

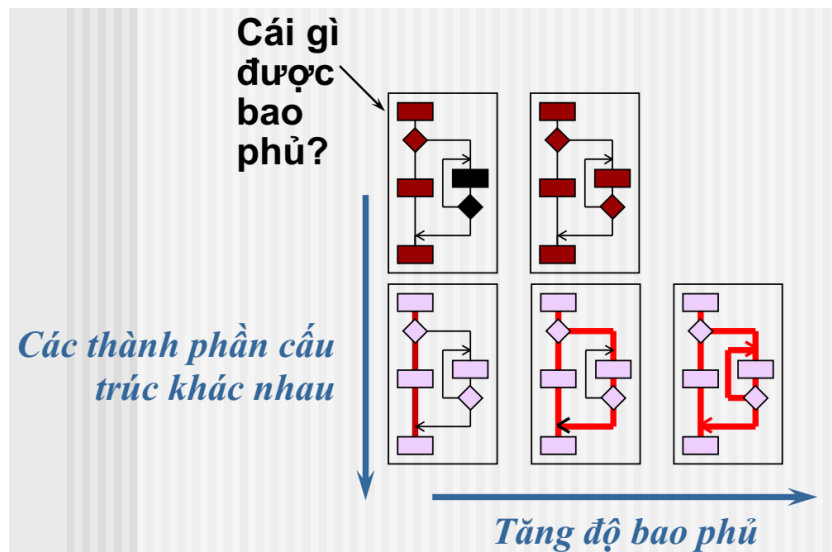
Chương IV. CÁC KỸ THUẬT THIẾT KẾ TEST P.2

IV.4. Kiểm thử hộp trắng (White-box testing)

- Là phương pháp kiểm thử dựa trên **cấu trúc logic của PM.**
- Vì có thể nhìn thấy mọi thứ bên trong phần mềm → Ta có thể kiểm thử dựa trên **source code** → Người kiểm thử chủ yếu là **Dev**, hoặc **Tester phải nắm về code.**
- **Phân loại:**
 - **Kiểm thử dựa trên luồng điều khiển - Control Flow Testing**
 - *Statement testing* – kiểm thử dòng lệnh
 - *Branch / Decision testing* – kiểm thử nhánh
 - *Branch condition testing* – kiểm thử điều kiện
 - *Branch condition combination testing* – kiểm thử tổ hợp các điều kiện.
 - **Kiểm thử dựa trên luồng dữ liệu - Data flow testing**

1. Kỹ thuật độ bao phủ cấu trúc (Coverage techniques)

- Khi ta kiểm thử dòng lệnh/nhánh cần đo lường độ bao phủ này. Nếu TC chưa đạt độ bao phủ 100%, cần bổ sung các TCs để đạt được tỷ lệ này.
- **Độ bao phủ 100% không có nghĩa là 100% được test**
- **Cái gì được bao phủ?** Dòng lệnh, nhánh (điều kiện).
- Chưa được bao phủ nghĩa là dòng lệnh/nhánh đó chưa được thực thi.

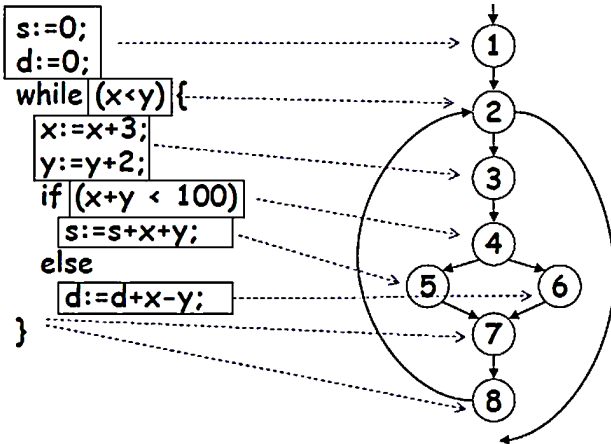


- **Mục tiêu:** Thiết kế **số lượng TCs nhỏ nhất** nhưng vẫn đạt độ bao phủ là 100%

2. Kiểm thử luồng điều khiển (Control Flow Testing):

- Kỹ thuật này sử dụng **đồ thị luồng điều khiển từ source code**. Đồ thị này gồm 2 thành phần: nút & cạnh.
- Sau khi xây dựng đồ thị luồng, ta thiết kế các TCs làm sao đó **bao phủ được tất cả các thành phần của đồ thị**.

Ví dụ 1:



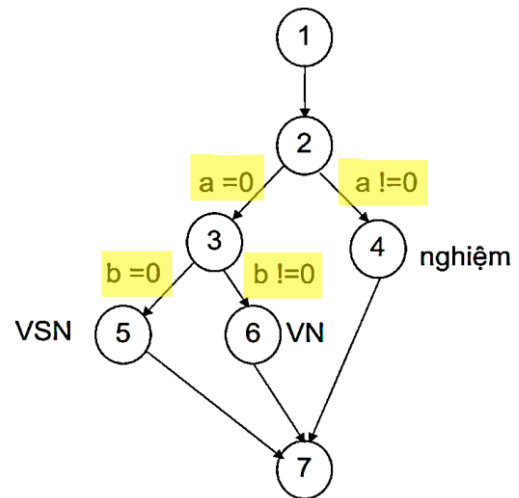
- Có thể gom 2 câu lệnh gán lại thành 1 nút (1).
- Nút điều kiện (4): cần chính xác 2 nhánh đi ra**
- Nút gộp (7) có hay không đều được.
- Đồ thị cần thể hiện rõ vòng lặp** nếu trong source code có vòng lặp (2-8, 8-2).

Ví dụ 2:

Chương trình giải phương trình bậc nhất

- if(a == 0)
- if(b == 0)
- cout<< "Vo so nghiem";
- else
- cout<< "Vo nghiem";
- else
- cout<< "x = "<<-b/a;

* Nút (1) khởi động – có hay không đều được.



- Độ phức tạp "Cyclomatic":** Ta sử dụng 2 trong 3 công thức sau để so khớp KQ

- **$V(G) = E - N + 2$** (số cạnh – số nút + 2)

○ Xét ví dụ 2: Số cạnh = 8, số nút = 7 $\rightarrow V(G) = 8 - 7 + 2 = 3$.

- **$V(G) = P + 1$** (số nút điều kiện) (Nên sử dụng vì tính nhanh nhất)

○ Số nút điều kiện = 2

- **$V(G) = R$** (tính dựa trên số miền)

- Xét lại ví dụ 2: Ta có $V(G) = 3 \rightarrow$ Ta có 3 đường độc lập cơ bản

$\left\{ \begin{array}{l} 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \\ 1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \end{array} \right.$

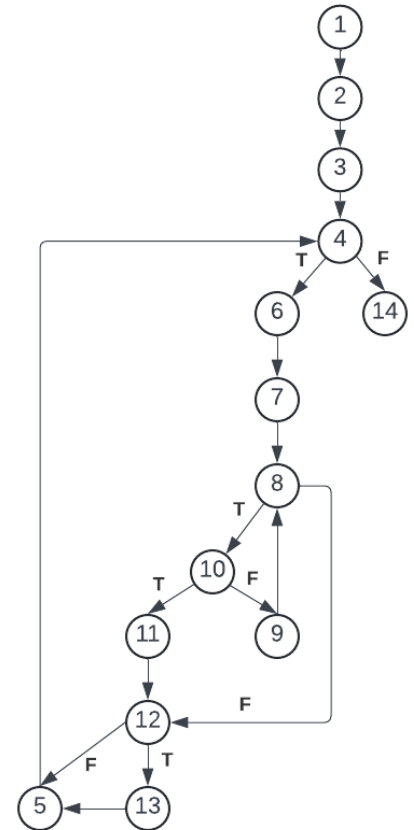
\rightarrow Ta cần thiết kế 3 TCs để bao phủ 3 đường độc lập cơ bản này.

Bài tập: Vẽ đồ thị luồng điều khiển

```

1. void prime(int n) {                               //1
2.     cout<<"Các số nguyên tố nhỏ hơn n là:<<endl; //2
3.     for(int i = 2 /*3*/; i < n /*4*/; i++ /*5*/){
4.         int flag = 1;                             //6
5.         for(int j = 2 /*7*/; j <= sqrt(i) /*8*/; j++ /*9*/){
6.             if(i%j==0)                             //10
7.                 { flag = 0; break;}                //11
8.         } if(flag == 1){                             //12
9.             cout<<i<<"\t";                          //13
10.        }
11.    }
12. } //14

```



3. Bao phủ câu lệnh (Statement coverage)

- Tỉ lệ phần trăm các câu lệnh được thực thi bởi bộ test = **Số câu lệnh được thực thi / Tổng số câu lệnh**
- Thiết kế bộ TC sao cho **mỗi câu lệnh thực thi ít nhất 1 lần.**
- Ví dụ 1:**
 - Chương trình có 100 câu lệnh
 - tests thực thi 87 câu lệnh
 - Bao phủ câu lệnh = 87%
- Ví dụ 2:**

	Test case	Input	Expected output
<div> <div>1</div> <div>cin>>a;</div> </div> <div> <div>2</div> <div>if (a > 6)</div> </div> <div> <div>3</div> <div>b = a;</div> </div> <div> <div>4</div> <div>cout<<b;</div> </div>	1	7	7

Số câu lệnh

Với 1 test case này thì tất cả 4 câu lệnh đều được thực thi → đạt độ bao phủ dòng lệnh 100%

• **Ví dụ 3:**

<pre>Prints (int a, int b) { int result = a+ b; If (result> 0) Print ("Positive", result) Else Print ("Negative", result) }</pre>	<pre>1 Prints (int a, int b) { 2 int result = a+ b; 3 If (result> 0) 4 Print ("Positive", result) 5 Else 6 Print ("Negative", result) 7 }</pre> <p>Với A = 2, B = 5 → phủ 5/7 câu lệnh (71%)</p>	<pre>1 Prints (int a, int b) { 2 int result = a+ b; 3 If (result> 0) 4 Print ("Positive", result) 5 Else 6 Print ("Negative", result) 7 }</pre> <p>Với A = 3, B = -5 → phủ 6/7 câu lệnh (85%)</p>
--	---	--

→ **Tối thiểu 2 test-case** để phủ 100% câu lệnh.

Bài tập 1: Đoạn chương trình giải phương trình bậc nhất $ax+b = 0$

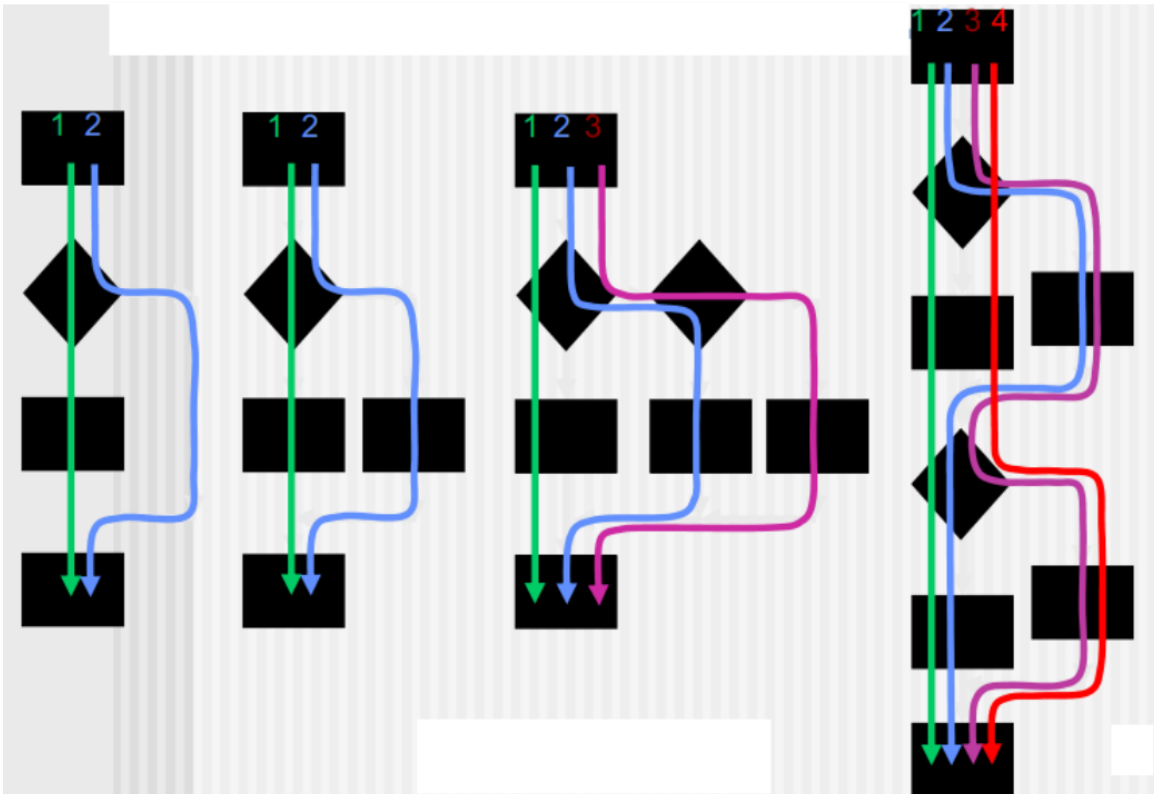
<pre>1. { 2. if(a == 0) 3. if(b == 0) 4. cout<<"Vo so nghiem"; 5. else 6. cout<<"Vo nghiem"; 7. else 8. cout<<"x ="<<-b/a; 9. }</pre>	<p>1. Có bao nhiêu câu lệnh? → 9</p> <p>2. Test case (a=0, b=8) bao phủ bao nhiêu % câu lệnh ? → Các câu lệnh được bao phủ: 1, 2, 3, 5, 6, 9 → 6/9 câu lệnh → 66,67%</p> <p>3. Cần tối thiểu bao nhiêu test case để bao phủ 100% các câu lệnh ?</p> <p>→ 3TCs $\begin{cases} a = 0, b = 8 \\ a = 0, b = 0 \\ a \neq 0, b \neq 0 \end{cases}$</p>
---	---

Bài tập 2:

<pre>1. void func(int A, int B, int X) 2. { 3. if (A > 1 && B == 0) 4. X = X / A; 5. if (A == 2 X > 1) 6. X = X + 1; 7. }</pre>	<p>1. Có bao nhiêu câu lệnh ? → 7</p> <p>2. Test case (A=2,B=1,X=3) bao phủ bao nhiêu % câu lệnh ? → Các câu lệnh được thực thi: 1, 2, 3, 5, 6, 7 → 6/7 câu lệnh (85%)</p> <p>3. Test case (A=3,B=0,X=0) bao phủ bao nhiêu % câu lệnh ? → Các câu lệnh được thực thi: 1, 2, 3, 4, 5, 7 → 6/7 câu lệnh (85%)</p> <p>4. Tối thiểu bao nhiêu test case để bao phủ 100% các câu lệnh ? → 1 TC: A=2, B=0, X bất kỳ</p>
--	--

4. Bao phủ nhánh (Decision/Branch coverage)

- **Tương ứng với mỗi 1 điều kiện có 2 nhánh** → Khi test cần KT hết 2 nhánh T, F.
- **Tỉ lệ phần trăm các nhánh được thực thi bởi bộ test = Số nhánh được thực thi/Tổng số nhánh**



Ví dụ 1: Đoạn chương trình giải phương trình bậc nhất $ax+b = 0$

<pre> 1. { 2. if(a == 0) 3. if(b == 0) 4. cout<<"Vo so nghiem"; 5. else 6. cout<<"Vo nghiem"; 7. else 8. cout<<"x ="<<-b/a; 9. }</pre>	<p>1. Có bao nhiêu nhánh ? → 4.</p> <p>2. Test case (a=0,b=8),(a=0,b=0) bao phủ bao nhiêu % nhánh ? $a=0 \begin{cases} T \\ F \end{cases}; b=0 \begin{cases} T \\ F \end{cases}$</p> <p>→ (a=0,b=8) bao phủ trường hợp: a=0(True), b=0(False)</p> <p>→ (a=0,b=0) bao phủ trường hợp: a=0(True), b=0(True)</p> <p>→ ¾ nhánh được bao phủ (75%)</p> <p><i>* Khi xét độ bao phủ nhánh của các biến cần KT xem nó có đạt tới điều kiện để xét biến đó không. Chẳng hạn nếu $a \neq 0$ thì b sẽ không bao giờ được xét, và ngược lại.</i></p> <p>3. Tối thiểu bao nhiêu test case để bao phủ 100% các nhánh? → 3TCs $\begin{cases} a = 0, b = 8 \\ a = 0, b = 0 \text{ (như BT1 ở trên)} \\ a \neq 0, b \neq 0 \end{cases}$</p>
--	---

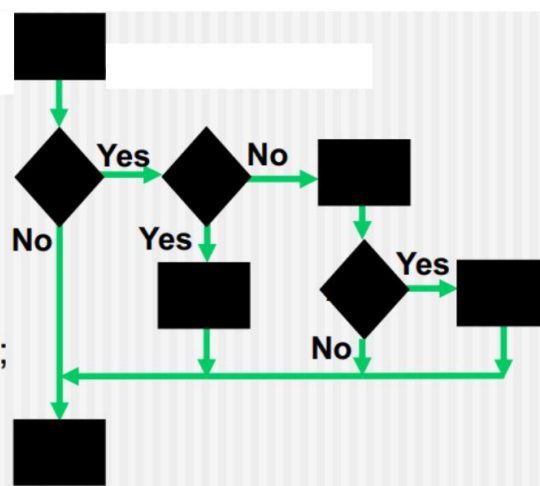
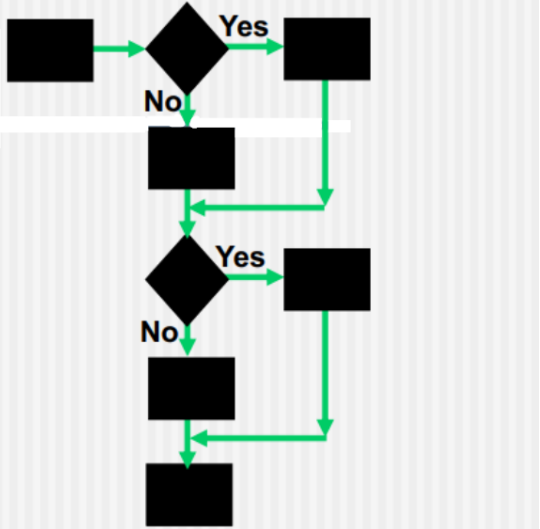
Ví dụ 2:

<pre> 1. void func(int A, int B, int X) 2. { 3. if (A > 1 && B == 0) 4. X = X / A; 5. if (A == 2 X > 1) 6. X = X + 1; 7. }</pre>	<p>1. Có bao nhiêu nhánh? → 4</p> <p>2. Test case (A=2,B=0,X=3) bao phủ bao nhiêu % nhánh ?</p> $\left\{ \begin{array}{l} (A > 1 \ \&\& \ B == 0) \\ (A == 2 \ \ X > 1) \end{array} \right\} \begin{array}{l} T \\ F \end{array}$ <p>→ 50% nhánh (chỉ mới bao phủ 2 trường hợp True của 2 điều kiện).</p> <p>3. Tối thiểu bao nhiêu test case để bao phủ 100% các nhánh ?</p> <p>→ Cần thêm 1TC: A <= 0, B bất kỳ, X bất kỳ.</p> <p>Chẳng hạn: A=0,B=0,X=1.</p>
---	---

• Điểm yếu bao phủ nhánh:

<pre> 1. void func(int A, int B, int X) 2. { 3. if (A > 1 && B == 0) 4. X = X / A; 5. if (A == 2 X > 1) 6. X = X + 1; 7. }</pre>	<pre> graph TD a((a)) --> D1{A > 1 and B == 0} D1 -- T --> C[X = X / A] D1 -- F --> b((b)) C --> c((c)) b --> D2{A == 2 or X > 1} D2 -- T --> E[X = X + 1] D2 -- F --> d((d)) E --> e((e)) c --> exit(()) e --> exit d --> exit style exit fill:none,stroke:none </pre>	<p>Test cases bao phủ nhánh</p> <p>1) A = 3, B = 0, X = 3 (acd)</p> <p>2) A = 2, B = 1, X = 1 (abe)</p> <p>Vì tồn tại điều kiện OR nên ta có thể bỏ sót các trường hợp.</p>
---	---	--

• Xét các ví dụ sau:

<pre> 1. cin>>A; 2. cin>> B; 3. if (A > 0) 4. if (B == 0) 5. cout<< "No values"; 6. else 7. { 8. cout<<"B"<<endl; 9. if (A > 21) 10. cout<<"A"; 11. } </pre>  <p>• Độ phức tạp Cyclomatic : <u>4</u></p> <p>• Số tests nhỏ nhất:</p> <ul style="list-style-type: none"> ◦ Statement coverage: <u>2</u> ◦ Branch coverage: <u>4</u> 	<p>2 TCs bao phủ dòng lệnh:</p> <ul style="list-style-type: none"> • A=23, B=0 (dòng 1, 2, 3, 4, 5) • A=23, B=1 (dòng 1, 2, 3, 4, 6-11) <p>4 TCs bao phủ nhánh:</p> $A > 0 \begin{cases} T \\ F \end{cases}; B = 0 \begin{cases} T \\ F \end{cases}; A > 21 \begin{cases} T \\ F \end{cases}$ <ul style="list-style-type: none"> • A=23, B=0 (A>0 T, B = 0 T) • A=23, B=1 (A>0 T, B = 0 F, A>21 T) • A=20, B=1 (A>0, B=0 F, A>21 F) • A=0, B bất kỳ (A>0 F)
<pre> 1. Read A 2. Read B 3. IF A < 0 THEN 4. Print "A negative" 5. ELSE 6. Print "A positive" 7. ENDIF 8. IF B < 0 THEN 9. Print "B negative" 10. ELSE 11. Print "B positive" 12. ENDIF </pre>  <p>■ Độ phức tạp Cyclomatic : <u>3</u></p> <p>■ Số tests nhỏ nhất :</p> <ul style="list-style-type: none"> • Statement coverage: <u>2</u> • Branch coverage: <u>2</u> 	<p>2 TCs bao phủ dòng lệnh:</p> <ul style="list-style-type: none"> • A=-1, B=1 (dòng 1, 2, 3, 4, 7, 8, 10, 11, 12) • A=1, B=-1 (dòng 1, 2, 3, 5, 6, 7, 8, 9, 12) <p>2 TCs bao phủ nhánh:</p> $A < 0 \begin{cases} T \\ F \end{cases}; B < 0 \begin{cases} T \\ F \end{cases}$ <p>Sử dụng 2 TC trên.</p>

- Nhận xét: **ĐỘ PHỨC TẠP CHÍNH LÀ GIỚI HẠN TRÊN SỐ TCs CẦN THIẾT KẾ.**

5. Bao phủ điều kiện (Condition coverage)

- Tỷ lệ phần trăm các **điều kiện** được thực thi bởi bộ test = Số điều kiện được thực thi/ Tổng số điều kiện
- Ví dụ:

Cần thiết kế các test cases **bao phủ hết các điều kiện** sau:

$A > 1, A \leq 1, B = 0, B \neq 0$

$A = 2, A \neq 2, X > 1, X \leq 1$

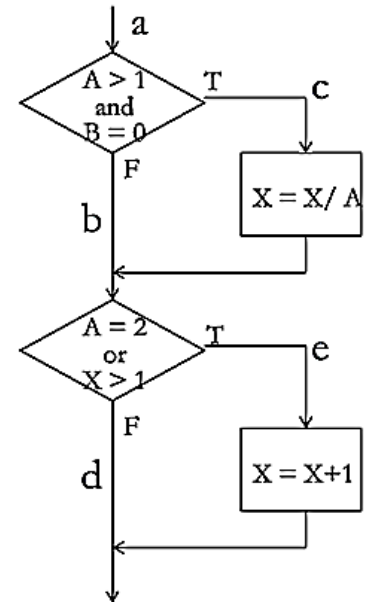
→ **Phân biệt với bao phủ nhánh.**

Test cases:

1) $A = 1, B = 0, X = 3$ (abe)

2) $A = 2, B = 1, X = 1$ (abe)

→ **Không thỏa mãn bao phủ nhánh**



Bài tập 1

```

1. float A, B, C;
2. printf("Enter three values\n");
3. scanf("%f%f%f", &A, &B, &C);
4. printf("\n largest value is: ");
5. if(A>B)
6. {
7.     if (A>C) printf("%f\n",A);
8.     else printf("%f\n",C);
9. }
10. else
11. {
12.     if (C>B) printf("%f\n",C);
13.     else printf("%f\n",B);
14. }
  
```

Lưu ý: BT này không mặc định các điều kiện if luôn thực hiện, vì nó gắn chung với lệnh printf.

Thiết kế bộ TCs nhỏ nhất sao cho bao phủ:

1. 100% dòng lệnh
2. 100% nhánh
3. 100% điều kiện

Ans:

- $A=3, B=2, C=1$ ($A>B, A>C$)
- $A=2, B=1, C=3$ ($A>B, A<C$)
- $A=1, B=2, C=3$ ($A<B, C>B$)
- $A=1, B=3, C=2$ ($A<B, C<B$)

Bài tập 2: Tìm phần tử âm đầu tiên trong mảng

```

1. void PTA(int a[], int n)
2. {
3.     int i = 0;
4.     while ((i < n) && (a[i] >= 0))
5.         i++;
6.     if (i < n)
7.         printf("Phan tu am a[%d] = %d", i, a[i]);
8.     else
9.         printf("Khong co phan tu am.");
10. }
  
```

Các TCs

- $a = [1, -2, 3]$
- $a = [1, 2, 3]$

Bao phủ nhánh sẽ sử dụng thêm 1 TC:

- $a = \emptyset$.

6. Bao phủ nhánh – điều kiện

- Bao phủ nhánh - điều kiện:** Viết các TCs sao cho **mỗi điều kiện (đơn) trong mỗi nhánh nhận được 2 giá trị (T và F) ít nhất 1 lần** và **mỗi nhánh được thực hiện ít nhất 1 lần**
- Bao phủ tổ hợp các điều kiện:** Viết các TCs để **thực thi được tất cả các tổ hợp giá trị (T và F) của các điều kiện (đơn) trong 1 nhánh**

Ví dụ:

<p>Bao phủ nhánh - điều kiện:</p> <p>Các TCs cần bao phủ tất cả các điều kiện A>1, A<=1, B=0, B!=0 A=2, A!=2, X>1, X<=1 và (A > 1 and B = 0) T, F (A = 2 or X > 1) T, F</p> <p>Test cases: 1) A = 2, B = 0, X = 4 (ace) 2) A = 1, B = 1, X = 1 (abd)</p>	<pre>graph TD a((a)) --> D1{A > 1 and B = 0} D1 -- T --> c((c)) c --> P1[X = X / A] P1 --> B((b)) D1 -- F --> B B --> D2{A = 2 or X > 1} D2 -- T --> e((e)) e --> P2[X = X + 1] D2 -- F --> d((d)) P2 --> Exit(()) d --> Exit</pre>
<p>Bao phủ tổ hợp điều kiện:</p> <p>TCs phải bao phủ các điều kiện 1) A > 1, B = 0 5) A = 2, X > 1 2) A > 1, B != 0 6) A = 2, X <= 1 3) A <= 1, B = 0 7) A != 2, X > 1 4) A <= 1, B != 0 8) A != 2, X <= 1</p> <p>Test cases: 1) A = 2, B = 0, X = 4 (bao phủ 1,5) 2) A = 2, B = 1, X = 1 (bao phủ 2,6) 3) A = 1, B = 0, X = 2 (bao phủ 3,7) 4) A = 1, B = 1, X = 1 (bao phủ 4,8)</p>	

IV.5 Kỹ thuật kiểm thử dựa trên kinh nghiệm

- Đoán lỗi (Error guessing)**
 - ☐ Được sử dụng sau khi áp dụng các kỹ thuật hình thức khác
 - ☐ Có thể tìm ra 1 số lỗi mà các kỹ thuật khác có thể bỏ lỡ
 - ☐ Không có qui tắc chung
- Kiểm thử thăm dò (Exploratory testing)**
 - ☐ Hoạt động thiết kế test và thực thi test được thực hiện song song
 - ☐ Các chú ý sẽ được ghi chép lại để báo cáo sau này
 - ☐ **Khía cạnh chủ chốt: learning** (cách sử dụng, điểm mạnh, điểm yếu của PM)

IV.6 Chọn kĩ thuật kiểm thử

- Mỗi kĩ thuật riêng lẻ chỉ **hiệu quả với 1 nhóm lỗi cụ thể**
- **Cần kết hợp các kỹ thuật kiểm thử để đem lại hiệu quả tốt nhất, phát hiện được nhiều loại lỗi.**
- **Các yếu tố ảnh hưởng đến việc chọn kĩ thuật kiểm thử:**
 - **Loại hệ thống**
 - *Tiêu chuẩn quy định*
 - *Yêu cầu khách hàng hoặc hợp đồng*
 - *Mức độ rủi ro, loại rủi ro*
 - *Mục tiêu kiểm thử*
 - *Tài liệu có sẵn*
 - **Kiến thức của testers**
 - *Thời gian và ngân sách*
 - *Mô hình phát triển PM*
 - *Kinh nghiệm về các loại lỗi đã được tìm ra trước đó*