pt2) { ... } /* Doesn't work */
```

```
int points_equal(point pt1, point pt2) {  
    return (pt1.x == pt2.x && pt1.y == pt2.y);  
}
```

```
if (points_equal(pt1, pt2)) { ... } /* OK */
```

Xuất/Nhập struct

```
void print_point(point p) {  
    printf("(%f,%f)", p.x, p.y);  
}  
void scan_point(point *ptptr) {  
    point temp;  
    scanf("%lf %lf", &temp.x, &temp.y);  
    *ptptr = temp;  
}
```

```
point a;  
scan_point(&a);  
print_point(a);
```

ptptr:

temp:

x:

y:

a:

x:

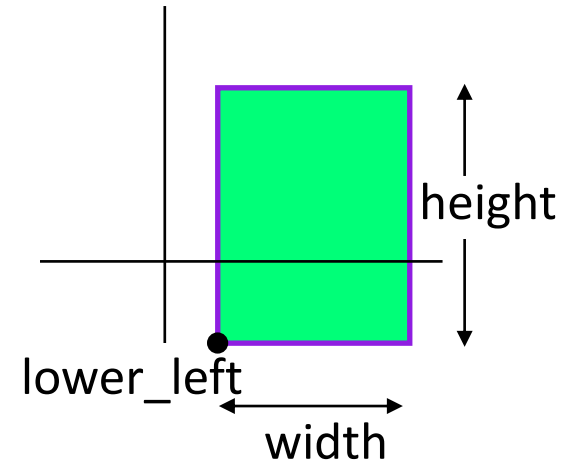
y:

struct Lồng Nhau

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

```
typedef struct {  
    double width, height;  
} dimension;
```

```
typedef struct {  
    dimension size;  
    point lower_left;  
    int line_color, fill_color;  
} rectangle;
```



Cấu Trúc Và Toán Học Con Trỏ

- Chương trình dịch sẽ tự động điền đầy lỗ trống, ví dụ cấu trúc sau 5-bytes, nhưng thực tế có thể là 6 or 8-bytes

```
struct Stype {  
    char c;  
    int i;  
};
```

- Toán tử **sizeof** luôn trả về kích thước đúng

Cấu Trúc Và Mảng

- Một cấu trúc thể hiện một bản ghi đơn nhất, các chương trình ứng dụng chạy trên máy tính làm việc với tập hợp các bản ghi.
- Ví dụ: bản ghi sinh viên, bản ghi nhân viên, bản ghi khách hàng...
- Trong mỗi trường hợp, sẽ có nhiều biến có cùng kiểu cấu trúc và do vậy sẽ rất tự nhiên nếu bạn dùng mảng cấu trúc để lưu giữ dữ liệu.

Mảng Cấu Trúc

- Mỗi khai báo phía dưới khai báo một mảng với các thành phần mảng có kiểu cấu trúc

```
point corner_points[10];
```

```
time meeting_times[MAX_MEETINGS];
```

```
student_record tdh_34[MAX_STUDENTS];
```

- Sử dụng mảng cấu trúc là một mở rộng tự nhiên các nguyên tắc đã được học.

Ôn Lại: Biến Kiểu Cấu Trúc Làm Đối Số

- Biến kiểu cấu trúc được truyền theo giá trị
 - Tất cả các thành phần của biến được sao chép từ đối số để khởi tạo tham số

```
point midpoint (point a; point b) {...}  
  
int main (void) {  
    point p1, p2, m; /* declare 3 points */  
    ...  
    m = midpoint(p1, p2);  
}
```

Truyền Mảng Cấu Trúc

- Một mảng cấu trúc trước hết là một mảng.
- Khi mảng được dùng làm đối số trong lời gọi hàm, nó được truyền theo tham chiếu
 - Tham số thực chất là một bí danh khác của đối số

```
int avg (student_rec class_db[MAX_N]) {...}

int main (void) {
    student_rec ktlt_k50[MAX_N];
    int average;
    ....
    average = avg(ktlt_k50); /*by reference*/
}
```

Sắp Xếp Mảng Cấu Trúc

David 920915 2.9	Kathryn 901028 4.0	Sarah 900317 3.9	Phil 920914 2.8	Casey 910607 3.6
------------------------	--------------------------	------------------------	-----------------------	------------------------

Phil 920914 2.8	David 920915 2.9	Casey 910607 3.6	Sarah 900317 3.9	Kathryn 901028 4.0
-----------------------	------------------------	------------------------	------------------------	--------------------------

```
typedef struct {  
    char    name[MAX_NAME + 1];  
    int     id;  
    double  score;  
} StudentRecord;
```

Ôn Lại: Sắp Xếp Lựa Chọn

```
int min_loc (int a[ ], int k, int n) {  
    int j, pos; pos = k;  
    for (j = k + 1; j < n; j = j + 1)  
        if (a[j] < a[pos])  
            pos = j;  
    return pos;  
}
```

```
void swap (int *x, int *y);
```

```
void sel_sort (int a[ ], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```


Sắp Xếp Mảng Cấu Trúc

- Trước tiên cần xác định trường cần sắp xếp
 - Có thể sử dụng điểm số
- Thay đổi kiểu mảng sang **StudentRecord**
- Viết lại đoạn mã so sánh trong hàm **min_loc**
- Viết hàm **swap** cho **StudentRecord**

Sắp Xếp Mảng Cấu Trúc

```
int min_loc (StudentRecord a[ ], int k, int n) {  
    int j, pos; pos = k;  
    for (j = k + 1; j < n; j = j + 1)  
        if (a[j].score < a[pos].score)  
            pos = j;  
    return pos;  
}
```

```
void swap (StudentRecord *x, StudentRecord *y);
```

```
void sel_sort (StudentRecord a[ ], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```

Sắp Xếp Theo Thứ Tự A-B-C

David 920915 2.9	Kathryn 901028 4.0	Sarah 900317 3.9	Phil 920914 2.8	Casey 910607 3.6
------------------------	--------------------------	------------------------	-----------------------	------------------------

Phil 920914 2.8	David 920915 2.9	Casey 910607 3.6	Sarah 900317 3.9	Kathryn 901028 4.0
-----------------------	------------------------	------------------------	------------------------	--------------------------

```
typedef struct {  
    char    name[MAX_NAME + 1];  
    int     id;  
    double  score;  
} StudentRecord;
```

- Cần viết một hàm để so sánh hai chuỗi ký tự.

Ôn Lại: So Sánh Chuỗi Ký Tự

- “Alice” nhỏ hơn “Bob”
- “Dave” nhỏ hơn “David”
- “Rob” nhỏ hơn “Robert”

```
#include <string.h>
```

```
int strcmp(char str1[ ], char str2[ ] );
```

- Giá trị trả về
 - Là số âm nếu **str1** nhỏ hơn **str2**
 - Bằng không nếu **str1** bằng **str2**
 - Là số dương nếu **str1** lớn hơn **str2**

Sắp Xếp Theo Thứ Tự A-B-C

```
int min_loc (StudentRecord a[ ], int k, int n) {  
    int j, pos; pos = k;  
    for (j = k + 1; j < n; j = j + 1)  
        if (0 > strcmp(a[j].name, a[pos].name))  
            pos = j;  
    return pos;  
}
```

```
void swap (StudentRecord *x, StudentRecord *y);
```

```
void sel_sort (StudentRecord a[ ], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```

Một Chút Về Cấu Trúc Dữ Liệu

- Nếu bạn muốn lưu trữ thông tin về một bài hát trong máy tính
 - Những thông tin gì cần được lưu trữ?
 - Chúng được tổ chức như thế nào?
 - Cách thực hiện trong C?
- Còn nếu
 - Bạn muốn thông tin của một đĩa CD
 - Hoặc thông tin của một tập hợp các đĩa CD

Sử Dụng Sắp Xếp Chèn

```
/* sort student records a[0..size-1] in */  
/* ascending order by score */  
void sort (student_record a[ ], int size)  
{  
    int j;  
    for (j = 1; j < size; j = j + 1)  
        insert(a, j);  
}
```

Sử Dụng Sắp Xếp Chèn

```
/* given that a[0..j-1] is sorted, move a[j]
to the correct location so that that a[0..j]
is sorted by score */
void insert (student_record a[ ], int j) {
    int i;
    student_record temp;
    temp = a[j];
    for (i = j; i > 0 &&
        a[i-1].score > temp.score; i = i-1) {
        a[i] = a[i-1];
    }
    a[i] = temp;
}
```


Sử Dụng Sắp Xếp Chèn

```
/* given that a[0..j-1] is sorted, move a[j] to  
the correct location so that that a[0..j] is  
sorted by score */
```

```
void insert (student_record a[ ], int j) {  
    int i;  
    student_record temp;  
    temp = a[j];  
    for (i = j; i > 0 &&  
        strcmp(a[i-1].name, temp.name) > 0;  
        i = i-1) {  
        a[i] = a[i-1];  
    }  
    a[i] = temp;  
}
```