

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

## Bài 8

# LẬP TRÌNH PHÒNG NGỪA

# Nội dung



1. Khái niệm
2. Phòng ngừa sai sót về dữ liệu
3. Xử lý lỗi
4. Bảo đảm

1.

# Khái niệm

Lập trình phòng ngừa



Lập trình  
phòng ngừa  
Defensive  
Programming

=

Defensive  
driving



# Lập trình phòng ngừa

## Defensive programming

- Ý tưởng chính: nếu chương trình (CTC) nhận dữ liệu vào bị lỗi thì nó vẫn chạy thông, ngay cả khi chương trình khác cũng nhận dữ liệu đầu vào đó đã bị lỗi.
- Lập trình phòng ngừa là cách tự bảo vệ chương trình của mình khỏi
  - *các ảnh hưởng tiêu cực của dữ liệu không hợp lệ*
  - *các rủi ro đến từ các sự kiện tương tự như "không bao giờ" xảy ra*
  - *sai lầm của các lập trình viên khác*

# Các lỗi có thể phòng ngừa

- Lỗi liên quan đến phần cứng
  - Đảm bảo các lỗi như *buffer overflows* hay *divide by zero* được kiểm soát
- Lỗi liên quan đến chương trình
  - Đảm bảo giá trị gán cho các biến luôn nằm trong vùng kiểm soát
  - *Do not trust anything; verify everything*
- Lỗi liên quan đến người dùng
  - Đừng cho rằng người dùng luôn thực hiện đúng các thao tác theo chỉ dẫn, hãy kiểm tra mọi thao tác của họ
- Lỗi liên quan đến các kỹ thuật phòng ngừa!
  - Mã nguồn cài đặt các kỹ thuật phòng ngừa cũng có khả năng gây lỗi, kiểm tra kỹ phần này



# Các **giai đoạn** lập trình phòng ngừa

- Lập kế hoạch thực hiện công việc:
  - *Dành thời gian để kiểm tra và gỡ rối chương trình cẩn thận : hoàn thành chương trình trước ít nhất 3 ngày so với hạn nộp*
- Thiết kế chương trình:
  - *Thiết kế giải thuật trước khi viết bằng ngôn ngữ lập trình cụ thể*
- Giữ vững cấu trúc chương trình:
  - *Viết và kiểm thử từng phần chương trình: phần chương trình nào dùng để làm gì*
  - *Viết và kiểm thử mối liên kết giữa các phần trong chương trình: quy trình nghiệp vụ như thế nào*
  - *Phòng ngừa bằng các điều kiện trước và sau khi gọi mỗi phần chương trình: điều gì phải đúng trước khi gọi chương trình, điều gì xảy ra sau khi chương trình thực hiện xong*
  - *Dùng chú thích để miêu tả cấu trúc chương trình khi viết chương trình*

# Kiểm tra cái gì, khi nào?

- Testing: chỉ ra các vấn đề làm chương trình không chạy
- Kiểm tra theo cấu trúc của chương trình: Kiểm tra việc thực hiện các nhiệm vụ đặt ra cho từng phần chương trình
  - Ví dụ: điều gì xảy ra với chương trình căn lề văn bản, nếu hàm `ReadWord()` bị lỗi ?
- Nếu chương trình không có tham số đầu vào, mà chỉ thực thi nhiệm vụ và sinh ra kết quả thì không cần kiểm tra nhiều. Hầu hết chương trình đều không như vậy
  - Ví dụ: điều gì xảy ra với chương trình căn lề văn bản, nếu
    - ▶ Không nhập đầu vào ?
    - ▶ Đầu vào không phải là xâu/file chứa các từ hay chữ cái đúng quy định ?



# Kiểm soát lỗi có thể xảy ra

- Error handling: xử lý các lỗi mà ta dự kiến sẽ xảy ra
- Tùy theo tình huống cụ thể, ta có thể trả về:
  - *một giá trị trung lập*
  - *thay thế đoạn tiếp theo của dữ liệu hợp lệ*
  - *trả về cùng giá trị như lần trước*
  - *thay thế giá trị hợp lệ gần nhất*
  - *ghi vết một cảnh báo vào tệp*
  - *trả về một mã lỗi*
  - *gọi một thủ tục hay đối tượng xử lý*
  - *hiện một thông báo hay tắt máy*

# Chắc chắn hay chính xác?

- **Chắc chắn:** chương trình luôn chạy thông, kể cả khi có lỗi
  - **Chính xác:** chương trình không bao giờ gặp lại lỗi
  - Ví dụ: Lỗi hiển thị trong các trình xử lý văn bản: khi đang thay đổi nội dung văn bản, thỉnh thoảng một phần của một dòng văn bản ở phía dưới màn hình bị hiển thị sai. Khi đó người dùng phải làm gì?
    - *Tắt chương trình*
    - *Nhấn PgUp hoặc PgDn, màn hình sẽ làm mới*
- Ưu tiên tính chắc chắn thay vì tính chính xác:
- *Bất cứ kết quả nào đó bao giờ cũng thường là tốt hơn so với Shutdown.*

# Khi nào phải loại bỏ hết lỗi

- Đôi khi, để loại bỏ một lỗi nhỏ, lại rất tốn kém
  - *Nếu lỗi đó chắc chắn không ảnh hưởng đến mục đích cơ bản của ứng dụng, không làm chương trình bị treo, hoặc làm sai lệch kết quả chính, người ta có thể bỏ qua, mà không cố sửa để có thể gặp phải các nguy cơ khác.*
- Phần mềm “chịu lỗi”?: Phần mềm sống chung với lỗi, để đảm bảo tính liên tục, ổn định



2.

## Phòng ngừa sai sót về dữ liệu

Kiểm tra tham số đầu vào, điều kiện biên, tràn số...

# Kiểm tra tham số đầu vào

- Một phần chương trình chạy thông một lần không có nghĩa là lần tiếp theo nó sẽ chạy thông.
- Chương trình trả ra kết quả đúng với đầu vào 'n' không có nghĩa là nó sẽ trả ra kết quả đúng với đầu vào 'm' <> 'n'.
- Vậy chương trình có thực sự chạy thông không ?
  - *Với bất cứ đầu vào nào chương trình cũng phải chạy thông, không bị “crash”. Nếu có lỗi thì chương trình phải dừng và thông báo lỗi*
  - *Bạn có thể biết chương trình có chạy thông hay không khi kiểm tra chương trình bằng các tham số đầu vào sai*

# Tham số đầu vào **sai**

- Trong thực tiễn: “Garbage in, garbage out.” – GIGO
- Trong lập trình, “rác vào → rác ra” là dấu hiệu của những chương trình tồi, không an toàn
- Với một chương trình tốt thì:
  - *rác vào → không có gì ra*
  - *rác vào → có thông báo lỗi*
  - *không cho phép rác vào*

# Phòng ngừa lỗi tham số vào

- Kiểm tra giá trị đầu vào
  - *Kiểm tra giá trị của tất cả các tham số truyền vào các hàm*
  - *Kiểm tra dữ liệu nhập từ nguồn ngoài khác*
- Quyết định kiểm soát đầu vào không hợp lệ
  - *Khi phát hiện một tham số hay một dữ liệu không hợp lệ, cần làm gì với nó?*
    - ▶ Chọn một trong các phương án phù hợp tình huống thực tế



# Phòng ngừa lỗi tham số vào

- Kiểm tra giá trị của mọi dữ liệu từ nguồn bên ngoài
  - *Khi nhận dữ liệu từ file, bàn phím, mạng, hoặc từ các nguồn ngoài khác, hãy kiểm tra để đảm bảo rằng dữ liệu nằm trong giới hạn cho phép.*
  - *Hãy đảm bảo rằng giá trị số nằm trong dung sai và xâu phải đủ ngắn để xử lý*
    - ▶ Nếu một chuỗi cần trong một phạm vi giới hạn của các giá trị (như một ID giao dịch tài chính...), hãy chắc chắn rằng chuỗi đầu vào là hợp lệ cho mục đích của nó; nếu không từ chối.
  - *Với ứng dụng bảo mật, hãy đặc biệt lưu ý đến những dữ liệu có thể tấn công hệ thống: Cố làm tràn bộ nhớ, injected SQL commands, injected html hay XML code, tràn số ...*

# Ví dụ

Chương trình tính giá trị trung bình

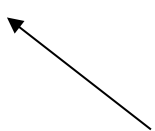
```
double avg (double a[], int n)
/* a là mảng gồm n số kiểu doubles */
{
    int i;
    double sum= 0;
    for (i= 0; i < n; i++) {
        sum+= a[i];
    }
    return sum/n;
}
```

# Phòng ngừa lỗi tham số vào

- Trong một số trường hợp, phải viết thêm các đoạn mã nguồn để lọc giá trị đầu vào trước khi tính toán

```
void class_of_degree (char degree[], double percent)
/* Xếp hạng sinh viên dựa vào tổng điểm tính theo % */
{
    if (percent < 0 || percent > 100)
        strcpy(degree, "Error in mark");
    else if (percent >= 70)
        strcpy(degree, "First");
    else if (percent >= 60)
        strcpy(degree, "Two-one");
    ....
}
```

Thêm các  
dòng này vào  
để báo lỗi



# Kiểm tra điều kiện biên

- Điều gì xảy ra nếu giá trị đầu vào quá lớn hay quá nhỏ?
- Hãy chắc chắn là chương trình của bạn có thể đối phó với các tham số đầu vào kiểu này
- Luôn kiểm tra trường hợp “divide by zero error”

# Tràn số Overflow

## •Arian 5

- Chi phí phát triển: 7 tỷ USD
- Phụ kiện hàng hóa đi kèm : 370 triệu USD
- Thực hiện chuyển đổi 64 bit dấu phẩy động sang 16 bit số nguyên:
  - Việc chuyển đổi không thành công do tràn số
- 04/06/1996: 37 giây sau khi phóng, nổ ở độ cao 3700m



# Tràn số Overflow

- Nếu cần tính toán với các số lớn, hãy chắc chắn là bạn biết giá trị lớn nhất mà biến bạn dùng có khả năng lưu trữ
- Ví dụ:
  - Với phần lớn trình dịch C, một *unsigned char* có giá trị từ 0 đến 255.
  - Kích thước tối đa của một biến kiểu *int* có thể thay đổi

# 3. Xử lý lỗi

Error handling





# Lỗi Error

- Khi lỗi xảy ra cần
  - *Định vị nguồn gây lỗi*
  - *Kiểm soát lỗi*
- Luôn có ý thức đề phòng các lỗi hay xảy ra trong chương trình, nhất là khi đọc file, dữ liệu do người dùng nhập vào và cấp phát bộ nhớ.
- Áp dụng các biện pháp phòng ngừa ngay cả khi điều đó có thể dẫn tới việc dừng chương trình
- In các lỗi bằng stderr stream.

```
fprintf (stderr, "There is an error!\n");
```

# Kiểm tra cái gì để phát hiện lỗi

- Kiểm tra mọi thao tác có thể gây lỗi khi viết chương trình
  - *Nhập dữ liệu*
  - *Sử dụng dữ liệu*
- Ví dụ:
  - *Kiểm tra mỗi lần mở một tệp tin hay cấp phát các ô nhớ.*
  - *Kiểm tra các phương thức người dùng nhập dữ liệu vào cho đến khi không còn nguy cơ gây ra dừng chương trình*
  - *Trong trường hợp tràn bộ nhớ (out of memory), nên in ra lỗi kết thúc chương trình (-1: error exit);*
  - *Trong trường hợp dữ liệu do người dùng đưa vào bị lỗi, tạo cơ hội cho người dùng nhập lại dữ liệu (lỗi tên file cũng có thể do người dùng nhập sai)*

# Kiểm soát lỗi có thể xảy ra

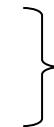
- Error handling: xử lý các lỗi mà ta dự kiến sẽ xảy ra
- Tùy theo tình huống cụ thể, ta có thể trả về:
  - *một giá trị trung lập*
  - *thay thế đoạn tiếp theo của dữ liệu hợp lệ*
  - *trả về cùng giá trị như lần trước*
  - *thay thế giá trị hợp lệ gần nhất*
  - *ghi vết một cảnh báo vào tệp*
  - *trả về một mã lỗi*
  - *gọi một thủ tục hay đối tượng xử lý*
  - *hiện một thông báo hay tắt máy*

# Bắt ngoại lệ

- Exception: bắt các tình huống bất thường và phục hồi chúng về trạng thái trước đó.
- Giúp chương trình đáng tin cậy hơn, tránh kết thúc bất thường
- Tách biệt khối lệnh có thể gây ngoại lệ và khối lệnh xử lý ngoại lệ

```
.....  
IF B IS ZERO GO TO ERROR  
C = A/B  
PRINT C  
GO TO EXIT
```

```
ERROR:  
  DISPLAY "DIVISION BY  
  ZERO"
```



Khối xử lý lỗi

```
EXIT:  
  END
```

# Ví dụ

## Bắt ngoại lệ

- C#

```
try
{
    cmd.ExecuteNonQuery();
    ErrorsManager.SetError(ErrorIDs.KhongCoLoi);
}
catch
{
    ErrorsManager.SetError(ErrorIDs.SQLThatBai,
                           database.DbName, "ten_strore");
}
```

- VB.NET

```
Try
    Return
    CBO.FillCollection(CType(SqlHelper.ExecuteReader(ConStr, "TimHDon",
iSoHoaDon), IDataReader), GetType(ThanhToan.ChiTietHDInfo))
    Catch ex As Exception
        messagebox.show(ex.message)
    End Try
```

# Phục hồi tài nguyên

- Phục hồi tài nguyên khi xảy ra lỗi?
  - Thường thì không phục hồi tài nguyên, nhưng sẽ hữu ích khi thực hiện các công việc nhằm đảm bảo cho thông tin ở trạng thái rõ ràng và vô hại nhất có thể
  - Nếu các biến vẫn còn được truy xuất thì chúng nên được gán các giá trị hợp lý
  - Trường hợp thực thi việc cập nhật dữ liệu, nhất là trong 1 phiên – transaction – liên quan tới nhiều bảng chính, phụ, thì việc khôi phục khi có ngoại lệ là vô cùng cần thiết (rollback)

# Chắc chắn hay chính xác?

- **Chắc chắn:** chương trình luôn chạy thông, kể cả khi có lỗi
  - **Chính xác:** chương trình không bao giờ gặp lại lỗi
  - Ví dụ: Lỗi hiển thị trong các trình xử lý văn bản: khi đang thay đổi nội dung văn bản, thỉnh thoảng một phần của một dòng văn bản ở phía dưới màn hình bị hiển thị sai. Khi đó người dùng phải làm gì?
    - *Tắt chương trình*
    - *Nhấn PgUp hoặc PgDn, màn hình sẽ làm mới*
- Ưu tiên tính chắc chắn thay vì tính chính xác:
- *Bất cứ kết quả nào đó bao giờ cũng thường là tốt hơn so với Shutdown.*



# Khi nào phải loại bỏ hết lỗi

- Đôi khi, để loại bỏ một lỗi nhỏ, lại rất tốn kém
  - *Nếu lỗi đó chắc chắn không ảnh hưởng đến mục đích cơ bản của ứng dụng, không làm chương trình bị treo, hoặc làm sai lệch kết quả chính, người ta có thể bỏ qua, mà không cố sửa để có thể gặp phải các nguy cơ khác.*
- Phần mềm “chịu lỗi”?: Phần mềm sống chung với lỗi, để đảm bảo tính liên tục, ổn định

4.

# Bảo đảm

Assertion



# Bảo đảm

## Assertions

- **Assertion:** một macro hay một chương trình con dùng trong quá trình phát triển ứng dụng, cho phép chương trình tự kiểm tra khi chạy.
- Return true >> OK, false >> có một lỗi gì đó trong chương trình.
- Ghi lại những giả thiết được đưa ra trong code
- Loại bỏ những điều kiện không mong đợi

### Ví dụ

- *Nếu hệ thống cho rằng file dữ liệu về khách hàng không bao giờ vượt quá 50 000 bản ghi, chương trình có thể chứa một assertion rằng số bản ghi là  $\leq 50\,000$ .*
- *Khi mà số bản ghi  $\leq 50,000$ , assertion sẽ không có phản ứng gì.*
- *Nếu đếm đc hơn 50 000 bản ghi, nó sẽ lớn tiếng “khẳng định” rằng có một lỗi trong chương trình*

# Bảo đảm

## Assertions

- Assertions có thể được dùng để kiểm tra các giả thiết như :
  - *Các tham số đầu vào nằm trong phạm vi mong đợi (tương tự với các tham số đầu ra)*
  - *File hay stream đang được mở (hay đóng) khi một CTC bắt đầu thực hiện (hay kết thúc)*
  - *một file hay stream đang ở bản ghi đầu tiên (hay cuối cùng) khi một CTC bắt đầu ( hay kết thúc) thực hiện*
  - *một file hay stream được mở để đọc, để ghi, hay cả đọc và ghi*
  - *Giá trị của một tham số đầu vào là không thay đổi bởi một CTC*
  - *một pointer là non-NULL*
  - *một mảng đc truyền vào CTC có thể chứa ít nhất X phần tử*
  - *một bảng đã đc khởi tạo để chứa các giá trị thực*
  - *một danh sách là rỗng (hay đầy) lkhi một CTC bắt đầu (hay kết thúc) thực hiện*

# **Bảo đảm**

## **Assertions**

- End users không cần thấy các thông báo của assertion ;
- Assertions chủ yếu đc dùng trong quá trình phát triển hay bảo dưỡng ứng dụng.
- Dịch thành code khi phát triển, loại bỏ khỏi code trong sản phẩm để nâng cao hiệu năng của chương trình
- Rất nhiều NNLT hỗ trợ assertions : C++, Java và Visual Basic.
- Kể cả khi NNLT không hỗ trợ, thì cũng có thể dễ dàng xây dựng

**Ví dụ**

**Assert tự viết**

```
#define ASSERT(condition, message)
{
    if ( !(condition) ) {
        fprintf(
            stderr,
            "Assertion %s failed: %s\n",
            condition,
            message);
        exit( EXIT_FAILURE );
    }
}
```

# Sử dụng assertions

- Bẫy lỗi cho những tình huống lường trước (sự kiện ta chờ đợi sẽ xảy ra);
  - *Error-handling : checks for bad input data → Hướng tới việc xử lý lỗi*
- Dùng assertions cho các tình huống không lường trước (sự kiện không mong đợi xảy ra hoặc không bao giờ xảy ra)
  - *Assertions : check for bugs in the code → hướng đến việc hiệu chỉnh chương trình, tạo ra phiên bản mới của chương trình*
- Tránh đưa code xử lý vào trong assertions
  - *Điều gì xảy ra khi ta turn off the assertions ?*



# Sử dụng assertions

- Các chương trình lớn:
  - *trước tiên xác nhận lỗi (dùng assertion),*
  - *sau đó bắt lỗi (dùng error-handling)*
- Nguyên nhân gây lỗi đã được xác định:
  - *hoặc dùng assertion, hoặc dùng error-handling,*
  - *không dùng cả 2 cùng lúc*
- Các chương trình cực lớn, nhiều người cùng phát triển trong thời gian 5-10 năm, hoặc hơn nữa?
  - *Cả assertions và error handling code có thể đc dùng cho cùng một lỗi.*
  - *Ví dụ trong source code cho Microsoft Word, những điều kiện luôn trả về true thì đc dùng assertion, nhưng đồng thời cũng đc xử lý.*
  - *Assertions rất có lợi vì nó giúp loại bỏ rất nhiều lỗi trong quá trình phát triển hệ thống*



# Thanks!

Any questions?

Email me at [trungtt@soict.hust.edu.vn](mailto:trungtt@soict.hust.edu.vn)