

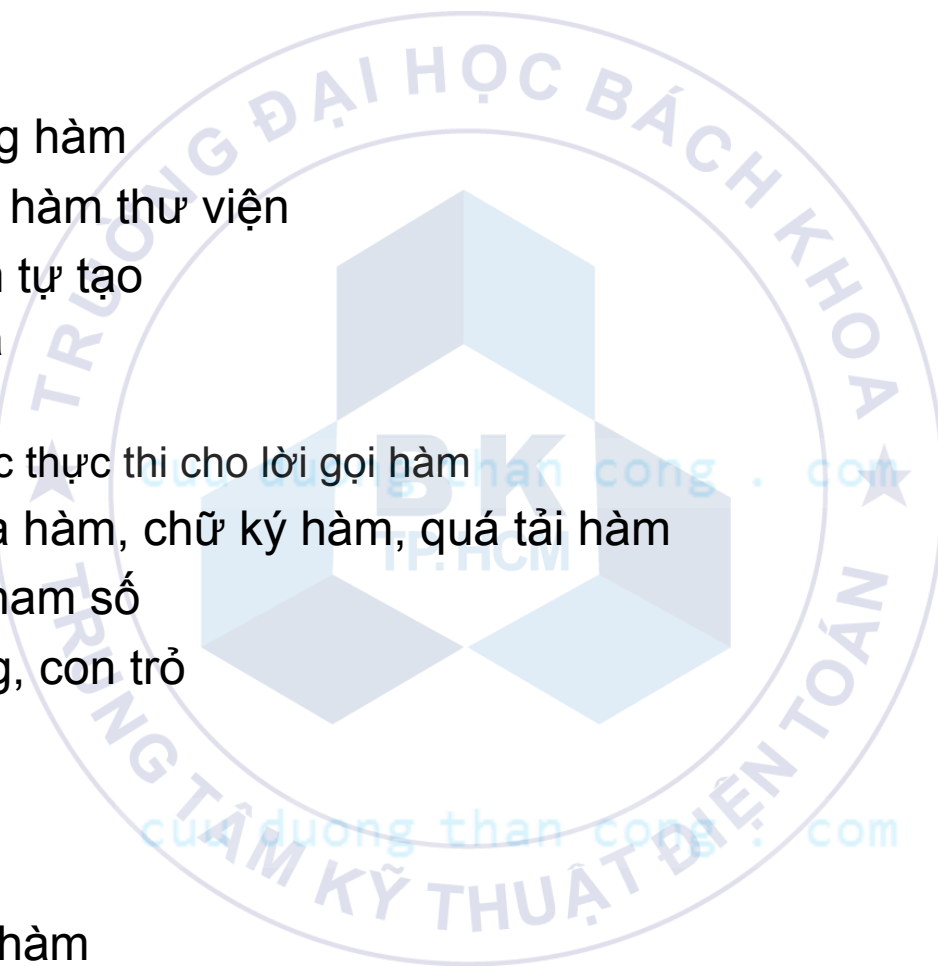


Chương 08 HÀM

Nguyễn Thanh Tùng

Nội dung

- Hàm là gì?
- Lý do sử dụng hàm
- Hàm main và hàm thư viện
- Sử dụng hàm tự tạo
 - Định nghĩa
 - Gọi hàm
 - Nguyên tắc thực thi cho lời gọi hàm
- Prototype của hàm, chữ ký hàm, quá tải hàm
- Kiểu truyền tham số
- Hàm và mảng, con trỏ
- Hàm inline
- Con trỏ hàm
- Hàm đệ quy
- Tạo thư viện hàm
 - Liên kết tĩnh và động



Hàm là gì?

■ Hàm là

- Một đơn vị xử lý
- Một chuỗi các lệnh **có liên quan, được thực hiện cùng nhau** để hoàn thành **một công việc** nào đó
- Ví dụ: trong thư viện `<math.h>`
 - Hàm `sin(x)`
 - Là chuỗi các lệnh tính toán để tính giá trị sin của một góc x được truyền vào, góc x có đơn vị tính là radian; hàm `sin(x)` trả về một số thực
 - Hàm `sqrt(x)`
 - Là chuỗi các lệnh tính toán để tính căn bậc 2 của đại lượng x được truyền vào, đại lượng x có đơn vị tính là một số thực (float hay double); hàm `sqrt` trả về một số thực

Hàm là gì?

■ Hàm là

- Một đơn vị tính toán
 - Nhận giá trị đầu vào
 - Tính toán
 - Trả về giá trị
- Minh họa

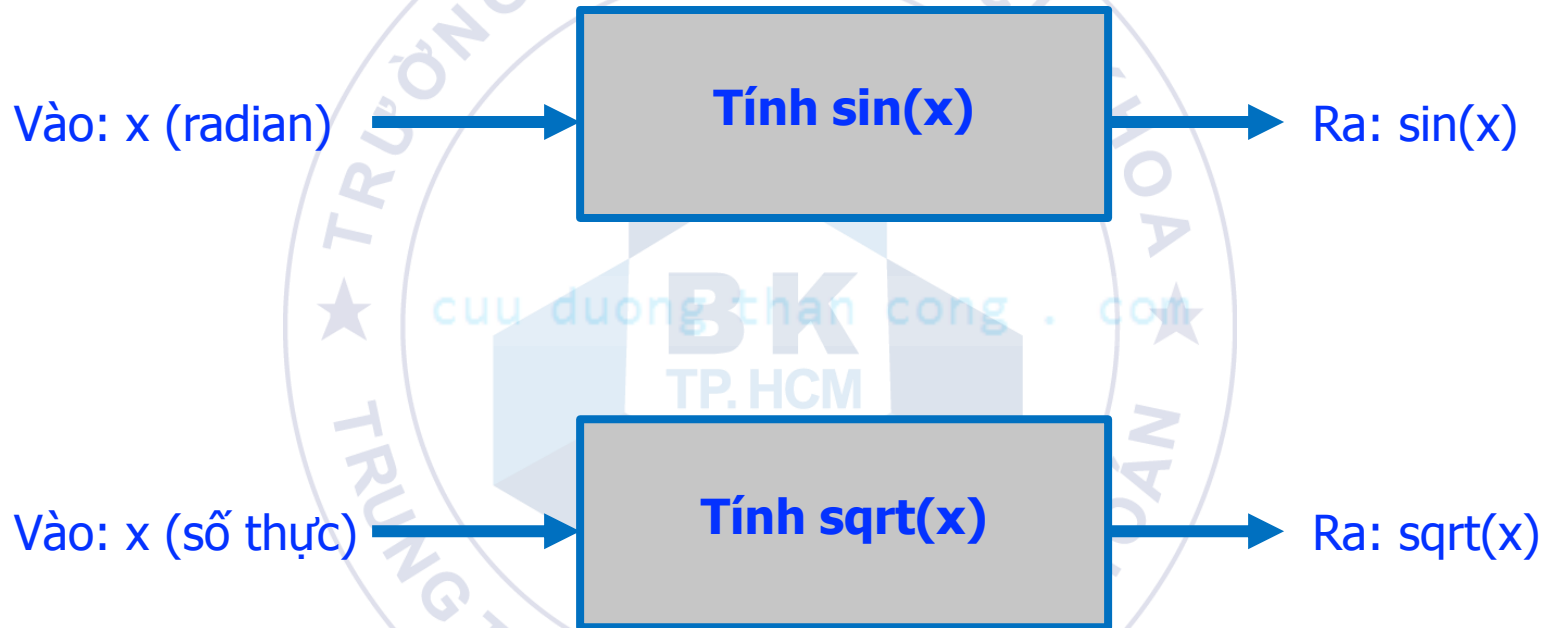
Các giá trị đầu vào

Chuỗi lệnh của hàm

Các giá trị đầu ra

Hàm là gì?

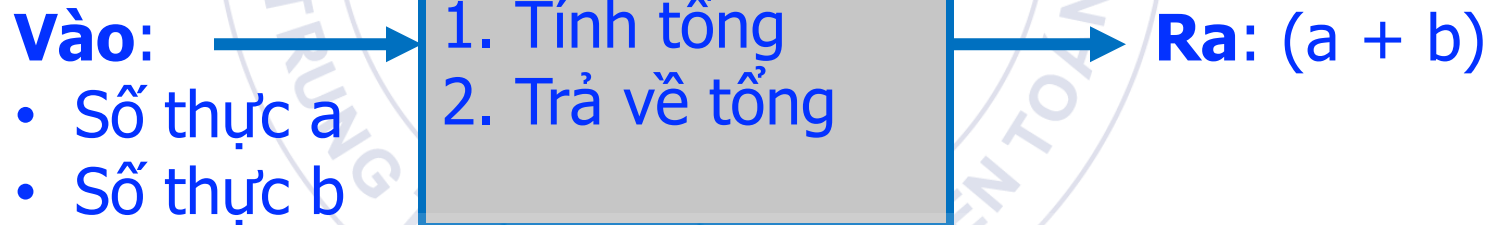
- Minh họa



Hàm là gì?

■ Minh họa cho hàm cộng hai số

- Vào: hai số a và b kiểu số thực
- Tính toán: phép cộng hai số
- Ra: tổng hai số



Lý do sử dụng hàm

- Tránh lặp lại mã nguồn
 - ➔ Tiết kiệm thời gian phát triển
 - ➔ Thay đổi đoạn mã nguồn trong hàm nhanh và dễ dàng, chỉ tại một nơi
- Sử dụng lại một đơn vị tính toán mà không phải viết lại
 - Tiết kiệm thời gian phát triển
 - Có thể chia sẻ đơn vị tính toán không chỉ cho một dự án mà cho nhiều dự án
 - Ví dụ: hãy xem xét trường hợp mà mọi dự án đều viết lại các hàm toán học: $\sin(x)$, \sqrt{x} , v.v. ➔ tốn kém và lãng phí
 - ➔ sử dụng thư viện `<math.h>`

Lý do sử dụng hàm

- Giúp cho việc phát triển giải thuật, việc tổ chức chương trình dễ dàng
 - Giải thuật:
 - Một kỹ thuật cơ bản của giải quyết vấn đề là: phân rã bài toán lớn thành bài toán con
 - ➔ Mỗi bài toán con có thể là một đơn vị tính toán (là một hàm)
 - Ví dụ: bài toán cho nhập một dãy số, tính toán và in ra giá trị trung bình và độ lệch chuẩn. Có thể phân rã thành các bài toán con
 - (1) Nhập dãy số
 - (2) Tính toán giá trị trung bình và độ lệch chuẩn
 - (3) In ra dãy số và các giá trị trung bình và độ lệch chuẩn

Lý do sử dụng hàm

- Giúp cho việc phát triển giải thuật, việc tổ chức chương trình dễ dàng
 - Giải thuật:
 - Ví dụ: bài toán cho nhập một dãy số, tính toán và in ra giá trị trung bình và độ lệch chuẩn. Có thể phân rã thành các bài toán con
 - (1) Nhập dãy số
 - (2) Tính toán giá trị trung bình và độ lệch chuẩn
 - (3) In ra dãy số và các giá trị trung bình và độ lệch chuẩn
 - ➔ Mỗi bài toán con ở trên có thể được viết thành hàm riêng

Lý do sử dụng hàm

- Giúp cho việc phát triển giải thuật, việc tổ chức chương trình dễ dàng
 - Tổ chức chương trình:
 - Nếu chương trình (ngôn ngữ C) được so sánh với một cuốn sách (Ngôn ngữ Tiếng việt)
 - Có cuốn sách nào trên thực tế mà tác giả viết toàn bộ cuốn sách thành các câu nối tiếp nhau; không phân ra chương, phần, phần con, đoạn hay không?
 - Hàm có ý nghĩa tương tự như các chương, các phần con trong các chương

Hàm main và hàm thư viện

Giá trị trả về: kiểu số nguyên **int**

Tên hàm: "**main**". Một chương trình phải và chỉ có 01 hàm **main** duy nhất

```
int main(){  
    // Các lệnh xử lý của hàm main  
    return 0;  
}
```

Trả về giá trị cho bên gọi hàm main

Giá trị trả về của main:

- Phải là kiểu **int**
- Có thể là một trong 2 hằng số
 - **EXIT_SUCCESS** (hoặc 0): nếu chương trình kết thúc thành công
 - **EXIT_FAILURE** (hoặc 1): nếu chương trình kết thúc với lỗi nào đó

Hàm main và hàm thư viện

Nếu muốn truyền tham số vào dòng lệnh

argc: số lượng các thông số, kể cả tên chương trình
argv: một danh sách các chuỗi, mỗi chuỗi là một thông số.
Khi truyền vào, tất cả các dữ liệu đều được hiểu như chuỗi

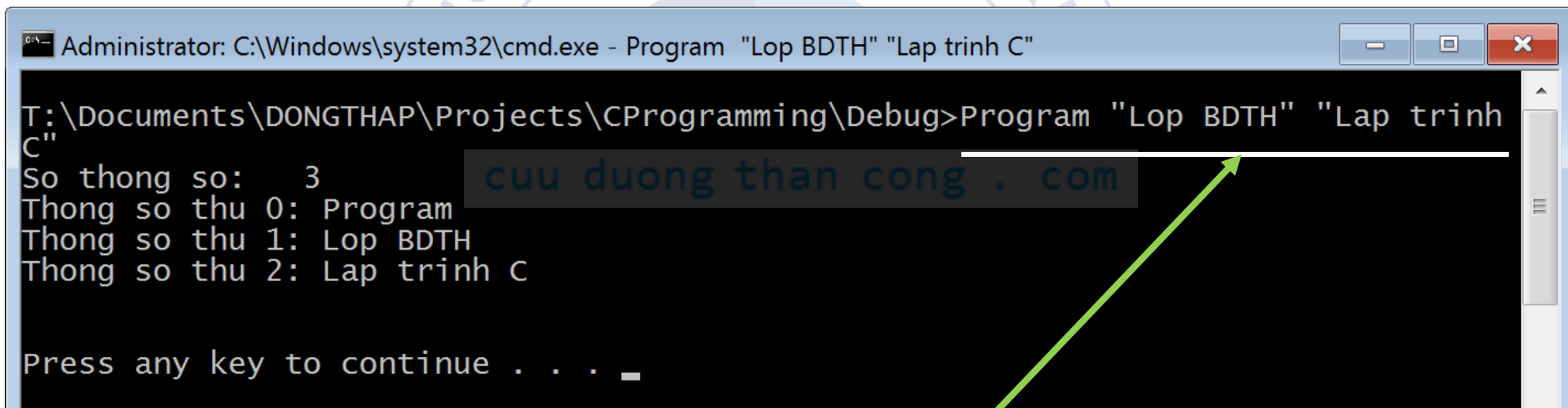
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]){
    printf("Số thông số: %3d\n", argc);
    for(int i=0; i < argc; i++)
        printf("Thông số thứ %d: %s\n", i, argv[i]);

    return EXIT_SUCCESS;
}
```

Hàm main và hàm thư viện

- Sau khi biên dịch chương trình thành công, tạo ra tập tin "Program.exe"
- Chạy chương trình "Program.exe" bằng dòng lệnh như sau:



```
Administrator: C:\Windows\system32\cmd.exe - Program "Lop BDTH" "Lap trinh C"

T:\Documents\DONGTHAP\Projects\CProgramming\Debug>Program "Lop BDTH" "Lap trinh
C"
So thong so: 3
Thong so thu 0: Program
Thong so thu 1: Lop BDTH
Thong so thu 2: Lap trinh C

Press any key to continue . . .
```

Thông cho chương trình

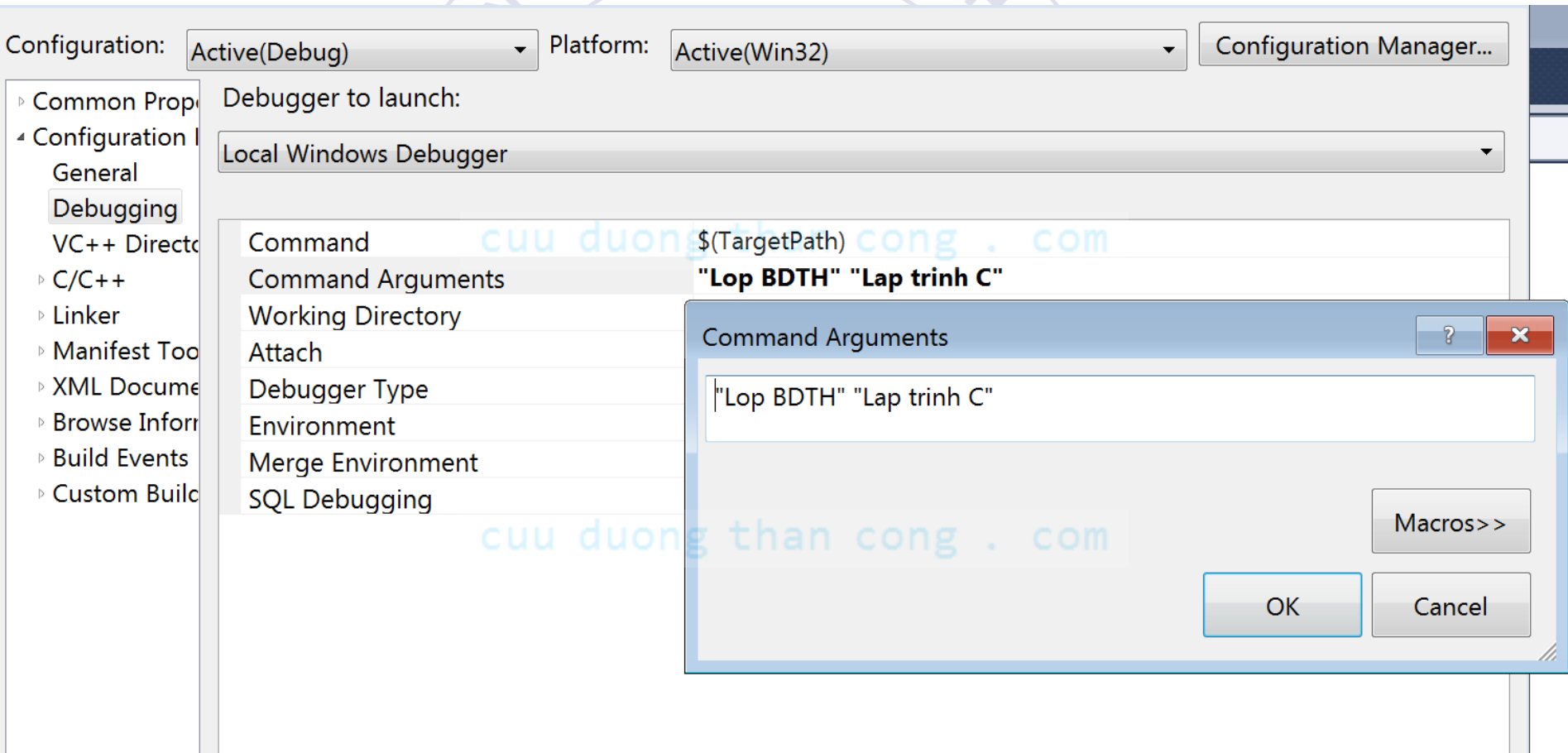
Hàm main và hàm thư viện

- Cách truyền tham số dòng lệnh trong Visual Studio
 - (1) Nhấn chuột phải trên <dự án> trong cửa sổ "Solution Explorer"
 - (2) Chọn "Debug" > "Command Arguments"
 - (3) Xổ chọn "Edit ..." trong danh sách chức năng của "Command Arguments"
 - (4) **Gõ vào danh sách thông số: các thông số cách nhau bởi khoảng trắng hay dấu ","**



Hàm main và hàm thư viện

- Cách truyền tham số dòng lệnh trong Visual Studio



Hàm main và hàm thư viện

- Ví dụ hàm trong thư viện <math.h>
 - (1) Dùng chỉ thị #include <math.h> để thông báo với bộ biên dịch là có sử dụng thư viện <math.h>
 - (2) Gọi các hàm cần thiết. Khi gọi một hàm chỉ cần biết
 - Tên hàm + công dụng của hàm
 - Các giá trị cần cung cấp cho hàm
 - Giá trị trả về của hàm

Hàm main và hàm thư viện

■ Ví dụ hàm trong thư viện <math.h>

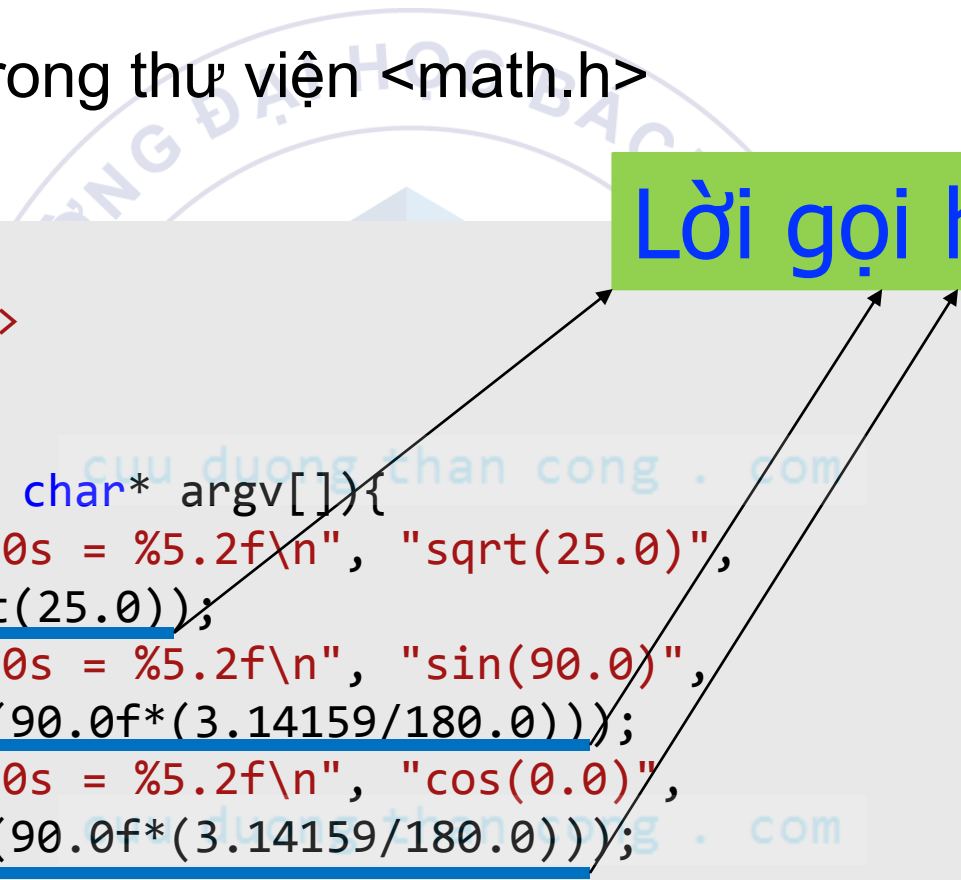
■ Ví dụ

Lời gọi hàm

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

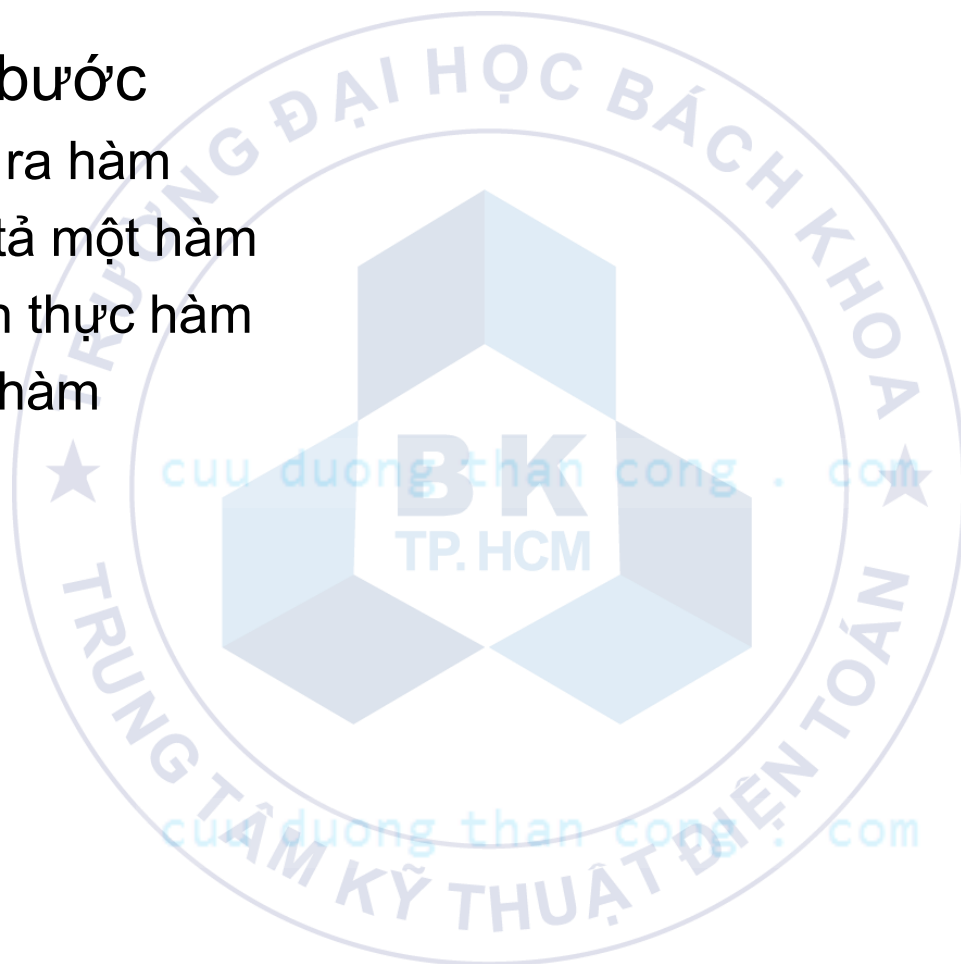
int main(int argc, char* argv[]){
    printf("%-10s = %5.2f\n", "sqrt(25.0)",
        sqrt(25.0));
    printf("%-10s = %5.2f\n", "sin(90.0)",
        sin(90.0f*(3.14159/180.0)));
    printf("%-10s = %5.2f\n", "cos(0.0)",
        cos(90.0f*(3.14159/180.0)));

    printf("\n\n");
    system("pause");
    return EXIT_SUCCESS;
}
```

A diagram with three arrows pointing from specific function calls in the code to a green box labeled 'Lời gọi hàm'. The first arrow points from 'sqrt(25.0)' in the first printf statement. The second arrow points from 'sin(90.0f*(3.14159/180.0))' in the second printf statement. The third arrow points from 'cos(90.0f*(3.14159/180.0))' in the third printf statement.

Sử dụng hàm tự tạo

- Gồm hai bước
 - (1) Tạo ra hàm
 - Mô tả một hàm
 - Hiện thực hàm
 - (2) Gọi hàm



Sử dụng hàm tự tạo

Định nghĩa hàm

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

```
int add(int a, int b)
```

```
{
    int c;
    c = a + b;
    return c;
}
```

```
int main(){
    printf("10 + 15 = %d", add(10, 15));

    system("pause");
    return EXIT_SUCCESS;
}
```

Phần mô tả một hàm, gồm:

- (1) Kiểu giá trị trả về: ví dụ này là **int**
- (2) Tên hàm: ví dụ này là "add"
- (3) Các thông số, là giá trị đầu vào.

Ví dụ này có

- Thông số thứ nhất: tên là "a", kiểu là **int**
- Thông số thứ hai: tên là "b", kiểu là **int**
- Danh sách thông số: bắt đầu bằng "(", kết thúc bằng ")"
- Các thông số cách nhau bằng dấu phẩy ","

Tên hàm và tên thông số tuân theo quy tắc đặt tên danh hiệu

Sử dụng hàm tự tạo

Định nghĩa hàm

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

```
int add(int a, int b)
```

```
{
    int c;
    c = a + b;
    return c;
}
```

```
int main(){
    printf("10 + 15 = %d", add(10, 15));

    system("pause");
    return EXIT_SUCCESS;
}
```

Phần thân của hàm, gồm:

Gồm các câu lệnh được thực hiện cùng nhau, lần lượt. Ở ví dụ này: có 3 lệnh trong thân hàm

Dùng câu lệnh **return** để chấm dứt thực thi hàm và trả điều khiển về cho hàm gọi → chuyển thực thi về lệnh kế sau lệnh gọi hàm

Các lệnh trong thân của hàm phải được gom lại với nhau bằng cặp dấu "{" và "}"

Sử dụng hàm tự tạo

Gọi hàm

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

```
int add(int a, int b)
```

```
{
    int c;
    c = a + b;
    return c;
}
```

```
int main(){
    printf("10 + 15 = %d", add(10, 15));

    system("pause");
    return EXIT_SUCCESS;
}
```

Lời gọi hàm:

Dùng tên hàm và truyền vào các giá trị cho các tham số của hàm:

- Thứ tự truyền vào quyết định giá trị nào sẽ được truyền cho thông số nào.
- Ví dụ này: 10 được truyền cho a; 15 được truyền cho b
- Phải truyền đủ (**không thiếu, không thừa**) tất cả các tham số
- **Không chấp nhận**: add(), add(10), add(10, 20, 30)

Phần mô tả hàm phải xuất hiện trước khi lời gọi hàm xảy ra để biên dịch không gặp lỗi danh hiệu chưa được định nghĩa : "undefined identifier"

Sử dụng hàm tự tạo

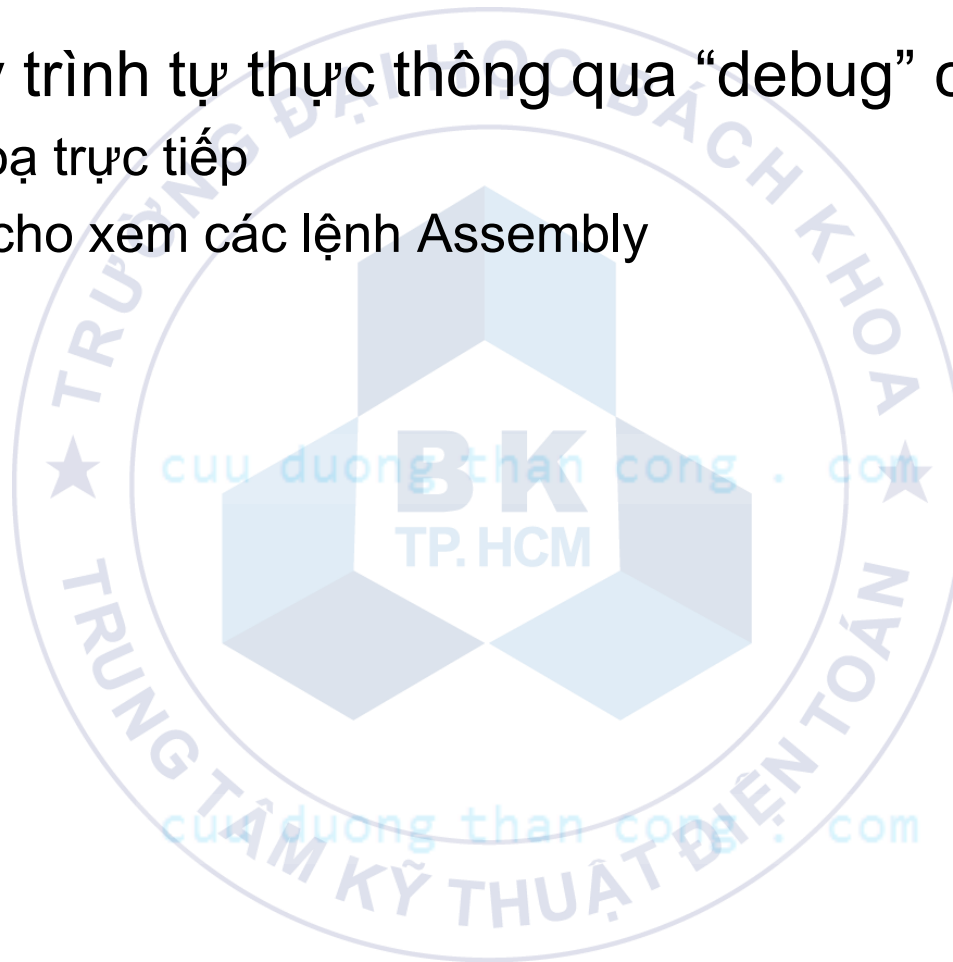
Nguyên tắc thực thi khi gọi hàm

- Khi lời gọi hàm được thực thi thì bộ thực thi sẽ làm các công việc
 - **Lưu vết**: lệnh kế tiếp của lệnh gọi hàm
 - **Copy** các thông số cho **hàm được gọi**
 - **Chuyển điều khiển thực thi** cho hàm được gọi để nó thực thi lệnh đầu tiên trong hàm được gọi
 - **Hàm được gọi thực thi các lệnh**
 - Khi hàm được gọi thực hiện lệnh **return**.
 - **Giải phóng** tất cả các biến cục bộ của nó
 - **Trả điều khiển** về lệnh theo sau lệnh gọi hàm
 - Hàm gọi **giải phóng các thông số** đã truyền và **thực thi lệnh kế tiếp** theo lệnh gọi hàm

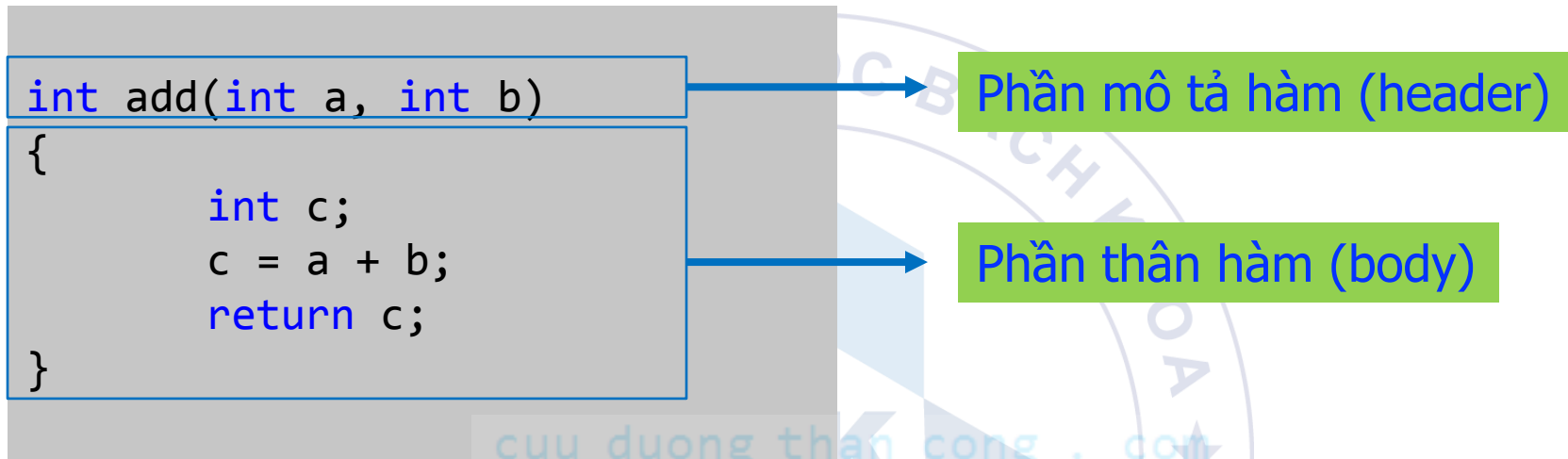
Sử dụng hàm tự tạo

Nguyên tắc thực thi khi gọi hàm

- Trình bày trình tự thực thông qua “debug” chương trình
 - Minh họa trực tiếp
 - Chú ý: cho xem các lệnh Assembly



Tổ chức mã nguồn



Phần mô tả **nên được tách riêng** khỏi toàn bộ phần định nghĩa một hàm.

Lý do:

- Không cần quan tâm thứ tự các hàm trong mã nguồn.
- Dùng lại các hàm trong dự án hoặc nhiều dự án
- Phát triển thư viện các hàm → không cần chuyển giao cho bên thứ 3 (người mua thư viện) mã nguồn phần hiện thực các hàm

Tổ chức mã nguồn

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

```
int add(int a, int b);
```

```
int main(){
    printf("10 + 15 = %d", add(10, 15));

    printf("\n\n");
    system("pause");
    return EXIT_SUCCESS;
}
```

```
int add(int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

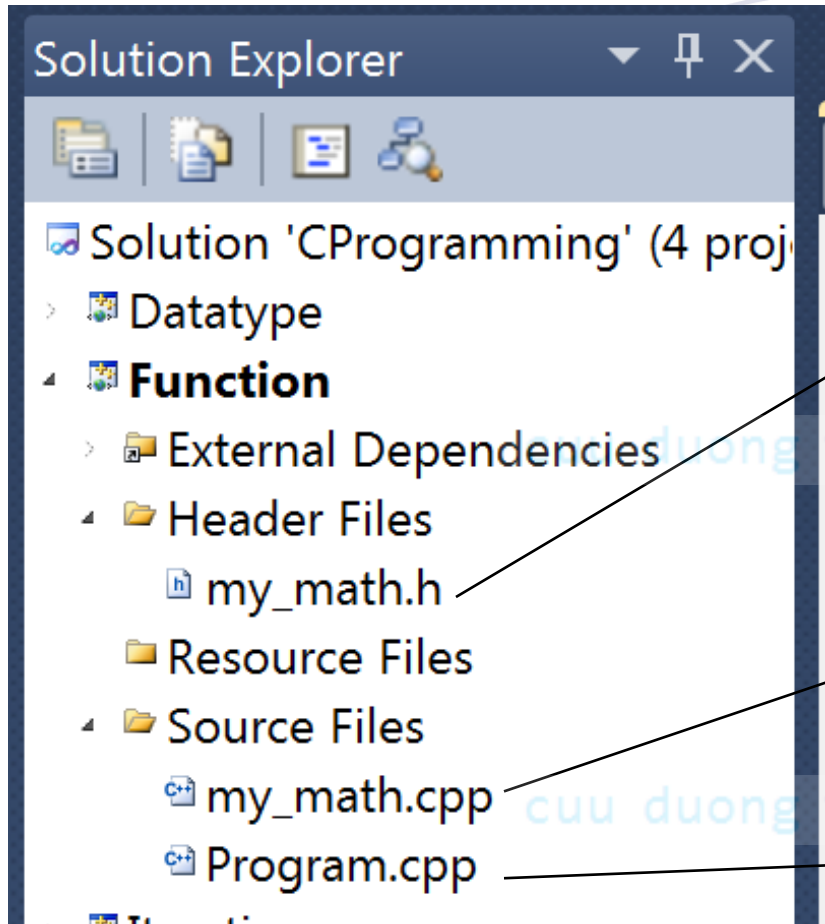
Tách rời phần mô tả của hàm "add" và đặt trước hàm "main" (trước khi sử dụng)

→ Không cần thiết đặt trước toàn bộ phần định nghĩa cho hàm "add" phía trước hàm "main"

Tổ chức mã nguồn

- Đưa phần mô tả vào một tập tin riêng
 - Gọi là tập tin mô tả (header): *.h
 - Có thể sử dụng lại ở nhiều tập tin khác trong dự án
 - Sử dụng chỉ thị `#if !defined(.) ... endif` để tránh lỗi “định nghĩa lặp lại” (redefinition)
- Đưa phần hiện thực vào một tập tin riêng
 - Gọi là tập tin hiện thực (implementation): *.c; *.cpp
 - Có thể sử dụng lại ở nhiều tập tin khác trong dự án
- Đưa phần hiện thực vào một tập tin riêng
 - Khai báo có sử dụng đến các hàm ở *.h nói trên
 - Gọi hàm

Tổ chức mã nguồn

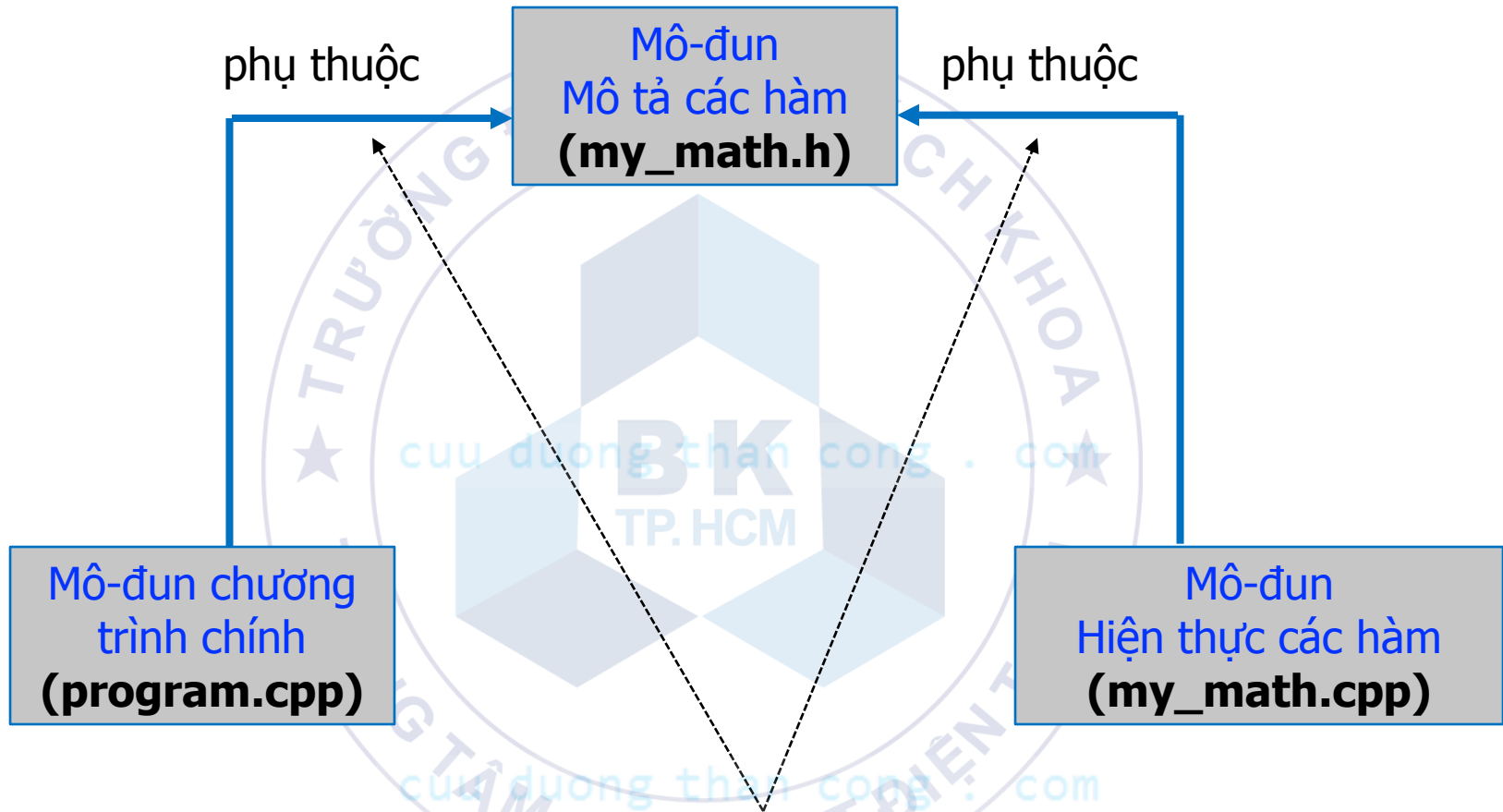


Tập tin chứa phần mô tả cho hàm, kiểu dữ liệu, v.v. các phần mô tả nói chung

Tập tin chứa phần định nghĩa hàm "add", có khai báo sử dụng phần mô tả *.h

Tập tin chứa hàm main, có sử dụng hàm "add".

Tổ chức mã nguồn



Sự phụ thuộc được biểu thị bởi chỉ thị
#include "my_math.h" trong mã nguồn

Tổ chức mã nguồn

Tập tin: "my_math.h"

```
#if !defined(MY_MATH_HEADER)
#define MY_MATH_HEADER

int add(int a, int b);

#endif
```

MY_MATH_HEADER
là một tên (danh
hiệu)

Phần mô tả
cho hàm
add

Ý nghĩa của cấu trúc chỉ thị #if:

NẾU như trong quá trình biên dịch, đến thời điểm hiện tại, chưa thấy một tên (MY_MATH_HEADER) xuất hiện thì định nghĩa một tên mới (MY_MATH_HEADER) và thực hiện biên dịch cho cả đoạn mã nguồn nằm trong phần tương ứng khối #if

NGƯỢC LẠI thì không định nghĩa tên mới và không biên dịch đoạn mã nguồn tương ứng khối if

Tổ chức mã nguồn

Tập tin: "my_math.h"

```
#if !defined(MY_MATH_HEADER)
#define MY_MATH_HEADER

int add(int a, int b);

#endif
```

MY_MATH_HEADER
là một tên (danh
hiệu)

Phần mô tả
cho hàm
add

Nhờ chỉ thị #if ... mà phần mô tả của các tên như hàm add ở đây không bị lặp lại nhiều lần khi được dùng ở nhiều tập tin khác nhau, kể cả trong tập tin *.h

Tổ chức mã nguồn

Tập tin: "my_math.cpp"

```
#include "my_math.h"  
int add(int a, int b)  
{  
    int c;  
    c = a + b;  
    return c;  
}
```

Khai báo sử dụng phần mô tả trong tập tin *.h ("my_math.h")

Phần định nghĩa một hàm (hàm `add`)

Tổ chức mã nguồn

Tập tin: "program.cpp"

```
#include <stdio.h>
#include <stdlib.h>
#include "my_math.h"

int main(){
    printf("10 + 15 = %d", add(10, 15));

    system("pause");
    return EXIT_SUCCESS;
}
```

Khai báo sử dụng phần mô tả trong tập tin *.h ("my_math.h")

Lời gọi hàm (hàm **add**)

Tổ chức mã nguồn

- Bài toán xây dựng các hàm tính toán cho số phức (Complex)
- Phân tích:
 - Cần cung cấp kiểu dữ liệu cho số phức: $z = x + y*i$
 - Cung cấp các hàm với kiểu mới này
 - Hàm lấy giá trị độ lớn của số phức

$$r = |z| = \sqrt{x^2 + y^2}$$

Tổ chức mã nguồn

- Bài toán xây dựng các hàm tính toán cho số phức (Complex)
- Phân tích:
 - Cần cung cấp kiểu dữ liệu cho số phức: $z = x + y*i$
 - Cung cấp các hàm với kiểu mới này
 - Hàm lấy giá trị độ lớn của số phức
 - Hàm lấy giá trị góc của số phức

$$\varphi = \arg(z) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{indeterminate} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Tổ chức mã nguồn

- Hàm lấy giá trị góc của số phức

$$\varphi = \arg(z) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{indeterminate} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

- Cần định nghĩa các hằng

- PI

- Hằng biểu diễn giá trị không xác định “indeterminate”

- Có thể biểu diễn bằng $2 \cdot \text{PI}$ hay bất cứ giá trị nào nằm ngoài khoảng $[-\text{PI}, \text{PI}]$.

- Cần định nghĩa macro để hỗ trợ phép so sánh “==” với số thực để tránh sai số biểu diễn số học

Tổ chức mã nguồn

- Hàm lấy giá trị góc của số phức
 - Cần định nghĩa các hằng
 - PI
 - Hằng biểu diễn giá trị không xác định “indeterminate”
 - Cần định nghĩa macro để hỗ trợ phép so sánh “==”
 - Hằng hỗ trợ chuyển đổi từ RADIAN sang độ và ngược lại

```
#define PI 3.14159265
#define UN_DEF_ANGLE (2*PI)
#define EPSILON (1.0E-13)
#define equal(d1, d2) (abs((d1) - (d2)) < EPSILON)
#define RAD_2_DEG (180.0/PI)
#define DEG_2_RAD (PI/180.0)
```

Tổ chức mã nguồn

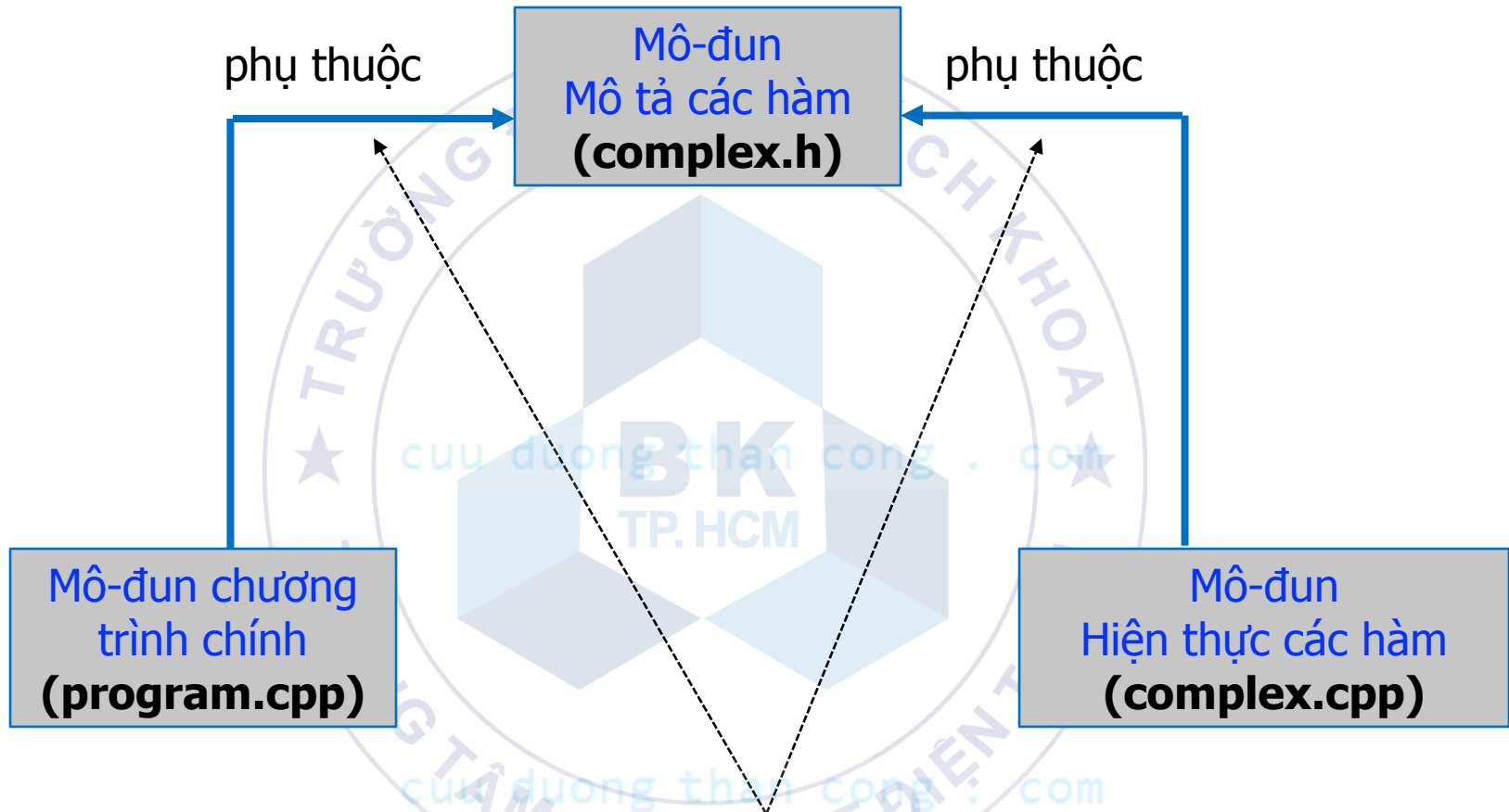
■ Phân tích:

- Tổ chức mã nguồn, gồm các mô-đun
 - Tập tin mô tả (complex.h) chứa mô tả kiểu mới và mô tả các hàm với số phức
 - Tập tin hiện thực (complex.cpp) chứa phần hiện thực của các hàm
 - Tập tin chương trình (program.cpp) chứa hàm main và sử dụng các hàm của số phức

cuuduongthancong.com

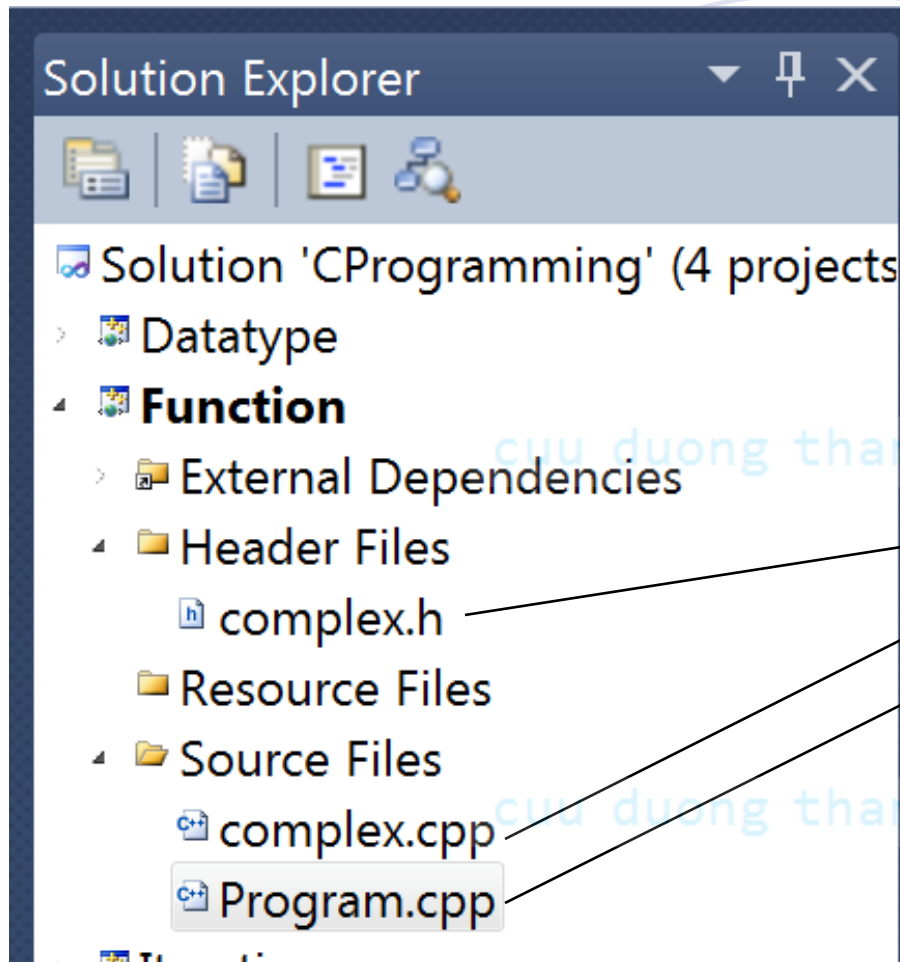
cuuduongthancong.com

Tổ chức mã nguồn



Sự phụ thuộc được biểu thị bởi chỉ thị
#include "complex.h" trong mã nguồn

Tổ chức mã nguồn



Các tập tin trong dự án

Tổ chức mã nguồn

Tập tin: **complex.h**

```
#if !defined(MY_MATH_HEADER)
#define MY_MATH_HEADER

#define PI 3.14159265
#define UN_DEF_ANGLE (2*PI)
#define EPSILON (1.0E-13)
#define equal(d1, d2) (abs((d1) - (d2)) < EPSILON)
#define RAD_2_DEG (180.0/PI)

typedef struct{
    double x, y;
} Complex;

double get_magnitude(Complex c);
double get_angle(Complex c);
void print_complex(Complex c);

#endif
```


Tổ chức mã nguồn

Thuộc tập tin: **complex.cpp**

```
#include "complex.h"  
#include <math.h>  
#include <stdio.h>
```

Khai báo sử dụng mô tả số phức

Khai báo sử dụng các hàm toán trong math.h

Khai báo sử dụng hàm printf trong khi in số phức ra màn hình

```
double get_magnitude(Complex c){  
    double mag;  
    mag = sqrt(c.x*c.x + c.y*c.y);  
    return mag;  
}
```

Phần định nghĩa hàm get_magnitude: trả về độ lớn số phức c ở đầu vào

```
double get_angle(Complex c){  
    double angle;  
    if(c.x > 0){  
        angle = atan(c.y/c.x);  
    }  
    if((c.x < 0) && (c.y >= 0)){  
        angle = atan(c.y/c.x) + PI;  
    }  
    if((c.x < 0) && (c.y < 0)){  
        angle = atan(c.y/c.x) - PI;  
    }  
    if(equal(c.x, 0.0) && (c.y > 0)){  
        angle = PI/2;  
    }  
    if(equal(c.x, 0.0) && (c.y < 0)){  
        angle = -PI/2;  
    }  
    if(equal(c.x, 0.0) && equal(c.y, 0.0)){  
        angle = UN_DEF_ANGLE;  
    }  
    angle *= RAD_2_DEG;  
    return angle;  
}
```

Hàm lấy góc của số phức

Tổ chức mã nguồn

Thuộc tập tin: **complex.cpp**

```
void print_complex(Complex c){  
    printf("[%-.2f,%.2f*i]", c.x, c.y);  
}
```

cuu duong than cong . com

Hàm in số phức ra màn hình

cuu duong than cong . com

Các kiểu truyền tham số



Tham số và đối số

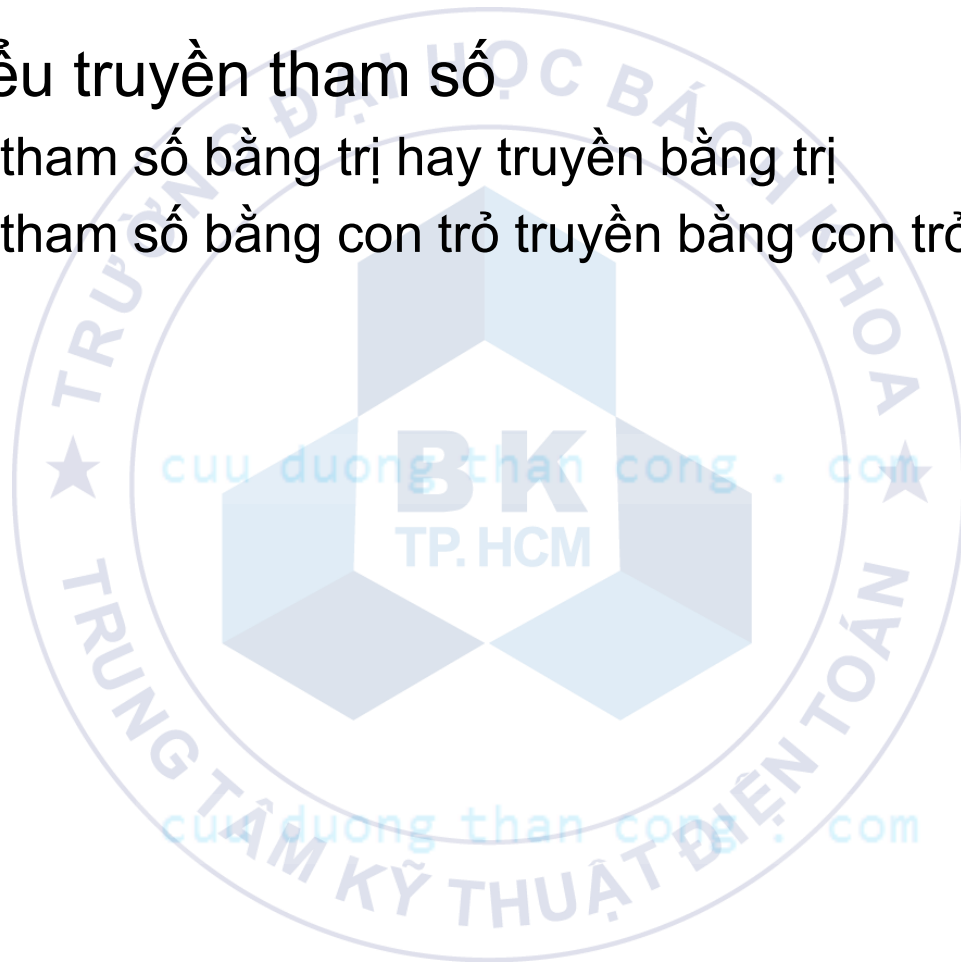
```
int main(){  
    printf("10 + 15 = %d", add(10, 15));  
  
    system("pause");  
    return EXIT_SUCCESS;  
}
```

10: là **đối số** của **thông số a**
15: là **đối số** của **thông số b**

```
int add(int a, int b)  
{  
    int c;  
    c = a + b;  
    return c;  
}
```

Các kiểu truyền tham số

- Có hai kiểu truyền tham số
 - Truyền tham số bằng trị hay truyền bằng trị
 - Truyền tham số bằng con trỏ truyền bằng con trỏ, truyền bằng địa chỉ



Các kiểu truyền tham số

■ Lý do có hai kiểu truyền

■ Truyền bằng trị

- Được sử dụng khi **KHÔNG CHO PHÉP** hàm được gọi thay đổi giá trị đối

■ Truyền bằng địa chỉ

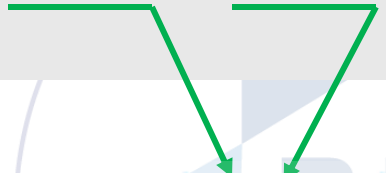
- Được sử dụng khi **MUỐN CHO PHÉP** hàm được gọi thay đổi giá trị đối
- Hoặc
 - Khi không muốn bộ thực thi tốn nhiều thời gian cho việc chuẩn bị tham số của hàm được gọi (nghĩa là **COPY** giá trị của đối số vào thông số), như truyền một mảng nhiều phần tử vào hàm được gọi

Các kiểu truyền tham số

Cách nhận biết hai kiểu truyền

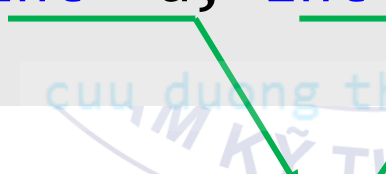
Dấu sao (*) chỉ ra thông số nào sẽ được truyền bằng địa chỉ

```
void swap(int a, int b){  
}
```



a và **b** sẽ được truyền bằng trị

```
void swap(int *a, int *b){  
}
```



a và **b** sẽ được truyền bằng địa chỉ

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng trị

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

```
void swap(int a, int b){
}

int main(){
    int x = 10, y = 100;

    swap(x, y);
    swap(10, 100);
    swap(x + 10, y*2);

    return EXIT_SUCCESS;
}
```

a và **b** sẽ được truyền bằng trị

Đối số có thể là: **biến**

Đối số có thể là: **hằng số**

Đối số có thể là: **biểu thức**
có cùng kiểu với kiểu tham số

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng trị

```
#include <stdio.h>
#include <stdlib.h>
void swap(int a, int b){
    int t = a;
    a = b;
    b = t;
}
int main(){
    int x = 10, y = 100;
    printf("Truoc khi goi ham swap(x,y)\n");
    printf("x = %3d; y = %3d\n", x, y);
    swap(x, y);
    printf("Truoc khi goi ham swap(x,y)\n");
    printf("x = %3d; y = %3d\n", x, y);

    printf("\n\n");
    system("pause");
    return EXIT_SUCCESS;
}
```

Hàm swap hoán đổi giá trị của hai biến a và b qua biến tạm t

Lời gọi hàm: truyền x và y là đối số tương ứng cho a và b

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng trị

```
\\psf\Home\Documents\DONGTHAP\Projects\CProgramming\Debug\Function.exe
Truoc khi goi ham swap(x,y)
x = 10; y = 100
Truoc khi goi ham swap(x,y)
x = 10; y = 100
```

Giá trị của x và y vẫn thay đổi sau khi hàm $\text{swap}(x,y)$ thực thi xong.

Lý do: Bộ thực thi đã thực hiện

- **COPY** giá trị của hai đối số x và y vào vùng nhớ cho 2 thông số a và b tương ứng
- Hàm **$\text{swap}(\text{int } a, \text{int } b)$** chỉ hoán đổi giá trị của 2 biến a và b của nó rồi kết thúc
- => **Vùng nhớ của hai biến x và y không bị ảnh hưởng**

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng trị

```
void swap(int a, int b){  
    int t = a;  
    a = b;  
    b = t;  
}
```

```
int main(){  
    int x = 10, y = 100;  
    swap(x, y);  
    return EXIT_SUCCESS;  
}
```

a : 10

b : 100

x : 10

y : 100

COPY giá trị

Lời gọi hàm **swap(x, y)** trong main khiến cho biến a và b của giữ giá trị của x và y tương ứng; nghĩa là 10 và 100

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int a, int b){  
    int t = a;  
    a = b;  
    b = t;  
}
```

a : 10

b : 100

t : 10

```
int main(){  
    int x = 10, y = 100;  
    swap(x, y);  
    return EXIT_SUCCESS;  
}
```

x : 10

y : 100

int t = a; khiến cho biến t được tạo ra và chứa giá trị của a, nghĩa là 10

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int a, int b){  
    int t = a;  
    a = b;  
    b = t;  
}
```

a :

100

b :

100

t :

10

```
int main(){  
    int x = 10, y = 100;  
    swap(x, y);  
    return EXIT_SUCCESS;  
}
```

x :

10


y :

100

a = b; Khiến cho biến a có giá trị của b nghĩa là 100

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int a, int b){  
    int t = a;  
    a = b;  
    b = t;   
}
```

a : **100**

b : **10**

t : **10**

```
int main(){  
    int x = 10, y = 100;  
    swap(x, y);  
    return EXIT_SUCCESS;  
}
```


x : **10**

y : **100**

b = t; Khiến cho biến b có giá trị của t nghĩa là 10

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int a, int b){  
    int t = a;  
    a = b;  
    b = t;   
}
```

a : 100

b : 10

t : 10

```
int main(){  
    int x = 10, y = 100;  
    swap(x, y);  
    return EXIT_SUCCESS;  
}
```

x : 10

y : 100

swap(int a, int b) Hoàn tất hoán đổi giá trị a và b, không chạm gì đến x và y ở hàm main. Do đó, khi nó kết thúc hai biến x và y trong main vẫn giữ nguyên giá trị

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

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

void swap(int *a, int *b){
}

int main(){
    int x = 10, y = 100;

    swap(&x, &y);
    swap(10, 100);
    swap(x + 10, y*2);

    return EXIT_SUCCESS;
}
```

a và **b** sẽ được truyền bằng địa chỉ

Đối số: **CHỈ SỐ THỂ LÀ BIẾN**

Đối số: **KHÔNG THỂ LÀ
hằng số**

Đối số: **KHÔNG THỂ LÀ
biểu thức**

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}
int main(){
    int x = 10, y = 100;
    printf("Truoc khi goi ham swap(x,y)\n");
    printf("x = %3d; y = %3d\n", x, y);
    swap(&x, &y);
    printf("Truoc khi goi ham swap(x,y)\n");
    printf("x = %3d; y = %3d\n", x, y);

    printf("\n\n");
    system("pause");
    return EXIT_SUCCESS;
}
```

Hàm swap hoán đổi giá trị của hai đối số được truyền cho a và b tương ứng.

Toán tử sao (*):

a là địa chỉ → (*a) là giá trị của vùng nhớ có địa chỉ là a

Lời gọi hàm: truyền địa chỉ của biến x và y vào hai thông số a và b tương ứng.

Dùng toán tử &: để lấy địa chỉ

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
\\psf\Home\Documents\DONGTHAP\Projects\CProgramming\Debug\Function.exe
Truoc khi goi ham swap(x,y)
x = 10; y = 100
Truoc khi goi ham swap(x,y)
x = 100; y = 10
```

Giá trị của x và y đã được hoán đổi sau khi hàm `swap(&x, &y)` thực thi xong.
Lý do: Bộ thực thi đã thực hiện

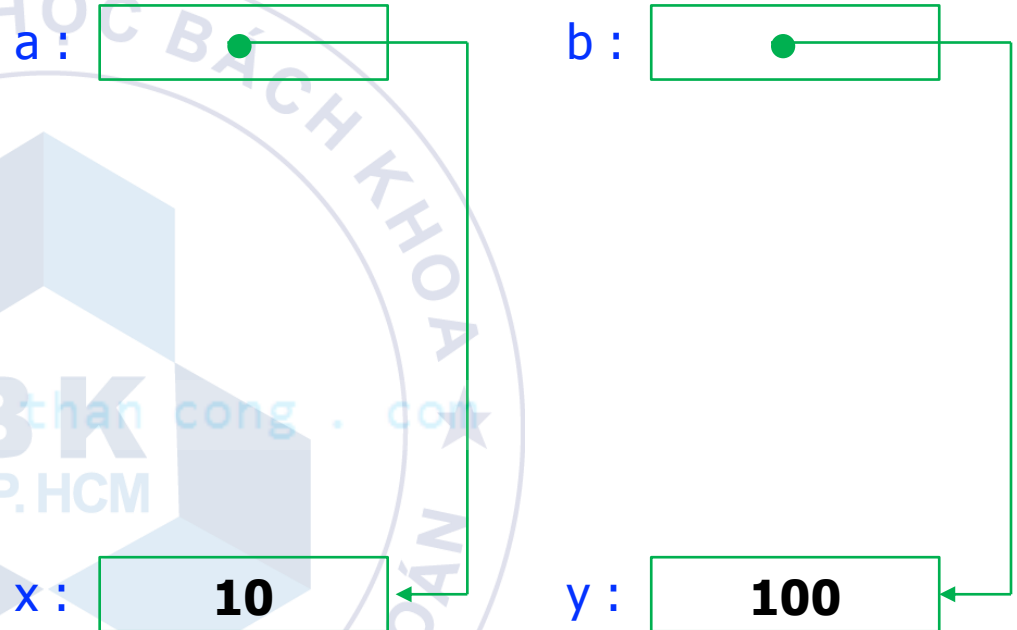
- **COPY ĐỊA CHỈ** của hai đối số x và y vào vùng nhớ của hai thông số a và b tương ứng
- Hàm **`swap(int *a, int *b)`** chỉ hoán đổi giá trị của biến x và y thông qua địa chỉ a và b : dùng toán tử $*$
- **Lưu ý: Lời gọi hàm `swap` cần dùng toán tử $&$ để lấy địa chỉ**

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}
```


```
int main(){  
    int x = 10, y = 100;  
    swap(&x, &y);  
    return EXIT_SUCCESS;  
}
```



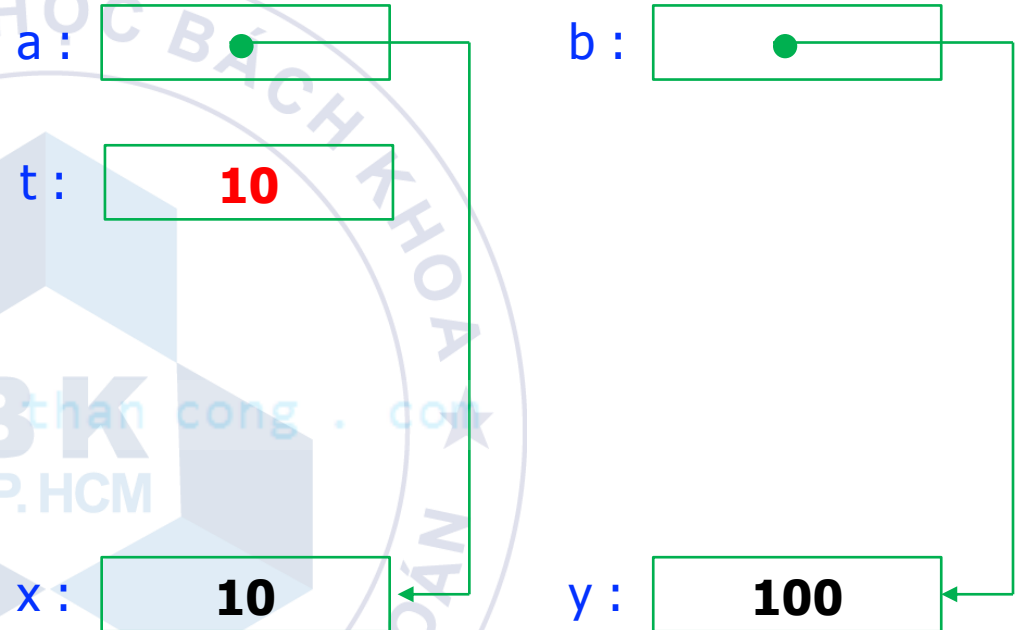
Lời gọi hàm **swap(&x, &y)** trong main khiến cho biến **a** và **b** của hàm **swap(int *a, int *b)** chứa địa chỉ của **x** và **y** tương ứng

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int *a, int *b){  
    int t = *a;   
    *a = *b;  
    *b = t;  
}
```

```
int main(){  
    int x = 10, y = 100;  
    swap(&x, &y);  
    return EXIT_SUCCESS;  
}
```




int t = *a; khiến cho biến t được tạo ra và chứa giá trị của ô nhớ có địa chỉ là a, nghĩa là biến x

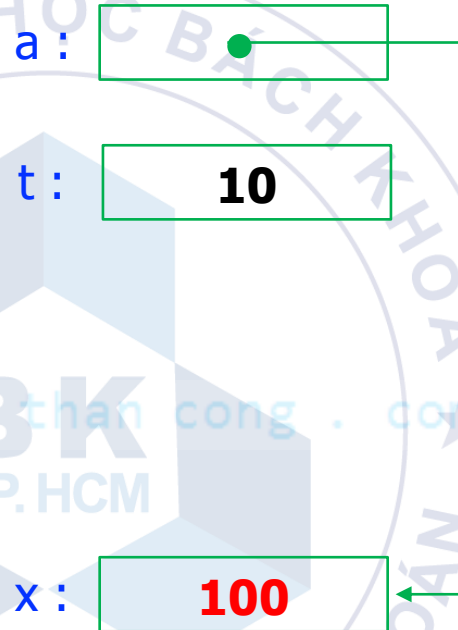
Lưu ý về sử dụng toán tử *: **a là địa chỉ**, nhưng ***a là số nguyên**

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;   
    *b = t;  
}
```


```
int main(){  
    int x = 10, y = 100;  
    swap(&x, &y);  
    return EXIT_SUCCESS;  
}
```



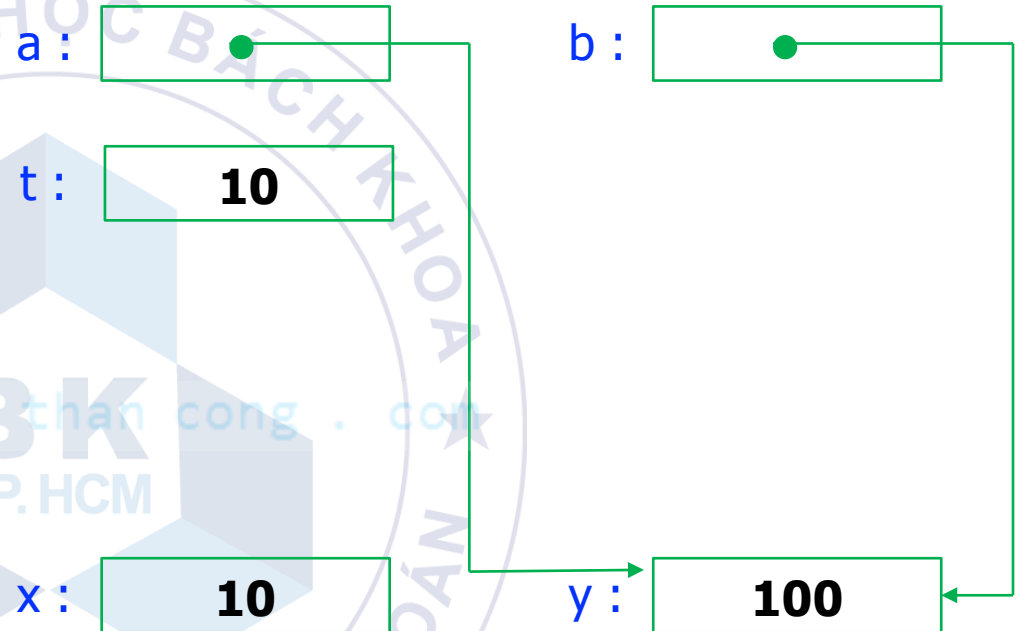
***a = *b;** khiến cho giá trị của ô nhớ có địa chỉ là `b` được gán vào giá trị của ô nhớ có địa chỉ `a`; nghĩa là biến `y` được gán vào `x`

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int *a, int *b){  
    int t = *a;  
    a = b;   
    *b = t;  
}
```

```
int main(){  
    int x = 10, y = 100;  
    swap(&x, &y);  
    return EXIT_SUCCESS;  
}
```



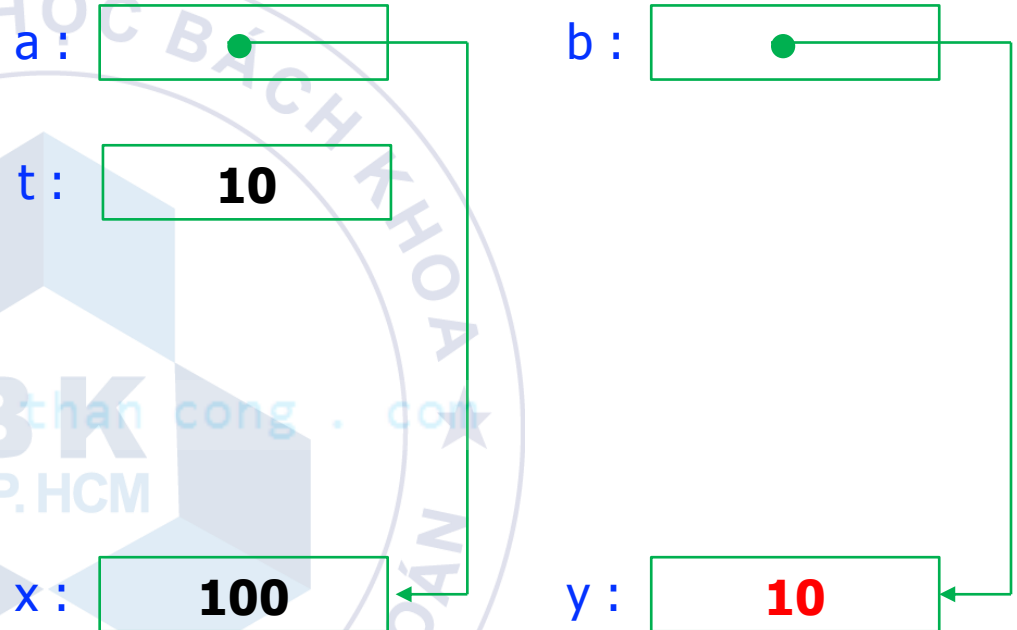
Nếu thực hiện $a = b$; sẽ khiến cho cả hai ô nhớ **a** và **b** đều chứa địa chỉ của biến **y** → không phù hợp với ý đồ hoán đổi trị

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}
```

```
int main(){  
    int x = 10, y = 100;  
    swap(&x, &y);  
    return EXIT_SUCCESS;  
}
```



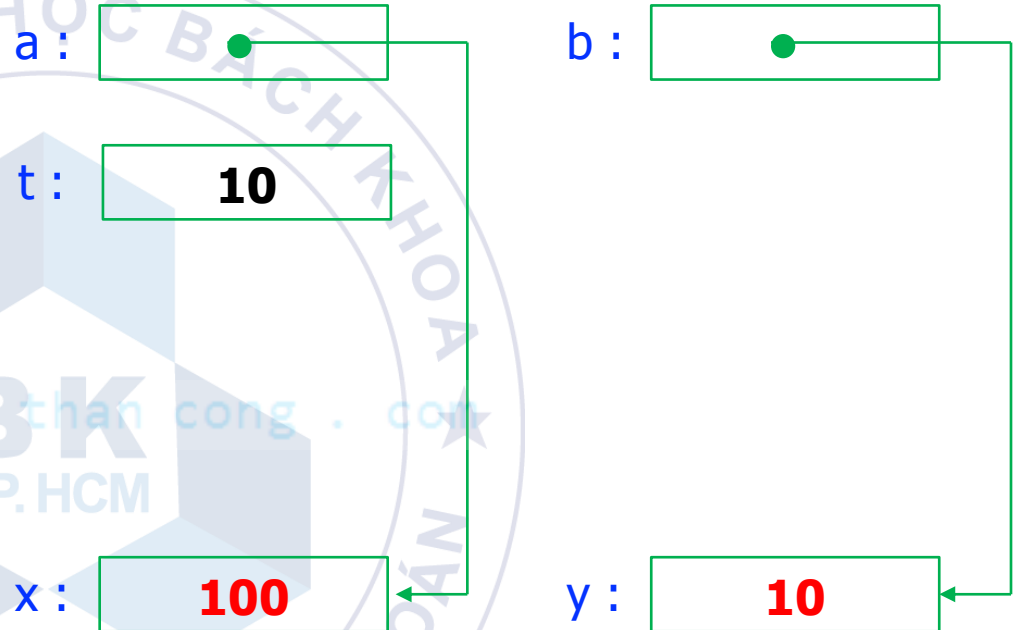
***b = t;** khiến cho giá trị của biến `t` được COPY vào ô nhớ có địa chỉ là `b`; tương đương, biến `y` được gán giá trị `t` (nghĩa là `10`)

Các kiểu truyền tham số

Lời gọi hàm: truyền bằng địa chỉ

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}
```

```
int main(){  
    int x = 10, y = 100;  
    swap(&x, &y);  
    return EXIT_SUCCESS;  
}
```



Do đó, khi chương `swap` kết thúc và trả điều khiển về hàm `main` thì giá trị `x` và `y` là `100` và `10` tương ứng, nghĩa là đã được hoán đổi

Hàm và mảng, con trỏ

Lưu ý:

Mảng và con trỏ đều là những ô nhớ chứa địa chỉ



Hàm và mảng, con trỏ

- Sử dụng hàm để xử lý mảng, cần truyền vào hàm
 - (1) Mảng giá trị
 - Trong C:
Truyền mảng vào hàm
 - ⇔ truyền địa chỉ của phần tử đầu tiên vào hàm
 - ⇔ truyền con trỏ đến phần tử đầu tiên vào hàm
 - C luôn luôn truyền mảng vào hàm bằng phương pháp truyền bằng địa chỉ
- (2) Số lượng phần tử của mảng

Hàm và mảng, con trỏ

- Sử dụng hàm để xử lý mảng, cần truyền vào hàm
 - Ví dụ:
 - Viết hàm để in ra các phần tử của mảng
 - Phân tích
 - (1) Cần truyền mảng giá trị
 - (2) Cần truyền số lượng phần tử của mảng

cuduongthanhong.com

cuduongthanhong.com

cuduongthanhong.com

cuduongthanhong.com

cuduongthanhong.com

cuduongthanhong.com

Hàm và mảng, con trỏ

- Cú pháp khai báo thông số

```
void print_array1(int arr[MAX_SIZE], int size){  
}  
void print_array2(int arr[], int size){  
}  
void print_array3(int *arr, int size){  
}
```

Cả 3 hàm trên **ĐỀU** có thông số đầu tiên là con trỏ đến số nguyên, nghĩa là giống nhau

➔ Để 3 hàm cùng tên thì sẽ có lỗi trong lúc biên dịch – lỗi tái định nghĩa một danh hiệu

Thông số thứ 2: size là số phần tử thuộc mảng

Hàm và mảng, con trỏ

■ Cú pháp khai báo thông số

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
void print_array1(int arr[MAX_SIZE], int size){}
void print_array2(int arr[], int size){}
void print_array3(int *ptr, int size){}

int main(){
    int size = 5;
    int a[MAX_SIZE];
    for(int i=0; i<size; i++) a[i] = i*i;

    print_array1(a, size);
    print_array2(a, size);
    print_array3(a, size);

    return EXIT_SUCCESS;
}
```

Lời gọi hàm:

- Thông số thứ 1: **a**

Truyền địa chỉ của phần tử đầu tiên vào hàm
(truyền bằng địa chỉ)

- Thông số thứ 2: **size**

Truyền bằng trị

Hàm và mảng, con trỏ

- Tương tự, nếu phần tử là các kiểu cấu trúc như sau

```
typedef struct{  
    char code[10];  
    char name[50];  
    float gpa;  
} Student;
```

```
typedef struct{  
    double x,y,z;  
} Point3D;
```

Hàm và mảng, con trỏ

- Tương tự, thì hàm in các mảng có kiểu Student và Point3D có thể có cú pháp:

```
void print_array1(Student arr[MAX_SIZE], int size);  
void print_array2(Student arr[], int size);  
void print_array3(Student *arr, int size);
```

```
void print_array1(Point3D arr[MAX_SIZE], int size);  
void print_array2(Point3D arr[], int size);  
void print_array3(Point3D *arr, int size);
```


Hàm và mảng, con trỏ

Không cho phép hàm thay đổi phần tử trên mảng

- Sử dụng từ khoá **const** trước tên kiểu phần tử

```
void print_array1(const Student arr[MAX_SIZE], int size);  
void print_array2(const Student arr[], int size);  
void print_array3(const Student *arr, int size);
```

```
void print_array1(const Point3D arr[MAX_SIZE], int size);  
void print_array2(const Point3D arr[], int size);  
void print_array3(const Point3D *arr, int size);
```

Hàm và mảng, con trỏ

Không cho phép hàm thay đổi phần tử trên mảng

- Sử dụng từ khoá `const` trước tên kiểu phần tử

```
void print_array3(const Point3D *arr, int size){  
    arr[0].x = 100;  
}
```

const Point3D *arr

Error: expression must be a modifiable lvalue

Đã khai báo `const` + cố tình thay đổi phần tử trên mảng thì có lỗi

Lỗi: “**expression must be a modifiable lvalue**”

Nghĩa: `arr` không được dùng trong bên trái của biểu thức gán

Hàm inline

■ Hàm inline là gì?

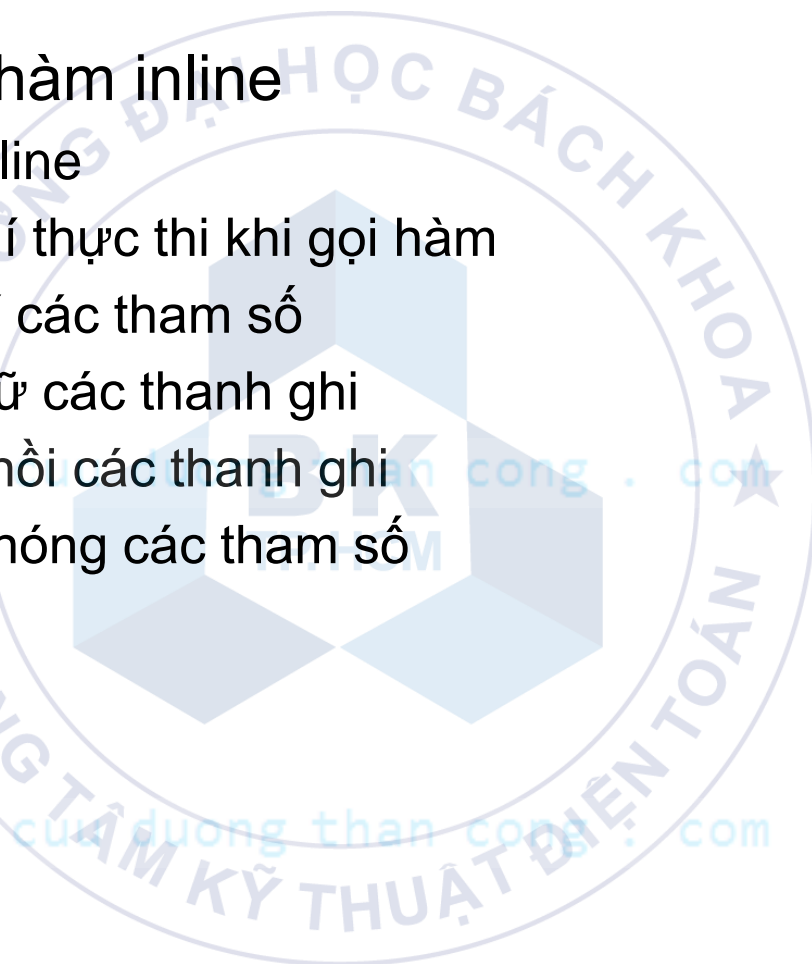
- Là hàm có từ khoá “**inline**” đứng trước kiểu trả về của hàm, như ví dụ

```
inline void print_array1(const Point3D arr[MAX_SIZE], int size);  
inline void print_array2(const Point3D arr[], int size);  
inline void print_array3(const Point3D *arr, int size);
```

Từ khoá **inline**

Hàm inline

- Tác dụng của hàm inline
 - Hàm không inline
 - Tồn chi phí thực thi khi gọi hàm
 - COPY các tham số
 - Lưu trữ các thanh ghi
 - Phục hồi các thanh ghi
 - Giải phóng các tham số
 - V.v



Hàm inline

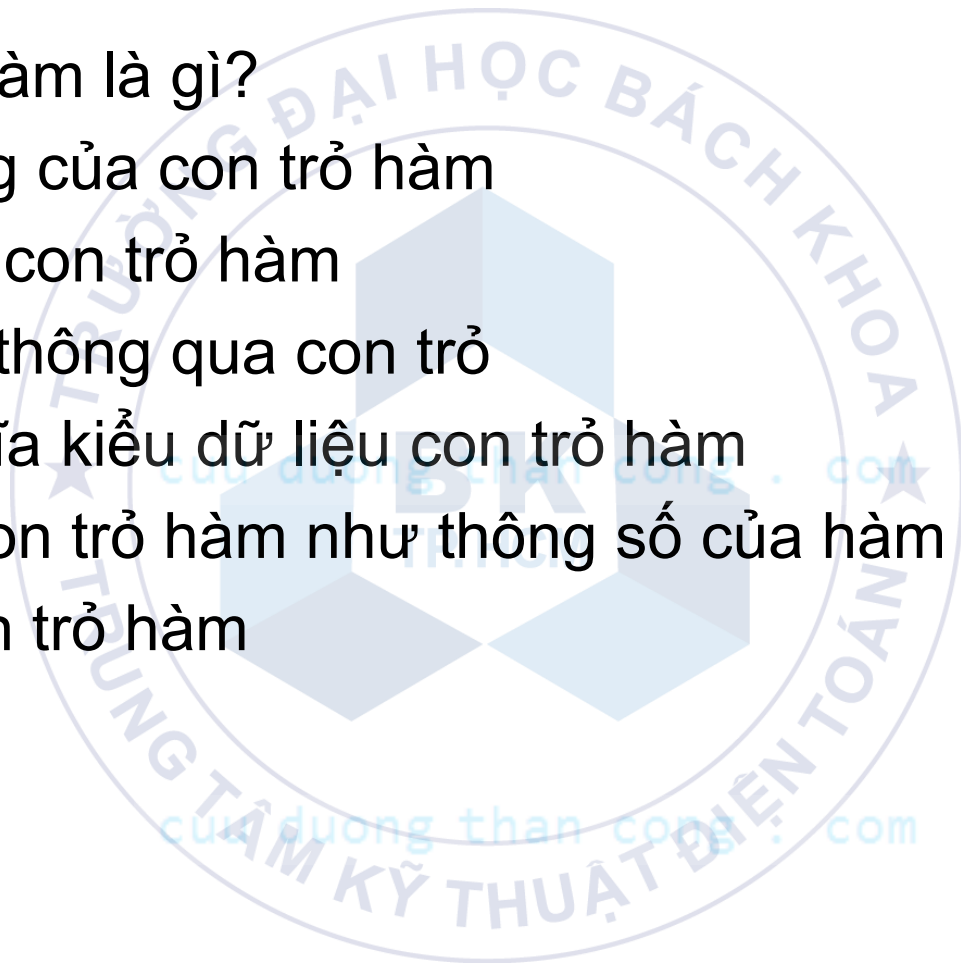
■ Tác dụng của hàm inline

■ Hàm inline

- Thay vì làm các thủ tục để gọi hàm và trả về từ hàm được gọi, mã thực thi của hàm inline được chèn trực tiếp tại vị trí gọi hàm này.
- => Tiết kiệm chi phí gọi hàm
- => Làm tăng kích thước tập tin thực thi (*.EXE) nếu gọi hàm inline có đoạn mã thực thi lớn và nhiều lần
- => chỉ nên sử dụng hàm inline khi cần tối ưu thời gian thực thi

Con trỏ hàm

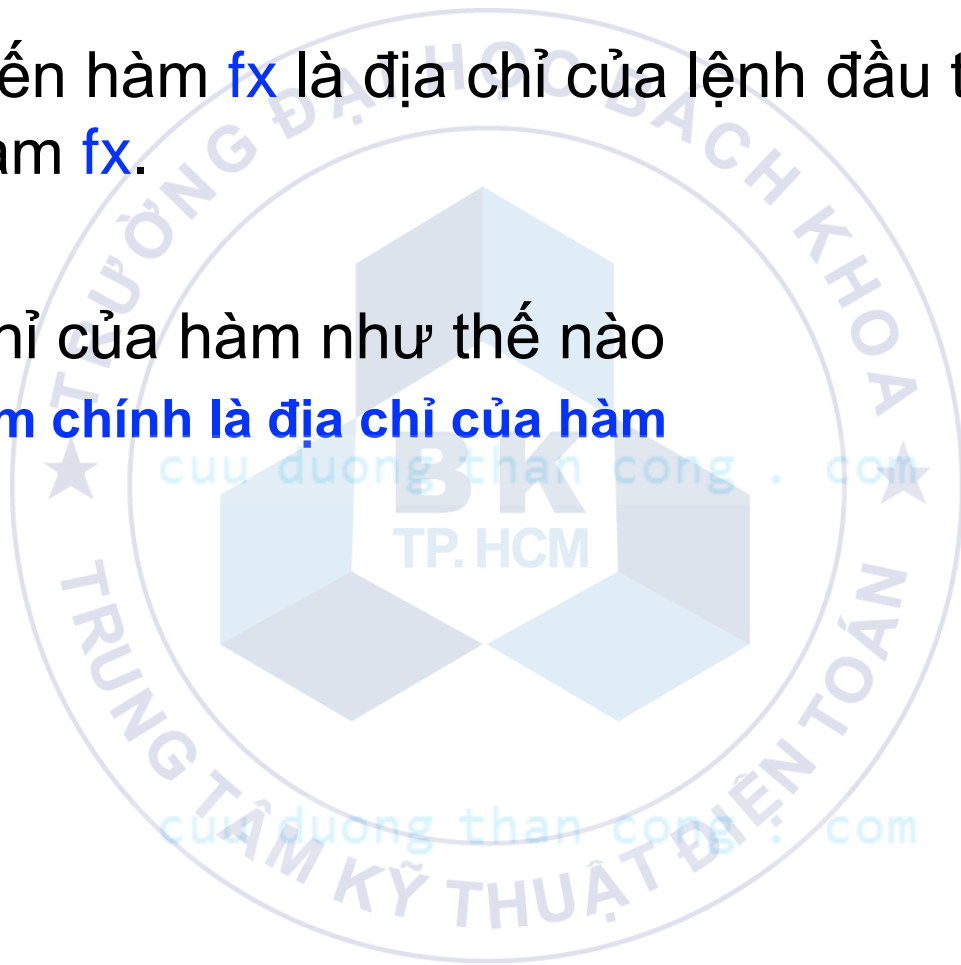
- Con trỏ hàm là gì?
- Ứng dụng của con trỏ hàm
- Khai báo con trỏ hàm
- Gọi hàm thông qua con trỏ
- Định nghĩa kiểu dữ liệu con trỏ hàm
- Truyền con trỏ hàm như thông số của hàm
- Mảng con trỏ hàm



Con trỏ hàm

Con trỏ hàm là gì?

- Con trỏ đến hàm **fx** là địa chỉ của lệnh đầu tiên được thực thi của hàm **fx**.
- Lấy địa chỉ của hàm như thế nào
 - **Tên hàm chính là địa chỉ của hàm**



Con trỏ hàm

Ứng dụng của con trỏ hàm

- Khi không có con trỏ hàm
 - Hàm được gọi thông qua tên hàm trong mã nguồn
 - ➔ Cần biết trước tên hàm tại thời điểm biên dịch
- Khi dùng con trỏ hàm
 - Có thể gọi hàm qua con trỏ đến hàm
 - ➔ Không cần biết trước tên hàm tại thời điểm biên dịch
 - ➔ Chỉ cần biết địa chỉ hàm tại thời điểm thực thi và gọi nó
 - ➔ Chương trình uyển chuyển hơn

Con trỏ hàm

Ứng dụng của con trỏ hàm

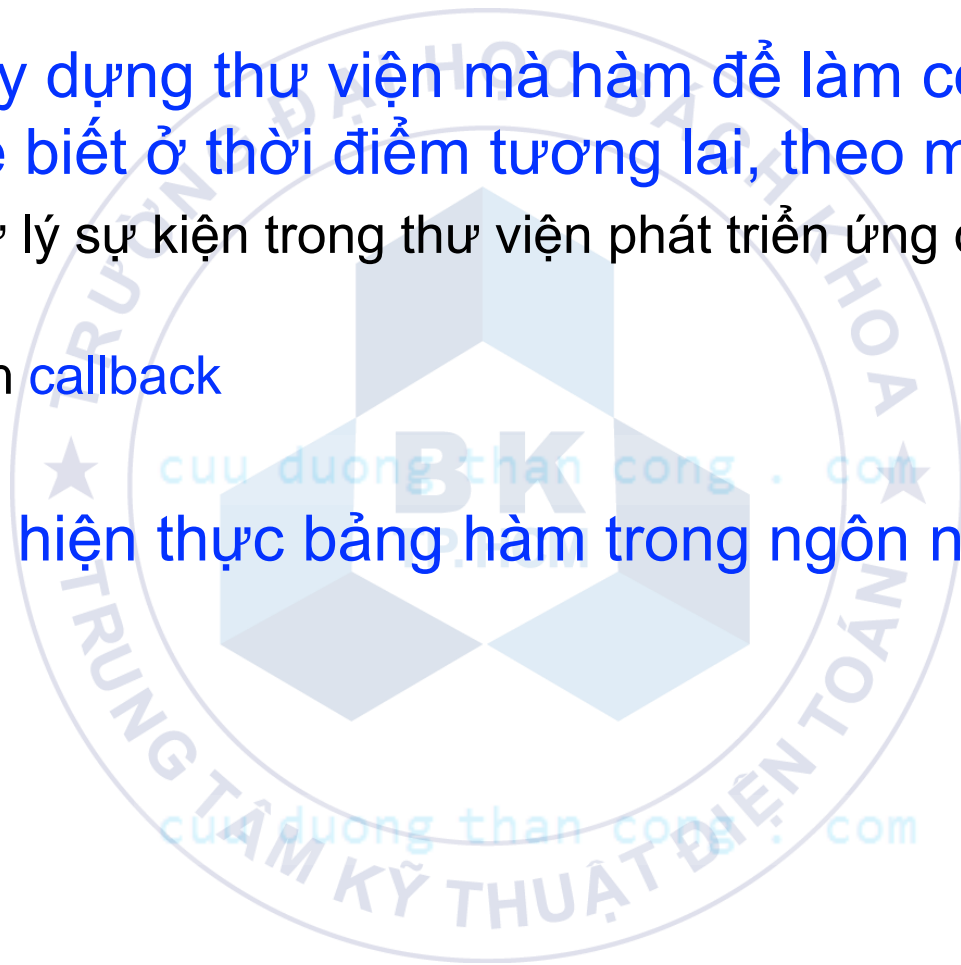
■ Ví dụ: bài toán vẽ đồ thị

- Hàm vẽ chỉ cần biết: khi cho x thì nó lấy được giá trị y của hàm số nào đó, chưa cần biết tên lẫn cách tính tại thời điểm biên dịch
- Chương trình xây dựng bảng hàm (mảng các địa chỉ hàm), và có thể gọi hàm thông qua địa chỉ để biết y cho x .
- Thậm chí bảng hàm này có thể thêm vào và lấy ra tại thời điểm thực thi
 - Khi chương trình đang chạy, người dùng chọn thư viện có tên hàm đánh giá (tính y khi biết x), cho biết tên hàm (chuỗi)
 - => Chương trình có thể vẽ đồ thị các hàm mà cách tính chỉ biết tại thời điểm chạy chương trình

Con trỏ hàm

Ứng dụng của con trỏ hàm

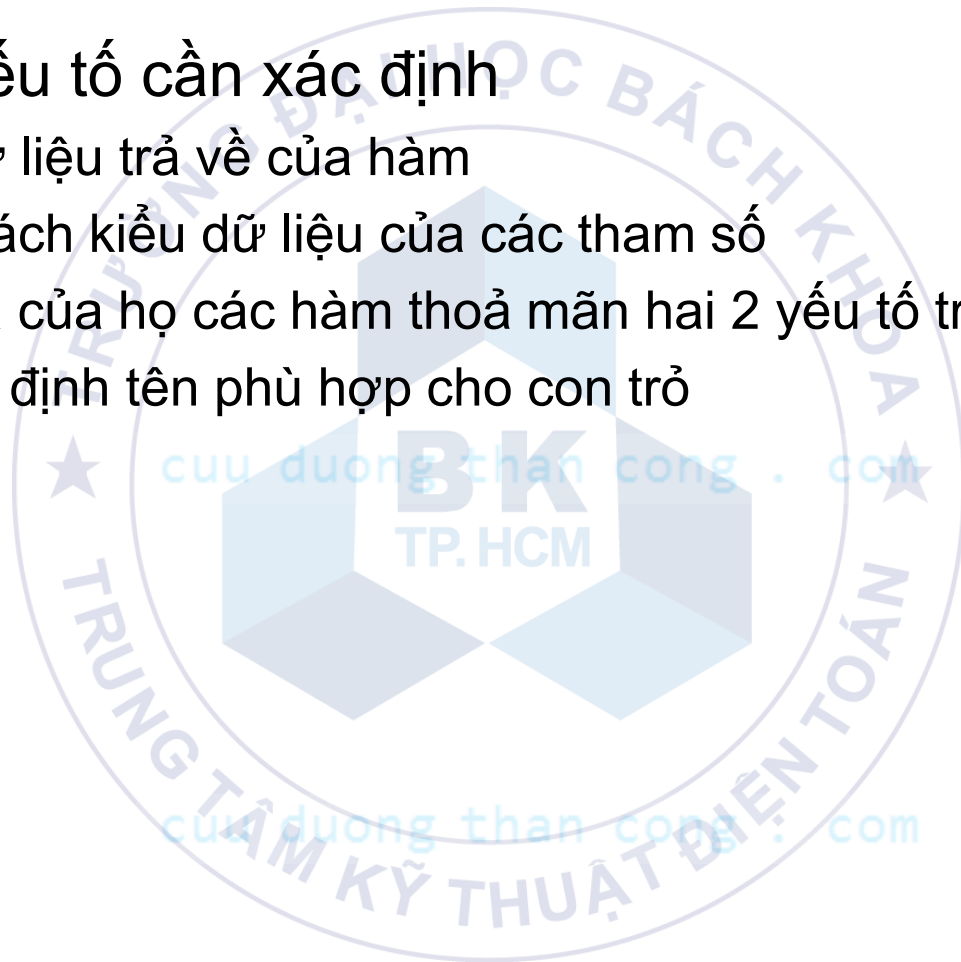
- Ví dụ: Xây dựng thư viện mà hàm để làm công việc nào đó chỉ có thể biết ở thời điểm tương lai, theo mỗi dự án.
 - Hàm xử lý sự kiện trong thư viện phát triển ứng dụng cơ giao diện đồ hoạ
 - Hàm `callback`
- Ví dụ: Để hiện thực bảng hàm trong ngôn ngữ C++



Con trỏ hàm

Khai báo con trỏ hàm

- Những yếu tố cần xác định
 - Kiểu dữ liệu trả về của hàm
 - Danh sách kiểu dữ liệu của các tham số
 - Ý nghĩa của họ các hàm thoả mãn hai 2 yếu tố trên
 - Xác định tên phù hợp cho con trỏ



Con trỏ hàm

Khai báo con trỏ hàm – ví dụ

```
typedef struct{
    char code[5];
    char name[20];
    float gpa;
} Student;
```

Student:

Kiểu dữ liệu chứa thông tin sinh viên

```
void print_one_row(Student student);
```

print_one_row:

Một hàm trả về void, đầu vào là một Student

```
void (*print_ptr1)(Student);
```

```
void (*print_ptr2)(Student) = NULL;
```

```
void (*print_ptr3)(Student) = print_one_row;
```

print_ptr1, print_ptr2, print_ptr3:

- là các biến con trỏ hàm (là các biến chứa địa chỉ của hàm). Các hàm có thể gán cho nó là (không cần quan tâm tên)
- Trả về void, không trả về
- Một thông số đầu vào có kiểu là Student

Con trỏ hàm

Gọi hàm qua con trỏ

Mã nguồn thực toàn bộ

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

```
typedef struct{
    char code[5];
    char name[20];
    float gpa;
} Student;
```

```
void print_one_row(Student student);
void print_one_row(Student student){
    printf("%-6s%-20s%-4.1f\n",
        student.code,
        student.name,
        student.gpa);
}
```

Hàm in thông tin sinh viên trên một dòng



cuu duong than cong . com

cuu duong than cong . com

Con trỏ hàm

Gọi hàm qua con trỏ

Hàm main

```
int main(){
    void (*print_ptr3)(Student) = print_one_row;

    Student s = {"001", "Nguyen Thanh An", 9.8f};

    print_ptr3(s);

    (*print_ptr3)(s);

    system("pause");
    return EXIT_SUCCESS;
}
```

Lời gọi hàm qua con trỏ:
địa chỉ hàm ở vị trí tên hàm
thông thường trong lời gọi
hàm thông thường

Lời gọi hàm qua con trỏ:
dùng đến toán tử *

Con trỏ hàm

Gọi hàm qua con trỏ - kết quả


Hàm main

```
int main(){
    void (*print_ptr3)(Student) = print_one_row;

    Student s = {"001", "Nguyen Thanh An", 9.8f};

    print_ptr3(s);

    (*print_ptr3)(s);
}
```



```
\\psf\Home\Documents\DONGTHAP\Projects\CProgramming\Debug\Function.exe
001 Nguyen Thanh An 9.8
001 Nguyen Thanh An 9.8
```

Con trỏ hàm

Định nghĩa kiểu dữ liệu con trỏ hàm

```
typedef struct{
    char code[5];
    char name[20];
    float gpa;
} Student;

typedef void (*PrintStudentPtr)(Student);
```

Từ khoá **typedef** giúp rút ngắn việc khai báo biến

PrintStudentPtr: có thể được sử dụng như tên kiểu mới

Nó là họ các hàm có kiểu trả về void, chấp nhận 1 thông số đầu vào có kiểu **Student**

Con trỏ hàm

Định nghĩa kiểu dữ liệu con trỏ hàm

```
void (*print_ptr3)(Student) = print_one_row;
```

```
PrintStudentPtr print_ptr = print_one_row;
```

print_ptr3, print_ptr:

là tên biến, được khởi động là địa chỉ của hàm
print_one_row

Nhờ có PrintStudentPtr

Khai báo biến print_ptr như khai báo biến có kiểu dữ liệu nào khác, dễ hiểu và ngắn hơn.

Con trỏ hàm

Định nghĩa kiểu dữ liệu con trỏ hàm

Ví dụ hoàn chỉnh

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

typedef struct{
    char code[5];
    char name[20];
    float gpa;
} Student;

typedef void (*PrintStudentPtr)(Student);

void print_one_row(Student student);
void print_one_row(Student student){
    printf("%-6s%-20s%-4.1f\n",
        student.code,
        student.name,
        student.gpa);
}
```

Con trỏ hàm

Định nghĩa kiểu dữ liệu con trỏ hàm

Ví dụ hoàn chỉnh

```
int main(){  
    void (*print_ptr3)(Student) = print_one_row;  
  
    PrintStudentPtr print_ptr = print_one_row;  
  
    Student s = {"001", "Nguyen Thanh An", 9.8f};  
    print_ptr3(s);  
    (*print_ptr3)(s);  
    print_ptr(s);  
    (*print_ptr)(s);  
}
```

In ra 4 hàng

\\pst\Home\Documents\DONGTHAP\Projects\CProgramming\Debug\Function.exe

001	Nguyen Thanh An	9.8
001	Nguyen Thanh An	9.8
001	Nguyen Thanh An	9.8
001	Nguyen Thanh An	9.8

Con trỏ hàm

Truyền con trỏ như tham số - khai báo thông số

```
void print_list1(Student *list, int size,  
PrintStudentPtr print_ptr);
```

```
void print_list2(Student *list, int size,  
void (*print_ptr)(Student));
```

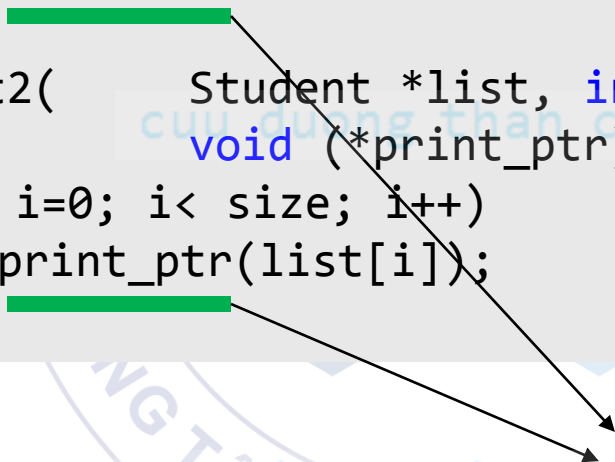
Không dùng tên kiểu

Dùng tên kiểu đã định nghĩa
bằng typedef

Con trỏ hàm

Truyền con trỏ như tham số - dùng thông số

```
void print_list1(    Student *list, int size,
                    PrintStudentPtr print_ptr){
    for(int i=0; i< size; i++)
        print_ptr(list[i]);
}
void print_list2(    Student *list, int size,
                    void (*print_ptr)(Student)){
    for(int i=0; i< size; i++)
        print_ptr(list[i]);
}
```



Đều gọi hàm theo con trỏ hàm như thông thường

Con trỏ hàm

Truyền con trỏ như tham số - dùng thông số

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

typedef struct{
    char code[5];
    char name[20];
    float gpa;
} Student;

typedef void (*PrintStudentPtr)(Student);

void print_one_row(Student student);
void print_list1(Student *list, int size,
                 PrintStudentPtr print_ptr);
void print_list2(Student *list, int size,
                 void (*print_ptr)(Student));
```

Con trỏ hàm

Truyền con trỏ như tham số - dùng thông số

```
int main(){
    Student aList[] = {
        {"001", "Nguyen Thanh An", 9.8f},
        {"002", "Tran Van Binh", 7.5f},
        {"003", "Le Tan Cong", 6.7f},
    };
    PrintStudentPtr func_ptr = print_one_row;
    print_list1(aList, 3, func_ptr);
    printf("\n");
    print_list2(aList, 3, func_ptr);

    printf("\n\n");
    system("pause");
    return EXIT_SUCCESS;
}
```

Con trỏ hàm

Truyền con trỏ như tham số - dùng thông số

Kết quả xuất ra màn hình

```
\\psf\Home\Documents\DONGTHAP\Projects\CProgramming\Debug\Function.exe
001 Nguyen Thanh An 9.8
002 Tran Van Binh 7.5
003 Le Tan Cong 6.7

001 Nguyen Thanh An 9.8
002 Tran Van Binh 7.5
003 Le Tan Cong 6.7
```


Con trỏ hàm

Mảng con trỏ hàm

```
PrintStudentPtr func_arr_ptr[10];
```

```
void (*print_ptr[10])(Student);
```

Khi sử dụng tên kiểu `PrintStudentPtr`

Khi không sử dụng tên kiểu `PrintStudentPtr`

`func_arr_ptr` và `print_ptr` là mảng của 10 con trỏ hàm.

Sử dụng mảng này như mảng của các kiểu dữ liệu khác

Hàm đệ quy

- Hàm đệ quy là hàm gọi lại chính nó
 - Trực tiếp:
 - `foo()` gọi `foo()` trực tiếp trong thân hàm `foo()`
 - Gán tiếp:
 - `foo()` gọi `bar`, `bar` gọi `foo()`; hoặc qua nhiều trung gian hàm khác



Hàm đệ quy

■ Ví dụ

- Chương trình tính tổng của $1+2+3+ \dots + N$
- Hàm `tong(N)` gọi lại `tong(N-1)`

```
int tong(int N){  
    int ket_qua;  
    if(N <=0) ket_qua = 0;  
    else ket_qua = N + tong(N-1);  
  
    return ket_qua;  
}
```

Hàm đệ quy

■ Ví dụ

- Chương trình tính giai thừa: $1 \times 2 \times 3 \times \dots \times N$
- Hàm `giai_thua(N)` gọi lại `giai_thua(N-1)`

```
long long giai_thua(int N){  
    int ket_qua;  
    if(N <=1) ket_qua = 1;  
    else ket_qua = N*giai_thua(N-1);  
  
    return ket_qua;  
}
```

Hàm đệ quy

- Yêu cầu cần thiết của một hàm đệ quy:
 - Điều kiện dừng quá trình gọi đệ quy
 - Ví dụ:

```
int tong(int N){  
    ...  
    if(N <=0) ket_qua = 0;  
    ...  
}
```

Dừng quá trình gọi đệ quy

Hàm đệ quy

- Yêu cầu cần thiết của một hàm đệ quy:
 - Điều kiện dừng quá trình gọi đệ quy
 - Ví dụ:

```
long long giai_thua(int N){  
    ...  
    if(N <=1) ket_qua = 1;  
    ...  
}
```



Dừng quá trình gọi đệ quy

Hàm đệ quy

- Yêu cầu cần thiết của một hàm đệ quy:
 - Điều kiện dừng quá trình gọi đệ quy
 - Có lời gọi hàm đệ quy
 - Hàm tong(N) gọi lại tong(N-1)
 - Hàm giai_thua(N) gọi lại giai_thua(N-1)



Hàm đệ quy

- Giải bài toán bằng đệ quy
 - Lời giải của bài toán có kích thước N được tổng hợp từ lời giải của các bài toán có kích thước nhỏ hơn.
 - Ví dụ:
 - Lời giải cho bài toán tìm tổng N phần tử
 - Giả sử ta biết tổng của $(N-1)$ phần tử
 - Vậy, lời giải của N phần tử được tổng hợp như thế nào từ kết quả trên?

Hàm đệ quy

- Giải bài toán bằng đệ quy
 - Lời giải của bài toán có kích thước N được tổng hợp từ lời giải của các bài toán có kích thước nhỏ hơn.
 - Ví dụ:
 - Lời giải cho giai thừa N
 - Giả sử ta biết lời giải cho giai thừa của $(N-1)$
 - Lời giải cho giai thừa N được tổng hợp như thế nào từ lời giải trên?

Hàm đệ quy

- Giải bài toán bằng đệ quy
 - Lời giải của bài toán có kích thước N được tổng hợp từ lời giải của các bài toán có kích thước nhỏ hơn.
 - Ví dụ:
 - Tìm số thứ N trong dãy số Fibonacci
 - $F(1) = 1$
 - $F(2) = 1$
 - $N > 2: F(N) = F(N-1) + F(N-2)$
 - Lời giải
 - Giả sử ta có $F(N-1)$ và $F(N-2)$
 - Lời giải cho $F(N)$ được tổng hợp như thế nào từ các kết quả trên?

Hàm đệ quy

- Giải bài toán bằng đệ quy
 - Ví dụ:
 - Bài toán tháp Hanoi
 - Di chuyển chồng đĩa từ Cột đầu sang cột cuối, dùng cột giữa làm trung gian
 - Luôn đảm bảo trật tự kích thước các đĩa

