Hàm &
Kỹ thuật tổ chức chương trình
Phần a: Hàm trong nhiều tập tin mã nguồn

Nhập môn lập trình

Trình bày: Nguyễn Sơn Hoàng Quốc
Email: nshquoc@fit.hcmus.edu.vn

Lập trình đơn thể

- Chương trình với một tập tin mã nguồn chỉ phù hợp với các chương trình nhỏ.
- Khi đặt một tập các hàm có mục đích tổng quát vào một tập tin riêng, ta có thể sử dụng lại các hàm này ở các chương trình khác.
- Khi viết chương trình gồm nhiều tập tin mã nguồn, mỗi tập tin mã nguồn được gọi là một đơn thể (module). Cách lập trình như vậy gọi là lập trình đơn thể (modular programming), có liên quan rất gần với lập trình cấu trúc.

Tổ chức mã nguồn nhiều tập tin

- Mỗi chương trình C chỉ có duy nhất một hàm main().
- Đơn thể chứa hàm main() được gọi là đơn thể chính, các đơn thể khác được gọi là đơn thể phụ.
- Một tập tin tiêu đề riêng rẽ thường được đi kèm với mỗi đơn thể phụ.

mymath.h và mymath.c

```
1. /* mymath.h: header file for mymath.c */
2. double sqrt3(double x);
3. double sqrtN(double x);
4. /* end of mymath.h */
```

```
1. /* mymath.c: module containing math functions */
2. #include "mymath.h"
3. double sqrt3(double x) { /* statements */ }
4. double sqrtN(double x) { /* statements */ }
5. /* end of mymath.c */
```

sample.c (đơn thể chính)

```
1. #include <stdio.h>
2. #include "mymath.h"
3. void main() {
4.     int x;
5.     printf("Enter an integer value: ");
6.     scanf("%d", &x);
7.     printf("The 3rd root of %d is %.lf\n",
                                x, sqrt3((double)x);
8.     /* other statements here… */
9. }
```

Phạm vi của hàm và biến toàn cục

- Hàm và biến toàn cục (hay biến ngoài) không tự động được thấy trong các đơn thể khác.
- Khai báo để các đơn thể khác có thể thấy được hàm hay biến toàn cục trong các đơn thể khác:
  - Hàm: sử dụng chỉ thị #include (ví dụ trước)
  - Biến toàn cục: sử dụng từ khóa extern

Ví dụ khai báo biến toàn cục

```
1. /* main module: sample.c */
2. int x = 99, y; /* the compiler automatically initializes y to 0 */
3. void main() {/* statements */ }
```

```
1. /* secondary module: mod1.c */
2. extern int x, y;
3. void func1() { /* statements */ }
```

```
1. /* secondary module: mod2.c */
2. extern int x;
3. void func2() { /* statements */ }
```