NAME: Nguyễn Văn Khánh Duy STUDENT CODE: 3122411027

1. VERIFICATION AND VALIDATION

Vấn đề của SYSTEM 1:

* Mặc dù có giao diện tuân theo yêu cầu của người dùng đó là giải hệ phương trình bậc 2. Tuy nhiên, hệ thống này lại có cách tính x1 sai công thức. Có thể nói rằng giao diện đúng với đặc tả nhưng kết quả lại sai với mong muốn của người dùng.
* Chưa khai báo DELTA trước khi dùng để tính x1, x2.

Vấn đề của SYSTEM 2:

* Mặc dù công thức tính nghiệm của phương trình bậc 2 đúng, nhưng lại sai với đặc tả. Đặc tả yêu cầu nhập a,b,c vào hệ thống; rồi hệ thống phải trả về x1 và x2 dựa trên input. Tuy nhiên, hệ thống khong trả về x1 và x2 ngay lập tức, mà trả về delta ở step1 và bắt người dùng phải nhập delta vào step2 thì mới trả về x1 và x2. Có thể nói rằng giao diện và cách xử lý không tuân theo đặc tả (không thỏa verification) nhưng kết quả lại đúng với yêu cầu người dùng (thỏa validation).

II. TEST-CASES

a/

Có 2 testcase cần cho hàm f1, để có thể cover được hết code:

* Testcase đầu tiên là giá trị x>10: x=11 với kết quả mong muốn là 22
* Test case thứ hai là giá trị x <=10: x=10 với kết quả mong muốn là -10

Các tập tin đính kèm:

* baia.cpp
* test.ipynb

b/

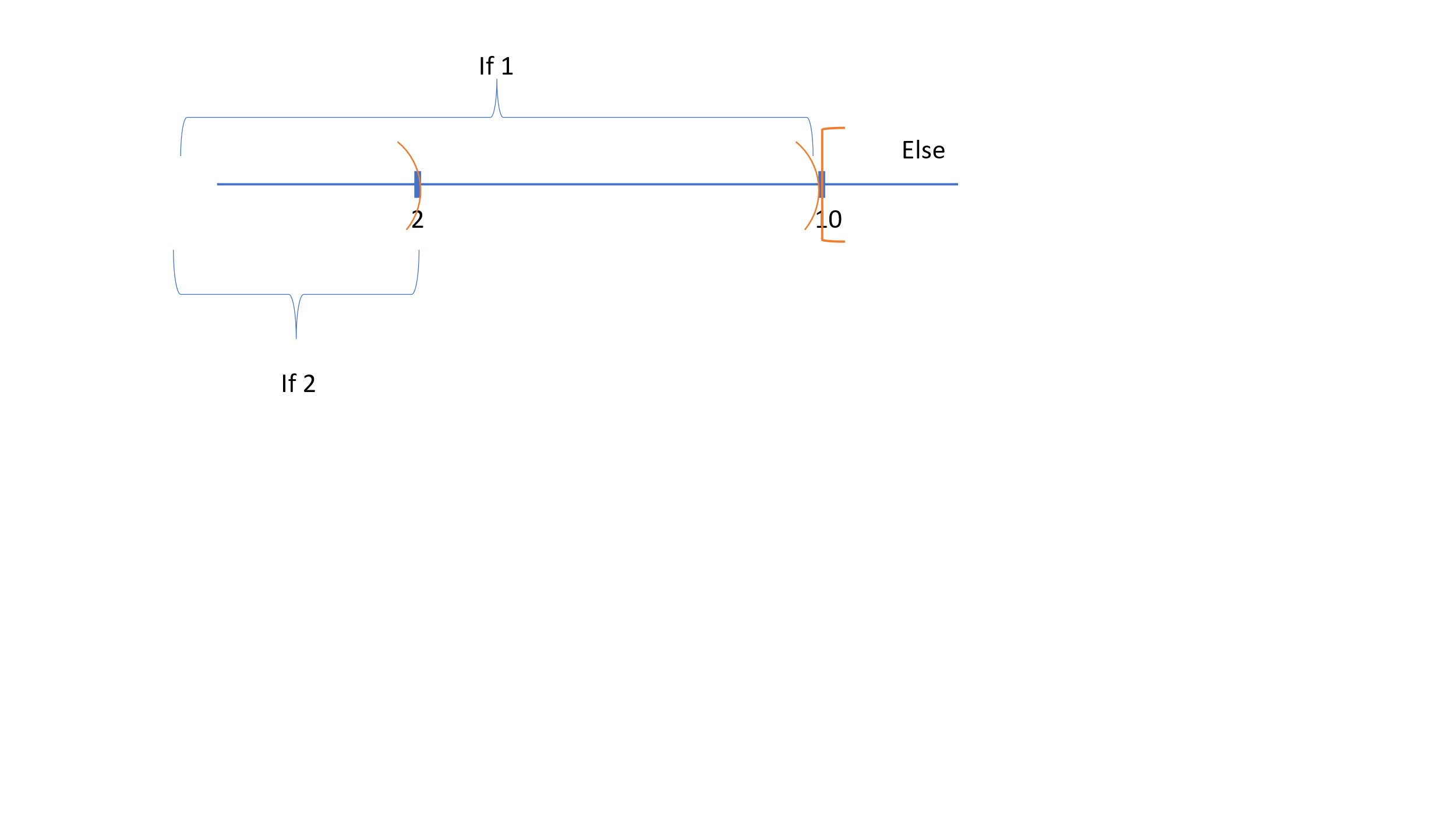
Dựa trên ngữ nghĩa của hàm f1 ở câu a, hàm f1 được tái lập trình ở câu b cần có 3 testcase để có thể cover được hết code:

* Test case 1, x>10: x=11 với kết quả mong muốn là 22
* Test case 2, 0<x<=10: x=10 với kết quả mong muốn là -10
* Test case 3: x<=0: x=-1 với kết quả mong muốn là 1. Testcase này có output thật sự là -2=> f1 có lỗi

Các tập tin đính kèm:

* baib.cpp
* test.ipynb

c/



Dựa vào hình trên, tất cả các giá trị nhỏ hơn 2 đều sẽ nhỏ hơn 10, mà câu lệnh if 1 lại đứng trước if 2, nên if 2 không bao giờ được xét đến. Vì câu lệnh if 1 sẽ luôn bao trùm if 2, nên hàm f2 cần có 2 testcase để cover hết code:

* Testcase 1 với giá trị x <10: x=-2, expected=-4
* Testcase 2 với giá trị x>=10: x=10, expected=20

Các file đính kèm:

* baic.cpp
* test.ipynb

d/

Vì hàm log trong c++ là log tự nhiên, nên có 2 testcase cần có cho hàm f3 để có thể cover được hết code:

* Testcase 1, giá trị x sao cho : x=1, expected=2
* Testcase 2, giá trị x sao cho : x=-1, expected=-2

Các file đính kèm:

* baid.cpp
* test.ipynb

e/

Ngoài việc các testcase phải cover được hết code (testcase chạy được vào các cầu điều kiện if), thì cũng cần phải test thử tổ hợp các if trong chương trình này. Gọi 1 là trạng thái câu If thỏa và 0 là ngược lại, có 3 câu if trong chương trình, nên số tổ hợp các if là 2\*2\*2=8. Xét từng tổ hợp và đưa ra testcase cho từng trường hợp:

* (0,0,0) (trong đó 0 ở vị trí thứ i biểu thị câu if thứ i không thỏa, còn nếu là 1 thì if thỏa): a=5, b=5, c=5, expected=5
* (0, 0, 1) : a=2, b=1, c=5, expected=5
* (0, 1, 0) : a=-2, b=0, c=-1, expected=0
* (0, 1, 1) : không thể xảy ra vì num2 không thể vừa lớn hơn num3 và num3 lại lớn hơn num2
* (1, 0, 0): a=5, b=2, c=1, expected=5
* (1, 0, 1): không thể xảy ra vì num1 không thể vừa lớn hơn num3 và num3 lại lớn hơn num1
* (1, 1, 0): không thể xảy ra vì num1 không thể vừa lớn hơn num2 và num2 lại lớn hơn num3
* (1, 1, 1): không thể xảy ra vì cả 3 số num1, num2, num3 đều lớn hơn nhau
* Số lượng testcase là 4:

1. a=5, b=5, c=5, expected=5 (Lỗi xảy ra ở testcase này, kết quả thực sự ra 0 thay vì 5)
2. a=2, b=1, c=5, expected=5
3. a=-2, b=0, c=-1, expected=0
4. a=5, b=2, c=1, expected=5

Các file đính kèm:

* baid.cpp
* test.ipynb

III. PRATICE 1

Mô tả bài toán, các input / output có thể có của bài toán:

* Dựa theo code của hàm solveQuartic, chương trình này đường dùng để tìm x của phương trình với .
* Các input có thể có:
  + a=b=c=0 => bài toán có dạng 0=0
  + a=b=0, => bài toán có dạng c=0
  + a=0, , c tự do=> bài toán có dạng
  + , b và c tự do=> bài toán có dạng
* Các output có thể có:
  + Vố số nghiệm
  + Vô nghiệm
  + Có nghiệm

Xây dựng các test cases kiểm tra tính đúng đắn chương trình:

* Testcase 1: a=b=c=0, expected: vô số nghiệm
* Testcase 2 cho a=b=0, : a=b=0, c=1 expected: vô nghiệm
* Testcase 3 cho a=0, , c tự do với nhỏ hơn 0: a=0, b=3, c=1, expected: vô nghiệm
* Testcase 4 cho a=0, , c tự do với lớn hơn hoặc bằng 0: a=c=0, b=3, expected: có một nghiệm là 0. Lỗi xảy ra: output thật sự là “The equation has 2 real solution(s): -0 0”
* Testcase 5 cho , delta>0, : a=2, b=7, c=3, expected: vô nghiệm
* Testcase 6 cho , delta>0, : a=1, b=-2, c=0, expected: có ba nghiệm là 0, , . Lỗi xảy ra: output thật sự là “The equation has 4 real solution(s): 1.41421 -1.41421 0 -0”
* Testcase 7 cho , delta=0, : a=1,b=2,c=1, expected: vô nghiệm
* Testcase 8 cho , delta=0, : a=1, b=-4, c=4, expected: có 2 nghiệm là ,
* Testcase 9 cho , delta<0: a=5, b=1, c=1, expected: vô nghiệm

Viết đoạn mã tự động kiểm tra chương trình cho bên dưới đúng hay sai?

*!*g*++* solveQuartic.cpp -o solveQuartic

!./solveQuartic < t1.inp > t1.out

*!*echo "Expected: Infinite solutions"

*!*cat t1.out

!./solveQuartic < t2.inp > t2.out

*!*echo "Expected: No solution"

*!*cat t2.out

!./solveQuartic < t3.inp > t3.out

*!*echo "Expected: No solutions"

*!*cat t3.out

!./solveQuartic < t4.inp > t4.out

*!*echo "Expected: The equation has 1 real solution(s): 0"

*!*cat t4.out

!./solveQuartic < t5.inp > t5.out

*!*echo "Expected: No solutions"

*!*cat t5.out

!./solveQuartic < t6.inp > t6.out

*!*echo "Expected: The equation has 3 real solution(s): 0, 1.41, -1.41"

*!*cat t6.out

!./solveQuartic < t7.inp > t7.out

*!*echo "Expected: No solutions"

*!*cat t7.out

!./solveQuartic < t8.inp > t8.out

*!*echo "Expected: The equation has 3 real solution(s): 1.41, -1.41"

*!*cat t8.out

!./solveQuartic < t9.inp > t9.out

*!*echo "Expected: No solutions"

*!*cat t9.out

Các file đính kèm:

* solveQuartic.cpp
* test.ipynb