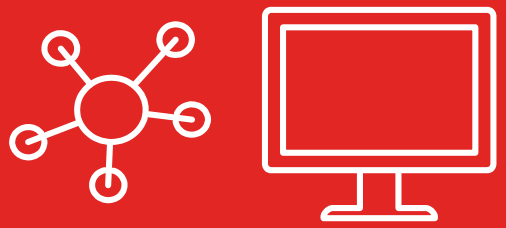
Trịnh Thành Trung (ThS)
trungtt@soict.hust.edu.vn

# Bài 6
# PHONG CÁCH LẬP TRÌNH

# Nội dung



1. Tổng quan
2. Cấu trúc mã nguồn
3. Đặt tên và chú thích

# 1. Tổng quan

Programming style

# What is this

```
#include <stdio.h>
_(__,___,____){___/__<=1?_(__,___+1,____):!(___%__)
?_(__,___+1,0):___%__==___/
__&&!____?(printf("%d\t",___/__),_(__,___+1,0))
:___%__>1&&___%__<___/__?_(__,1+___,____
+!(___/__%(___%__))):___<__*__?_(__,___+1,____):0;}
main(){_(100,0,0);
}
```


*Ai là người đọc mã nguồn này?*

# Tại sao cần có phong cách lập trình tốt

- Lỗi thường xảy ra do sự nhầm lẫn của lập trình viên
  - *Biến này được dùng làm gì?*
  - *Hàm này được gọi như thế nào?*
- Mã nguồn tốt ~ mã nguồn dễ đọc

# Mã nguồn
# dễ đọc

Làm thế nào để mã nguồn dễ đọc?

- ***Cấu trúc chương trình rõ ràng, dễ hiểu, khúc triết***
- ***Sử dụng thành ngữ phổ biến***
- ***Chọn tên phù hợp, gợi nhớ***
- ***Viết chú thích rõ ràng***
- ***Sử dụng module***

# Một số quy tắc

## Nhất quán

- Tuân thủ quy tắc đặt tên trong toàn bộ chương trình
- Nhất quán trong việc dùng các biến cục bộ.

## Khúc triết

- Mỗi chương trình con phải có một nhiệm vụ rõ ràng.
- Đủ ngắn để có thể nắm bắt được
- Số tham số của chương trình con là tối thiểu (dưới 6)

# Một số quy tắc

Rõ ràng

- Chú thích rõ ràng, vd. đầu mỗi chương trình con

Bao đóng

- Hàm chỉ nên tác động tới duy nhất 1 giá trị - giá trị trả về của hàm
- Không nên thay đổi giá trị của biến chạy trong thân của vòng lặp, ví dụ

```
for(i=1;i<=10;i++) i++;
```

# 2.
# Cấu trúc mã nguồn

Structure

# Khoảng trắng
# Spacing

- Sử dụng khoảng trắng dễ đọc và nhất quán
  - *VD: Gán mỗi phần tử mảng a[j] = j.*
  - *Bad code*

```
for (j=0;j<100;j++) a[j]=j;
```

  - *Good code*

```
for (j=0; j<100; j++)
   a[j] = j;
```

  - *Thường có thể dựa vào auto-indenting, tính năng trong trình soạn thảo*

# Cách lề
# Indentation

- Cách lề hợp lý -> tránh nhầm lẫn về cấu trúc
  - *VD:*

```
if (month == FEB) {
   if (year % 4 == 0)
      if (day > 29)
         legal = FALSE;
   else
      if (day > 28)
         legal = FALSE;
}
```

SAI
(nếu *day>29?)*

```
if (month == FEB) {
   if (year % 4 == 0) {
      if (day > 29)
         legal = FALSE;
   }
   else {
      if (day > 28)
         legal = FALSE;
   }
}
```

ĐÚNG

# Cách lề
# Indentation

- Use "else-if" cho cấu trúc đa lựa chọn

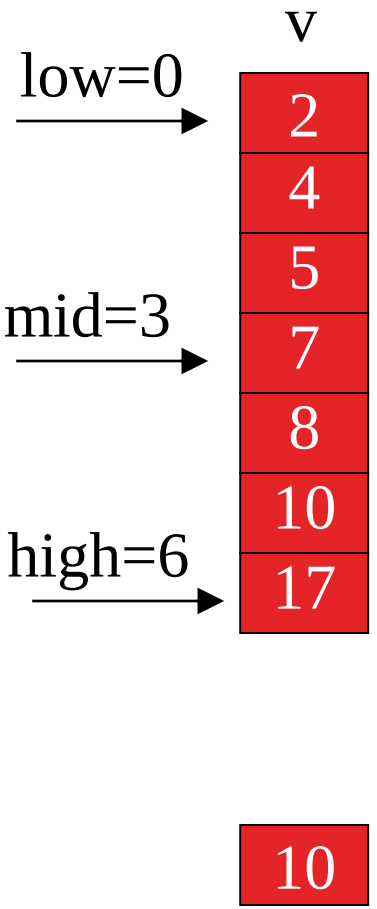
*VD: So sánh trong tìm kiếm nhị phân*

- *Bad code*

```
if (x < v[mid])
   high = mid - 1;
else
   if (x > v[mid])
      low = mid + 1;
   else
      return mid;
```

- *Good code*

```
if (x < v[mid])
   high = mid - 1;
else if (x > v[mid])
   low = mid + 1;
else
   return mid;
```



# Cách đoạn
# Paragraph

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

int main(void)

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0 if successful. */

{
   const double PI = 3.14159;
   int radius;
   int diam;
   double circum;

   printf("Enter the circle's radius:\n");
   if (scanf("%d", &radius) != 1)
   {
      fprintf(stderr, "Error: Not a number\n");
      exit(EXIT_FAILURE);  /* or:  return EXIT_FAILURE; */
   }

   diam = 2 * radius;
   circum = PI * (double)diam;

   printf("A circle with radius %d has diameter %d\n",  radius, diam);
   printf("and circumference %f.\n", circum);

   return 0;
}
```

# Biểu thức
# Expressions

- Dùng các biểu thức dạng nguyên bản
  - *VD: Kiểm tra nếu* ***n*** *thỏa mãn* ***j < n < k***
  - *Bad code*

```
if (!(n >= k) && !(n <= j))
```

  - *Good code*

```
if ((j < n) && (n < k))
```

  - *Biểu thức điều kiện có thể đọc như cách thức bạn viết thông thường*
    - Đừng viết biểu thức điều kiện theo kiểu mà bạn không bao giờ sử dụng

# Biểu thức
# Expressions

- Dùng ( ) để tránh nhầm lẫn
  - *VD: Kiểm tra nếu* ***n*** *thỏa mãn* ***j < n < k***
  - *Moderately bad code*

```
if (j < n && n < k)
```

  - *Moderately better code*

```
if ((j < n) && (n < k))
```

# Biểu thức
# Expressions

- Dùng ( ) để tránh nhầm lẫn
  - *VD: đọc và in các ký tự cho đến cuối tệp.*
  - *Wrong code*

```
while (c = getchar() != EOF)
    putchar(c);
```

  - *Right code*

```
while ((c = getchar()) != EOF)
    putchar(c);
```

  - *Nên nhóm các nhóm một cách rõ ràng*
    - Toán tử Logic (“!=“) có độ ưu tiên cao hơn toán tử gán (“=“)

# Biểu thức
# Expressions

- Đơn giản hóa các biểu thức phức tạp
  - *VD: Xác định các ký tự tương ứng với các tháng của năm*
  - *Bad code*

```
if ((c == 'J') || (c == 'F') || (c ==
'M') || (c == 'A') || (c == 'S') || (c
== 'O') || (c == 'N') || (c == 'D'))
```

  - *Good code*

```
if ((c == 'J') || (c == 'F') ||
    (c == 'M') || (c == 'A') ||
    (c == 'S') || (c == 'O') ||
    (c == 'N') || (c == 'D'))
```

  - *Nên xắp xếp các cơ cấu song song*

# 3.
# Đặt tên và chú thích

Structure

# Đăt tên
# Naming

- Dùng tên gợi nhớ, có tính miêu tả cho các biến và hàm
  - *VD : hovaten , **`CONTROL, CAPACITY`***
- Dùng tên nhất quán cho các biến cục bộ
  - *VD : **`i`** (not **`arrayIndex`**) cho biến chạy vòng lặp*
- Dùng chữ hoa, chữ thường nhất quán
  - *VD : Buffer_Insert (Tên hàm)*
    CAPACITY (hằng số)
    buf (biến cục bộ)
- Dùng phong cách nhất quánkhi ghép từ
  - *VD : **`frontsize`**, **`frontSize, front_size`***
- Dùng động từ cho tên hàm
  - *VD : DocSoLieu **`()`**, InKq**`()`**, **`Check_Octal()`**, ...*

# Chú thích
# Comments

- Làm chủ ngôn ngữ
  - *Hãy để chương trình tự diễn tả bản thân*
  - *Rồi…*
- Viết chú thích để thêm thông tin

```
i++;   /* add one to i */
```

- Chú thích các đoạn (“paragraphs”) code, đừng chú thích từng dòng
  - *Vd : “Sort array in ascending order”*
- Chú thích dữ liệu tổng thể
  - *Global variables, structure type definitions, ….*
- Viết chú thích tương ứng với code
  - *Và thay đổi khi bản thân code thay đổi.*

# Ví dụ

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

int main(void)

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0 if successful. */

{
   const double PI = 3.14159;
   int radius;
   int diam;
   double circum;

   /* Read the circle’s radius. */
   printf("Enter the circle's radius:\n");
   if (scanf("%d", &radius) != 1)
   {
      fprintf(stderr, "Error: Not a number\n");
      exit(EXIT_FAILURE);  /* or:  return EXIT_FAILURE; */
   }
…
```

# Ví dụ

```
   /* Compute the diameter and circumference. */
   diam = 2 * radius;
   circum = PI * (double)diam;

   /* Print the results. */
   printf("A circle with radius %d has diameter %d\n",
      radius, diam);
   printf("and circumference %f.\n", circum);

   return 0;
}
```

# Chú thích hàm

- Mô tả những gì cần thiết để gọi hàm 1 cách chính xác
  - *Mô tả Hàm làm gì, chứ không phải làm như thế nào*
  - *Bản thân Code phải rõ ràng, dễ hiểu để biết cách nó làm việc…*
  - *Nếu không, hãy viết chú thích bên trong định nghĩa hàm*
- Mô tả đầu vào: Tham số truyền vào, đọc file gì, biến tổng thể được dùng
- Mô tả đầu ra: giá trị trả về, tham số truyền ra, ghi ra files gì, các biến tổng thể nó tác động tới

# Chú thích hàm

- Bad comment

```
/* decomment.c */

int main(void) {

/* Đọc 1 ký tự.  Dựa trên ký tự ấy và trạng thái DFA
hiện thời, gọi hàm xử lý trạng thái tương ứng.  Lặp
cho đến hết tệp end-of-file. */

   …
}
```

  - *Giải thích hàm làm **như thế nào***

# Chú thích hàm

- Good function comment

```
/* decomment.c */

int main(void) {

/* Đọc 1 chương trình C qua stdin.
   Ghi ra stdout với mỗi chú thích thay bằng 1 dấu
cách.
   Trả về 0 nếu thành công, EXIT_FAILURE nếu không
thành công. */


   …
}
```

  - *Giải thích hàm **làm gì***

# 4.
# Các quy tắc chung

Good programming style

*từ cuốn*

***The Elements of Programming Style***

Brian Kernighan and P. J. Plauger

McGraw-Hill Book Company, New York, 1974

“

1. *Write clearly / don't be too clever – Viết rõ ràng – đừng quá thông minh (kỳ bí)*
2. *Say what you mean, simply and directly – Trình bày vấn đề 1 cách đơn giản, trực tiếp*
3. *Use library functions whenever feasible. – Sử dụng thư viện mọi khi có thể*
4. *Avoid too many temporary variables – Tránh dùng nhiều biến trung gian*
5. *Write clearly / don't sacrifice clarity for efficiency – Viết rõ ràng / đừng hy sinh sự rõ ràng cho hiệu quả*
6. *Let the machine do the dirty work – Hãy để máy tính làm những việc nặng nhọc của nó. (tính toán…)*

“

7. *Replace repetitive expressions by calls to common functions. – Hãy thay những biểu thức lặp đi lặp lại bằng cách gọi các hàm*
8. *Parenthesize to avoid ambiguity. – Dùng () để tránh rắc rối*
9. *Choose variable names that won't be confused – Chọn tên biến sao cho tránh được lẫn lộn*
10. *Avoid unnecessary branches. – Tránh các nhánh không cần thiết*
11. *If a logical expression is hard to understand, try transforming it – Nếu 1 biểu thức logic khó hiểu, cố gắng chuyển đổi cho đơn giản*

“

12. *Choose a data representation that makes the program simple – Hãy lựa chọn cấu trúc dữ liệu để chương trình thành đơn giản*
13. *Write first in easy-to-understand pseudo language; then translate into whatever language you have to use. – Trước tiên hãy viết chương trình bằng giả ngữ dễ hiểu, rồi hãy chuyển sang ngôn ngữ cần thiết.*
14. *Modularize. Use procedures and functions. – Mô đul hóa. Dùng các hàm và thủ tục*
15. *Avoid gotos completely if you can keep the program readable. – Tránh hoàn toàn việc dùng goto*

“

16. *Don't patch bad code, rewrite it. – Không chắp vá mã xấu – Viết lại đoạn code đó*
17. *Write and test a big program in small pieces. – Viết và kiểm tra 1 chương trình lớn thành từng chương trình con*
18. *Use recursive procedures for recursively-defined data structures. – Hãy dùng các thủ tục đệ quy cho các cấu trúc dữ liệu đệ quy*
19. *Test input for plausibility and validity. – Kiểm tra đầu vào để đảm bảo tính chính xác và hợp lệ*
20. *Make sure input doesn't violate the limits of the program. – Hãy đảm bảo đầu vào không quá giới hạn cho phép của chương trình*

“

21. *Terminate input by end-of-file marker, not by count. – Hãy kết thúc dòng nhập bằng ký hiệu EOF, không dùng phép đếm*
22. *Identify bad input; recover if possible. – Xác định đầu vào xấu, khôi phục nếu có thể*
23. *Make input easy to prepare and output self-explanatory. – Hãy làm cho đầu vào đơn giản, dễ chuẩn bị và đầu ra dễ hiểu*
24. *Use uniform input formats. – Hãy dùng các đầu vào theo các định dạng chuẩn.*

“

25. *Make sure all variable are initialized before use.- Hãy đảm bảo các biến được khởi tạo trước khi sử dụng*
26. *Test programs at their boundary values. – Hãy kiểm tra chương trình tại các cận*
27. *Check some answers by hand. – Kiểm tra 1 số câu trả lời bằng tay*
28. *10.0 times 0.1 is hardly ever 1.0. – 10 nhân 0.1 không chắc đã = 1.0*
29. *7/8 is zero while 7.0/8.0 is not zero. 7/8 =0 nhưng 7.0/8.0 ≠ 0*
30. *Make it right before you make it faster. – Hãy làm cho chương trình chạy đúng, trước khi làm nó chạy nhanh*

“

30. *Make it clear before you make it faster. – Hãy viết code rõ ràng, trước khi làm nó chạy nhanh*
31. *Let your compiler do the simple optimizations. – Hãy để trình dịch thực hiện các việc tôi ưu hóa đơn giản*
32. *Don't strain to re-use code; reorganize instead. – Đừng cố tái sử dụng mã, thay vì vậy, hãy tổ chức lại mã*
33. *Make sure special cases are truly special. – Hãy đảm bảo các trường hợp đặc biệt là thực sự đặc biệt*
34. *Keep it simple to make it faster. – Hãy giữ nó đơn giản để làm cho nó nhanh hơn*

“

35. *Make sure comments and code agree. – Chú thích phải rõ ràng, sát code*
36. *Don't comment bad code | rewrite it. – Đừng chú thích những đoạn mã xấu, hãy viết lại*
37. *Use variable names that mean something. – Hãy dùng các tên biến có nghĩa*
38. *Format a program to help the reader understand it.- Hãy định dạng chương trình để giúp người đọc hiểu đc chương trình*
39. *Don't over-comment. – Đừng chú thích quá nhiều*

# Thanks!

## Any questions?

*Email me at trungtt@soict.hust.edu.vn*