

Bài tập

Software Quality Assurance & Testing

Khối Đại Học

Năm 2020



Hướng dẫn thực hiện:

Tuần 1	Module 1 – Tìm lỗi chương trình
Tuần 2	Module 2 - Kỹ thuật hộp đen (thiết kế)
Tuần 3	Module 2 - Kỹ thuật hộp đen (thiết kế)
Tuần 4	Module 2 - Kỹ thuật hộp đen (thực thi)
Tuần 5	Module 3 - Kỹ thuật hộp trắng (thiết kế)
Tuần 6	Module 3 - Kỹ thuật hộp trắng (thực thi)
Tuần 7	Module 4 – Kiểm thử tĩnh
Tuần 8	Module 5 – Kế hoạch kiểm thử
Tuần 9	Ôn tập Module 6 – CodeUI test Chăm đề tài tìm hiểu công cụ kiểm thử
Tuần 10	Kiểm tra thực hành.

Module 1. Tìm lỗi chương trình

Nội dung kiến thức thực hành:

- + Tìm lỗi chương trình đơn giản.
- + Tìm những chương trình có lỗi.

Yêu cầu lưu trữ:

Sinh viên mở MSWord làm bài và đặt tên theo mẫu **HoTenSV_Module01.docx**, nộp lại vào cuối buổi thực hành.

Bài 1.

- a. Đưa ra một số lý do giải thích tại sao đặc tả là nơi chứa nhiều bug nhất.
- b. Giải thích tại sao chi phí sửa bug sau khi sản phẩm release lại cao hơn khi mới bắt đầu.
- c. Nêu và giải thích 7 nguyên tắc của kiểm thử.

Bài 2.

Hãy tìm các defect/bug (càng nhiều càng tốt) trong hình dưới đây. Các lỗi có thể là giao diện không nhất quán, lỗi chính tả, trùng lặp,...

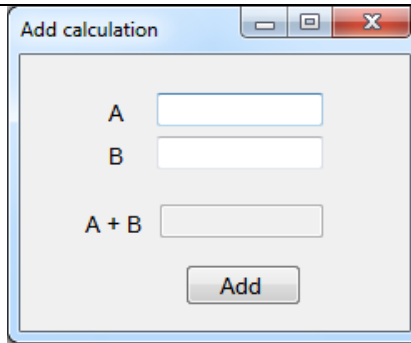


Bài 3.

Hãy chỉ ra ít nhất một lỗi bất kỳ trong ứng dụng hoặc website bạn đã từng sử dụng.

Bài 4.

Tìm các trường hợp kiểm thử (test case) cho chương trình cộng 2 số nguyên có giao diện sau:

A dialog box titled "Add calculation" with a standard Windows window border. Inside, there are three input fields. The first is labeled "A", the second "B", and the third "A + B". Below these fields is a button labeled "Add".

Lập bảng như sau:

STT	Dữ liệu nhập	Kết quả mong muốn

Bài 5.

Cho chương trình đọc một loạt các số đo nhiệt độ tùy ý (số nguyên) trong khoảng -60°C đến $+60^{\circ}\text{C}$ và in ra giá trị trung bình của các số này. Tìm các trường hợp cần kiểm thử cho chương trình này, lập theo bảng sau:

STT	Dữ liệu nhập	Kết quả mong muốn

Module 2. Kỹ thuật hộp đen

Nội dung kiến thức thực hành:

- + Phân tích, thiết kế test case.
- + Thực thi test case.
- + Sử dụng data-driven.
- + Chỉnh sửa test-script.
- + Viết báo cáo kiểm thử.

Yêu cầu lưu trữ:

Mở MSExcel làm bài và đặt tên theo mẫu **HoTenSV_Module02.xlsx**, nộp lại vào đầu buổi thực hành kế tiếp. Mỗi bài làm trong một sheet, gồm: đề bài, bảng phân tích, bảng thiết kế test case.

Bài 1.

Ở trung tâm Golden Splash Swimming, giá vé phụ thuộc vào bốn biến: Ngày (ngày thường, ngày cuối tuần), Trạng thái khách (OT = một lần, M = thành viên), Giờ vào (6.00-19.00, 19.01-24.00) và Độ tuổi của khách (0-16, 16.01-60, 60.01-120). Bảng giá vé được trình bày trong bảng bên dưới.

	Mon, Tue, Wed, Thurs, Fri				Sat, Sun			
Visitor's status	OT	OT	M	M	OT	OT	M	M
Entry hour	6.00–19.00	19.01–24.00	6.00–19.00	19.01–24.00	6.00–19.00	19.01–24.00	6.00–19.00	19.01–24.00
	Ticket prices – \$							
Visitor's age								
0.0–16.00	5.00	6.00	2.50	3.00	7.50	9.00	3.50	4.00
16.01–60.00	10.00	12.00	5.00	6.00	15.00	18.00	7.00	8.00
60.01–120.00	8.00	8.00	4.00	4.00	12.00	12.00	5.50	5.50

Yêu cầu: Xác định các lớp tương đương, các giá trị biên và thiết kế các testcase tương ứng.

Trích ví dụ trong giáo trình trang 199, the Golden Splash Swimming Center.

Bài 2.

Cho màn hình sau:

Subscription Form

User Name	<input style="width: 60%;" type="text"/>
Age	<input style="width: 20%;" type="text"/>
City	<input style="width: 60%;" type="text"/>
Postal Code	<input style="width: 30%;" type="text"/>
<input style="width: 100px;" type="button" value="Submit"/>	

Các ràng buộc:

- User Name phải có từ 6 đến 12 ký tự, bắt đầu bằng chữ cái và chỉ chứa chữ cái hoặc chữ số.
- Age phải ≥ 18 và < 65 .
- City phải là một trong các giá trị: Ottawa, Toronto, Montreal, Halifax.
- Postal Code phải có 6 ký tự, bắt đầu bằng chữ cái và luân phiên giữa chữ cái và số.

Nếu không thỏa ràng buộc, hệ thống sẽ thông báo lỗi.

Yêu cầu:

- i) Lập danh sách các lớp tương đương hợp lệ và không hợp lệ cho mỗi condition.
- ii) Đề xuất các giá trị biên cho mỗi condition.
- iii) Cần bao nhiêu test case để kiểm thử với các phân hoạch và giá trị biên ở trên.

Bài 3.

Cho hàm kiểm tra số nguyên tố với các giá trị nhập từ 0 đến 1000:

```
public Boolean primeCheck(int num) {  
    /* Return true if num is a prime,  
    * if not, return false.  
    * If invalid, throw an Exception.  
    */  
}
```

Dùng kỹ thuật phân lớp tương đương và phân tích giá trị biên thiết kế các test case để kiểm chứng.

Bài 4.

Dùng kỹ thuật phân lớp tương đương và phân tích giá trị biên để tìm các test case cho hàm kiểm tra năm nhuận sau, điều kiện năm ≥ 1582 .

```
bool IsLeapYear(int n)  
{  
    /* Returns true if n is a leap year,
```

```
* if not, returns false.  
* If data is invalid, throw Exception  
*/  
}
```

Ghi chú: năm nhuận là năm chia hết cho 400 hoặc chia hết cho 4 mà không chia hết cho 100.

Bài 5.

Thiết kế các test case cho hàm chuyển đổi từ số nhị phân sang số thập phân sau:

```
long BinToDec(string sbin) {  
    /* Chuỗi sbin chỉ gồm các giá trị 0 và 1  
    * Hàm trả về số thập phân tương ứng với chuỗi nhị phân  
    * Nếu giá trị nhập không phải chuỗi nhị phân, hàm ném ra ngoại lệ FormatException  
    */  
}
```

Bài 6.

Thiết kế các test case để kiểm chứng hàm kiểm tra tam giác, biết hàm có định dạng như sau:

```
String Triangle(int a, int b, int c) {  
    /* If a, b and c are the lengths of the sides of a triangle, this function returns  
    * one of three strings, "Scalene", "Isosceles" or "Equilateral" for the given three inputs.  
    * If not, return empty string.  
    */  
}
```

Bài 7.

Thiết kế các test case để kiểm chứng chương trình giải phương trình $ax^2+bx+c=0$, biết a,b,c là các số nguyên.

```
public static String SolveQuadratic(int a, int b, int c, out float x1, out float x2) {  
    /* Hàm trả về "Vô số nghiệm", "Vô nghiệm", "Có 1 nghiệm", "Có 2 nghiệm phân biệt",  
    * "Có nghiệm kép" tùy theo a,b,c.  
    * Hàm nhận hai kết quả x1, x2 là nghiệm của phương trình.  
    * Nếu nghiệm không được xác định (vô nghiệm, vô số nghiệm) thì x1 = x2 = NaN.
```

*/

}

Bài 8.

Thiết kế các test case cho hàm tính tiền điện sau đây:

```
public double TinhTienDien(int chiSoCu, int chiSoMoi)
{
    /* Tham số: chỉ số cũ, chỉ số mới.
    * Trả về: tiền điện, được tính như sau:
    * (chiSoMoi - chiSoCu) * Các đơn giá theo bậc
    * (tính thêm VAT 10%).
    * Nếu chiSoCu > chiSoMoi hoặc chiSoCu, chiSoMoi < 0, hàm trả về -1.
    */
}
```

Biết quy định đơn giá như sau:

Giá bán lẻ điện sinh hoạt	
Bậc 1: Cho kWh từ 0 - 50	1484
Bậc 2: Cho kWh từ 51 - 100	1533
Bậc 3: Cho kWh từ 101 - 200	1786
Bậc 4: Cho kWh từ 201 - 300	2242
Bậc 5: Cho kWh từ 301 - 400	2503
Bậc 6: Cho kWh từ 401 trở lên	2587

Ví dụ, số kw tiêu thụ là 80 thì thành tiền = $(50 \times 1484 + 30 \times 1533) + 0.1 \times (50 \times 1484 + 30 \times 1533)$

Bài 9.

Cho hàm tìm k bé nhất thỏa $s = 1 + 2 + 3 + \dots + k > s_0$, cho biết s khi đó có giá trị là bao nhiêu.

```
long Sum(long s0, out long s)
```

{

```
    /* Input: s0
    * Output: s and k
    */
```

}

Kiểm chứng lại hàm trên.

Bài 10.

Cho hàm sau:

```
String HuyChuoi(String s, int n, int p)
```


{

```
/* Hàm xóa n ký tự trong s bắt đầu từ ký tự thứ p (p ∈ [0, chiều dài s-1))
```

```
* Nhập: chuỗi s, số nguyên n, p.
```

```
* Trả về: chuỗi mới sau khi xóa.
```

```
* Nếu p ≥ chiều dài s, hàm trả về s
```

```
* Nếu n > chiều dài s-p, hàm trả về s từ vị trí 0 đến p-1
```

```
* Nếu p < 0 hoặc n < 0 (invalid), hàm trả về s
```

```
*/
```

}

Kiểm chứng lại hàm trên.

Bài 11.

Cho hàm sau:

`String` ThayThe(`string` s1, `string` s2, `string` s3)

{

```
/* Hàm thay thế chuỗi s3 vào s1 ở những vị trí mà s2 tồn tại trong s1.
```

```
* Nhập: s1, s2, s3.
```

```
* Trả về: chuỗi đã thay thế.
```

```
* Nếu cả 3 chuỗi rỗng, hàm trả về chuỗi rỗng
```

```
* Nếu s2 rỗng, hàm trả về s1
```

```
* Nếu s3 rỗng, hàm trả về s1 đã bị xóa đi những nội dung của chuỗi s2 (nếu có)
```

```
* Nếu s2 không có trong s1, hàm trả về s1
```

```
*/
```

}

Ví dụ: ThayThe("Truong dh cong nghiep", "dh", "dai hoc") = "Truong dai hoc cong nghiep".

Kiểm chứng lại hàm trên.

Chú ý: giá trị biên xác định tại những trường hợp s2 được tìm thấy ở đầu hoặc cuối chuỗi s1.

Bài 12.

Kiểm thử hàm tìm giá trị lớn nhất trong mảng, biết hàm có định dạng như sau:

`int` Largest (`int`[] a) {

```
/* requires: a is an array of integers
```

```
* effects: returns the maximum element in the list
```

```
* if a is empty, return max value of integer (32 bit)
* /
}
```

Bài 13.

Kiểm chứng cho hàm kiểm tra mảng đối xứng sau:

```
bool IsSymmetry( int[] a, int n )
{
    /* Tham số: các phần tử mảng a, kích thước mảng n
    * Trả về true nếu mảng đối xứng,
    * Ngược lại trả về false.
    */
}
```

Ví dụ mảng (4,3,7,8,2,8,7,3,4) là mảng đối xứng.

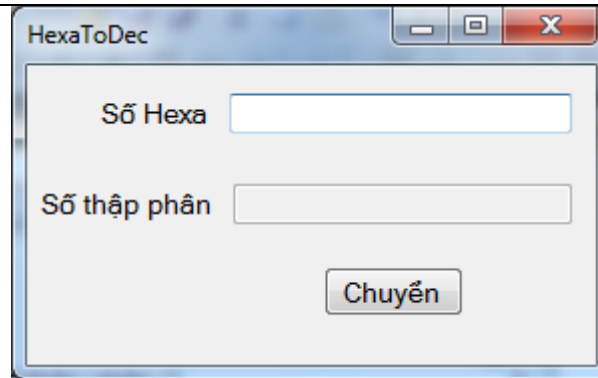
Bài 14.

Thiết kế các trường hợp kiểm thử cho hàm sắp xếp mảng số nguyên theo thuật toán đệ quy Quick sort như sau:

```
void QuickSort(int[] list, int left, int right)
{
    /* Input: mảng 1 chiều list, vị trí phần tử đầu tiên left và cuối cùng right (tính từ 0).
    * Output: mảng list được sắp xếp
    * Nếu left >= right, hàm dừng và trả về list
    */
}
```

Bài 15.

Cho chương trình chuyển số thập lục phân sang số thập phân với giao diện như sau:



Nêu đặc tả cho chương trình trên để có thể thiết kế các trường hợp kiểm thử.

Bài 16.

Phí cuộc gọi được tính dựa vào thời điểm và thời gian gọi như sau:

- Các cuộc gọi bắt đầu từ 18h00 đến trước 8h00 được giảm giá 50%.
- Các cuộc gọi bắt đầu từ 8h00 đến trước 18h00 không được giảm giá.
- Tất cả các cuộc gọi đều có thuế Federal 4%.
- Đơn giá cho một cuộc gọi là \$0.40/phút.
- Các cuộc gọi kéo dài hơn 60 phút thì được giảm 15% (được tính sau khi đã tính các khoản giảm giá khác nhưng trước khi tính thuế).

Chương trình cho phép nhập vào thời điểm bắt đầu cuộc gọi và thời gian cuộc gọi. Chương trình tính và xuất ra tổng chi phí (trước khi giảm giá và thuế), tiền phải trả (sau khi tính giảm giá và thuế).

Giả định rằng chỉ nhập số nguyên, khoảng thời gian gọi là số không âm và thời điểm bắt đầu cuộc gọi là giá trị đồng hồ thật. Kết quả được làm tròn đến giá trị gần nhất.

Phân tích và lập bảng testcase cho đặc tả trên.

Bài 17.

Một công ty bảo hiểm ô tô tính phí bảo hiểm dựa vào độ tuổi, giới tính và hồ sơ của khách hàng theo các luật sau:

- Nếu khách hàng nhỏ hơn 21 tuổi, hoặc lớn hơn 80 tuổi thì không được tham gia bảo hiểm này.
- Nếu hồ sơ của khách hàng có hơn 6 yêu cầu bồi thường trong hai năm thì không được tham gia.
- Nếu khách hàng là nam có tuổi từ 21 đến 27 và không có yêu cầu bồi thường trong hai năm thì phí bảo hiểm phải đóng là 2% giá trị xe cộng với \$ 900,00.
- Nếu khách hàng là nữ có tuổi từ 21 đến 27 và không có yêu cầu bồi thường trong hai năm thì phí bảo hiểm phải đóng là 2% giá trị xe cộng với \$ 700,00.
- Nếu khách hàng từ 21 đến 27 tuổi và hồ sơ có 1, 2 hoặc 3 yêu cầu trong hai năm thì phí bảo hiểm phải đóng là 2% giá trị xe cộng với \$ 1.100,00.
- Nếu khách hàng từ 21 đến 27 tuổi và có từ 4 yêu cầu bồi thường trong hai năm thì không được tham gia bảo hiểm.

• Nếu khách hàng từ 28 đến 80 tuổi và không có yêu cầu gì trong hai năm thì phí bảo hiểm phải đóng là 2% giá trị xe cộng với \$ 400,00.

• Nếu khách hàng từ 28 đến 80 tuổi và có 1, 2, hoặc 3 yêu cầu trong hai năm thì phí bảo hiểm phải đóng là 2% giá trị xe cộng với \$ 700,00.

• Nếu khách hàng từ 28 đến 80 tuổi và có 4, 5 hoặc 6 yêu cầu trong hai năm thì phí bảo hiểm phải đóng là 2% giá trị xe cộng với \$ 900,00.

Yêu cầu:

- (i) Xác định các condition và action/outcome cho mô tả trên.
- (ii) Lập bảng quyết định.
- (iii) Thiết kế các test case.

Bài 18.

Vẽ sơ đồ trạng thái rút tiền ở ATM.

Bài 19. Thực thi testcase

Thực thi các test case đã thiết kế trong module này.

Hướng dẫn:

B1. New Project Test.

B2. Add Reference file .dll vào Project Test này (chọn phần assembly và browse đến .dll cần viết unit test).

B3. Thực hiện viết hàm Unit Test.

Ghi chú: GV cung cấp file .dll hoặc file thực thi.

Bài 20. Test-driven development

Thực hiện Test-Driven Development (viết test trước khi viết code) theo hướng dẫn Exercise 1-Task 3 trong file TestingInVS2013_TDEV-H211.pdf.

Bài 21. Data-driven test

Thực hiện data-driven test: Thực thi test từ bài 1 đến bài 13 với các test case được lưu vào file .csv.

Hướng dẫn:

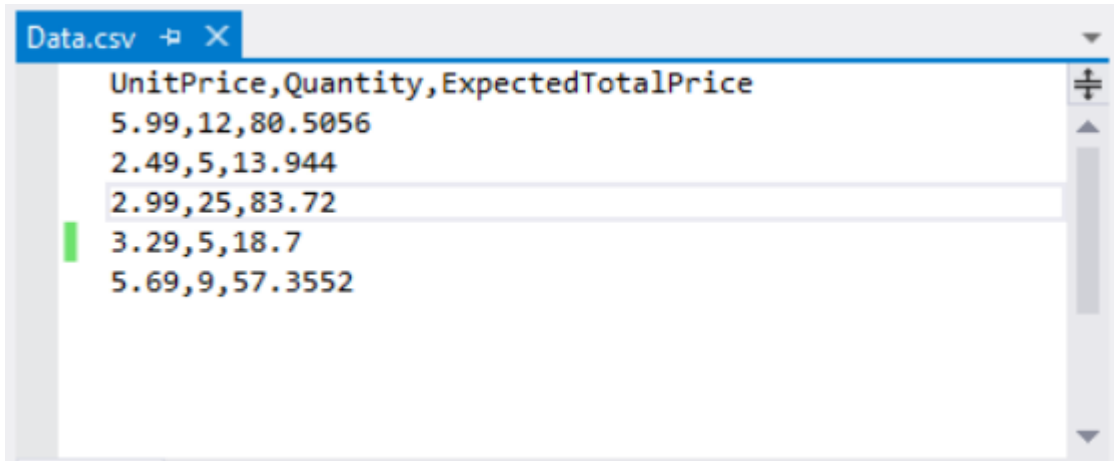
Bước 1: Tạo project test

Bước 2: Tạo file .csv:

- Click phải vào project test; chọn Add -> New Item; chọn dạng file muốn tạo là Text File; đặt tên file với phần mở rộng là csv. Nhập liệu vào file .csv.

- Chú ý: Thiết lập property cho file .csv là Copy always để file được copy vào thư mục bin sau khi build.

Ví dụ: tạo file .csv lưu thông tin đơn giá, số lượng, thành tiền



Bước 3: Add Reference cho System.Data assembly trong project test.

Bước 4: Khai báo thuộc tính TestContext trong class test để truy xuất dữ liệu như sau:

```
public TestContext TestContext { get; set; }
```

Bước 5: Thêm chuỗi kết nối trước hàm test, trong phần tử [TestMethod()] như sau:

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",  
    "|DataDirectory|\\data.csv", "data#csv",  
    DataAccessMethod.Sequential),  
    DeploymentItem("data.csv"), TestMethod]
```

Trong đó data.csv là tên file .csv cần kết nối.

Bước 6: Truy xuất dữ liệu:

Trong hàm test, sử dụng thuộc tính TestContext.DataRow["NameOfColumn" or IndexOfColumn"] để truy xuất dữ liệu trong file .csv.

Ví dụ: truy xuất cột UnitPrice, Quantity:

```
double dUnitPrice = Convert.ToDouble(TestContext.DataRow[0]);
```

```
double dQuantity = Convert.ToDouble(TestContext.DataRow[1]);
```

Chú ý:

What are the data source attributes for other data source types, such as SQL Express or XML?

Date Source Type	Data Source Attribute
CSV	[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", " DataDirectory \\data.csv", "data#csv", DataAccessMethod.Sequential), DeploymentItem("data.csv"), TestMethod]

Excel	<code>DataSource("System.Data.Odbc", "Dsn=ExcelFiles;Driver={Microsoft Excel Driver (*.xls)};dbq= DataDirectory \\Data.xls;defaultdir=.;driverid=790;maxbuffersize=2048;pagetimeout=5;readonly=true", "Sheet1\$", DataAccessMethod.Sequential), DeploymentItem("Sheet1.xls"), TestMethod]</code>
Test case in Team Foundation Server	<code>[DataSource("Microsoft.VisualStudio.TestTools.DataSource.TestCase", "http://vml13261329:8080/tfs/DefaultCollection;Agile", "30", DataAccessMethod.Sequential), TestMethod]</code>
XML	<code>[DataSource("Microsoft.VisualStudio.TestTools.DataSource.XML", " DataDirectory \\data.xml", "Iterations", DataAccessMethod.Sequential), DeploymentItem("data.xml"), TestMethod]</code>
SQL Express	<code>[DataSource("System.Data.SqlClient", "Data Source=.\sqlexpress;Initial Catalog=tempdb;Integrated Security=True", "Data", DataAccessMethod.Sequential), TestMethod]</code>

Why can't I modify the code in the UIMap.Designer file?

Any code changes you make in the UIMapDesigner.cs file will be overwritten every time you generate code using the UIMap - Coded UI Test Builder. In this sample, and in most cases, the code changes needed to enable a test to use a data source can be made to the test's source code file (that is, CodedUITest1.cs).

If you have to modify a recorded method, you must copy it to UIMap.cs file and rename it. The UIMap.cs file can be used to override methods and properties in the UIMapDesigner.cs file. You must remove the reference to the original method in the Coded UITest.cs file and replace it with the renamed method name.

Module 3. Kỹ thuật hộp trắng

Nội dung kiến thức thực hành:

- + Thiết kế test case.
- + Thực thi test case.
- + Đo độ bao phủ mã.
- + Viết báo cáo kiểm thử.

Yêu cầu lưu trữ:

Mở MSEXcel làm bài và đặt tên theo mẫu **HoTenSV_Module03.xlsx**, nộp lại vào đầu buổi thực hành kế tiếp. Mỗi bài làm trong một sheet, gồm: đề bài, CFG, bảng thiết kế test case.

Chú ý: Khi thực thi test case, với những bài chưa có mã nguồn, SV tự viết mã để chạy test.

Bài 1.

Vẽ lược đồ CFG cho đoạn mã sau:

1. int getPositionOf2Cir(int firstRadius, int secRadius, int distance)
2. START
3. If distance = 0
4. If firstRadius = secRadius
5. return 0
6. Else If firstRadius < secRadius
7. return 1
8. Else
9. return 2
10. If distance > 0
11. return 3
12. END

Bài 2.

Vẽ lược đồ CFG cho đoạn mã sau:

1. X,Y,Z is integer
2. **START**
3. Check the value of X input
4. If is 1 or 2
5. Return A
6. Else
7. Check the value of Y input
8. If Y <=10

9. Return B
10. Else
11. Check the value of Z input
12. If $Z < 5$
13. Return C
14. Else
15. Return D
16. End If
17. End If
18. End If
19. **END**

Bài 3.

Cho hàm tính giá trị lớn nhất và trung bình của 3 số A,B,C. Vẽ CFG và thiết kế test case sao cho đạt bao phủ lệnh 100%.

```
1. public int MaxAndMean(int A, int B, int C, out double Mean)
2. {
3.     Mean = (A + B + C) / 3.0;
4.     int Maximum;
5.     if (A > B)
6.         if (A > C)
7.             Maximum = A;
8.         else
9.             Maximum = B;
10.    else
11.        if (B > C)
12.            Maximum = B;
13.        else
14.            Maximum = C;
15.    return Maximum;
16. }
```

Bài 4.

Vẽ CFG cho đoạn mã bên dưới. Thiết kế các test case sao cho đạt bao phủ nhánh 100%. Đây là thuật toán tìm số lớn nhất trong 3 số a,b,c. Nếu một trong 3 giá trị nhỏ hơn 0 thì hàm trả về 0.

```
1. public int Max(int a, int b, int c)
2. {
3.     int max = 0;
4.     if (a > 0 && b > 0 && c > 0)
5.         max = a;
6.     else
```



```
7.     return 0;
8.     if (max < b)
9.         max = b;
10.    if (max < c)
11.        max = c;
12.    return max;
13. }
```

Bài 5.

Cho hàm **Triangle** kiểm tra tam giác là cân, đều hay tam giác thường sau đây. Hàm trả về "Not a Triangle" nếu không phải tam giác; "Triangle is Scalene" nếu là tam giác thường; "Triangle is Isosceles" nếu là tam giác cân; "Triangle is Equilateral" nếu là tam giác đều.

Viết các test case để đạt được 100% độ bao phủ lệnh, 100% độ bao phủ nhánh.

```
1.  String Triangle( int a, int b, int c ) {
2.      int match = 0;
3.      if (a == b)
4.          match = match + 1;
5.      if (a == c)
6.          match = match + 2;
7.      if (b == c)
8.          match = match + 3;
9.      if (match == 0)
10.         if ((a + b) <= c)
11.             return ("Not a Triangle");
12.         else if ((b + c) <= a)
13.             return ("Not a Triangle");
14.         else if ((a + c) <= b)
15.             return ("Not a Triangle");
16.         else return ("Triangle is Scalene");
17.     else if (match == 1)
18.         if ((a + c) <= b)
19.             return ("Not a Triangle");
20.         else return ("Triangle is Isosceles");
21.     else if (match == 2)
22.         if ((a + c) <= b)
23.             return ("Not a Triangle");
24.         else return ("Triangle is Isosceles");
25.     else if (match == 3)
26.         if ((b + c) <= a)
27.             return ("Not a Triangle");
28.         else return ("Triangle is Isosceles");
29.         else return ("Triangle is Equilateral");
```

30. }

Bài 6.

Cho hàm Average tính trung bình Sum/Count. Nếu Count \leq 0 hàm trả về 0. Vẽ CFG và thiết kế các test case sao cho hàm đạt bao phủ nhánh 100%.

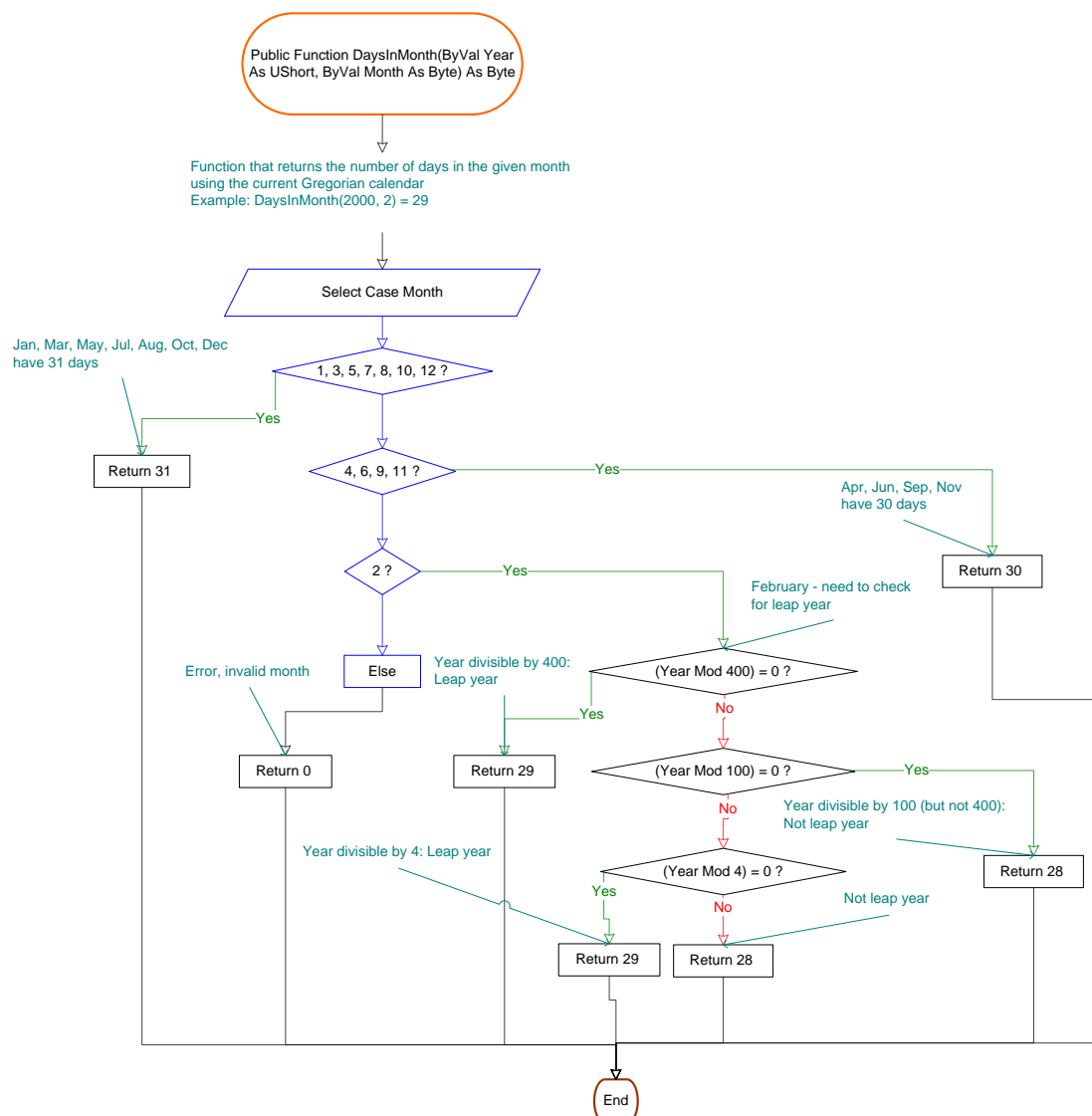
```

1. public double Average(double Sum, double Count)
2. {
3.     if (Count == 1) return Sum;
4.     else if (Count > 0) return Sum / Count;
5.     else return 0;
6. }

```

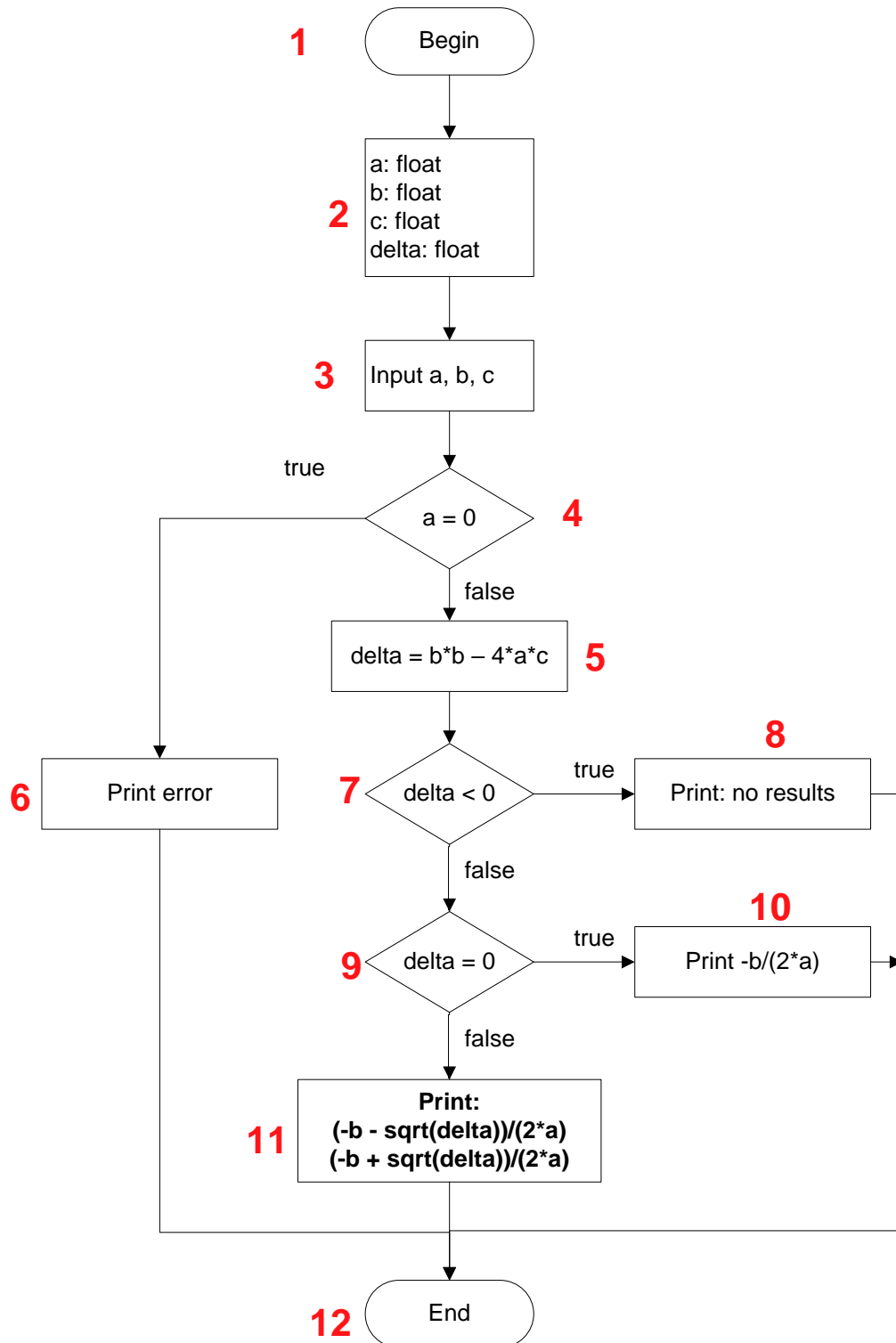
Bài 7.

Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%. Đây là thuật toán tính số ngày trong tháng của năm bất kỳ.



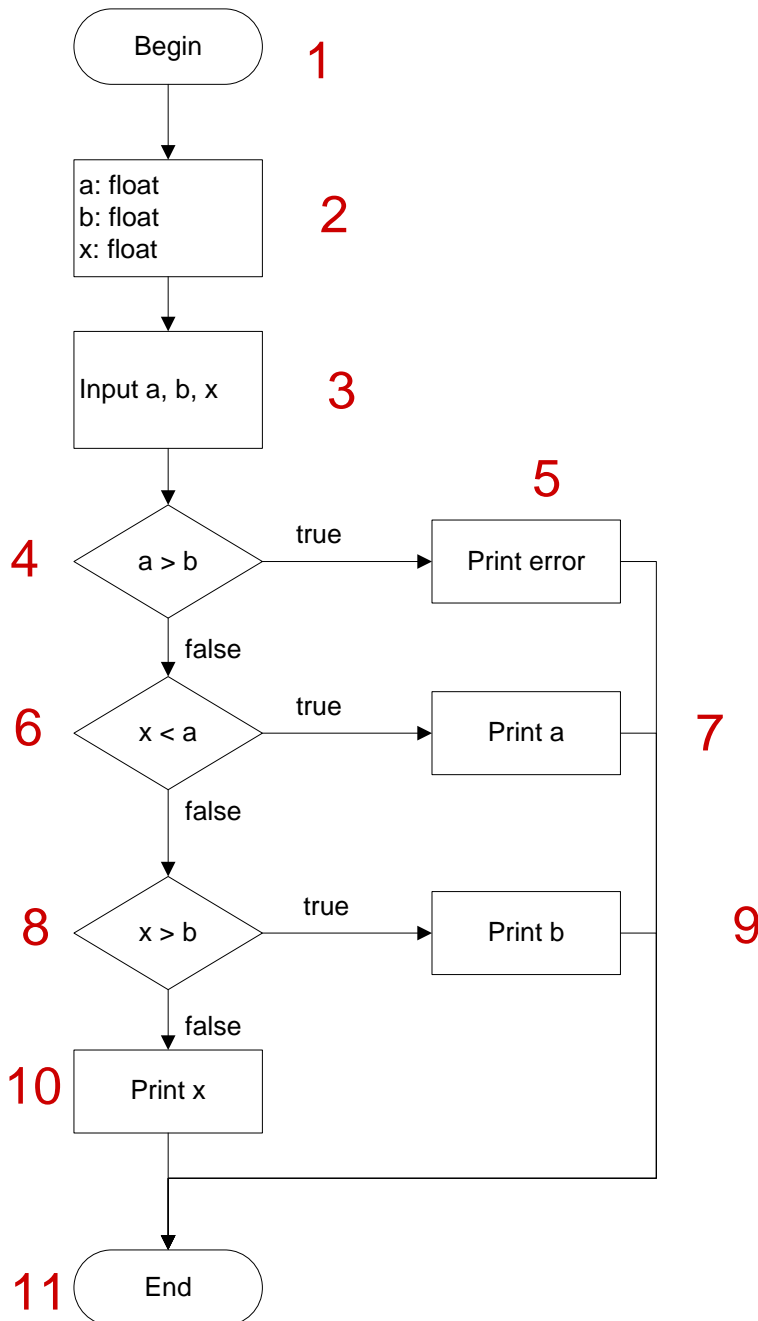
Bài 8.

Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%. Đây là thuật toán giải phương trình bậc hai. Nếu dữ liệu không hợp lệ thì hàm ném ra ngoại lệ.

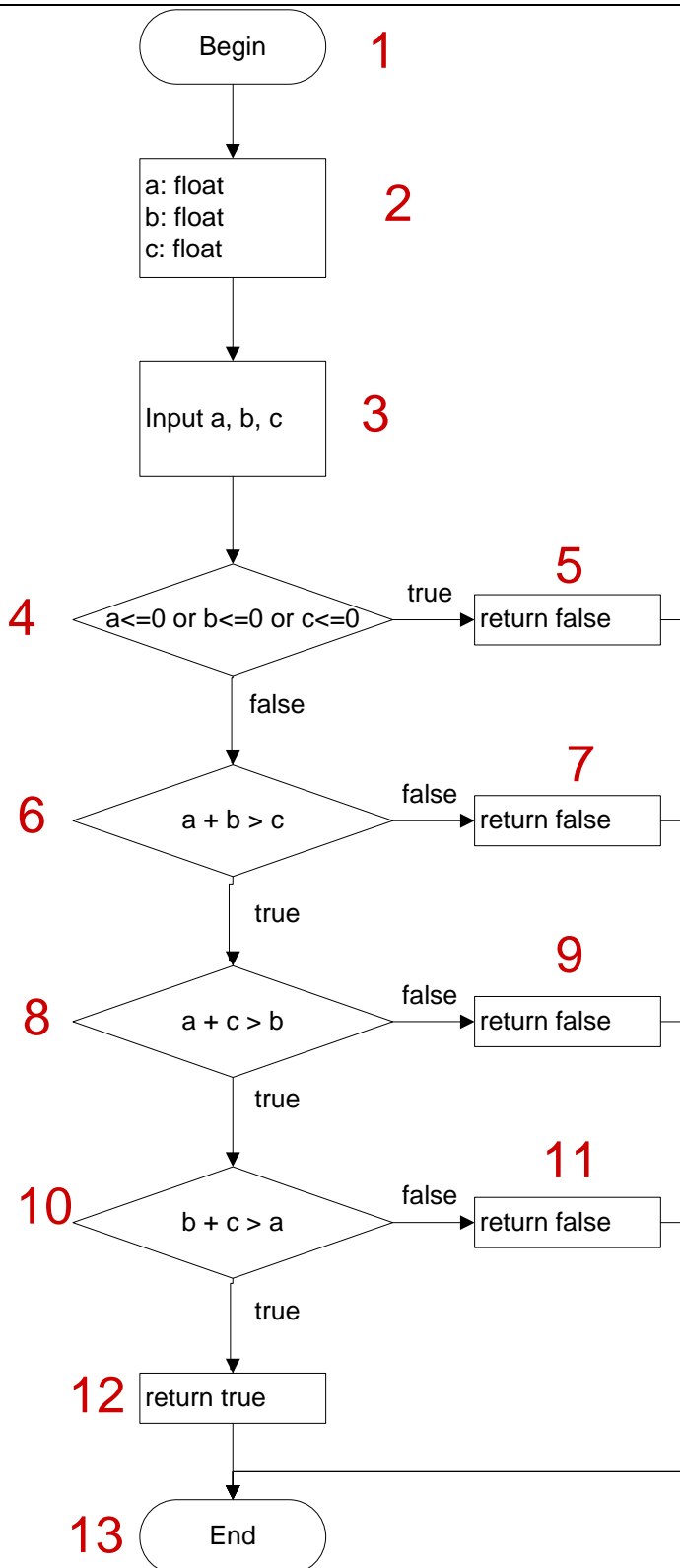


Bài 9.

Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%. Nếu dữ liệu không hợp lệ thì hàm ném ra ngoại lệ. Chú ý: Giới hạn một số trong khoảng (a, b): $x < a$ thì return a, $x > b$ thì return b, $a \leq x \leq b$ thì return x.

**Bài 10.**

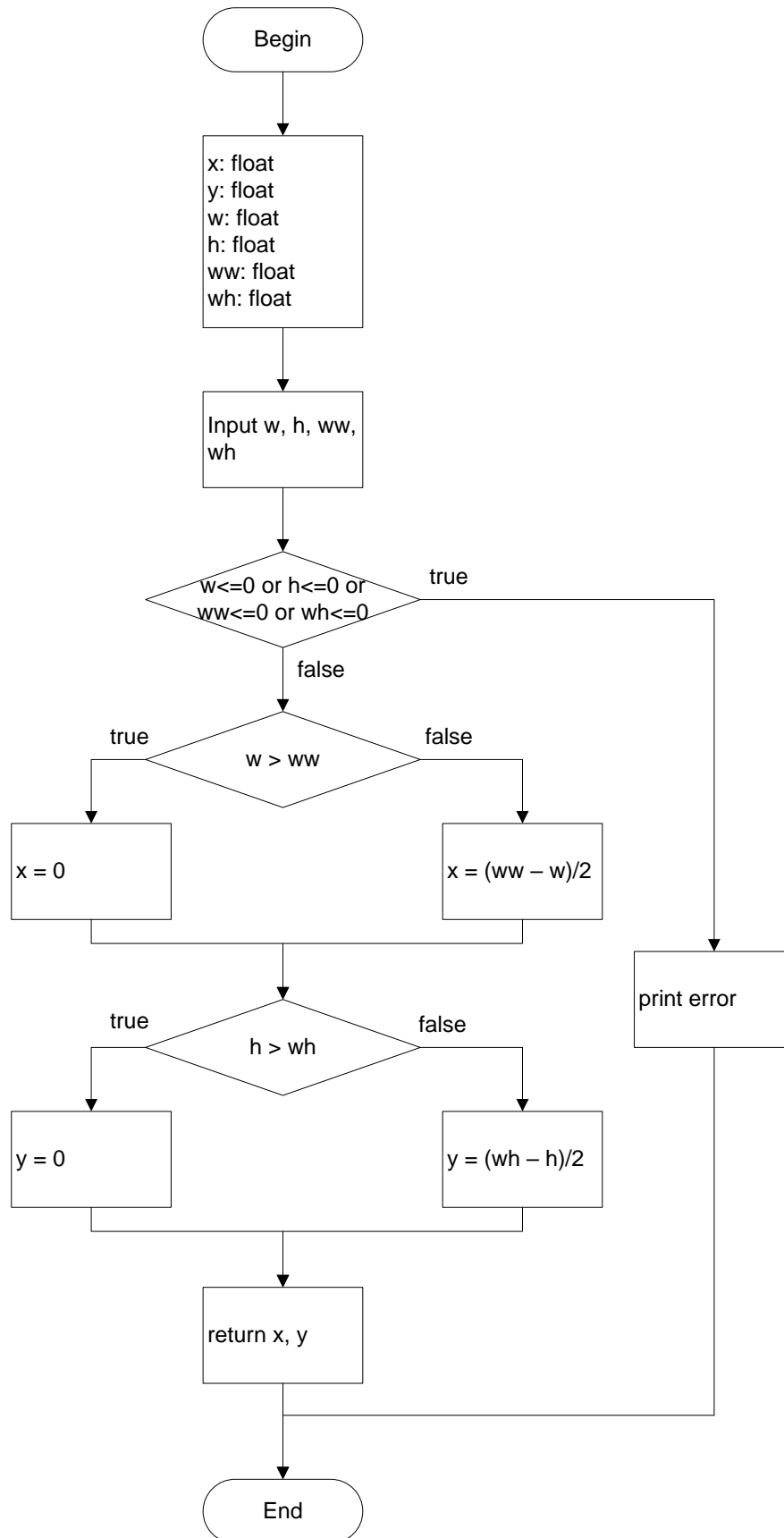
Đây là lược đồ mô tả cho chương trình kiểm tra xem 3 số a, b, c có là độ dài của 3 cạnh của một tam giác hay không, nếu có hàm trả về true, ngược lại trả về false. Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.



Bài 11.

Cho thuật toán căn giữa ảnh vào cửa sổ trong các chương trình xem ảnh: nếu ảnh lớn hơn cửa sổ, thì ảnh được vẽ theo góc trên trái cửa sổ, nếu ảnh nhỏ hơn cửa sổ thì được căn vào giữa cửa sổ. Nếu dữ liệu không hợp lệ thì hàm ném ra ngoại lệ.

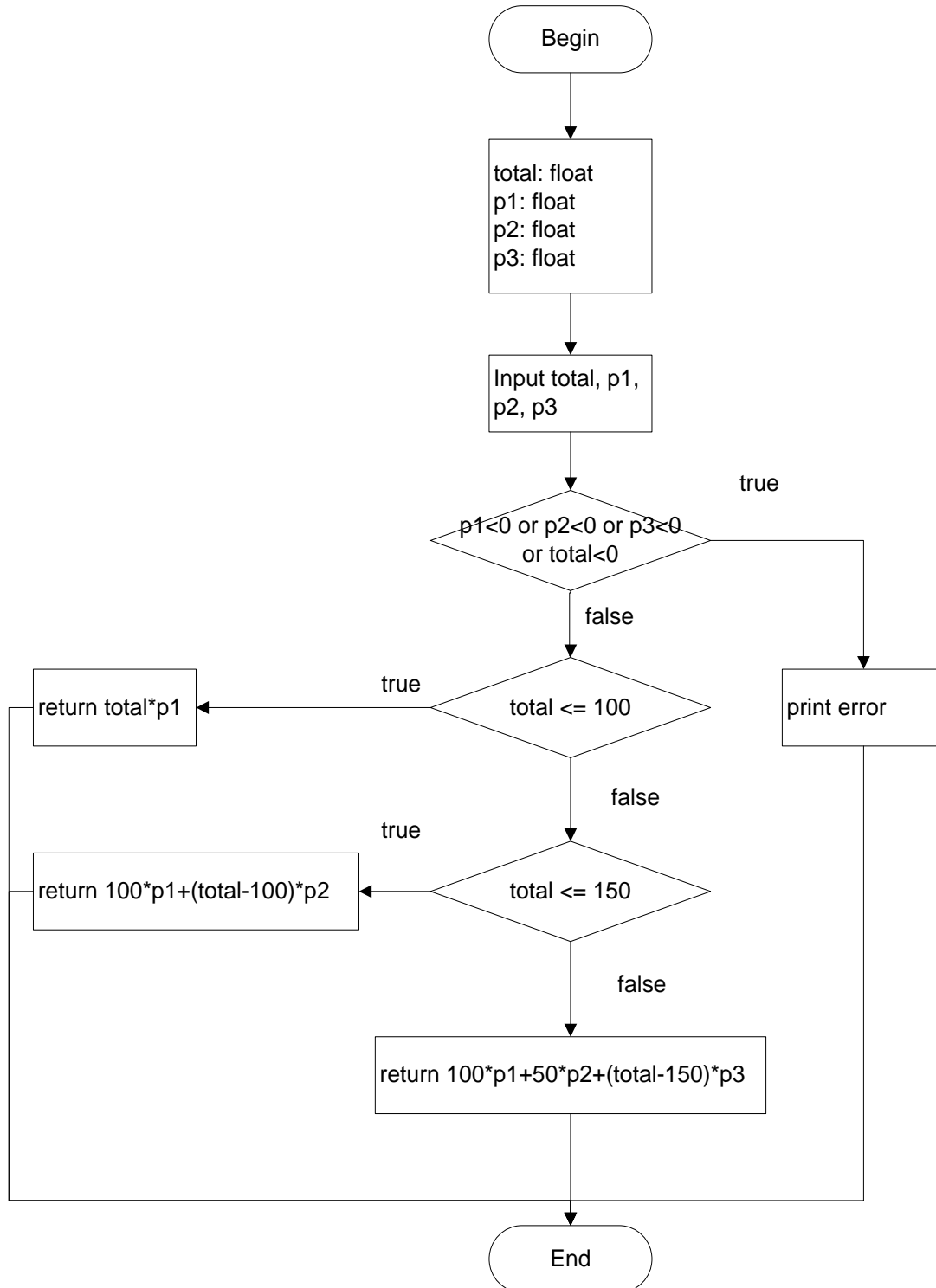
Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.



Bài 12.

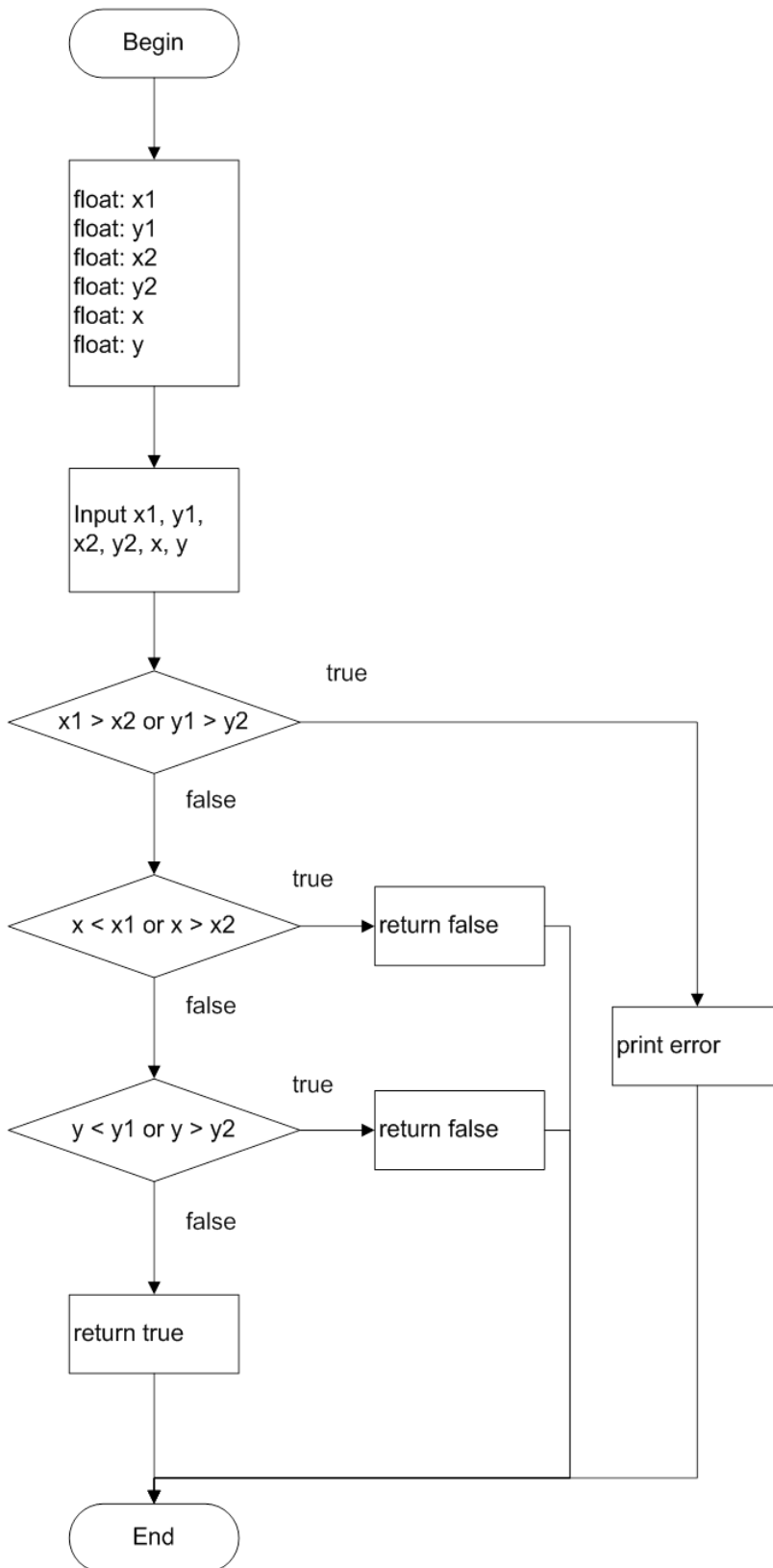
Cho thuật toán tính giá nhiều mức (như tính giá điện nước): từ 0 \rightarrow 100 giá p1, từ 101 \rightarrow 150 giá p2, từ 150 trở lên tính giá p3. Nếu dữ liệu không hợp lệ thì hàm ném ra ngoại lệ.

Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.



Bài 13.

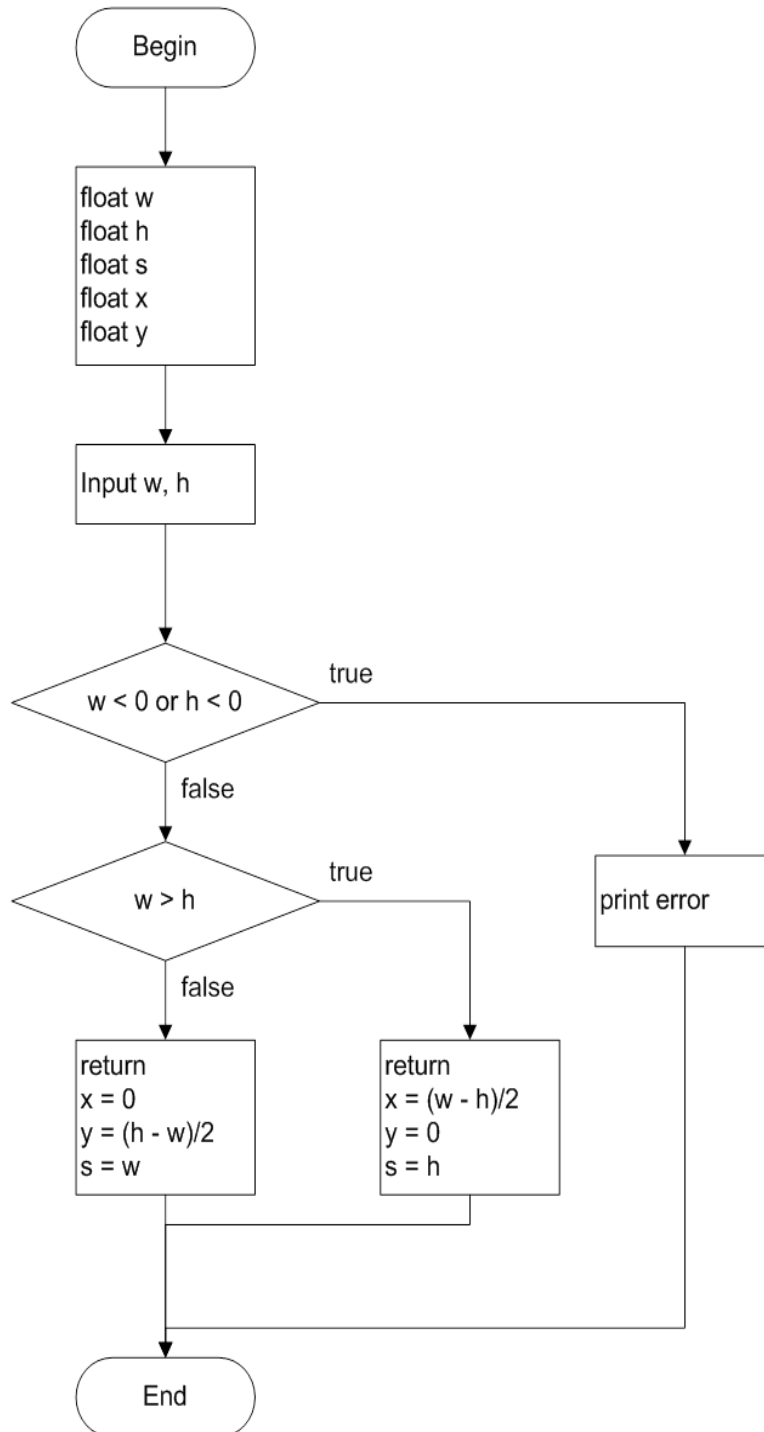
Cho thuật toán kiểm tra một điểm (x, y) có nằm trong hình chữ nhật $((x1, y1), (x2, y2))$ hay không. Nếu dữ liệu không hợp lệ thì hàm ném ra ngoại lệ. Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.



Bài 14.

Cho thuật toán tạo một Icon (ảnh vuông) từ một ảnh có kích cỡ bất kỳ mà không làm méo ảnh. Đầu vào là kích cỡ ảnh (w, h), đầu ra là vị trí và kích cỡ cắt ảnh (x, y, s).

Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.

**Bài 15.**

Cho thuật toán kiểm tra năm nhuận cho các giá trị nhập từ 1000 đến 10000. Hàm trả về true nếu là năm nhuận; là false khi không phải năm nhuận hoặc giá trị không hợp lệ. Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.

START

Validate boundary value

If **year** > 10000 or **year** < 1000

return false

Else

If **year** % 100 = 0

If **year** % 400 = 0

return true

Else

return false

Else If **year** % 4 = 0

return true

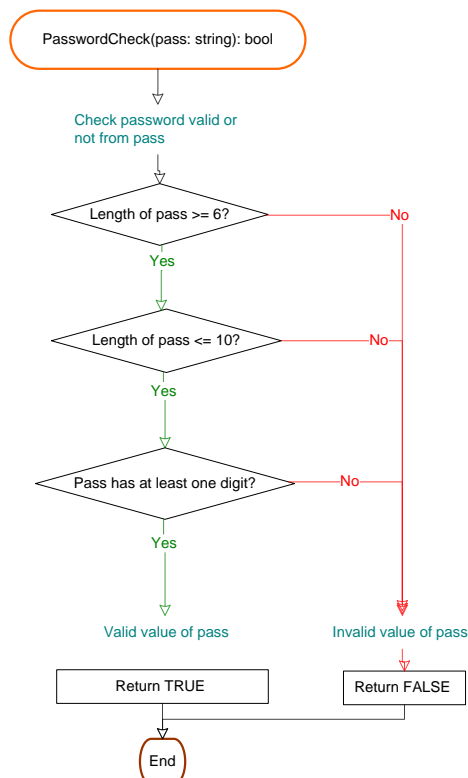
Else

return false

END**Bài 16.**

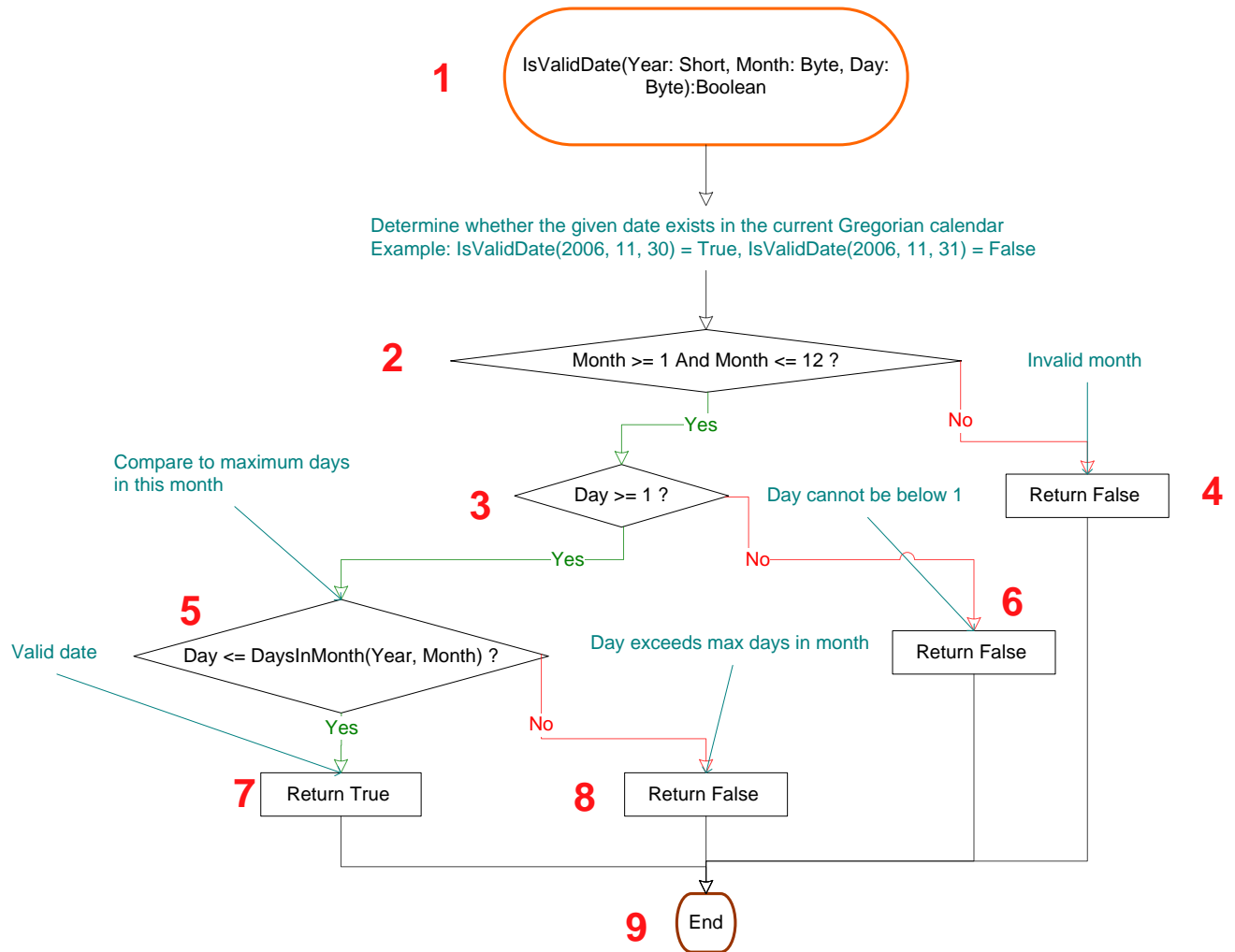
Cho thuật toán kiểm tra password: có từ 6 đến 10 ký tự trong đó có ít nhất 1 ký số.

Thiết kế các test case sao cho lược đồ sau đạt bao phủ nhánh 100%.



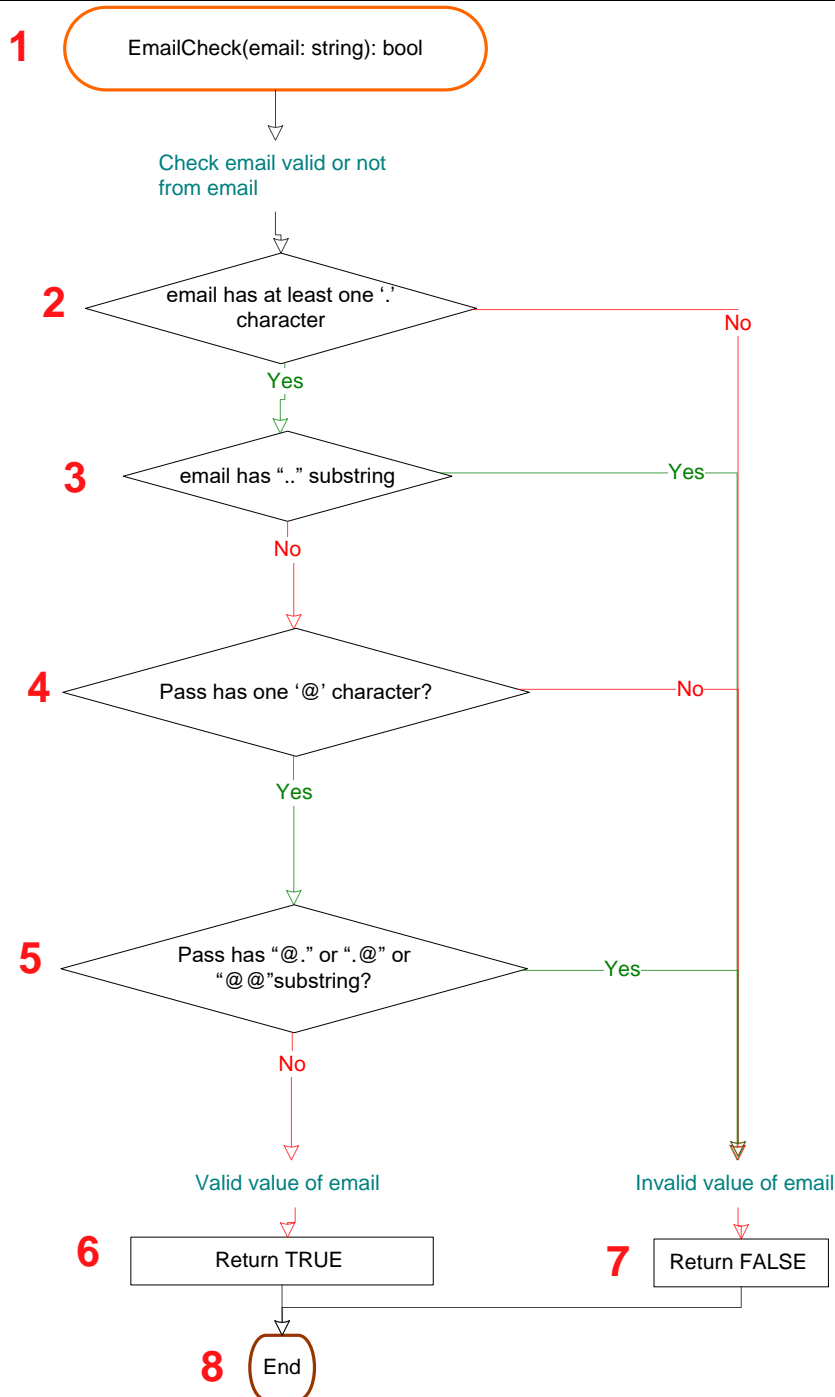
Bài 17.

Cho thuật toán kiểm tra 3 số nhập vào có phải ngày hợp lệ không theo lược đồ sau. Thiết kế các test case sao cho lược đồ đạt bao phủ nhánh 100%.



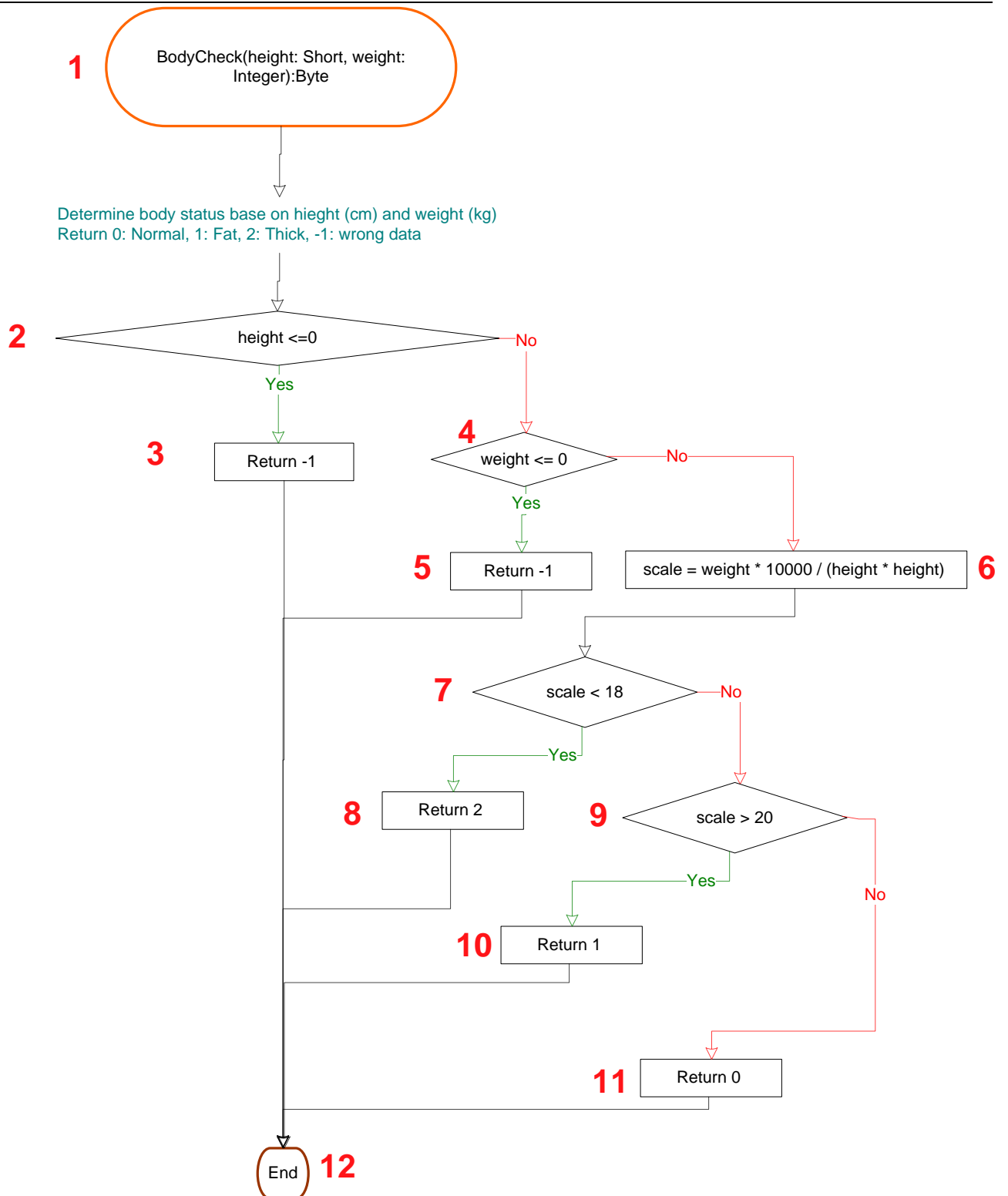
Bài 18.

Cho thuật toán kiểm tra email như lược đồ sau. Thiết kế các test case sao cho lược đồ đạt bao phủ nhánh 100%.



Bài 19.

Cho thuật toán kiểm tra chỉ số cơ thể như mô tả của lược đồ sau. Thiết kế các test case sao cho lược đồ đạt bao phủ nhánh 100%.



Bài 20.

Cho hàm tìm vị trí đầu tiên của một ký tự cho trước trong một chuỗi. Thiết kế các test case sao cho thuật toán đạt bao phủ nhánh 100%.

```
#define MAX_INT 32767
char str[] = "Statement Coverage";
unsigned char isInString(char tmp)
{
    int pos = MAX_INT;
    int i = 0;
    while(str[i] != '\0'){
        if(str[i] == tmp){
            pos = i;
            break;
        }
        i++;
    }
    return pos;
}
```

Bài 21.

Cho hàm **HexToDec** chuyển đổi chuỗi số dạng thập lục phân (*hexaString*) sang số dạng thập phân. Chỉ chấp nhận chuỗi ký tự thập lục phân.

- Viết các test case để đạt được 100% độ bao phủ lệnh, 100% độ bao phủ nhánh. Thực thi các test case này.
- Kiểm tra chuỗi dạng “059”, “ace”, “ACD” mức bao phủ đạt được là gì?

```
1 long HexToDec(string hexaString) {
2     int c;
3     long hexnum, nhex;
4     hexnum = nhex = 0;
5     int i = 0;
6     while (i < hexaString.Length)
7     {
8         c = hexaString[i++];      gán ri mi tng
9         switch (c)
10        {
11            case '0': case '1': case '2':
12            case '3': case '4': case '5':
13            case '6': case '7': case '8':
14            case '9':
15                /* Convert a decimal digit */
16                nhex++;
17                hexnum *= 0x10;
18                hexnum += (c - '0');
```

```
18 break;

19 case 'a': case 'b': case 'c':
20 case 'd': case 'e': case 'f':
    /* lower case hex digit */

21 nhex++;
22 hexnum *= 0x10;
23 hexnum += (c - 'a' + 0xa);
24 break;

25 case 'A': case 'B': case 'C':
26 case 'D': case 'E': case 'F':
    /*upper case hex digit */

27 nhex++;
28 hexnum *= 0x10;
29 hexnum += (c - 'A' + 0xA);
30 break;
31 default:
    /* non-hex characters */

32 break;
33 }
34 }
35 return hexnum;
36 }
```

Bài 22.

Thực thi các test case đã thiết kế trong module này.

Hướng dẫn: thực hiện theo hướng dẫn trong file TestingInVS2013_TDEV-H211.pdf như sau:

B1. Tạo class library: Exercise 1 - Task 1.

B2. Tạo unit test project: Exercise 1 - Task 2.

Bài 23. Hướng dẫn tính độ bao phủ mã (Code Coverage):

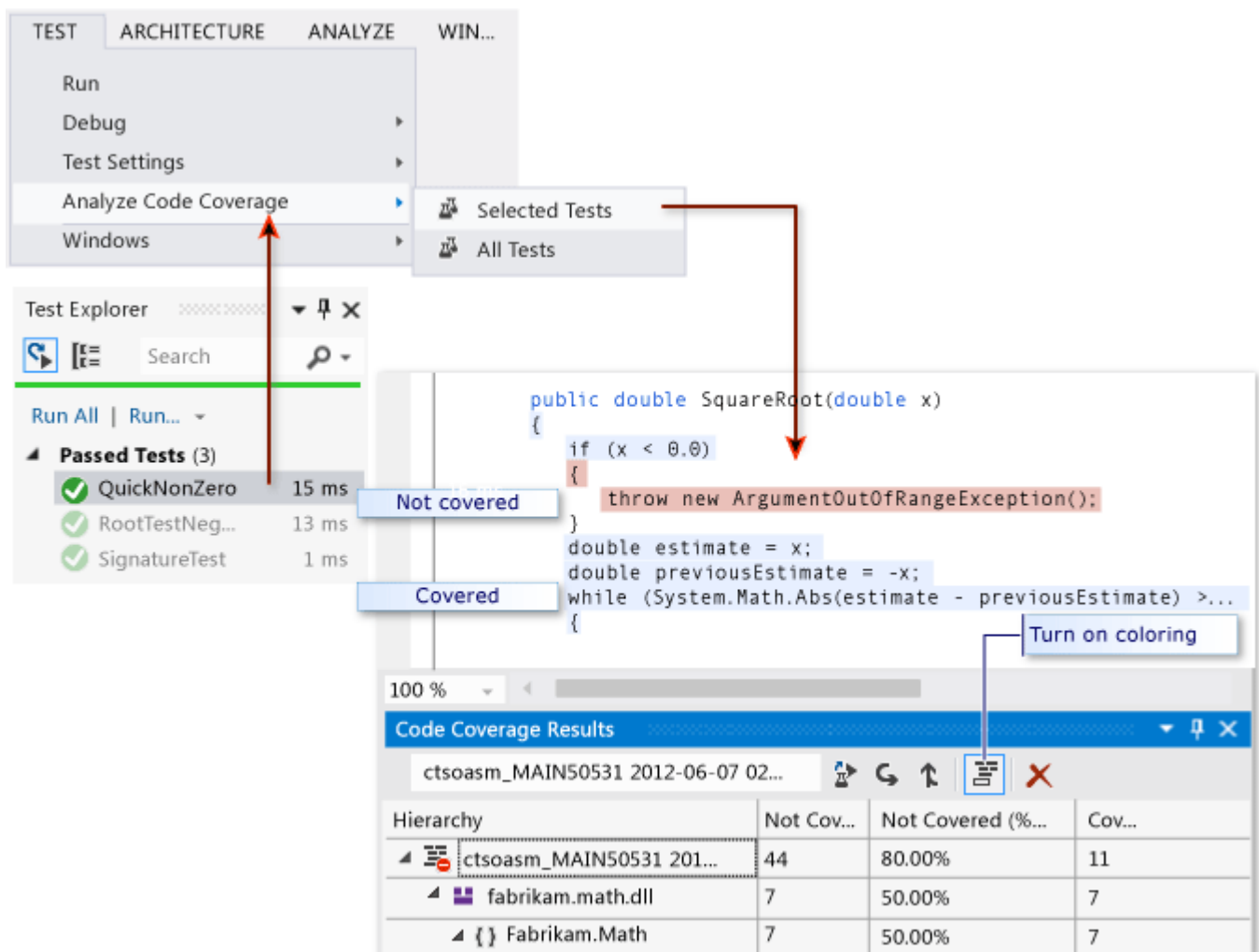
A. Analyze code coverage

[https://msdn.microsoft.com/en-us/library/dd537628\(v=vs.120\).aspx](https://msdn.microsoft.com/en-us/library/dd537628(v=vs.120).aspx)

To determine what proportion of your project's code is actually being tested by coded tests such as unit tests, you can use the code coverage feature of Visual Studio. To guard effectively against bugs, your tests should exercise or 'cover' a large proportion of your code.

Code coverage analysis can be applied to both managed (CLI) and unmanaged (native) code.

Code coverage is an option when you run test methods using Test Explorer. The results table shows the percentage of the code that was run in each assembly, class, and method. In addition, the source editor shows you which code has been tested.



The screenshot illustrates the process of analyzing code coverage in Visual Studio. The 'TEST' menu is open, showing options like 'Run', 'Debug', 'Test Settings', 'Analyze Code Coverage', and 'Windows'. The 'Analyze Code Coverage' option is selected, and a submenu shows 'Selected Tests' and 'All Tests'. The 'Test Explorer' window shows 'Passed Tests (3)' with a list of tests: 'QuickNonZero' (15 ms), 'RootTestNeg...' (13 ms), and 'SignatureTest' (1 ms). The 'Code Coverage Results' window shows a table with columns: Hierarchy, Not Covered, Not Covered (%), and Covered. The table shows results for 'ctsoasm_MAIN50531 2012-06-07 02...', 'fabrikam.math.dll', and 'Fabrikam.Math'. The source editor shows a 'SquareRoot' method with code blocks highlighted as 'Not covered' or 'Covered'. A 'Turn on coloring' button is also visible.

Hierarchy	Not Covered	Not Covered (%)	Covered
ctsoasm_MAIN50531 2012-06-07 02...	44	80.00%	11
fabrikam.math.dll	7	50.00%	7
{ } Fabrikam.Math	7	50.00%	7

B. Reporting in blocks or lines

Code coverage is counted in blocks. A block is a piece of code with exactly one entry and exit point. If the program's control flow passes through a block during a test run, that block is counted as covered. The number of times the block is used has no effect on the result.

You can also have the results displayed in terms of lines by choosing Add/Remove Columns in the table header. If the test run exercised all the code blocks in any line of code, it is counted as one line. Where a line contains some code blocks that were exercised and some that were not, that is counted as a partial line.

Some users prefer a count of lines because the percentages correspond more closely to the size of the fragments that you see in the source code. A long block of calculation would count as a single block even if it occupies many lines.

C. Managing code coverage results

The Code Coverage Results window usually shows the result of the most recent run. The results will vary if you change your test data, or if you run only some of your tests each time.

The code coverage window can also be used to view previous results, or results obtained on other computers.

You can merge the results of several runs, for example from runs that use different test data.

To view a previous set of results, select it from the drop-down menu. The menu shows a temporary list that is cleared when you open a new solution.

To view results from a previous session, choose Import Code Coverage Results, navigate to the TestResults folder in your solution, and import a .coverage file.

The coverage coloring might be incorrect if the source code has changed since the .coverage file was generated.

To make results readable as text, choose Export Code Coverage Results. This generates a readable .coveragexml file which you could process with other tools or send easily in mail.

To send results to someone else, send either a .coverage file or an exported .coveragexml file. They can then import the file. If they have the same version of the source code, they can see coverage coloring.

Module 4. Kiểm thử tĩnh

Nội dung kiến thức thực hành:

- + Review đặc tả.
- + Review code, xác định và fix các common defects.

Yêu cầu lưu trữ:

Sinh viên mở MSWord hoặc MSeExcel làm bài và đặt tên theo mẫu ***HoTenSV_Module04.docx***, nộp lại vào cuối buổi thực hành.

Bài 1.

Cho đặc tả yêu cầu trong tập tin *ReviewSRS.pdf*.

Review cho đặc tả này để tìm các yêu cầu không thể kiểm chứng, nêu lý do của sự lựa chọn này.

Bài 2.

Cho code java trong file *cocodepackage-com.doc*. Dùng file *Check-list.pdf* để kiểm tra file code này.

Bài 3.

Cho code triển khai hệ thống ATM bằng ngôn ngữ C#.NET. Trong source codes vẫn còn nhiều common defects tiềm ẩn.

Yêu cầu:

Review codes để xác định các common defects (theo như mô tả trong Fsoft coding conventions – trong file *Day02_04_Standard_CSharp Coding Convention.pdf* và các common defects trong bài học – trong file *Day02_01_Coding Process.pdf*), và đề xuất phương án fix lỗi.

Tìm tối thiểu 30 lỗi thuộc các loại lỗi khác nhau, mỗi loại lỗi có không quá 2 lỗi.

Các kết quả được log vào file excel như template *Template_Defects.xls*.

Module 5. Kế hoạch kiểm thử

Nội dung kiến thức thực hành:

- + Review test plan.
- + Xác định các thành phần trong một test plan.
- + Xác định chiến lược kiểm thử từ một SRS cho trước.

Bài 1.

Tự tìm một test plan bất kỳ hoặc được cho sẵn, kiểm tra xem test plan này có đủ các mục theo chuẩn IEEE không, nếu thiếu thì là thành phần nào.

Bài 2.

Cho trước các file sau:

- *Day01_Lesson01_Test Plan_v1.0.pdf*: chứa cấu trúc chung của một test plan.
- *Day01_Template_Test Plan.dotx*: mô tả thông tin chi tiết của một test plan (để tham khảo thêm).
- *Day01_Lesson01_Test Plan_sample_v1.0.doc*: **test plan sample**.

Yêu cầu: Kiểm tra file **test plan sample** xem có đủ các thành phần theo yêu cầu hay không (thành phần nào không có thì phải ghi “Không”).

Thực hiện theo mẫu sau:

Các thành phần cần có của 1 test plan		Test plan sample
Đề mục chính	Đề mục con	Chỉ số đề mục tương ứng (trong test plan sample)
1. Introduction	Purpose	
	Background information	
	Document reference	
	Scope of testing	
	Constraints	
	Risk list	
2. Requirement for test		
3. Test strategy	Test types	
	Test stage	
	Completion criteria	
	Test tool	
4. Resource	Human resource	
	System	
5. Milestones		
6. Deliverables	Product deliverable name	
	Deliverable date	

	Delivered by	
	Delivered to	

Ghi chú: Các file được để trong thư mục Assignment_01.

Bài 3.

Cho tài liệu SRS *AB-SD_Software Requirements Specification.doc* và một template *Day01_Lesson01_Assignment_Template.xlsx*, yêu cầu SV điền thông tin về các functional requirement, non-functional requirement vào template, sau đó tìm các test type phù hợp và chỉ ra reference đến SRS.

Thực hiện theo mẫu sau:

Test Type <Test Types in high-level>	Sub <Test Type in details-level>	Requirement References	
		Requirement Title/Index	Page No.
(ghi Functional hoặc Non-functional testing)			

Ghi chú: Các file được để trong thư mục Assignment_02.

Module 6. Coded UI Test

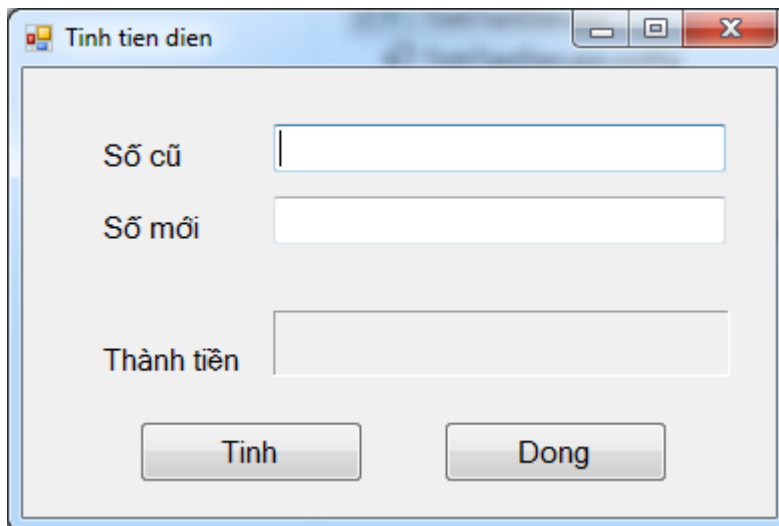
Nội dung kiến thức thực hành:

- + Thực hiện test cho ứng dụng.
- + Sử dụng check-point.

Chú ý: SV tự code cho ứng dụng và lập bảng test case trước khi thực thi test.

Bài 1.

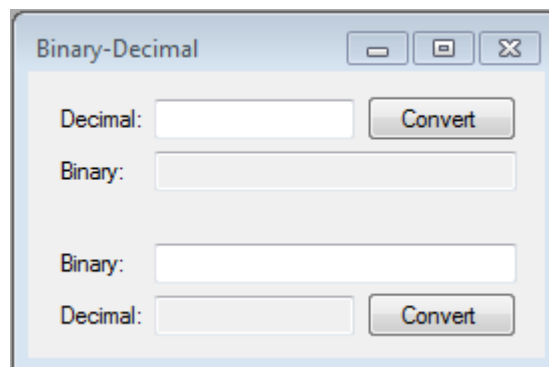
Kiểm thử cho chương trình tính tiền điện có giao diện như sau:



Cho mô tả cách tính tiền điện giống bài tập trong module 2.

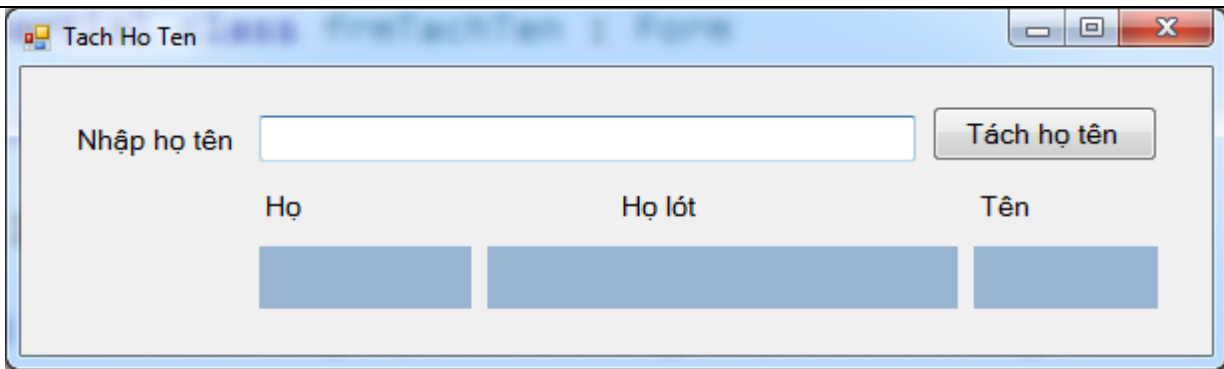
Bài 2.

Cho chương trình chuyển đổi số nhị phân sang thập phân (và ngược lại) với giao diện như hình. Dùng Code UI Test kiểm thử thông qua giao diện chương trình:



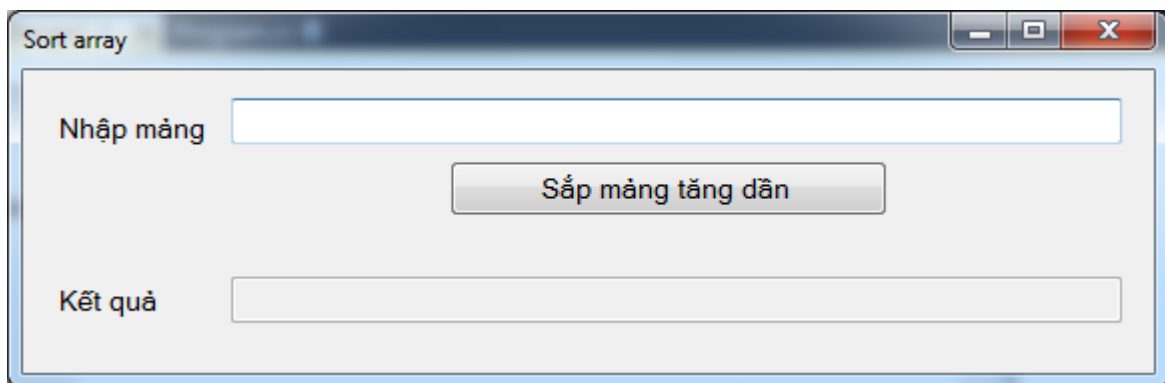
Bài 3.

Cho chương trình tách họ tên như hình, yêu cầu họ tên phải có ít nhất 2 từ. Kiểm chứng chương trình này bằng Coded UI Test.



Bài 4.

Kiểm thử chương trình sắp xếp mảng số nguyên tăng dần bằng phương pháp QuickSort sau đây:



Bài 5. Hướng dẫn thực hiện Coded UI Test trong Visual Studio 2013

A. Creating Coded UI Tests

[https://msdn.microsoft.com/en-us/library/dd286726\(v=vs.120\).aspx](https://msdn.microsoft.com/en-us/library/dd286726(v=vs.120).aspx)

1. Create a Coded UI Test project.

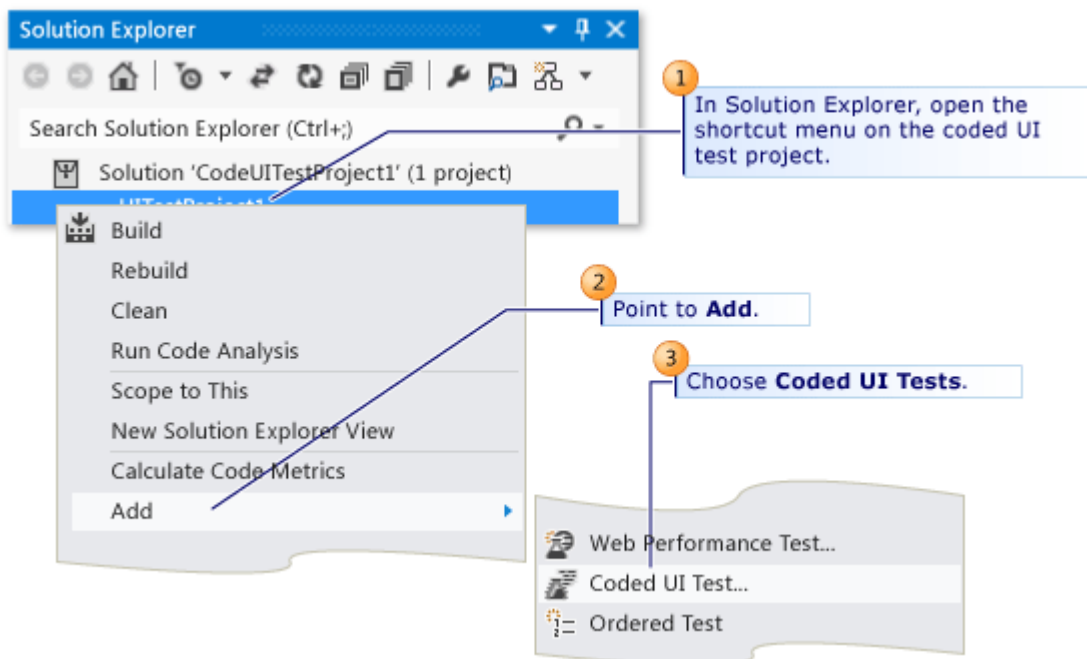
Coded UI tests must be contained in a coded UI test project. If you don't already have a coded UI test project, create one. In **Solution Explorer**, on the shortcut menu of the solution, choose **Add, New Project** and then select either **Visual Basic** or **Visual C#**. Next, choose **Test, Coded UI Test**.

[Q: I don't see the **Coded UI Test** project templates.

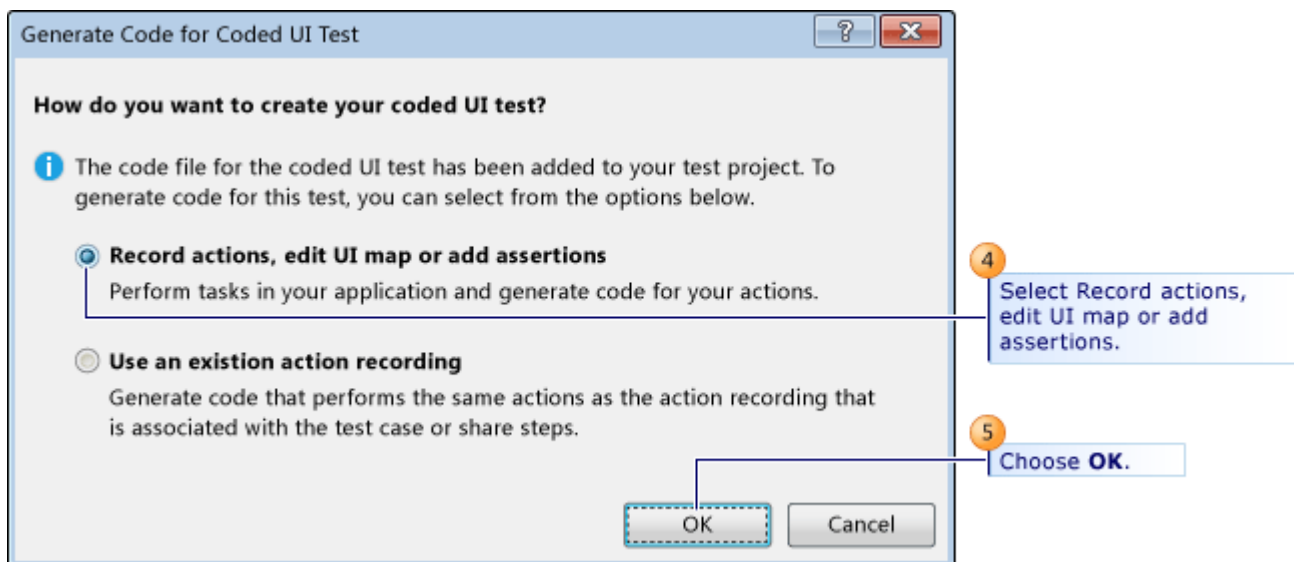
A: You might be using a version of Microsoft Visual Studio 2012 that does not support coded UI tests. To create coded UI tests, you must use either Visual Studio Ultimate or Visual Studio Premium.]

2. Add a coded UI test file.

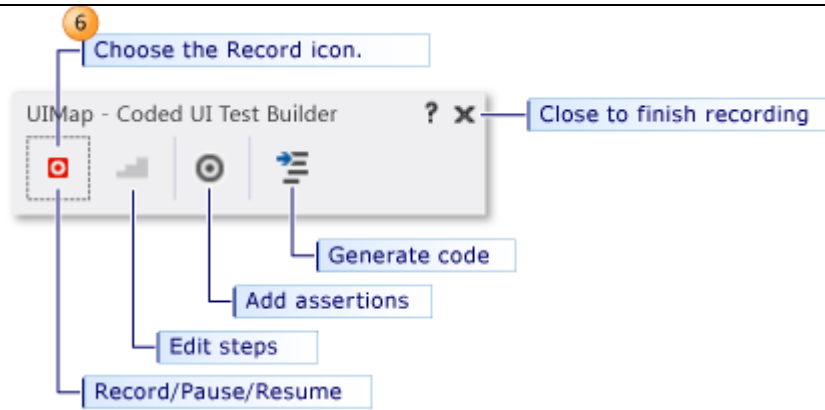
If you just created a Coded UI project, the first CUIT file is added automatically. To add another test file, open the shortcut menu on the coded UI test project, point to **Add**, and then choose **Coded UI Test**.



In the **Generate Code for Coded UI Test** dialog box, choose **Record actions, edit UI map or add assertions**.



The Coded UI Test Builder appears and Visual Studio is minimized.



3. Record a sequence of actions.

To start recording, choose the **Record** icon. Perform the actions that you want to test in your application, including starting the application if that is required.

For example, if you are testing a web application, you might start a browser, navigate to the web site, and log in to the application.

To pause recording, for example if you have to deal with incoming mail, choose **Pause**.

Caution

All actions performed on the desktop will be recorded. Pause the recording if you are performing actions that may lead to sensitive data being included in the recording.

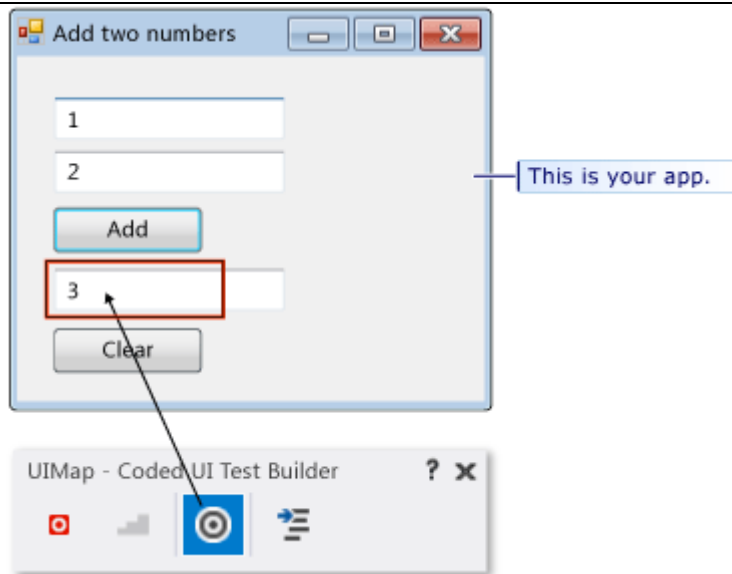
To delete actions that you recorded by mistake, choose **Edit Actions**.

To generate code that will replicate your actions, choose the **Generate Code** icon and type a name and description for your coded UI test method.

4. Verify the values in UI fields such as text boxes.

Choose **Add Assertions** in the Coded UI Test Builder, and then choose a UI control in your running application. In the list of properties that appears, select a property, for example, **Text** in a text box. On the shortcut menu, choose **Add Assertion**. In the dialog box, select the comparison operator, the comparison value, and the error message.

Close the assertion window and choose **Generate Code**.



Tip

Alternate between recording actions and verifying values. Generate code at the end of each sequence of actions or verifications. If you want, you will be able to insert new actions and verifications later.

For more details, see [Validating Properties of Controls](#).

5. View the generated test code.

To view the generated code, close the UI Test Builder window. In the code, you can see the names that you gave to each step. The code is in the CUIT file that you created:

C#

```
[CodedUITest]
public class CodedUITest1
{ ...
    [TestMethod]
    public void CodedUITestMethod1()
    {
        this.UIMap.AddTwoNumbers();
        this.UIMap.VerifyResultValue();
        // To generate more code for this test, select
        // "Generate Code" from the shortcut menu.
    }
}
```

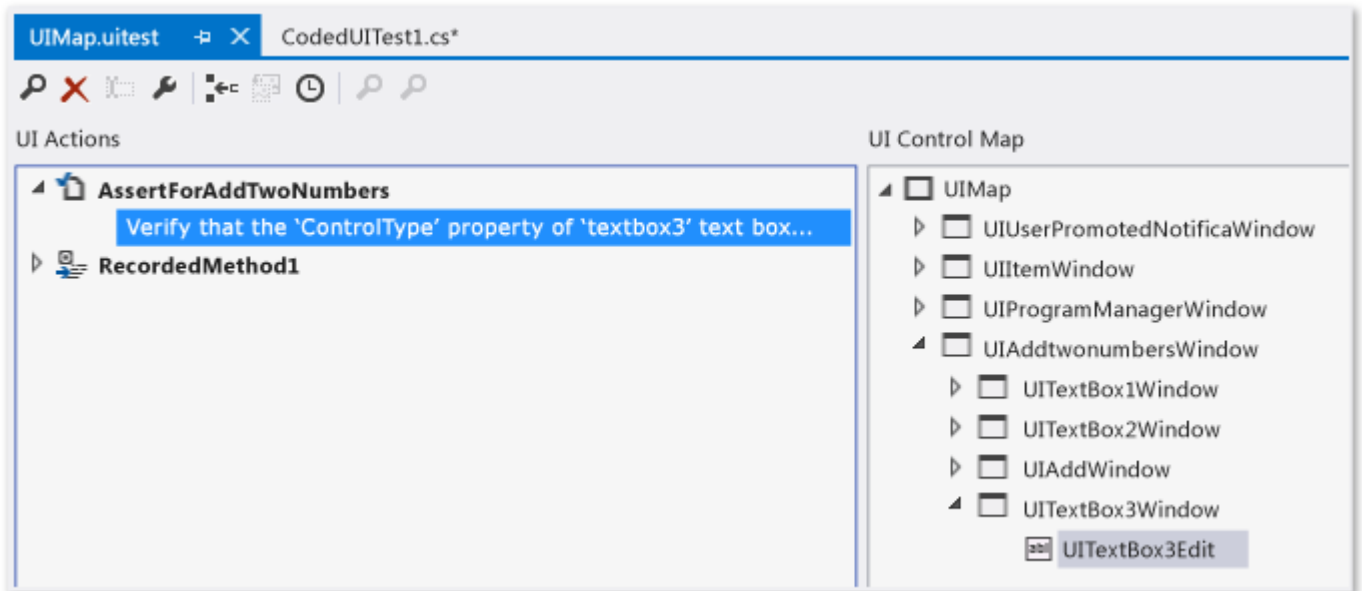
}

6. Add more actions and assertions.

Place the cursor at the appropriate point in the test method and then, on the shortcut menu, choose **Generate Code for Coded UI Test**. New code will be inserted at that point.

7. Edit the detail of the test actions and the assertions.

Open UIMap.uitest. This file opens in the Coded UI Test Editor, where you can edit any sequence of actions that you recorded as well as edit your assertions.



For more information, see [Editing Coded UI Tests Using the Coded UI Test Editor](#).

8. Run the test.

Use Test Explorer, or open the shortcut menu in the test method, and then choose **Run Tests**. For more information about how to run tests, see [Running Unit Tests with Test Explorer](#) and *Additional options for running coded UI tests* in the [What's next?](#) section at the end of this topic.

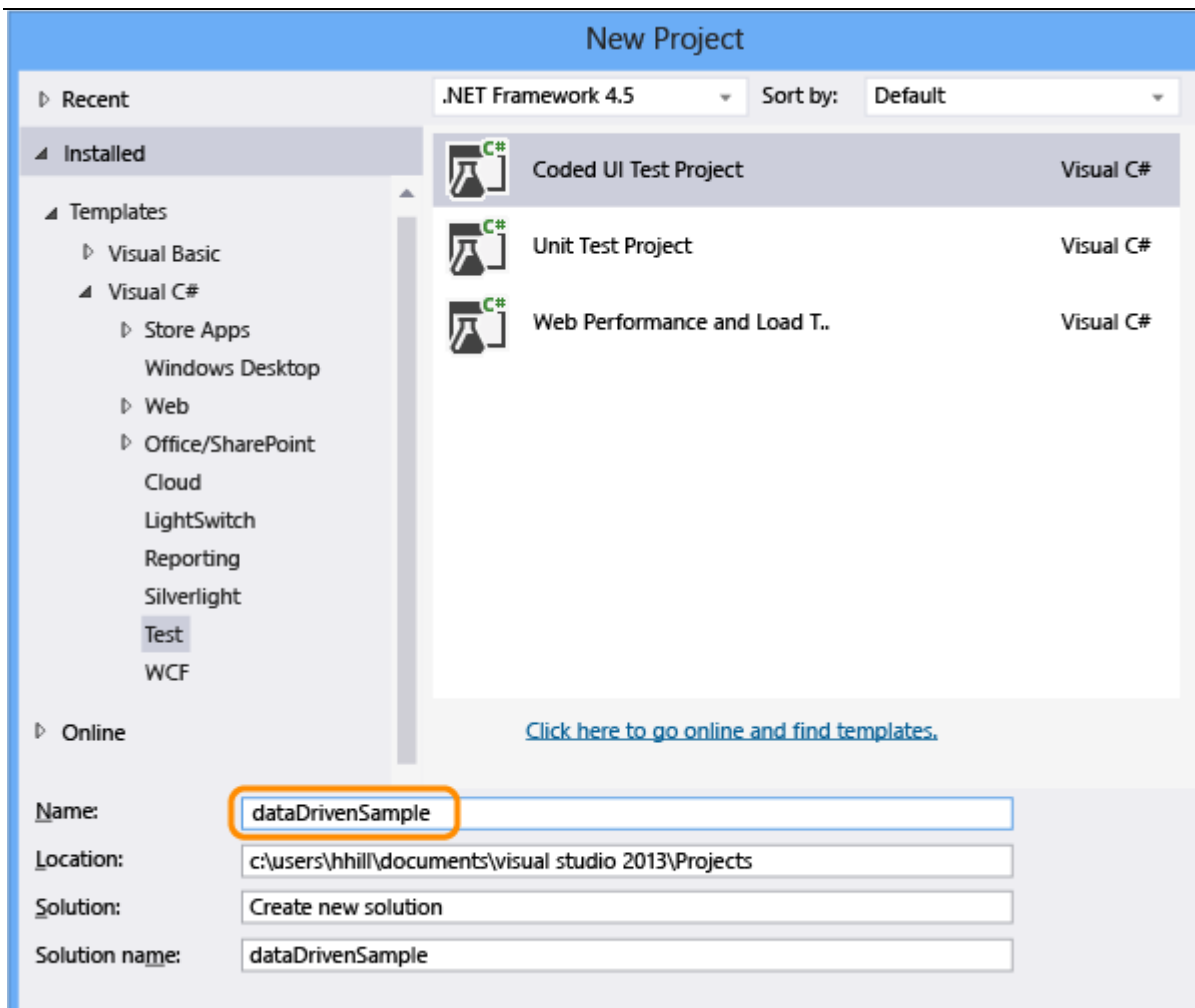
B. Creating a Data-Driven Coded UI Test

[https://msdn.microsoft.com/en-us/library/ee624082\(v=vs.120\).aspx](https://msdn.microsoft.com/en-us/library/ee624082(v=vs.120).aspx)

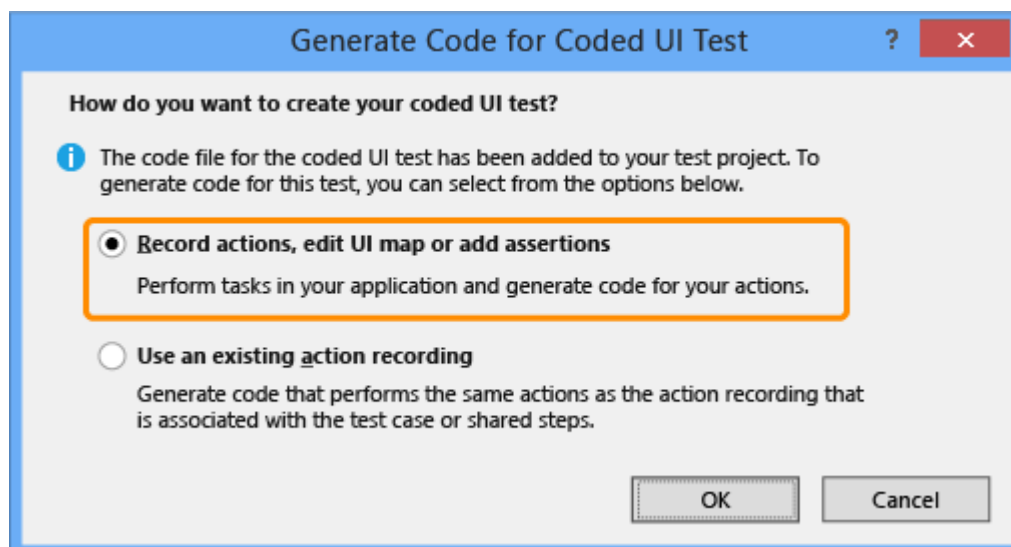
This sample creates a coded UI test that runs on the Windows Calculator application. It adds two numbers together and uses an assertion to validate that the sum is correct. Next, the assertion and the parameter values for the two numbers are coded to become data-driven and stored in a comma-separated value (.csv) file.

Step 1 - Create a coded UI test

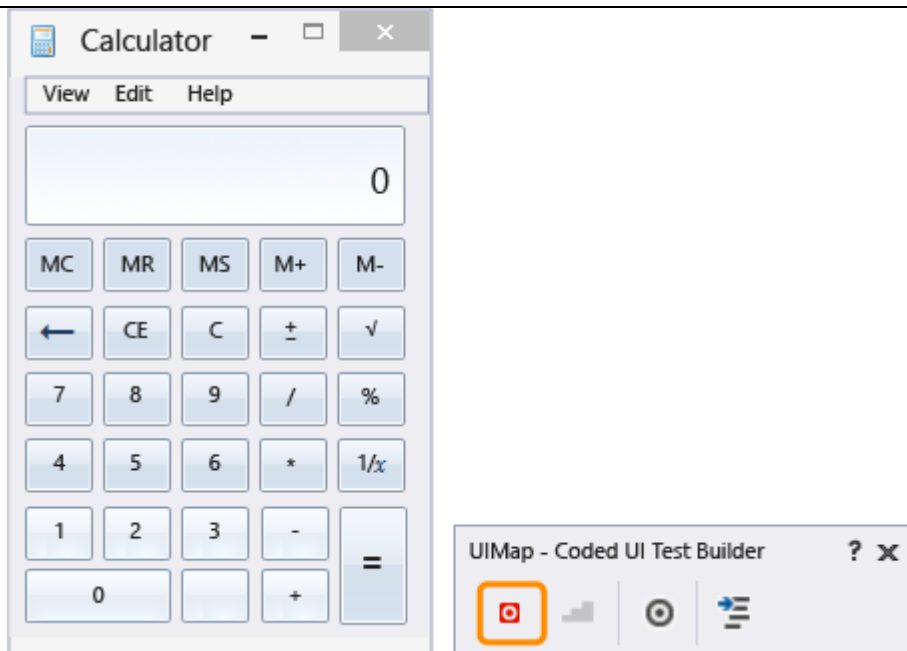
1. Create a project.



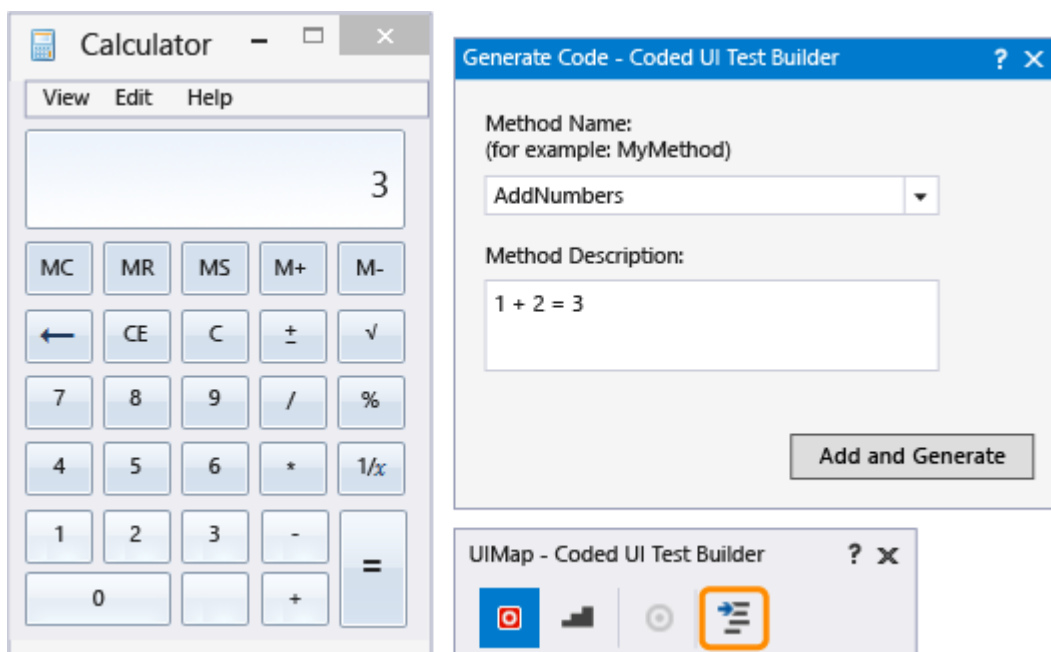
2. Choose to record the actions.



3. Open the calculator app and start recording the test.



4. Add 1 plus 2, pause the recorder, and generate the test method. Later we'll replace the values of this user input with values from a data file.



Close the test builder. The method is added to the test:

C#

```
[TestMethod]
```

```
public void CodedUITestMethod1()
```

```
{
```

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

```
this.UIMap.AddNumbers();
```

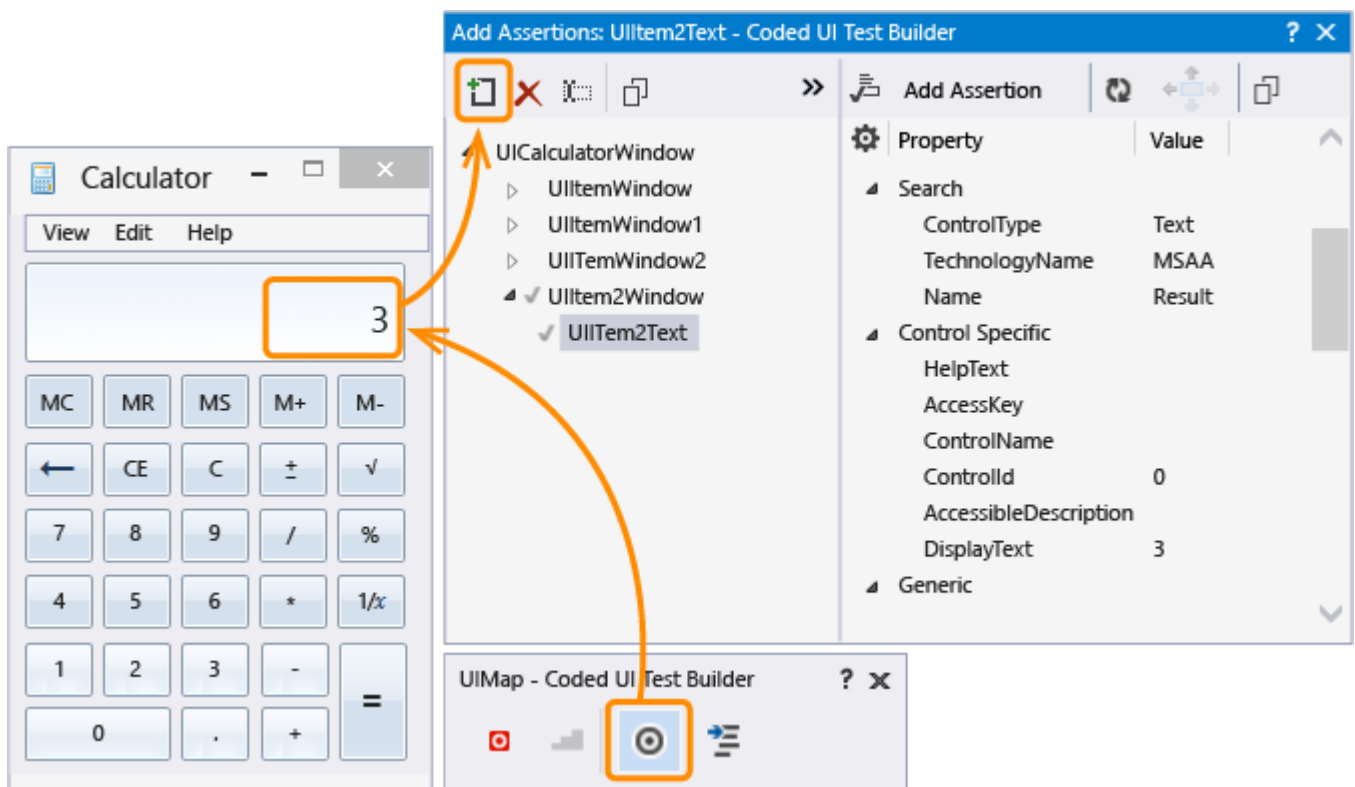
```
}
```

5. Use the `AddNumbers()` method to verify that the test runs. Place the cursor in the test method shown above, open the context menu, and choose **Run Tests**. (Keyboard shortcut: Ctrl + R, T).

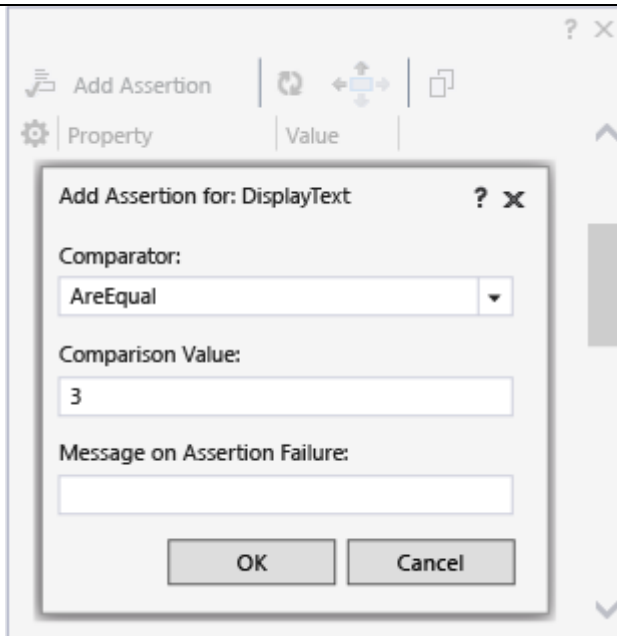
The test result that shows if the test passed or failed is displayed in the Test Explorer window. To open the Test Explorer window, from the **TEST** menu, choose **Windows** and then choose **Test Explorer**.

6. Because a data source can also be used for assertion parameter values—which are used by the test to verify expected values—let's add an assertion to validate that the sum of the two numbers is correct. Place the cursor in the test method shown above, open the context menu and choose **Generate Code for Coded UI Test**, and then **Use Coded UI Test Builder**.

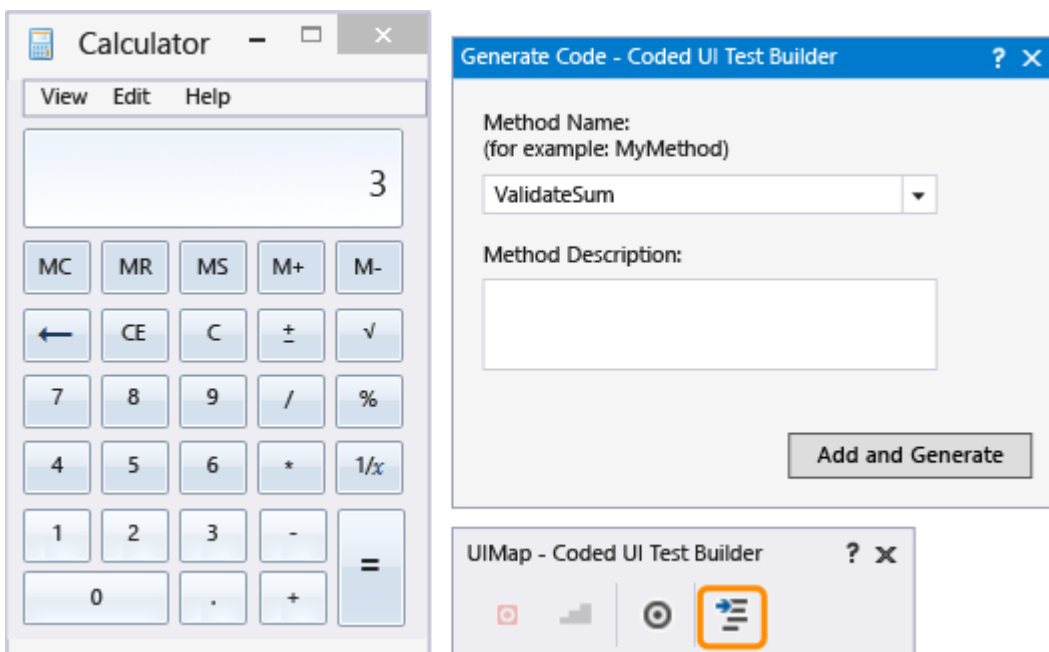
Map the text control in the calculator that displays the sum.



7. Add an assertion that validates that the value of the sum is correct. Choose the **DisplayText** property that has the value of 3 and then choose **Add Assertion**. Use the **AreEqual** comparator and verify that the comparison value is 3.



8. After configuring the assertion, generate code from the builder again. This creates a new method for the validation.



Because the `ValidateSum` method validates the results of the `AddNumbers` method, move it to the bottom of the code block.

C#

```
public void CodedUITestMethod1()
{
```

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

```
    this.UIMap.AddNumbers();
    this.UIMap.ValidateSum();
```

```
}

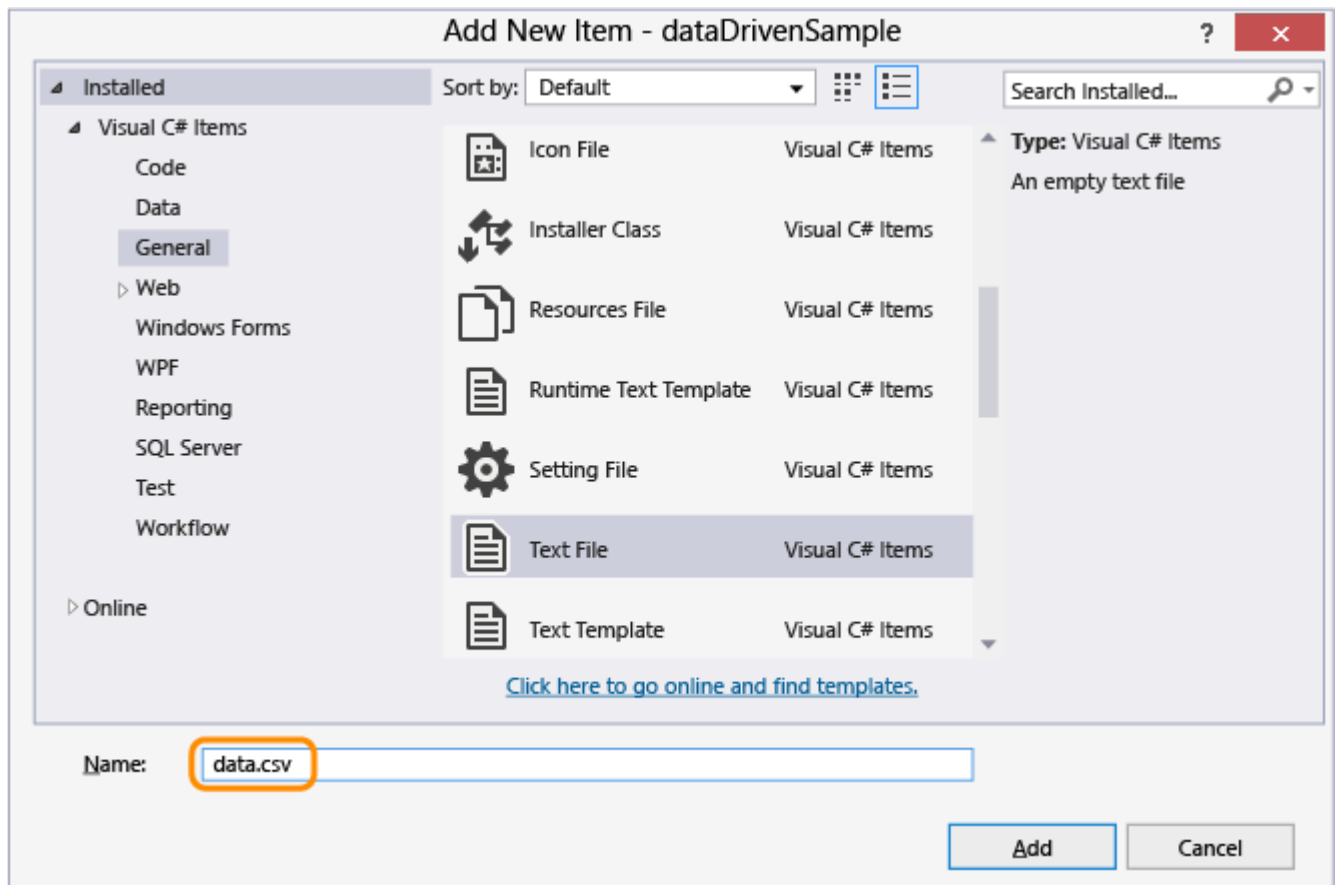
```

- Verify that the test runs by using the `ValidateSum()` method. Place the cursor in the test method shown above, open the context menu, and choose **Run Tests**. (Keyboard shortcut: Ctrl + R, T).

At this point, all the parameter values are defined in their methods as constants. Next, let's create a data set to make our test data-driven.

Step 2 - Create a data set

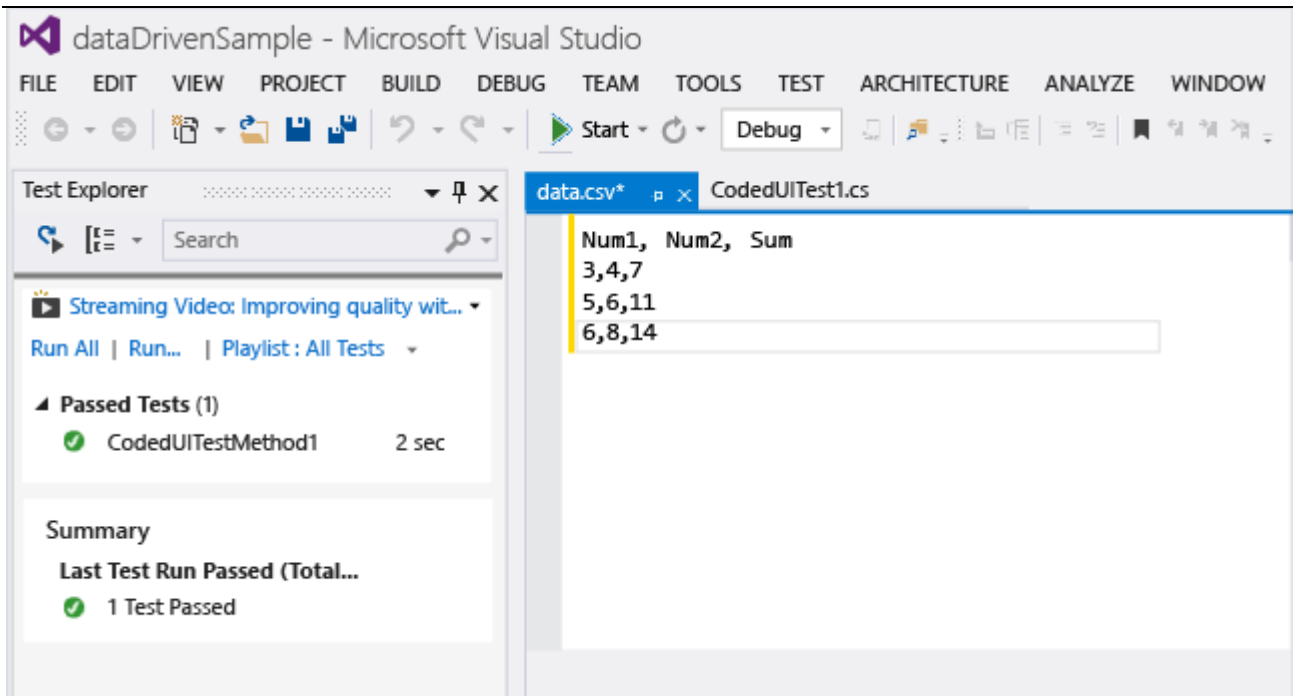
- Add a text file to the dataDrivenSample project named **data.csv**.



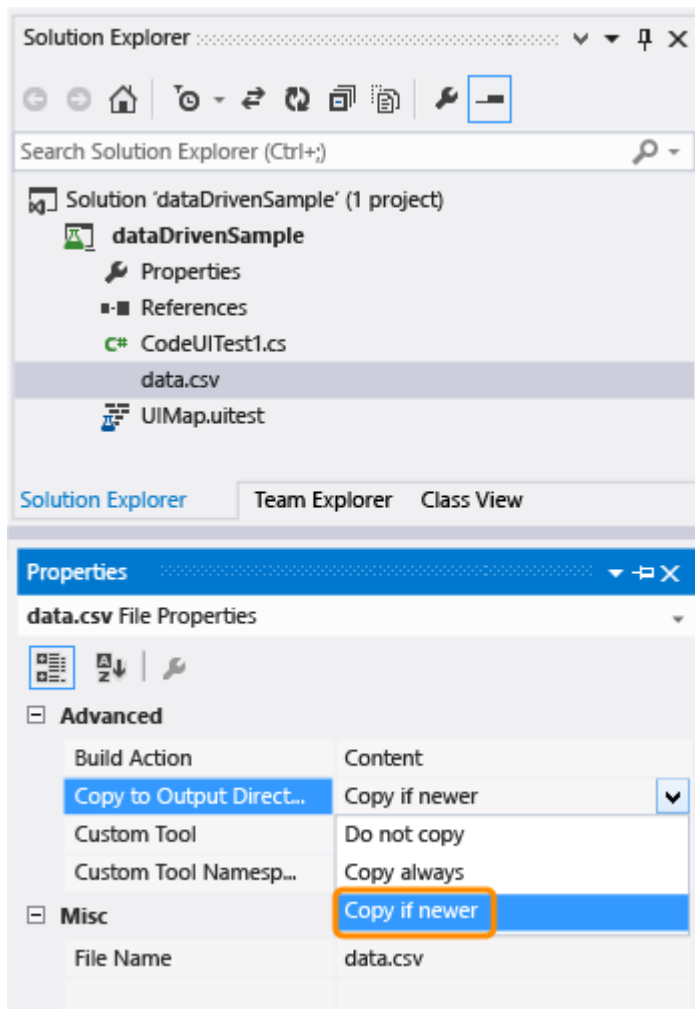
- Populate the .csv file with the following data (chú ý Num1, Num2 chỉ test được 1 chữ số)

Num1	Num2	Sum
3	4	7
5	6	11
6	8	14

After adding the data, the file should appear as the following:



- It is important to save the .csv file using the correct encoding. On the **FILE** menu, choose **Advanced Save Options** and choose **Unicode (UTF-8 without signature) – Codepage 65001** as the encoding.
- The .csv file, must be copied to the output directory, or the test can't run. Use the Properties window to copy it.



Now that we have the data set created, let's bind the data to the test.

Step 3 – Add data source binding

1. To bind the data source, add a `DataSource` attribute within the existing `[TestMethod]` attribute that is immediately above the test method.

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",  
"|DataDirectory|\\data.csv", "data#csv", DataAccessMethod.Sequential), DeploymentItem("data.csv"),  
TestMethod]
```

```
public void CodedUITestMethod1()  
{
```

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

```
    this.UIMap.AddNumbers();  
    this.UIMap.ValidateSum();  
}
```

The data source is now available for you to use in this test method.

2. Run the test.

Notice that the test runs through three iterations. This is because the data source that was bound contains three rows of data. However, you will also notice that the test is still using the constant parameter values and is adding 1 + 2 with a sum of 3 each time.

Next, we'll configure the test to use the values in the data source file.

Step 4 – Use the data in the coded UI test

1. Add `using Microsoft.VisualStudio.TestTools.UITesting.WinControls` to the top of the `CodedUITest.cs` file:

```
using System;  
using System.Collections.Generic;  
using System.Text.RegularExpressions;  
using System.Windows.Input;  
using System.Windows.Forms;  
using System.Drawing;  
using Microsoft.VisualStudio.TestTools.UITesting;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
using Microsoft.VisualStudio.TestTools.UITest.Extension;  
using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;
```

using Microsoft.VisualStudio.TestTools.UnitTesting.WinControls;

2. Add `TestContext.DataRow[]` in the `CodedUITestMethod1()` method which will apply values from the data source. The data source values override the constants assigned to UIMap controls by using the controls `SearchProperties`:

```
public void CodedUITestMethod1()
{
```

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

```
this.UIMap.UICalculatorWindow.UIItemWindow.UIItem1Button.SearchProperties[WinButton.Pr  
opertyNames.Name] = TestContext.DataRow[0].ToString();
```

```
this.UIMap.UICalculatorWindow.UIItemWindow21.UIItem2Button.SearchProperties[WinButton.  
PropertyNames.Name] = TestContext.DataRow[1].ToString();
```

```
    this.UIMap.AddNumbers();
```

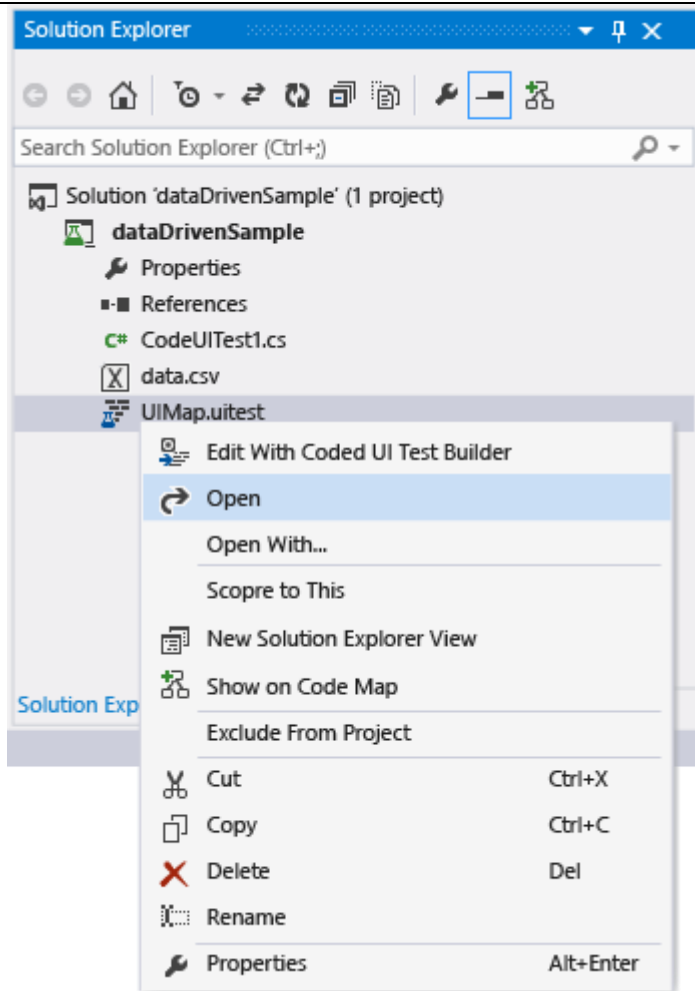
```
    this.UIMap.ValidateSumExpectedValues.UIItem2TextDisplayText  
TestContext.DataRow[2].ToString();
```

```
    this.UIMap.ValidateSum();
```

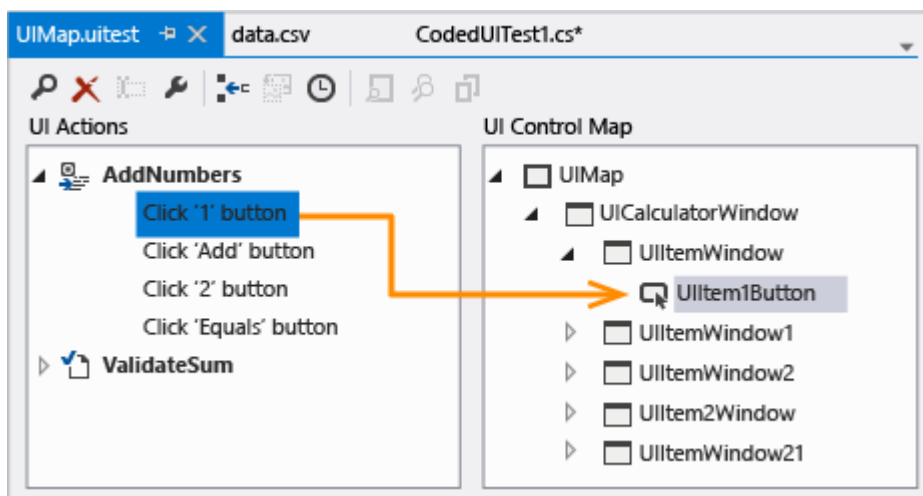
```
}
```

To figure out which search properties to code the data to, use the Coded UI Test Editor.

- Open the UIMap.uitest file.

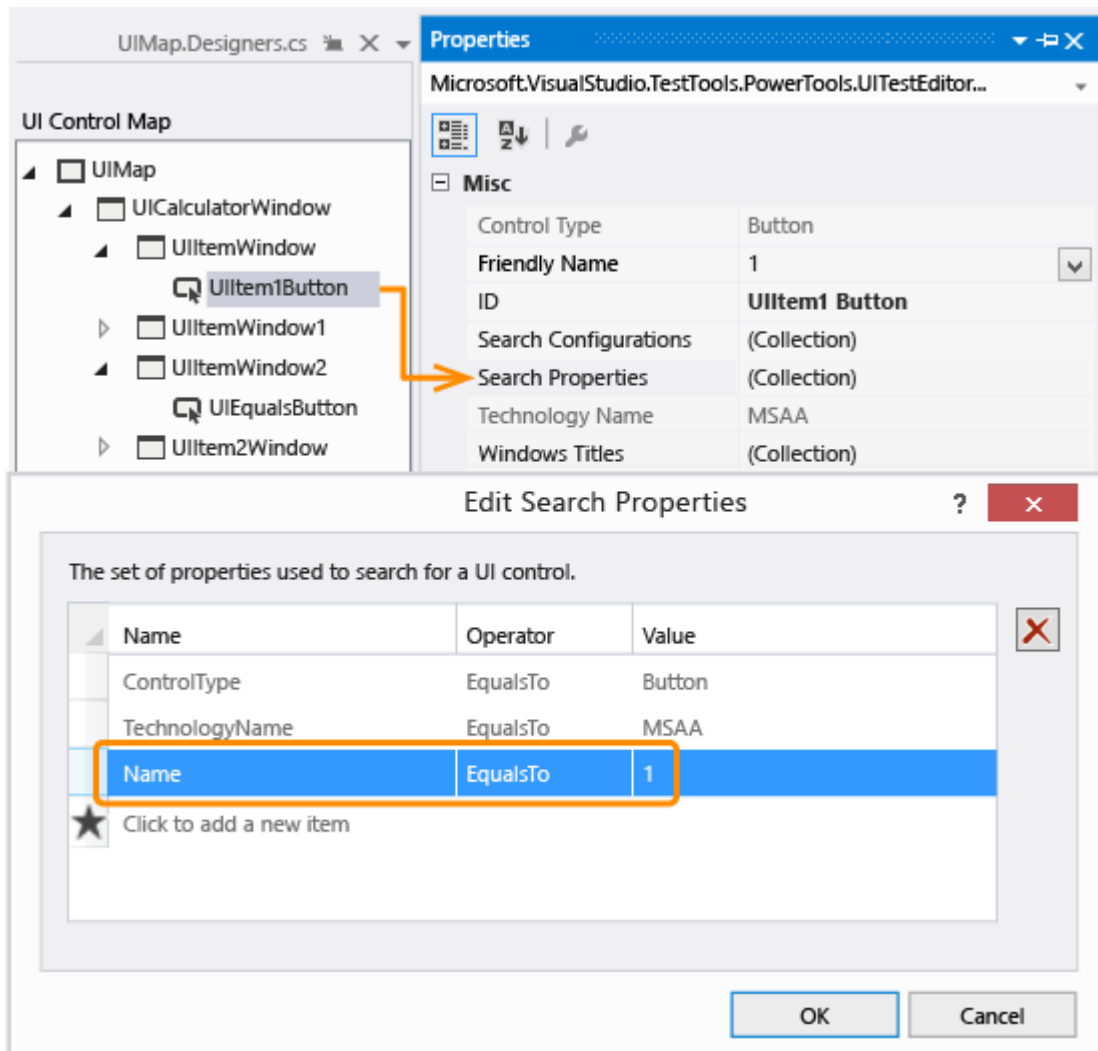


- Choose the UI action and observe the corresponding UI control mapping. Notice how the mapping corresponds to the code, for example, `this.UIMap.UICalculatorWindow.UIItemWindow.UIItem1Button`.



- In the Properties Window, open **Search Properties**. The search properties **Name** value is what is being manipulated in the code using the data source. For example, the `SearchProperties` is being assigned the values in the first column of each data row: `UIItem1Button.SearchProperties[WinButton.PropertyNames.Name]` =

`TestContext.DataRow["Num1"].ToString();`. For the three iterations, this test will change the **Name** value for the search property to 3, then 5, and finally 6.



3. Save the solution.

Step 5 – Run the data-driven test

Verify that the test is now data-driven by running the test again.

You should see the test run through the three iterations using the values in the .csv file. The validation should work as well and the test should display as passed in the Test Explorer.