Chương 6

Kiểm thử đơn vị

Nội dung

- Kiểm thử đơn vị
- Thiết kế *test-case*
- Kiểm thử tăng tiến (*Increment testing*)
- Kiểm thử từ trên xuống (*Top-Down testing*)
- Kiểm thử từ dưới lên (*Bottom-up testing*)

Kiểm thử đơn vị

- ***Kiểm thử đơn vị*** **(*unit testing, module testing*) là một phương pháp kiểm thử phần mềm bằng cách các đơn vị (*unit*) riêng biệt của mã nguồn, tập hợp các đơn thể (*module*) của chương trình cùng với các dữ liệu liên quan, các thủ tục thường dùng và các thủ tục hoạt động sẽ được kiểm tra để thỏa mãn yêu cầu sử dụng.**

Kiểm thử đơn vị

Mục tiêu của kiểm thử đơn vị

- Bảo đảm đoạn mã lệnh thỏa mãn thiết kế của nó.
- So sánh chức năng của đơn thể với một số đặc tả chức năng và đặc tả giao tiếp khi định nghĩa đơn thể.
  - Không cho thấy đơn thể thỏa mãn đặc tả của nó.
  - Cho thấy đơn thể không mâu thuẫn với đặc tả của nó.

Kiểm thử đơn vị

- **Đơn vị (*unit*)**
  - Có thể xem đơn vị là một phần kiểm thử nhỏ nhất của chương trình.
  - Trong lập trình thủ tục (*procedural programming*), đơn vị là toàn bộ một đơn thể, nhưng thông thường là một hàm (*function*) hoặc một thủ tục (*procedure*) riêng biệt.
  - Trong lập trình hướng đối tượng (*object-oriented programming*), đơn vị thường là toàn bộ một giao tiếp (*interface*), ví dụ một lớp (*class*), nhưng thông thường là một phương thức (*method*) riêng biệt.
  - Một kiểm thử đơn vị (*unit test*) là một đoạn mã lệnh ngắn do người lập trình (*programmer*) hoặc đôi khi do những người kiểm thử hộp trắng (*white box tester*) viết trong quá trình phát triển phần mềm.

Kiểm thử đơn vị

- **Động cơ của kiểm thử đơn vị**
  - Kiểm thử đơn vị là cách quản lý các đơn vị cần kiểm thử kết hợp với nhau, bắt đầu từ các đơn vị nhỏ của chương trình.
  - Kiểm thử đơn vị làm cho việc chẩn đoán chương trình được dễ dàng.
  - Kiểm thử đơn vị có thể song song hóa quá trình kiểm thử chương trình, kiểm thử đồng thời nhiều đơn thể.
- **Các vấn đề của kiểm thử đơn vị**
  - Cách thức thiết kế các *test-case*
  - Thứ tự các đơn thể sẽ được kiểm thử và được tích hợp
  - Lời khuyên về việc thực hiện kiểm thử

Thiết kế *test-case*

- **Các thông tin cần thiết để thiết kế *test-case***
  - **Đặc tả của đơn thể**
    - **Đặc tả của đơn thể chủ yếu là xác định các tham số vào / ra (*input / output parameter*) của đơn thể và chức năng của nó.**
  - **Mã nguồn của đơn thể**

Thiết kế *test-case*

- **Kiểm thử đơn vị hướng đến kiểm thử hộp trắng trên qui mô lớn.**
- **Các lý do**
  - **Khi kiểm thử các thành phần lớn hơn, ví dụ toàn bộ một chương trình (là trường hợp của các quá trình kiểm thử tiếp theo), kiểm thử hộp trắng là kém khả thi.**
  - **Các quá trình kiểm thử tiếp theo hướng đến việc tìm ra các loại lỗi khác nhau (ví dụ các lỗi không nhất thiết liên quan đến luận lý của chương trình, chẳng hạn lỗi do không thỏa mãn các yêu cầu của người sử dụng).**

Thiết kế *test-case*

- **Qui trình thiết kế *test-case***
  - Bước 1: Phân tích luận lý của đơn thể bằng cách sử dụng một hoặc nhiều kỹ thuật kiểm thử hộp trắng.
  - Bước 2: Thiết kế các *test-case* bằng cách áp dụng các kỹ thuật hộp đen cho đặc tả của đơn thể.
- **Lưu ý**
  - Thiết kế một tập các *test-case* hiệu quả.
  - Cách thức mà các đơn thể được tích hợp để tạo ra chương trình.
    - Thiết kế các *test-case* cho các đơn thể nào.
    - Sử dụng các loại công cụ kiểm thử nào.
    - Thứ thự các đơn thể được lập trình và được kiểm thử.
    - Chi phí tạo ra các *test-case*.
    - Chi phí của việc chẩn đoán (bao gồm việc tìm và sửa các lỗi)

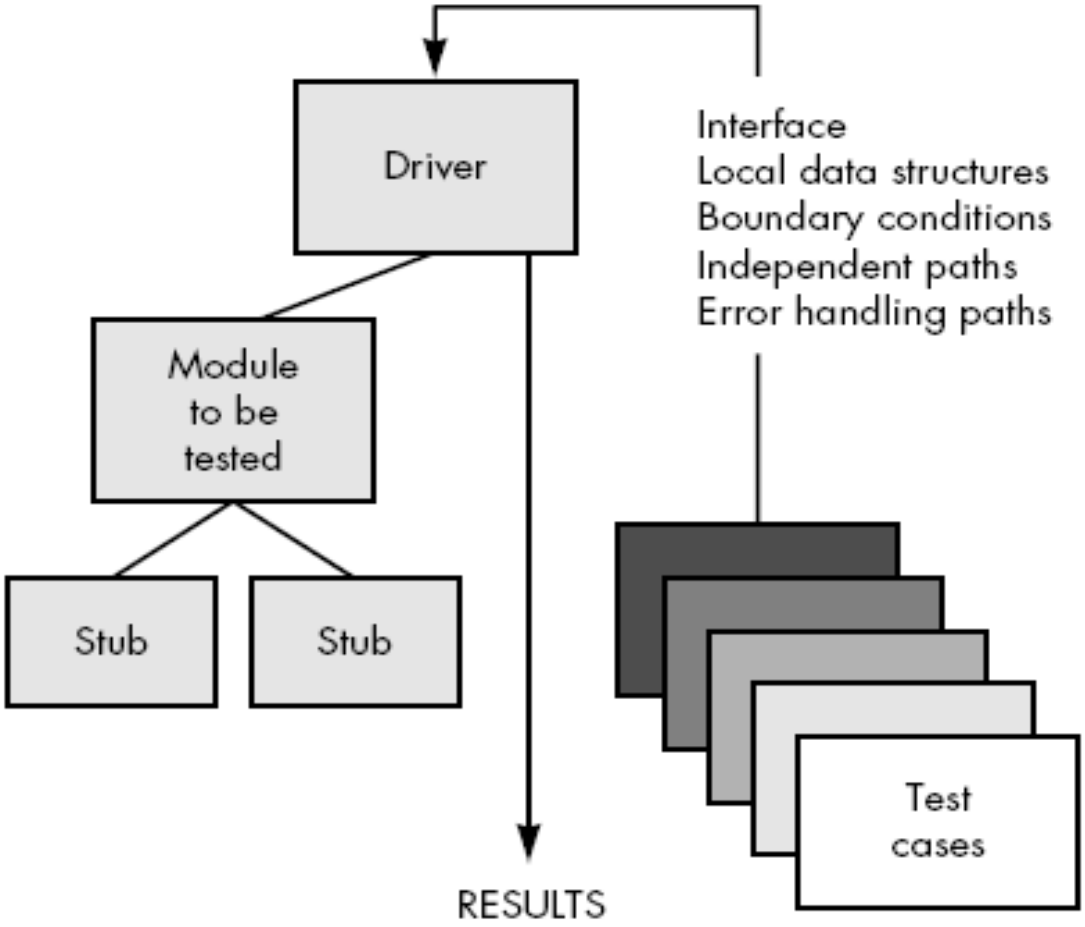
Kiểm thử tăng tiến và không tăng tiến

- **Kiểm thử không tăng tiến (*non-incremental testing*)**
  - **Kiểm thử mỗi đơn thể một cách độc lập, sau đó tích hợp các đơn thể này để tạo ra chương trình.**
- **Kiểm thử tăng tiến (*incremental testing*)**
  - **Tích hợp đơn thể cần kiểm thử kế tiếp vào các đơn thể đã được kiểm thử trước đó, sau đó kiểm thử các đơn thể này chạy chung với nhau.**

Kiểm thử không tăng tiến

- Kiểm thử đơn vị được thực hiện trên mỗi đơn thể.
- Mỗi đơn thể được kiểm thử một cách riêng biệt (6 đơn thể).
- Cuối cùng, các đơn thể này được tích hợp để tạo ra chương trình.
- Việc kiểm thử mỗi đơn thể cần có một đơn thể *driver* đặc biệt và một hoặc nhiều đơn thể *stub*.
- ***Driver*** là một đơn thể nhỏ được viết ra (đơn thể gọi) để truyền các *test-case* cho đơn thể cần kiểm thử (đơn thể bị gọi) và hiển thị kết quả của đơn thể cần kiểm thử.
- ***Stub*** là một đơn thể được viết ra để giả lập chức năng của một đơn thể bị gọi mà đơn thể cần kiểm thử là đơn thể gọi.

Kiểm thử không tăng tiến

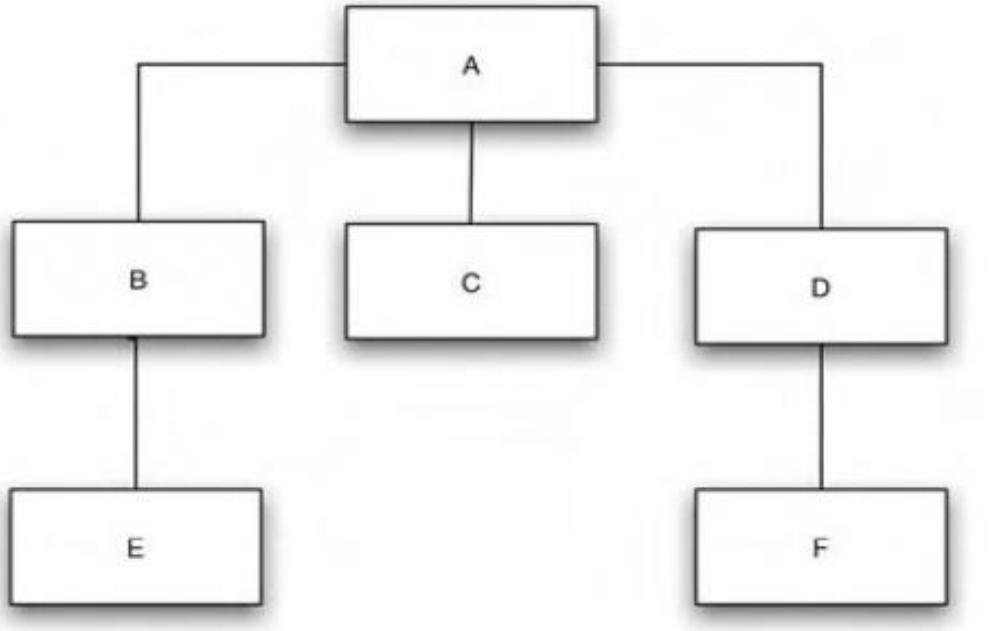


Kiểm thử tăng tiến

- Có thể bắt đầu từ đơn thể trên cùng (*top*) hoặc đơn thể dưới cùng (*bottom*) của chương trình.
- **Ví dụ:** bắt đầu từ đơn thể dưới cùng
  - **Bước 1:** Kiểm thử các đơn thể *E*, *C*, *F* một cách tuần tự hoặc đồng thời (cần ba người kiểm thử). Chuẩn bị một *driver* cho mỗi đơn thể, không có *stub*.
  - **Bước 2:** Kiểm thử các đơn thể *B* và *D*. *B* được tích hợp với *E* và *D* được tích hợp với *F*. Khi kiểm thử *B*, cần phải viết một *driver* cùng với các *test-case* để kiểm thử cặp *B-E*.
  - **Bước 3:** Kiểm thử đơn thể *A* (trên cùng).

Kiểm thử tăng tiến

Sample six-module program.



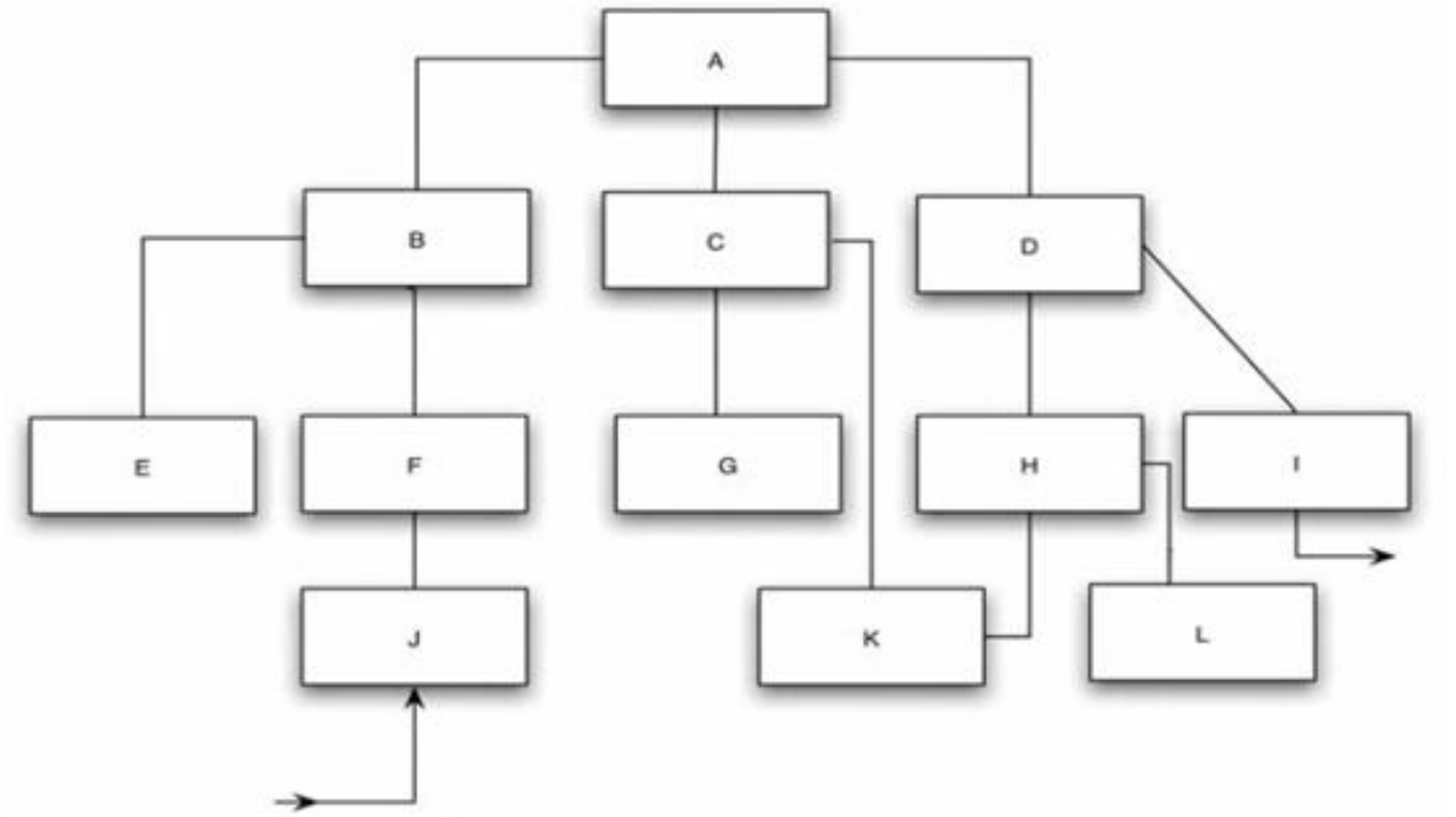
Kiểm thử tăng tiến và không tăng tiến

- **Nhận xét**
  - **Đối với kiểm thử tăng tiến**
    - Cần nhiều công sức hơn so với kiểm thử không tăng tiến.
    - Các lỗi liên quan đến giao tiếp không đúng giữa các đơn thể sẽ được phát hiện sớn hơn.
    - Việc chẩn đoán được dễ dàng hơn đối với các lỗi liên quan đến giao tiếp giữa các đơn thể.
    - Việc kiểm thử được thực hiện trọn vẹn hơn, phát hiện hiệu ứng lề (*side effect*) xảy ra khi chạy các đơn thể chung với nhau.

Kiểm thử tăng tiến và không tăng tiến

- Nhận xét
  - **Đối với kiểm thử không tăng tiến**
    - **Thời gian kiểm thử nhanh hơn vì có thể kiểm thử đồng thời các đơn thể một cách độc lập nhau.**
    - **Khi kiểm thử một đơn thể thì chỉ cần chạy một *driver* và đơn thể này. Do đó, số hàng lệnh được thực hiện ít hơn.**
    - **Ở đầu giai đoạn kiểm thử đơn thể, có nhiều cơ hội để kiểm thử đồng thời các đơn thể, nhất là đối với các dự án lớn có nhiều người và nhiều đơn thể.**

Kiểm thử từ trên xuống



Kiểm thử từ trên xuống

❖ Qui trình kiểm thử từ trên xuống

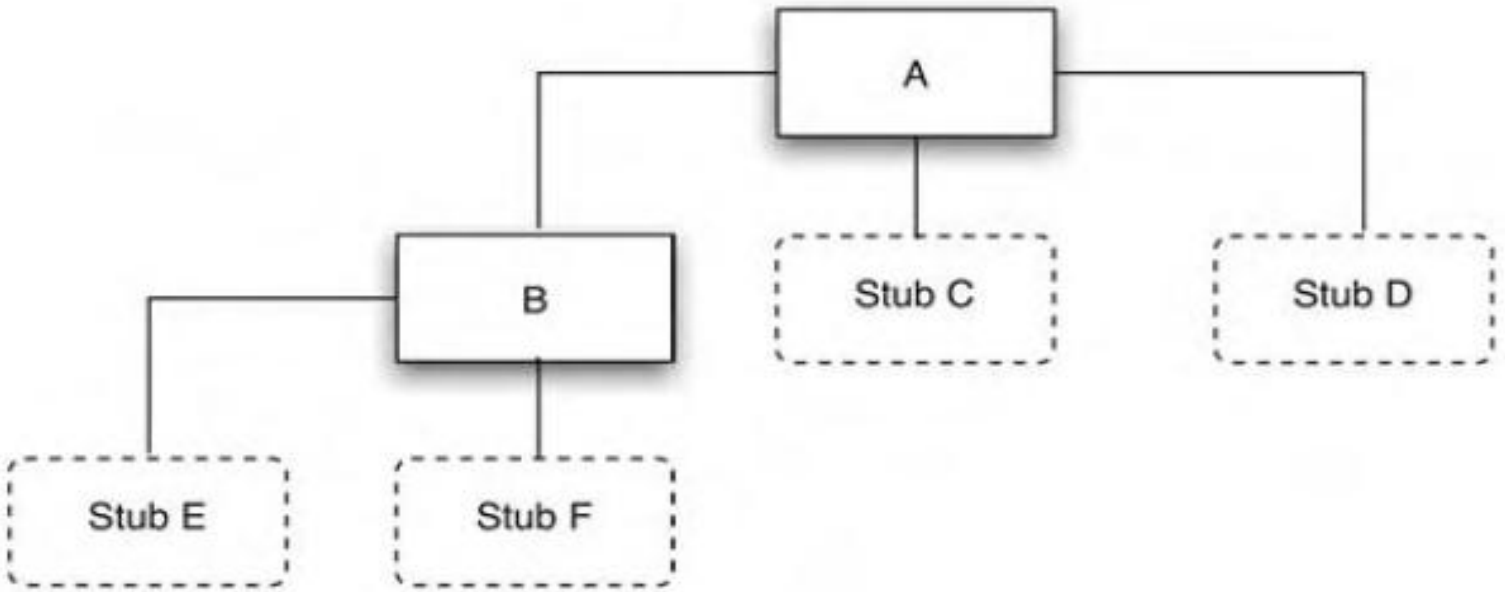
- ▸ **Bước 1: Kiểm thử đơn thể đầu tiên, trên cùng của chương trình.**
  - **Khi kiểm thử đơn thể *A*, phải viết các *stub* cho các đơn thể *B*, *C*, *D*.**
  - **Tạo các *test-case* cho đơn thể *A*. Dữ liệu kiểm thử cho *A* có thể được cung cấp từ các *stub*, khi đó dữ liệu kiểm thử được lưu trữ trong các tập tin và được các *stub* này đọc vào để cung cấp cho *A*.**

Kiểm thử từ trên xuống

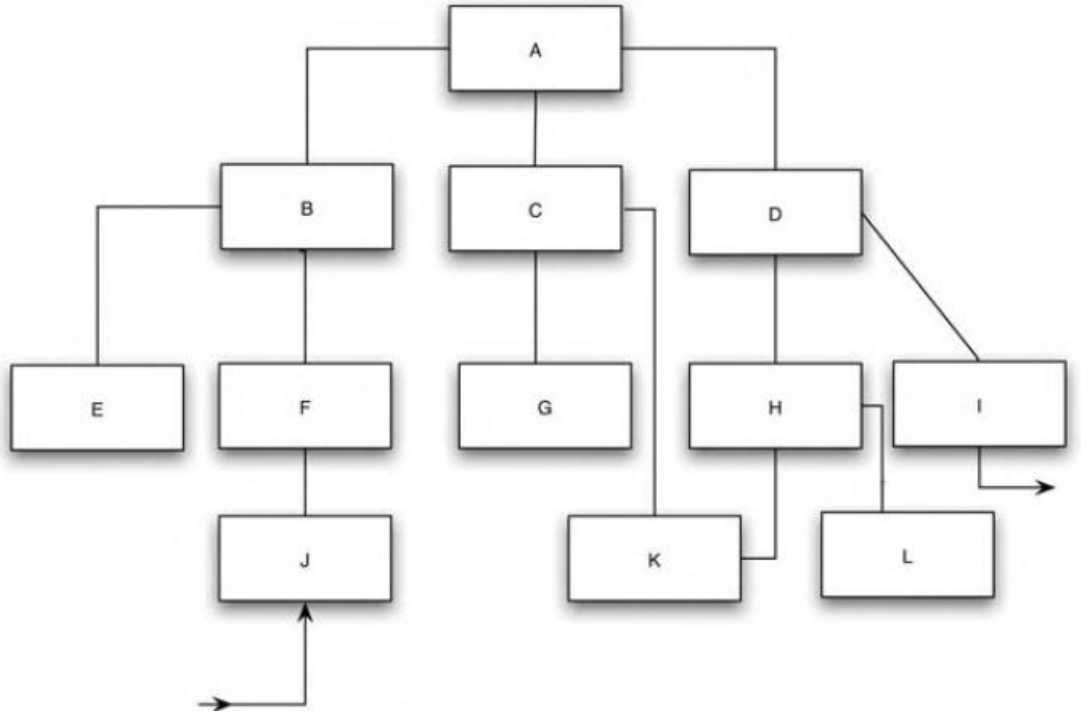
- Qui trình kiểm thử từ trên xuống
  - **Bước 2:** Chọn đơn thể cần kiểm thử kế tiếp.
    - Đơn thể cần kiểm thử kế tiếp là đơn thể được gọi từ đơn thể đã được kiểm thử.
    - Chọn một trong các *stub* và thay thế *stub* này bằng đơn thể thật, sau đó kiểm thử đơn thể thật này.
    - Có thể có nhiều chuỗi các đơn thể khác nhau.

Kiểm thử từ trên xuống

Second step in the top-down test.



Kiểm thử từ trên xuống



1. A B C D E F G H I J K L
2. A B E F J C G K D H L I
3. A D H I K L C G B F J E
4. A B F J D I E C G K H L

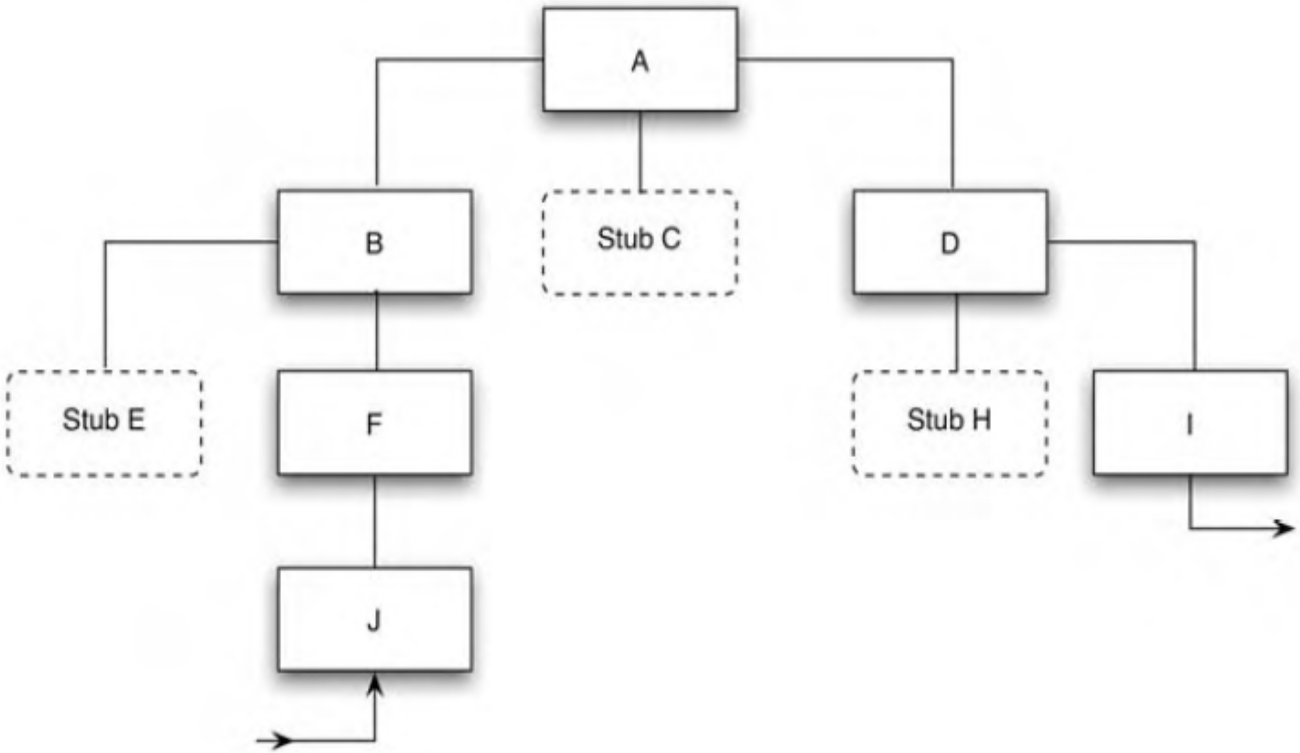
Kiểm thử từ trên xuống

- **Hướng dẫn**
  - Các đơn thể nhập / xuất được kiểm thử càng sớm càng tốt.
  - Các đơn thể quan trọng (dùng giải thuật phức tạp, giải thuật mới hoặc có thể xảy ra nhiều lỗi) được kiểm thử càng sớm càng tốt.
    - Nếu đơn thể *J* và *I* là các đơn thể nhập / xuất và đơn thể *G* là quan trọng, nên chọn chuỗi các đơn thể:

      A B F J D I C G E K H L

Kiểm thử từ trên xuống

Intermediate state in the top-down test.



Kiểm thử từ trên xuống

Top-down Testing

Advantages	*Disadvantages*
1. This is advantageous if major flaws occur toward the top of the program. 2. Once the I/O functions are added, representation of test cases is easier. 3. Early skeletal program allows demonstrations and boosts morale.	1. Stub modules must be produced. 2. Stub modules are often more complicated than they first appear to be. 3. Before the I/O functions are added, the representation of test cases in stubs can be difficult. 4. Test conditions may be impossible, or very difficult, to create. 5. Observation of test output is more difficult. 6. It allows one to think that design and testing can be overlapped. 7. It induces one to defer completion of the testing of certain modules.

Kiểm thử từ dưới lên

❖ Qui trình kiểm thử từ dưới lên

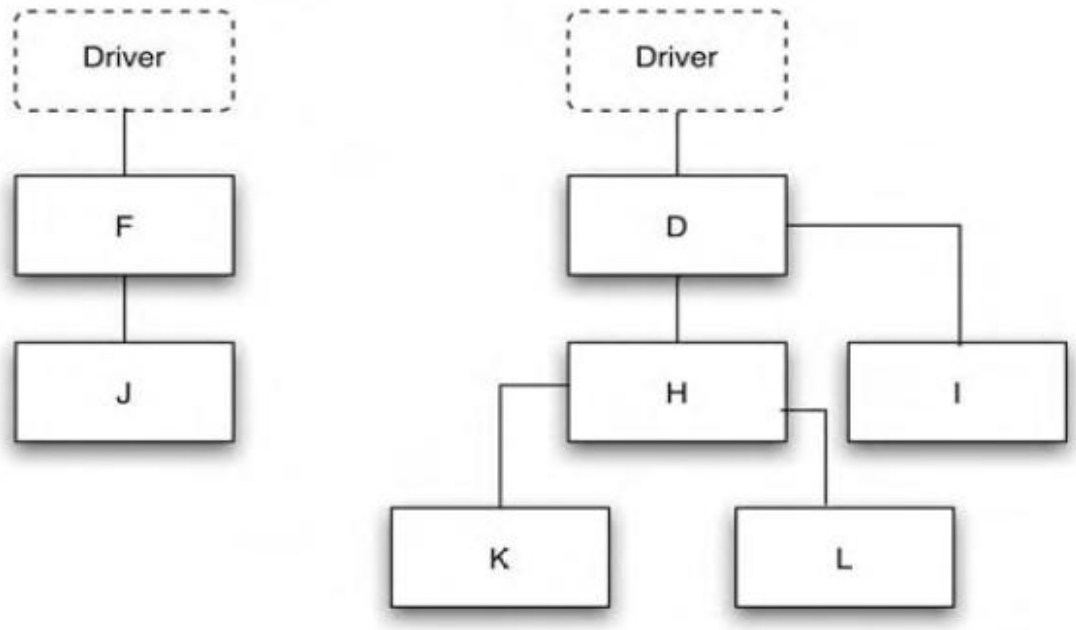
- **Bước 1:** Kiểm thử các đơn thể dưới cùng của chương trình, là đơn thể không gọi đơn thể khác.
  - Kiểm thử một số hoặc tất cả các đơn thể *E*, *I*, *G*, *K*, *L*, *I* một cách tuần tự hoặc song song.
  - Mỗi đơn thể cần kiểm thử cần có một *driver*: nhận dữ liệu kiểm thử, gọi đơn thể cần kiểm thử và hiển thị kết quả hoặc so sánh kết quả hiện tại với kết quả mong muốn.
  - Không cần phải viết nhiều *driver* khác nhau cho một đơn thể cần kiểm thử.

Kiểm thử từ dưới lên

- Qui trình kiểm thử từ dưới lên
  - **Bước 2:** Chọn đơn thể cần kiểm thử kế tiếp.
    - Đơn thể cần kiểm thử kế tiếp là đơn thể gọi trực trực tiếp các đơn thể đã được kiểm thử.
    - Các đơn thể nhập / xuất được kiểm thử càng sớm càng tốt.
    - Các đơn thể quan trọng (dùng giải thuật phức tạp, giải thuật mới hoặc có thể xảy ra nhiều lỗi) được kiểm thử càng sớm càng tốt.
    - Nếu hai đơn thể *D* và *F* là quan trọng nhất thì kiểm thử các đơn thể này.

Kiểm thử từ dưới lên

Intermediate state in the bottom-up test.



Kiểm thử từ dưới lên

Bottom-up Testing	
Advantages	*Disadvantages*
1. This is advantageous if major flaws occur toward the bottom of the program. 2. Test conditions are easier to create. 3. Observation of test results is easier.	1. Driver modules must be produced. 2. The program as an entity does not exist until the last module is added.