Chương 5:
Phong cách lập trình

Nội dung

1. Khái niệm phong cách lập trình
2. Một số quy tắc cơ bản về phong cách lập trình
3. Viết tài liệu chương trình

Khái niệm phong cách lập trình

Tại sao cần phong cách lập trình?

- Ai đọc chương trình của chúng ta?
  - Trình dịch
  - Các lập trình viên khác và… bản thân chúng ta

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s--sph)b=vdot(D,U=vcomb(-1.,P,s-cen)),u=b*b-vdot(U,U)+s-rad*s -
rad,u=u0?sqrt(u):1e31,u=b-u1e-7?b-u:b+u,tmin=u=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s-ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s-cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l--sph)if((e=l -
kl*vdot(N,U=vunit(vcomb(-1.,P,l-cen))))0&&intersect(P,U)==l)color=vcomb(e ,l-
color,color);U=s-color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta* eta*(1-
d*d);return vcomb(s-kt,e0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s-ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s-kd,
color,vcomb(s-kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}
```

This is a working ray tracer! (courtesy of Paul Heckbert)

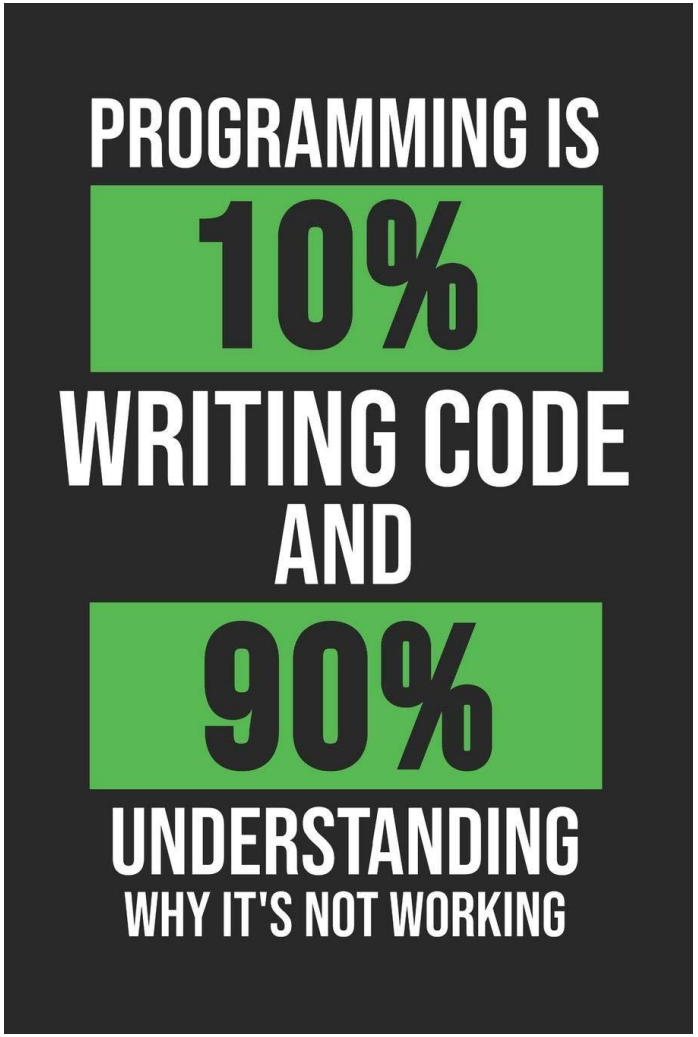
Tại sao cần phong cách lập trình?

- Chương trình thường phải chỉnh sửa vì những lí do:
  - Chưa hoàn thiện hoặc bị lỗi: phải bảo trì
  - Thêm chức năng mới: mở rộng

- Phong cách lập trình có ảnh hưởng rất lớn tới nguồn lực cần thiết để đọc hiểu và chỉnh sửa chương trình.

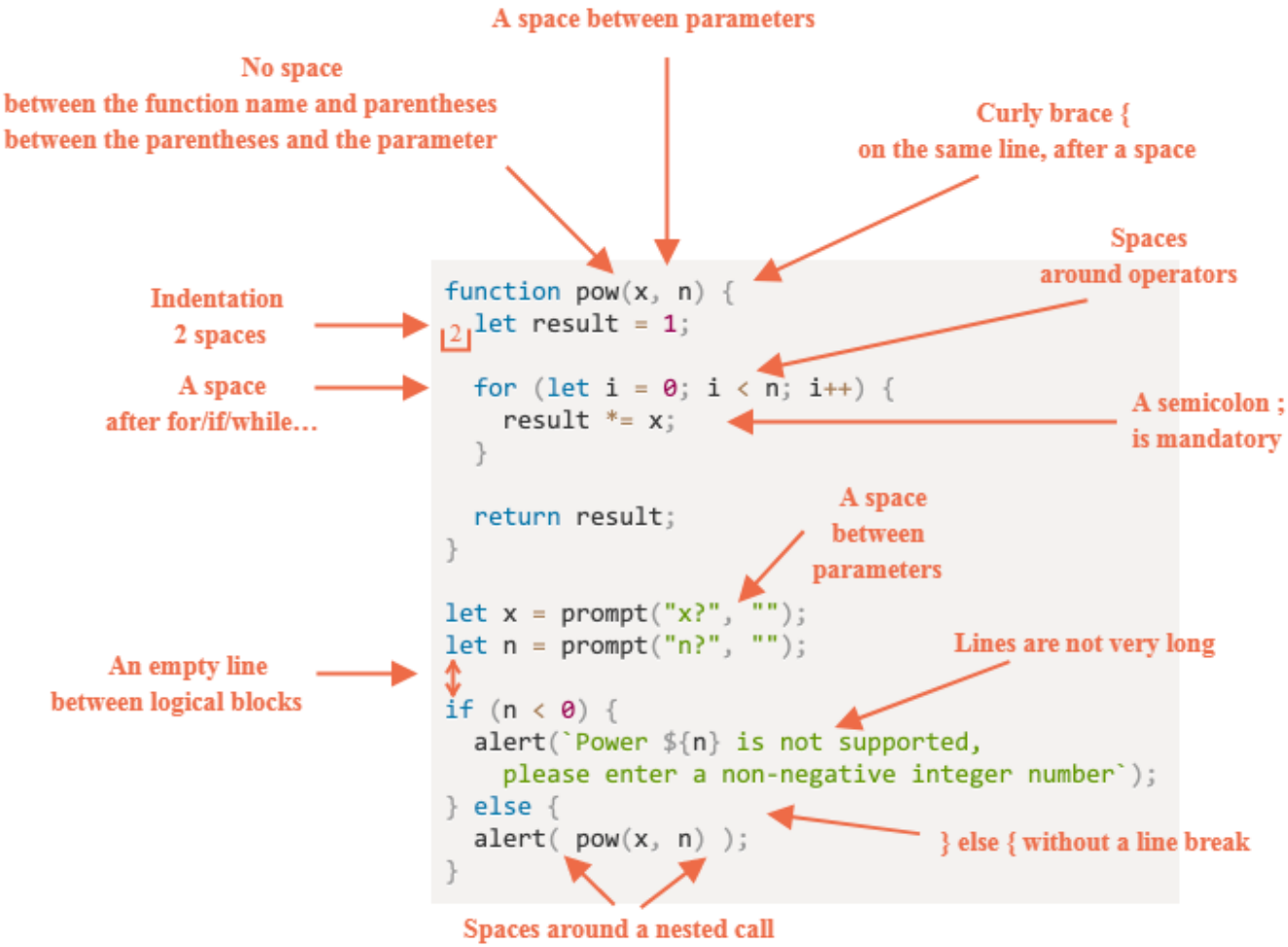
Tại sao cần phong cách lập trình?

- “Programming is an art of telling another human what one wants the computer to do.”
  Donald Knuth.
- “Programming is 10% writing code, and 90% reading code. Reading your own code and reading other code.”
- “Taking that extra time to write a proper description of what you worked on will save huge amounts of time in the future.”
  Tomer Ben Rachel, a full stack developer.



Thế nào là một phong cách lập trình?

- Là một tập hợp các quy tắc và hướng dẫn được sử dụng khi viết mã nguồn chương trình



Chọn phong cách lập trình nào?

- Có nhiều phong cách lập trình khác nhau. Thường mỗi công ty hoặc tổ chức có phong cách lập trình riêng.
- Ví dụ:
  - Google: https://google.github.io/styleguide/cppguide.html
  - Linux kernel: https://slurm.schedmd.com/coding_style.pdf
  - GNU: https://www.gnu.org/prep/standards/standards.html

Một số quy tắc cơ bản

Các quy tắc cơ bản

- Chúng ta sẽ đề cập một số quy tắc đơn giản của một phong cách lập trình tốt:
  - Định dạng (format)
  - Cách đặt tên (naming conventions)
  - Viết đặc tả hàm (specification)
  - Chú thích (comments)

- Chúng ta tập trung chủ yếu vào làm cách nào để viết một chương trình dễ đọc.
- Chúng ta sẽ minh họa các quy tắc bằng cách ví dụ.
- Phong cách lập trình thật có thể bao gồm hàng trăm quy tắc.

Định dạng: thụt đầu dòng và dấu ngoặc

```
int gcd(int a, int b)
        {
  while (a != b)
        {
     if (a > b) a = a - b;
else b = b - a;
        }
return a;}
```

```
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

- Thụt đầu dòng bằng 2 hoặc 4 dấu cách (phải nhất quán!). Tránh dùng 

- Đóng mở ngoặc nhất quán (ví dụ mở ngoặc ở cuối dòng)

Ví dụ thụt đầu dòng

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

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

(else matches “if day > 29”)

Ví dụ thụt đầu dòng

- Use "`else-if`" cho cấu trúc đa lựa chọn
  - Ví dụ: Tìm kiếm nhị phân

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

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

Đây có phải là một phong cách lập trình tốt?

- Lập trình viên Python sử dụng Java

```
public class Permuter                                          {
    private static void permute(int n, char[] a)               {
        if (n == 0)                                            {
            System.out.println(String.valueOf(a))              ;}
        else                                                   {
            for (int i = 0; i <= n; i++)                       {
                permute(n-1, a)                                ;
                swap(a, n % 2 == 0 ? i : 0, n)                 ;}}}
    private static void swap(char[] a, int i, int j) {
        char saved = a[i]                                      ;
        a[i] = a[j]                                            ;
        a[j] = saved                                           ;}}
```

Định dạng: dòng trống và dấu cách

- Dùng dòng trống ngăn các phần khác nhau trong chương trình:
  - Giữa các hàm khác nhau
  - Giữa các phần khác nhau của cùng một hàm (khởi tạo, vòng lặp chính, return…)

- Dùng khoảng cách để chương trình dễ đọc hơn:
  - Trong các biểu thức phức tạp (nhấn mạnh thứ tự ưu tiên các phép toán)
  - Phân tách các phần tử trong một danh sách

Ví dụ dấu cách

```
// Dense code
int numCars=0,time=0;


// Confusing expression
a = b * c+d * 2;

// No space after if/while
while(a!=0) {…}

// Space after function name
x = power (y,2);
```

```
// Spaced declarations
int numCars = 0, time = 0;


// Emphasize precedences
a = b*c + d*2;

// Space after if/while
while (a != 0) {…}

// No space after function name
// but space between parameters
x = power(y, 2);
```

Ví dụ dòng trống

- Dùng dòng trống để chia code thành các phần chính

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

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

int main() {
   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);
   }
…
```

Ví dụ dòng trống

- Dùng dòng trống để chia code thành các phần chính

```
   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;
}
```

Định dạng biểu thức

- Nên dùng các biểu thức dạng nguyên bản
- Ví dụ: Kiểm tra nếu n thỏa mãn j < n < k

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

```
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

Định dạng biểu thức

- Dùng () để tránh nhầm lẫn
- Ví dụ: Kiểm tra nếu n thỏa mãn j < n < k

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

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

- Nên nhóm các nhóm một cách rõ ràng
- Toán tử quan hệ (ví dụ “>”) có độ ưu tiên cao hơn các toán tử logic (ví dụ “&&”), nhưng không phải ai cũng nhớ điều đó.

Định dạng biểu thức

- Dùng () để tránh nhầm lẫn
- Ví dụ: đọc và in các ký tự cho đến cuối tệp.

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

```
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 (“=“)

Định dạng biểu thức

- Đơn giản hóa các biểu thức phức tạp
- Ví dụ: Xác định các ký tự tương ứng với các tháng của năm

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

```
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.

Quy tắc đặt tên

- Một vấn đề quan trọng trong phong cách lập trình là làm thế nào đặt tên thích hợp cho các thành phần của chương trình:
  - Các tệp (files)
  - Các hàm
  - Các biến
  - etc

- Quy tắc đặt tên có thể gây tranh cãi và thường đề cập tới:
  - Độ dài các định danh
  - Làm thế nào kết hợp ký tự (hoa và thường) với các số
  - Làm thế nào để phân tách các từ trong một định danh nhiều từ (dấu cách không được dùng)
  - etc

Quy tắc đặt tên

```
Files:
  x2.cc f.h

Variables:
  int nl;
  double n, m;

Functions:
  double f(double n);
```

```
Files:
  gcd.cc numerical.h

Variables:
  int numLetters;
  double x, y; // coordinates

Functions:
  double sqrt(double x);
```

Một số khuyến nghị về quy tắc đặt tên

- Đặt tên có ý nghĩa để từ tên gọi có thể hiểu được vai trò của nó.
- Có thể đặt tên ngắn nếu ý nghĩa của nó tường minh trong ngữ cảnh. Ví dụ:

```
for (int i = 0; i < n; ++i) …

double d; // represents distance
// this could be obvious in a program
// written by Physicists
```

Một số khuyến nghị về quy tắc đặt tên

- Các tên `i, j, k` được dùng cho các chỉ số.
- Các biến bắt đầu bởi `n` thường dùng để chỉ các số tự nhiên.
- Các biến bắt đầu bởi `x, y, z` thường để chỉ các số thực (ví dụ tọa độ).
- Ví du: tên cho khoảng cách.
  - `di` (rất tệ: difference? d_i?...)
  - `dis` (tệ: display? disjoint?...)
  - `dist` (tốt hơn nhưng có thể cải thiện thêm)
  - `distance` (tốt nhất)
  - `d` (chấp nhận được trong ngữ cảnh toán hay vật lý khi d luôn dùng để ký hiệu khoảng cách. Dùng ký tự khác có thể gây nhầm lẫn.

Một số khuyến nghị về quy tắc đặt tên

- Dùng tên có nhiều từ để cải thiện khả năng đọc: `numLetters, first_element, IsPrime, StartTime, …`
- Lập trình với các tên ngắn sẽ dẫn tới hậu quả mất nhiều thời gian hơn để phát triên: do khó hiểu và khó debug. Thêm một vài ký tự vào tên biến có thể tiết kiệm nhiều thời gian. Ví dụ:

```
int t;    →    int timeInSeconds;
```

- Lập trình với tên biến dài cũng không hiệu quả. Đa số IDE có chế độ tự hoàn thành và có thể định dang tên biến chỉ sau khi gõ vài ký tự.
- Bạn muốn sử dụng thời gian của mình vào việc nào hơn: viết chương trình hay ngồi debug?

Kết hợp các từ: một vài phong cách

Camel Case	`myFunctionName`
Pascal Case	`MyClassName`
Snake Case	`my_variable_name`
SCREAMING SNAKE CASE	`MY_CONSTANT_NAME`

Phương án nào tốt hơn?

- Hãy tuân theo quy tắc của phong cách lập trình
- Hãy nhất quán

Đặc tả hàm

- **Mẹ tôi nói:**
  "Con ơi, ra chợ mua cho mẹ 1 chai sữa. Nếu có trứng thì mua 6"

  Tôi quay về với 6 chai sữa.

  Mẹ nói: "Sao lại mua 6 chai sữa?"

  Tôi nói: "VÌ HỌ CÓ TRỨNG!!!"

Đặc tả hàm

- Trong khóa này ta tập trung vào vấn đề làm sao viết đặc tả tốt cho hàm. Những quy tắc quan trọng:
  - Đặt tên tốt cho hàm và các tham số
  - Mô tả ý nghĩa và miền giá trị của các tham số
  - Mô tả hàm làm cái gì và/hoặc trả về cái gì.

- Một đặc tả hàm tốt sẽ cho phép lập trình viên hiểu chính xác hàm làm gì mà không cần nhìn vào thân hàm
- Cần thiết: mỗi tham số cần được đề cập trong đặc tả của hàm.

Ví dụ đặc tả hàm

```
// Checks whether it is a multiple
bool check(int n, int m);

// Questions:
//    What does the function name mean?
//    Who is multiple of whom?
//    Can m be zero?
```

```
// Pre: m > 0.
// Checks whether n is multiple of m
bool isMultiple(int n, int m);
```

Ví dụ đặc tả hàm

```
// Draws a rectangle
void d(int n, int m);

// Questions: what is the height and width
// of the rectangle? What is the output?
```

```
// Better specification (notice the names):
// Pre: nrows > 0, ncols > 0
// Draws a rectangle of '*' in cout
// consisting of nrows rows and ncols columns
void drawRectangle(int nrows, int ncols);
```

Chú thích

- Chú thích giúp việc đọc hiểu mã nguồn một cách trôi chảy.
- Chú thích không những quan trọng với người đọc mà còn đối với cả người viết ra.
- Viết chú thích là một nghệ thuật. Mỗi lập trình viên có phong cách riêng. Điểm chung là: không chú thích quá nhiều hoặc quá ít. Chỉ chú thích khi cần thiết.
- Đặt tên hợp lý cho các biến và hàm là bước khởi đầu tốt cho việc viết tài liệu chương trình.

Ví dụ về chú thích

- Mất bao nhiêu thời gian để hiểu đoạn mã sau làm gì?

```
for (int i = P.size()-1; i >= 0; --i) eval = eval*x + P[i];
```

Ví dụ về chú thích

- Sẽ không mất nhiều thời gian nếu ta thêm dùng chú thích đơn giản

```
// Evaluates the polynomial P(x) using Horner's scheme
for (int i = P.size()-1; i >= 0; --i) eval = eval*x + P[i];
```

Đừng chú thích thừa

```
// We declare the variable
// and initialize to zero
int n = 0;
```

```
if (a%2 == 0) // If it is an even number
```

```
++i; // we increase i at each iteration
```

```
// and we add the elements
A[i] = B[i] + C[i];
```

Chú thích gần ngôn ngữ tự nhiên nhất có thể

```
// What is f?
while (f == false) { … }

// What's a and p?
a = 2*p;

// Hmmm, that long?
if (a >= 0) return true;
else return false;

// Too verbose if nobody
// else is using r
double r = sqrt(x);
if (r > z) { … }
```

```
// Ah, yeah!
while (not found) { … }

// Clear enough
angle = 2*Pi;

// Much nicer
return a >= 0;



// Much simpler
if (sqrt(x) > z) { … }
```

- Chắc bạn không muốn code của bạn có đoạn chú thích như này?

```
8   // Dear programmer:
9   // When I wrote this code, only god and
10  // I knew how it worked.
11  // Now, only god knows it!
12  //
13  // Therefore, if you are trying to optimize
14  // this routine and it fails (most surely),
15  // please increase this counter as a
16  // warning for the next person:
17  //
18  // total_hours_wasted_here = 254
19  //
20
```

Một vài trích dẫn nổi tiếng về lập trình

- “Documentation is a love letter that you write to your future self.”
  Damian Conway

- “Commenting your code is like cleaning your bathroom - you never want to do it, but it really does create a more pleasant experience for you and your guests.”
  Ryan Campbell

- “Looking at code you wrote more than two weeks ago is like looking at code you are seeing for the first time.”
  Dan Hurvitz

- “The sooner you start to code, the longer the program will take.”
  Roy Carlson

GOOD PROGRAMMING STYLE

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 rang / đừng hy sinh sự rõ rang cho hiệu quả

6. Let the machine do the dirty work – Hãy để máy tính làm những việc nặng nhọc của nó. (tính toán …)

GOOD PROGRAMMING STYLE

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

GOOD PROGRAMMING STYLE

13. Write first in easy-to-understand pseudo language; then translate into whatever language you have to use. – Trước tiên hãy viết ct 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 CT lớn thành từng CT con

GOOD PROGRAMMING STYLE

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 CT

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

GOOD PROGRAMMING STYLE

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 CT tại các cận

26. Check some answers by hand. – Kiểm tra 1 số câu trả lời bằng tay

27. 10.0 times 0.1 is hardly ever 1.0. – 10 nhân 0.1 không chắc đã = 1.0

28. 7/8 is zero while 7.0/8.0 is not zero. 7/8 =0 nhưng 7.0/8.0 <> 0

29. Make it right before you make it faster. – Hãy làm cho CT chạy đúng, trước khi làm nó chạy nhanh

GOOD PROGRAMMING STYLE

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 tối 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 CT để giúp người đọc hiểu đc CT

39. Don't over-comment. – Đừng chú thích quá nhiều

Viết tài liệu chương trình

Tài liệu ngoài cho các lập trình viên khác

- Giới thiệu với các lập trình viên khác mã nguồn dùng để làm gì
- Nhiều công ty lớn tự đặt chuẩn riêng để viết tài liệu ngoài
- Mục tiêu là cho phép các lập trình viên khác sử dụng và thay đổi mã nguồn mà không cần đọc và hiểu từng dòng lệnh

Viết tài liệu ngoài

Bước 1: Miêu tả một cách tổng quát cách thức hoạt động của chương trình

- Chương trình phải làm gì?
- Phải đọc từ nguồn dữ liệu nào, ghi vào đâu?
- Giả thiết gì với đầu vào?
- Dùng giải thuật nào?

Bước 2:

- Miêu tả một cách tổng quát quy trình nghiệp vụ của chương trình (giống như cách miêu tả 1 flowchart)
- Có thể vẽ biểu đồ
- Giải thích các giải thuật phức tạp được ưử dụng trong chương trình, hoặc cho biết có thể tìm được lời giải thích ở đâu

Viết tài liệu ngoài

Bước 3:

- Nếu chương trình bao gồm nhiều file, giải thích nội dung từng file
- Giải thích cấu trúc dữ liệu được sử dụng phổ biến trong chương trình
- Giải thích việc sử dụng các biến toàn cục trong các chương trình con

Bước 4:

- Miêu tả các hàm chính trong chương trình
  - Lập trình viên tự quyết định hàm nào là hàm chính trong chương trình của mình
  - Xem xét hàm nào là hàm nghiệp vụ thực sự, ko nhất thiết phải là hàm dài nhất hay khó viết nhất
- Miêu tả các tham số đầu vào và giá trị trả về

Viết tài liệu cho người dùng

- Đây chính là hướng dẫn sử dụng (user manual)
- Là phần không thể thiếu khi viết tài liệu cho 1 dự án phần mềm, nhưng không phải phần quan trọng nhất

Viết tài liệu kiểm thử

- Tài liệu kiểm thử là 1 trong số các tài liệu quan trong của 1 dự án phần mềm
- Nếu được, bạn nên viết ra 1 số bằng chứng về việc bạn đã kiểm thử chương trình của bạn với nhiều đầu vào khác nhau
- Việc không viết tài liệu kiểm thử có thể gây ra nhiều hậu quả nặng nề