

RÈN LUYỆN PHONG CÁCH LẬP TRÌNH CHUYÊN NGHIỆP CHO HS-SV THÔNG QUA VIỆC GIẢNG DẠY CÁC MÔN LẬP TRÌNH

I. GIỚI THIỆU

Phong cách lập trình chuyên nghiệp là phong cách lập trình tuân thủ theo các quy định chung khi viết code. Trong đó, code sạch (clean code) là yêu cầu cơ bản của những lập trình viên chuyên nghiệp.

Clean code là những đoạn code được trình bày một cách đơn giản và hiệu quả. Clean code là code dễ đọc, dễ hiểu, dễ sửa và dễ bảo trì. Các logic trong code được trình bày một cách xuyên suốt để giúp tránh được các lỗi tiềm ẩn, giảm sự phụ thuộc để dễ dàng bảo trì.

Phong cách lập trình còn phụ thuộc vào IDE (Integrated Development Environment) mà lập trình viên sử dụng. Các IDE cung cấp sẵn phong cách định dạng mã nguồn cho ngôn ngữ mà chúng hỗ trợ một cách tự động. Chẳng hạn như cách đóng mở ngoặc cặp ngoặc “{}”, cách thêm khoảng trắng giữa các biến, toán tử... Mặc dù vậy, phần lớn chúng đều có những điểm chung.

II. LÝ DO TẠI SAO PHẢI RÈN LUYỆN PHONG CÁCH LẬP TRÌNH CHUYÊN NGHIỆP

- Một phong cách lập trình tốt sẽ giúp lập trình viên tạo ra một chương trình dễ đọc, dễ hiểu, dễ tìm và sửa lỗi, dễ nhận thấy cấu trúc bậc cao của chương trình. Những điều này sẽ thuận lợi cho cả người viết chương trình, người đánh giá chương trình, và người bảo trì chương trình.

- Thông thường 80% thời gian dòng đời của một phần mềm là thời gian bảo trì và nâng cấp. Một khi triển khai ra thực tế sẽ gặp rất nhiều vấn đề cần chỉnh sửa, phát triển thêm tính năng. Nếu người lập trình phần mềm không có phong cách cách lập trình tốt hay còn gọi là Clean Code thì sẽ gây những hậu quả sau:

- Người bảo trì và nâng cấp chương trình không hiểu hoặc phải tốn nhiều thời gian để hiểu được chương trình. Bản thân người viết chương trình cũng sẽ gặp các khó khăn tương tự đối với các chương trình lớn hoặc sau một thời gian lâu cần xem lại chương trình.
- Người viết chương trình khó kiểm soát được code đối với chương trình lớn và phức tạp, gây nguy cơ phát sinh lỗi. Chương trình dễ bị “rối” và làm cho người lập trình phải mất nhiều thời gian hơn để gỡ rối và sửa lỗi.
- Nếu chương trình được thực hiện bởi một tập thể nhiều lập trình viên, mà các thành viên không tuân theo một quy chuẩn chung thì không thể hoàn thành tốt chương trình.

- Đã có rất nhiều project phát triển một thời gian, các chức năng chạy tạm ổn. Nhưng khi có các yêu cầu thêm mới và update thì các dòng code tăng ngày càng một nhiều và trở thành một mớ code lộn xộn. Do ngày từ đầu lập trình viên không chú ý vấn đề “clean code” nên đến một lúc lập trình viên không thể nắm bắt, kiểm soát được mã nguồn và vì thế không thể bảo trì và phát triển dự án. Đó là trường hợp của là ứng tên là Killer rất nổi tiếng và được nhiều người sử dụng vào thập niên 80, sau đó sản phẩm bị hủy và công ty phát hành phải đóng cửa.

- Việc học và rèn luyện một phong cách lập trình tốt nên được tiến hành ngay từ khi HS-SV bắt đầu học lập trình vì khi một thói quen không tốt được hình thành thì phải mất rất nhiều thời gian để thay đổi nó.

- Việc áp dụng phong cách lập trình còn thể hiện trình độ và sự chuyên nghiệp của lập trình viên, một mã nguồn có phong cách lập trình tốt luôn được đánh giá cao hơn một mã nguồn không tuân theo phong cách lập trình.

III. CÁC GIẢI PHÁP ĐỂ RÈN LUYỆN PHONG CÁCH LẬP TRÌNH CHUYÊN NGHIỆP CHO HS-SV

Theo kinh nghiệm của bản thân tôi, để dạy cho SV phong cách lập trình chuyên nghiệp không nhất thiết giáo viên phải giành quá nhiều thời gian hay bố trí những bài học riêng. HS-SV sẽ được rèn luyện phong cách lập trình tốt, viết clean

code thông qua phong cách lập trình hay các ví dụ của giáo viên. Một số giải pháp cần thiết là:

- Mỗi giáo viên cần tìm hiểu về clean code và chú ý cách viết code của mình. Mỗi đoạn code mà giáo viên viết đều phải là “clean code” để HS-SV làm theo. Điều quan trọng là sự đồng thuận của tất cả giáo viên, tránh hiện tượng “người làm người không” sẽ không đạt được mục tiêu đề ra.

- Trong quá trình giảng dạy hay sửa bài, cần chú ý nhắc nhở HS-SV viết “clean code”, đặc biệt là trong các môn học khi HS-SV mới làm quen với lập trình như Nhập môn Tin học và Kỹ thuật Lập trình.

- Nên chú ý cho điểm cộng đối với các chương trình có phong cách lập trình tốt để khuyến khích HS-SV.

IV. NHỮNG QUY TẮC CHUNG CẦN RÈN LUYỆN CHO HS-SV ĐỂ CÓ ĐƯỢC PHONG CÁCH LẬP TRÌNH CHUYÊN NGHIỆP

- Để phù hợp với đối tượng HS-SV của trường CDKT Lý Tự Trọng, SKKN này trình bày những quy tắc chính, cơ bản nhất và minh họa qua bằng các ví dụ trong các môn học Nhập môn Tin học, Kỹ thuật Lập trình, Cấu trúc dữ liệu và giải thuật để các bạn đồng nghiệp và HS-SV dễ dàng áp dụng.

1. Cách trình bày tổng quan chương trình

Quy tắc 1: Chương trình nên được chia thành nhiều modul, mỗi modul thực hiện một công việc và càng độc lập với nhau càng tốt. Điều này sẽ giúp chương trình dễ bảo dưỡng.

Ví dụ: Sau khi HS-SV được học về hàm cần bắt buộc HS-SV modul hóa các chức năng của chương trình thành những hàm riêng.

Quy tắc 2: Nên viết code theo chuẩn dù cho compiler không bắt buộc.

Ví dụ:

- Nên dùng `int main()` thay vì `void main()` vì mặc dù Microsoft Visual Studio chấp nhận cách dùng `void main()`, nhưng các trình biên dịch khác như Dev-C++ lại không chấp nhận cách viết này.

- Hạn chế dùng các header file không có trong c chuẩn như `conio.h`. Khi đó hàm `getch()` có thể thay bằng lệnh `system("pause");`

Quy tắc 3: Cách trình bày chương trình càng nhất quán sẽ càng dễ đọc và dễ hiểu và ít bị mắc lỗi. Từ đó, ta sẽ mất ít thời gian để nghĩ về cách viết chương trình, tập trung thời gian hơn để nghĩ về các vấn đề cần giải quyết.

Ví dụ: Do cách đặt tên hàm của HS-SV không nhất quán nên thường bị mắc lỗi cú pháp vì khi viết hoa khi thì không viết hoa, sinh viên thường mất thời gian để kiểm tra và sửa lỗi này:

Quy tắc 4: Chương trình nên giữ được tính đơn giản và rõ ràng trong hầu hết các tình huống.

Quy tắc 5: Mỗi câu lệnh nên được đặt riêng trên một dòng để chương trình dễ đọc và dễ tìm lỗi khi chạy debug.

Ví dụ:

Không nên
<code>if (i <= j){ swap(a[i], a[j]); i++; j--; }</code>
Nên
<pre> if (i <= j) { swap(a[i], a[j]); i++; j--; } </pre>

Quy tắc 6: Câu lệnh phải thể hiện đúng cấu trúc chương trình. Các câu lệnh con phải được lùi vào một bậc. Việc lùi vào một bậc có thể được thực hiện bằng một số khoảng trắng (space) hoặc dấu tab. Việc lùi đầu dòng các dòng code một cách thích hợp theo đúng cấu trúc sẽ giúp cho lập trình viên thấy được cấu trúc khối của chương trình, điều đó giúp cho chương trình dễ đọc, dễ hiểu hơn và giúp cho việc tìm lỗi và sửa lỗi dễ dàng hơn.

Việc yêu cầu HS-SV trình bày chương trình theo cấu trúc rất cần thiết, thông qua đó giáo viên cũng đánh giá được mức độ hiểu cấu trúc chương trình của HS-SV. Một chương trình được trình bày sai cấu trúc cho thấy người viết nó chưa hiểu rõ hoàn toàn các câu lệnh của chương trình.

Ví dụ:

Không nên
<pre>for (int i = 0; i < n -1; i++) { int min = i; for (int j = i+1; j < n; j++) if (a[min] > a[j]) min = j; swap(a[i], a[min]); }</pre>
Nên
<pre>for(int i = 0; i < n -1; i++) { int min = i; for (int j = i+1; j < n; j++) if (a[min] > a[j]) min = j; swap(a[i], a[min]); }</pre>

Quy tắc 7: Nên viết cặp dấu { } trước rồi viết các lệnh vào giữa để tránh thiếu các dấu ngoặc này. Các câu lệnh nằm giữa cặp dấu { } được viết thụt vào một tab.

Ví dụ:

Không nên
<pre>for(int i = 0; i < n -1; i++) { cout<<a[i]<<"\t"; }</pre>

}
Nên
<pre>for(int i = 0; i < n -1; i++) { cout<<a[i]<<"\t"; } Hay: for(int i = 0; i < n -1; i++){ cout<<a[i]<<"\t"; }</pre>

Quy tắc 8:

- Chiều dài mỗi dòng code không quá 80 ký tự. Trong trường hợp xuống dòng nên mới bằng một toán tử và dòng mới phải được lùi vào bằng sâu hơn phạm vi khối code đang phát biểu.
- Chiều dài của file không quá 2000 dòng (bao gồm cả phần chú thích).

Ví dụ:

Không nên
<pre>if (p->data.SiSo < 20) cout<<"\n"<<i++<<"\t"<<p->data.MaLop<< "\t"<<p->data.GVCN<<"\t\t"<<p->data.SiSo;</pre>
Nên
<pre>if (p->data.SiSo < 20) cout<<"\n"<<i++<<"\t"<<p->data.MaLop<<"\t" <<p->data.GVCN<<"\t\t"<<p->data.SiSo;</pre> Hay: <pre>if (p->data.SiSo < 20) cout<<"\n"<<i++<<"\t"<<p->data.MaLop<<"\t" <<p->data.GVCN<<"\t\t"<<p->data.SiSo;</pre>

2. Cách ghi chú thích (Comment)

Quy tắc 1: Các dòng chú thích không nên quá dài, vượt quá phạm vi hiển thị của trình soạn thảo, hãy xuống dòng sau các dấu câu.

Quy tắc 2: Không nên chú thích quá dài dòng hoặc chú thích những đoạn không cần thiết. Bản thân mã nguồn đã tự nói về công dụng của nó vì thế trước khi thực hiện chú thích hãy xem lại đoạn code đã thực sự rõ ràng và trong sáng chưa. Tránh trường hợp code “không sạch”, sau đó dùng comment để chú thích đoạn code đó làm gì.

Ví dụ:

Không nên
<pre>int dem(int a[], int n) //Dem so chan { int d = 0; // Bien dem ... }</pre>
Nên
<pre>int demSoChan(int a[], int n) { int dem = 0; ... }</pre>

Quy tắc 3: Nên sử dụng dấu // để ghi chú thích vì dấu này không bị ảnh hưởng khi sử dụng cặp ký hiệu /* */ để vô hiệu hóa một đoạn lệnh trong quá trình sửa lỗi chương trình (trong C/C++ không cho phép các cặp dấu /* */ lồng nhau)

Quy tắc 4: Với các chú thích ngắn, nên đặt nó trên cùng dòng lệnh cần chú thích. Với các chú thích dài hơn, hoặc chú thích cho cả một đoạn lệnh hãy đặt câu chú thích trên một dòng riêng ngay phía trên câu lệnh cần chú thích.

Các trường hợp nên chú thích:

Trường hợp 1: Chú thích về tính pháp lý dự án hay chương trình

Cách ghi chú thích cho một dự án/chương trình

```
//*****  
// Tên dự án  
// Thông tin bản quyền  
// Địa chủ trang chủ  
// Các thông tin về chương trình  
// Phiên bản, IDE sử dụng  
// Tên lập trình viên  
// Ngày cập nhật  
//*****
```

Trường hợp 2: Chú thích về thông tin chức năng, tính chất của lớp hoặc hàm

Cách ghi chú thích cho một hàm

```
// Tên hàm  
// Mô tả chức năng của hàm  
// Mô tả giá trị trả về cho các trường hợp  
// Mô tả các tham số  
// Mô tả giải thuật sử dụng trong hàm
```

Trường hợp 3: Chú thích về khối xử lý logic trong hàm

Trường hợp 4: Chú thích cảnh báo những hậu quả có thể xảy ra khi thực thi một đoạn/ khối lệnh

Trường hợp 5: Tạo tài liệu chú thích cho các giao diện lập trình ứng dụng (API)

Ví dụ: Các ghi chú cho chương trình và hàm

```
//*****  
// Chương trình: Thao tác trên cây nhị phân tìm kiếm  
// Tác giả: Nguyễn Văn Giang  
// Email: vangiangnguyen@yahoo.com  
// Ngôn ngữ : C/C++  
// Ngày cập nhật: 06/02/2017
```

```
//*****

...

// Tên hàm      : insertBST
// Chức năng    : Chèn giá trị x vào cây gốc root
// Giá trị trả về : 0 nếu x đã có trong cây,
//               1 nếu chèn thành công
//               -1 nếu bộ nhớ đầy

int insertBST(tree &root, int x)
{
    if(root != NULL)    // tìm vị trí chèn
    {
        if(root->data == x)
            return 0;
        if(root->data > x)
            return insertBST(root->left, x);
        return insertBST(root->right, x);
    }
    else                // tìm được vị trí chèn là root
    {
        root = new node;
        if(root == NULL)
            return -1;
        root->data = x;
        root->left = NULL;
        root->right = NULL;
        return 1;
    }
}
```

3. Cách đặt tên cho hằng, biến, hàm, lớp, phương thức

Cách đặt tên cần tuân thủ các quy tắc:

Quy tắc 1: Tên cần phải có ý nghĩa để gợi nhớ cho lập trình viên.

Ví dụ:

Không nên	Nên
typedef struct Date	typedef struct Date
{	{

<code>int a; // ngay</code>	<code>int day;</code>
<code>int b; // thang</code>	<code>int month;</code>
<code>int c; // nam</code>	<code>int year;</code>
<code>};</code>	<code>};</code>
<code>...</code>	<code>...</code>
<code>Date d; // date_of_birth;</code>	<code>Date date_of_birth;</code>

Quy tắc 2: Dùng từ chính quy, tránh từ "tự chế":

Ví dụ: Tên hàm là `generateKey` sẽ dễ hiểu hơn là `genKey` hay `gnKey`

Quy tắc 3: Tên lớp, hàm, phương thức theo quy tắc camel (viết hoa ký tự đầu tiên của từng từ).

- **Tên lớp** dùng danh từ hoặc cụm danh từ

Ví dụ:

Không nên	Nên
<code>class sinhvien;</code>	<code>class SinhVien;</code>

- **Tên hàm, phương thức** là động từ hoặc cụm động từ, nên bắt đầu bằng chữ cái thường.

Ví dụ:

Không nên	Nên
<code>void demsochan(char *name);</code>	<code>void demSoChan(char *name);</code>

Quy tắc 4: Tên của hằng nên được viết hoa toàn bộ, nếu tên dài thì nên dùng dấu phân cách là `_` để phân biệt với tên biến.

Ví dụ:

Không nên	Nên
-----------	-----

# define maxrow 5	# define MAX_ROW 5
# define MaxRow 5	

Quy tắc 5: Tên biến không nên đặt bằng toàn kí tự in hoa dễ nhầm với hằng số

Ví dụ:

Không nên	Nên
Date DATEOFBIRTH;	Date date_of_birth;
Date dateofbirth;	

4. Cách viết hàm

Quy tắc 1: Hàm không nên dài hơn 100 dòng và thường khoảng 20 dòng.

Quy tắc 2: Mỗi hàm chỉ nên làm đúng 1 chức năng duy nhất để dễ tái sử dụng và dễ đọc hơn.

Ví dụ:

Không nên
<pre>void demSoChan(int a[], int n) { int dem = 0, tong = 0; for (int i=0; i< n; i++) if (a[i]%2 == 0) { tong = tong + a[i]; dem++; } cout<<"Co "<<dem<<" so chan\n"; cout<<"Tong cac so chan = "<<tong;</pre>

}
Nên
<pre>int demSoChan(int a[], int n) { int dem = 0; for (int i=0; i< n; i++) if (a[i]%2 == 0) dem++; return dem; } int tongSoChan(int a[], int n) { int tong = 0; for (int i=0; i< n; i++) if (a[i]%2 == 0) tong = tong + a[i]; return tong; }</pre>

Quy tắc 3: Tránh viết những đoạn code xử lý được lặp đi lặp lại, tốt nhất là đưa chúng vào một hàm riêng để tiện cho việc tái sử dụng.

Quy tắc 4: Nên sử dụng các tham số khi truyền thông tin cho các chương trình con. Tránh sử dụng các biến toàn cục để truyền thông tin giữa các chương trình con vì như vậy sẽ làm mất tính độc lập giữa các chương trình con, giảm tính tái sử dụng và rất khó khăn khi kiểm soát giá trị của chúng khi chương trình thi hành.

Ví dụ: Hàm `xuatMang()` nếu dùng `a` và `n` là biến toàn cục thì chỉ xuất được mảng `a` nhưng nếu truyền tham số thì có thể dùng hàm này để xuất nhiều mảng khác.

Không nên
<pre>#define N 100 int a[N]; int n; void xuấtMang() { for (int i=0; i< n; i++) cout<<a[i]<<"\t"; }</pre>
Nên
<pre>void xuấtMang (int a[], int n) { for (int i=0; i< n; i++) cout<<a[i]<<"\t"; }</pre>

Quy tắc 5: Các toán tử và toán hạng trong một biểu thức nên được tách rời nhau bởi 1 khoảng trắng để biểu thức dễ đọc hơn (trừ các toán tử ++, --, -)

Ví dụ:

Không nên	Nên
<code>a=b*c;</code>	<code>a = b * c;</code>
<code>a = b ++;</code>	<code>a = b++;</code>
<code>a = - 1;</code>	<code>a = -1;</code>

Quy tắc 6: Có khoảng trắng ngăn cách giữa dấu phẩy hay chấm phẩy với các tham số.

Quy tắc 7: Nên có khoảng trắng giữa từ khóa và dấu “(”, nhưng không nên có khoảng trắng giữa tên hàm và dấu “(“

Ví dụ:

Không nên	Nên
<code>void hoanVi (int a,int b);</code>	<code>void hoanVi(int a, int b);</code>
...	...
<code>for(int i = 1;i < n;i++)</code>	<code>for (int i = 1; i < n; i++)</code>

Quy tắc 8: Nên sử dụng các dấu () khi muốn tránh các lỗi về độ ưu tiên toán tử

Không nên
<code>int i = a >= b && c < d && e <= g + h;</code>
Nên
<code>int i = (a >= b) && (c < d) && (e <= (g + h));</code>

Quy tắc 9: Nên dùng các dòng trắng để phân chia các đoạn lệnh trong một hàm như: đoạn nhập/xuất dữ liệu, đoạn tương ứng với các bước xử lý khác nhau.

Quy tắc 10: Các hằng số không nên viết trực tiếp vào chương trình mà nên sử dụng `#define` hay `const` để định nghĩa. Điều này sẽ giúp lập trình viên dễ kiểm soát những chương trình lớn vì giá trị của hằng số khi cần thay đổi thì chỉ phải thay đổi một lần duy nhất ở giá trị định nghĩa (ở `#define` hay `const`).

Tuy vậy, không nên dùng `#define` thường xuyên để định nghĩa các hằng số, bởi vì trong quá trình debug, ta sẽ không thể xem được giá trị của một hằng số định nghĩa bằng `#define`

Ví dụ: Xét chương trình tính diện tích và chu vi hình tròn.

Không nên
<code>int main()</code>
<code>{</code>
...

<pre>chu_vi = 2 * 3.14159 * r; dien_tich = 3.14159 * r * r; ... }</pre>
Nên
<pre>#define PI 3.14159 int main() { ... chu_vi = 2 * PI * r; dien_tich = PI * r * r; ... }</pre>

Quy tắc 11: Biến nên được khai báo ở gần vị trí mà nó bắt đầu được sử dụng nhằm tránh việc khai báo một loạt các biến dư thừa ở đầu hàm hay chương trình.

Lưu ý: Trong C chuẩn, tất các biến bắt buộc phải khai báo ở đầu khối trước khi sử dụng. Trong C++ biến có thể khai báo ở bất kỳ nơi đâu trước khi sử dụng.

Quy tắc 12: Các biến không nên được sử dụng lại với nhiều nghĩa khác nhau trong cùng một hàm.

Ví dụ: Ví dụ biến x trong chương trình sau được vừa được dùng để chỉ vị trí, vừa được dùng để chỉ giá trị của phần tử trong mảng nên dễ gây nhầm lẫn. Trong trường hợp cần thay đổi kiểu của mảng thì chương trình phải sửa đổi nhiều.

Không nên
<pre>int main() { ... int a[N], n, x;</pre>

<pre>... printf("\nNhap vi tri can xoa trong mang: "); scanf("%d", &x); xoaViTri(a, n, x); ... printf("\nNhap gia tri can xoa trong mang: "); scanf("%d", &x); xoaGiaTri(a, n, x); ... }</pre>	
Nên	
<pre>int main() { int a[N], n; ... int vi_tri; printf("\nNhap vi tri can xoa trong mang: "); scanf("%d", &vi_tri); xoaViTri(a, n, vi_tri); ... int gia_tri; printf("\nNhap gia tri can xoa trong mang: "); scanf("%d", &gia_tri); xoaGiaTri(a, n, gia_tri); }</pre>	

```
...  
}
```

TÀI LIỆU THAM KHẢO

[1] Robert C. Martin, “Clean Code: A Handbook of Agile Software Craftsmanship”, Prentice Hall.

[2] Đặng Bình Phương, “Phong cách lập trình”, Khoa Công nghệ Thông Tin, Trường ĐHKHTN TP. HCM