
THỰC THI
ThS. Dương Hữu Thành,
Khoa CNTT, Đại học Mở TP.HCM,
thanh.dh@ou.edu.vn.
1

1


Nội dung chính
1 Thực thi
2 Kỹ năng lập trình
3 Phong cách lập trình
4 Tái sử dụng
5 Phát triển mã nguồn mở
2

2

## Thực thi

- Thực thi (implementation) là quá trình hiện thực hoá thiết kế thành chương trình.
- Thực thi bao gồm việc lập trình để phát triển hệ thống theo đặc tả yêu cầu.



## Kỹ năng lập trình

- Hiểu rõ ngôn ngữ lập trình lựa chọn.
- Mã nguồn nên viết mạch lạc, dễ đọc, dễ theo dõi luống xử lý của nó.
- Sử dụng tên biến thích hợp, có ý nghĩa tránh nhầm lẫn.
- Viết mã lệnh theo chuẩn: thống nhất cách đặt tên module, tên hàm, tên biến, v.v.

## Kỹ năng lập trình

- Ghi chú (comment) trong chương trình.
- Định dạng mã nguồn (canh lề, thụt dòng, ...) để mã nguồn sáng sủa và dễ đọc.

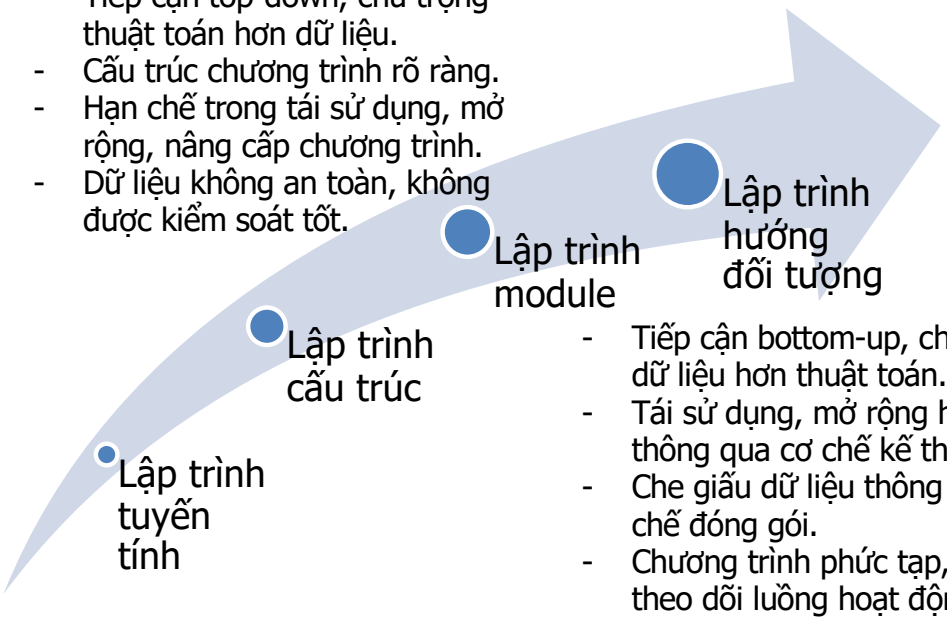


## Kỹ năng lập trình

- Khả năng tái sử dụng
  - Các module càng ít phụ thuộc vào môi trường càng tốt.
  - Cần ghi chú lại sự thay đổi, cập nhật module.
  - Cần tuân theo một chuẩn chung về đặt tên biến, hàm, phương thức.
  - Nên tách biệt phần giao diện và phần xử lý.

# Kỹ năng lập trình

- Tiếp cận top-down, chú trọng thuật toán hơn dữ liệu.
- Cấu trúc chương trình rõ ràng.
- Hạn chế trong tái sử dụng, mở rộng, nâng cấp chương trình.
- Dữ liệu không an toàn, không được kiểm soát tốt.




- Tiếp cận bottom-up, chú trọng dữ liệu hơn thuật toán.
- Tái sử dụng, mở rộng hiệu quả thông qua cơ chế kế thừa.
- Che giấu dữ liệu thông qua cơ chế đóng gói.
- Chương trình phức tạp, khó theo dõi luồng hoạt động, chỉ phù hợp chương trình lớn.

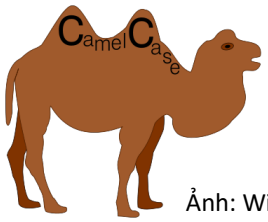


# Phong cách lập trình

- Phong cách lập trình là những quy tắc hay chỉ dẫn để viết mã nguồn, viết mã nguồn theo chuẩn giúp chương trình trở nâng dễ đọc, dễ hiểu, dễ bảo trì, khả năng tái sử dụng mã nguồn cao, tránh được nhiều lỗi không cần thiết.

## Phong cách lập trình

- Quy tắc đặt tên
  - Quy tắc lạc đà (Camel Case)
    - Upper Camel Case: các chữ cái đầu tiên của mỗi từ phải viết hoa, chẳng hạn SinhVien, GiangVien.
    - Lower Camel Case: tương tự Upper Camel Case nhưng chữ cái đầu của từ đầu tiên viết thường, chẳng hạn soLuong, tongDoanhThu.




Ảnh: Wikipedia





## Phong cách lập trình

- Quy tắc đặt tên
  - Quy tắc Hungary: quy tắc này thường kết hợp Upper Camel Case. Mục đích quy tắc này để tăng khả năng nhận biết kiểu dữ liệu của biến bằng cách gắn trước mỗi tên biến một hay vài ký tự thể hiện kiểu dữ liệu của chúng.
  - Ví dụ
    - iSoLuong → biến kiểu số nguyên
    - strHoTen → biến kiểu chuỗi

## Phong cách lập trình

- Trong một số ứng dụng lập trình giao diện đặt tên cho các điều khiển trong ứng dụng thường hay dùng quy tắc Hungary.
  - txtHeSo → điều khiển ô nhập điều (TextBox).
  - chkDongY → điều khiển CheckBox.
  - rdoMale → điều khiển RadioButton.
  - btnTinhLuong → điều khiển Button.



## Phong cách lập trình

- Quy tắc đặt tên
  - Quy tắc con rắn (Snake Case): tất cả các ký tự đều viết thường, các từ cách nhau bằng dấu gạch chân _.
  - Ví dụ: so_luong, ngay_bat_dau_lam

## Phong cách lập trình

- Chương trình nên được tách thành nhiều chương trình con càng độc lập với nhau càng tốt. Điều này giúp chương trình dễ đọc, dễ bảo trì và nâng cấp.
- Sử dụng các tham số khi truyền dữ liệu vào hàm, hạn chế tối đa sử dụng các biến toàn cục truyền thông tin giữa các hàm vì nó sẽ làm mất đi tính độc lập giữa các hàm, sự thay đổi dữ liệu rất khó kiểm soát.



## Phong cách lập trình

Bad

```
int m, n;

void nhap()
{
    cin >> m;
    cin >> n;
}

int main()
{
    nhap();

    return 0;
}
```

Good

```
void nhap(int &m, int &n)
{
    cin >> m;
    cin >> n;
}

int main()
{
    int m, n;
    nhap(m, n);

    return 0;
}
```

## Phong cách lập trình

- Tuân thủ chuẩn lập trình chung đảm bảo chương trình nhất quán từ cách đặt tên biến, tên hàm, biến cục bộ, biến toàn cục.
- Chương trình viết nên đơn giản, rõ ràng, tường minh, thể hiện mục đích của nó. Tránh viết mã nguồn ngầm định, tối nghĩa.



## Phong cách lập trình

- Lưu ghi chú (comment) trong chương trình
  - Ghi chú cho module: tên module, mục đích, cách dùng, mô tả cấu trúc dữ liệu, giải thuật, phiên bản, thời điểm phát hành, tác giả và thông tin liên hệ.
  - Ghi chú dòng lệnh: giúp hiểu rõ về đoạn mã nguồn.

## Phong cách lập trình

- Mỗi câu lệnh nên được đặt riêng trên một dòng giúp mã nguồn dễ đọc, dễ theo dõi, gỡ lỗi (debug) thuận tiện, hiệu quả hơn.

```
// bad
if (a % 2 == 0) cout << "Even";
else cout << "Odd";

// good
if (a % 2 == 0)
   cout << "Even";
else
   cout << "Odd";
```



## Phong cách lập trình

- Đối với các ngôn ngữ lập trình sử dụng {} để bọc khối mã nguồn nên được canh thẳng hàng theo một trong hai cách sau:

```
if ()
{
   ...
}
else
{
   ...
}
```

```
if () {
   ...
}
else{
   ...
}
```

## Phong cách lập trình

- Các khối lệnh con của các câu lệnh nào đó nên dùng tab thụt vào và nên có một dòng trắng giữa các khối lệnh có mục đích khác nhau.

Bad

```
if (a == b)
{
a += c;
c--;
}
else
{
a -= c;
c++;
}
```

Good

```
if (a == b)
{
    a += c;
    c--;
}
else
{
    a -= c;
    c++;
}
```



## Phong cách lập trình

Bad

```
char ten[100];
float diem;
cout << "Ten:";
cin.getline(ten, 100);
cout << "Diem:";
cin >> diem;
```

Good

```
char ten[100];
float diem;

cout << "Ten:";
cin.getline(ten, 100);

cout << "Diem:";
cin >> diem;
```

## Phong cách lập trình

- Các toán tử và toán hạng trong một biểu thức nên cách nhau một khoảng trắng, trừ các toán tử ++, --, -.

```
a=b+c; // bad
a = b + c; // good

a ++; // bad
a++; // good

a = - b; // bad
a = -b; // good
```



## Phong cách lập trình

- Nên có một khoảng trắng ở sau dấu phẩy (,) trong danh sách các tham số của hàm, khai báo biến. Tương tự cho dấu chấm phẩy (;).

```
int a,b; // bad
int a, b; // good
double tong(double a[],int n); // bad
double tong(double a[] , int n); // bad
double tong(double a[], int n); // good
tong(2,3); // bad
tong(2, 3); // good
for (int i=0;i<n;i++) // bad
for (int i = 0; i < n; i++); // good
```

## Phong cách lập trình

- Nên có khoảng trắng giữa từ khóa và dấu mở ngoặc (, nhưng không nên có giữa tên hàm và dấu mở ngoặc (.

```
if(a == b) // bad
if (a == b) // good
strcmp (a, b); // bad
strcmp(a, b); // good
```



## Phong cách lập trình

- Một dòng lệnh cũng không nên viết quá dài, thường nếu dài hơn 80 ký tự nên viết xuống dòng.
- Số lượng dòng code trong hàm, tổng số dòng code trong tập tin, số lượng phương thức trong lớp, số lượng lớp trong gói cần có giới hạn để dễ kiểm soát.
- Hàm thường không nên quá 30 dòng code, lớp không nên quá 500 dòng code.
- Số lượng tham số trong hàm không nên quá 5.
- Các câu lệnh lồng nhau không nên quá 4 cấp.

## Tái sử dụng

- Tái sử dụng một thành phần hoặc các hệ thống đã tồn tại giúp việc phát triển hệ thống nhanh hơn, ít rủi ro và giảm chi phí.
- Cách thức tái sử dụng ý tưởng hoặc phần mềm là điều đầu tiên nên quan tâm trước khi phát triển một hệ thống mới.
- Xem xét các khả năng tái sử dụng trước khi tiến hành thiết kế chi tiết phần mềm.



## Tái sử dụng

- Việc tái sử dụng sẽ đánh đổi một số chi phí sau:
  - Thời gian tìm kiếm phần mềm để tái sử dụng và đánh giá mức độ thích hợp của nó với hệ thống đang phát triển.
  - Chi phí mua phần mềm tái sử dụng.
  - Chi phí tương thích và cấu hình các phần mềm tái sử dụng để phù hợp với yêu cầu hệ thống đang phát triển.
  - Chi phí tích hợp các thành phần của phần mềm tái sử dụng với mã nguồn mới đã phát triển cho hệ thống.

## Tái sử dụng

- Việc tái sử dụng thực hiện ở nhiều cấp độ:
  - Mức trừu tượng (abstraction level)
  - Mức đối tượng (object level)
  - Mức thành phần (component level)
  - Mức hệ thống (system level)



## Tái sử dụng

- Mức trừu tượng: không trực tiếp tái sử dụng phần mềm, nhưng sử dụng ý tưởng trong thiết kế phần mềm, chẳng hạn sử dụng các mẫu thiết kế, mẫu kiến trúc, v.v.
- Mức đối tượng: trực tiếp sử dụng các đối tượng trong các thư viện, chẳng hạn cần xử lý gửi email trong ngôn ngữ Java, ta có thể sử dụng các đối tượng hoặc phương thức từ thư viện JavaMail.

## Tái sử dụng

- Mức thành phần: tái sử dụng bằng cách tích hợp hoặc kế thừa một số thành phần đã có vào ứng dụng. Ở mức này, ta cần viết mã nguồn cho một số thực thi riêng, chẳng hạn lập trình giao diện JavaFX, ta cần viết mã nguồn xác định cách thức hiển thị giao diện, xử lý sự kiện, v.v.
- Mức hệ thống: ở mức này, tái sử dụng toàn bộ hệ thống ứng dụng, thường bao gồm việc cấu hình hệ thống để sử dụng.



## Phát triển mã nguồn mở

- Phát triển phần mềm mã nguồn mở là cách tiếp cận phát triển phần mềm mã nguồn được xuất bản và sẵn sàng cho cộng đồng tham gia quy trình phát triển nó.
- Xuất phát từ Free Software Foundation (www.fsf.org) chủ trương mã nguồn không quan tâm quyền sở hữu và sẵn sàng cho phép mọi người đánh giá, chỉnh sửa theo ý họ muốn.

## Phát triển mã nguồn mở

- Sản phẩm mã nguồn mở nổi tiếng nhất là hệ điều hành Linux được sử dụng rộng rãi cho hệ thống server, cũng như môi trường desktop.
- Một số sản phẩm mã nguồn mở nổi tiếng khác: Java, Apache Web Server, MySQL, v.v.



## Phát triển mã nguồn mở

- Các mô hình cấp phép mã nguồn mở
  - GNU General Public License (GPL): nếu sử dụng mã nguồn mở cấp phép GPL thì sản phẩm phần mềm của phải là mã nguồn mở.
  - GNU Lesser General Public License (LGPL): khi viết thành phần liên kết với mã nguồn mở thì không cần công khai mã nguồn của các thành phần này.
  - Berkley Standard Distribution (BSD): không bắt buộc công bố lại những chỉnh sửa, bổ sung cho mã nguồn mở, người phát triển có quyền sở hữu để bán sản phẩm đã làm.


Q&A
33