

GỠ RỐI (DEBUGING)

Chỉnh sửa lại hướng dẫn gỡ rối trong công cụ mà nhóm sử dụng

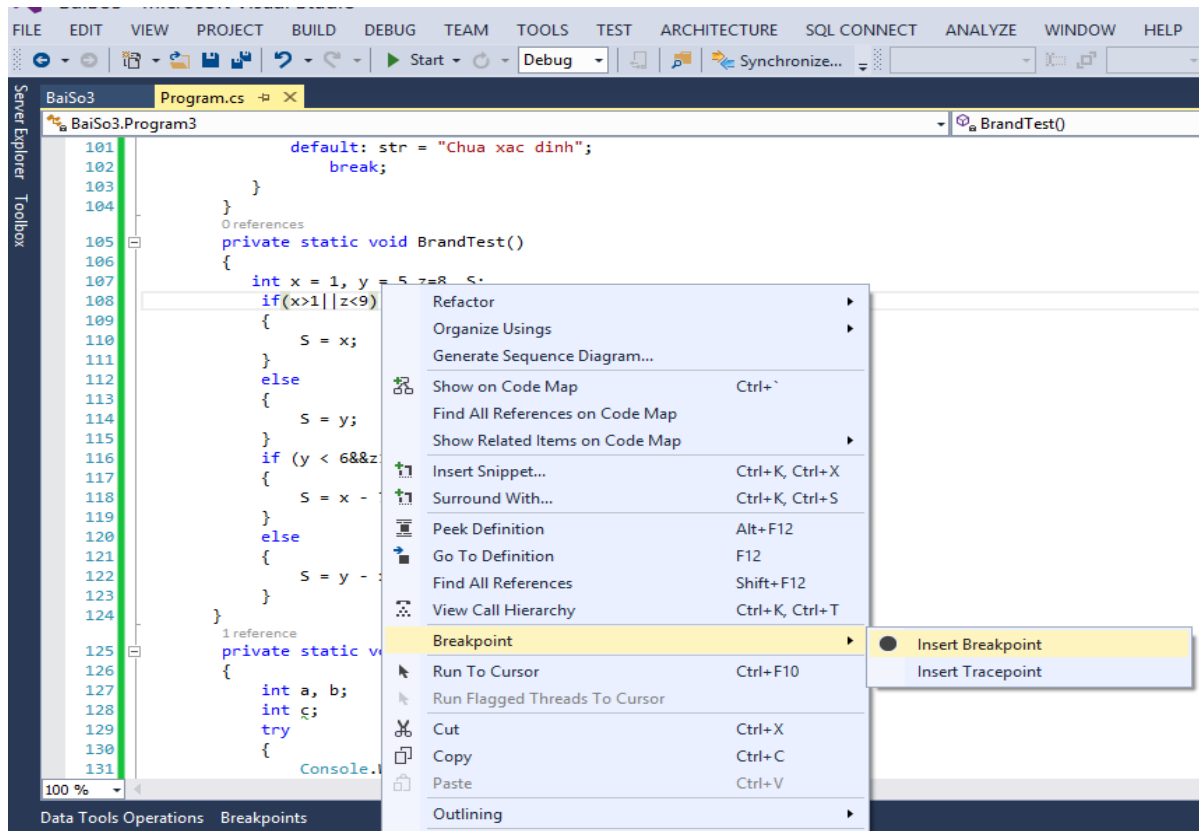
Các chức năng gỡ rối (debuging) trong C-Sharpce

1. Thêm/xóa một điểm ngắt.
2. Quản lí các điểm ngắt.
3. Điểm ngắt điều kiện.
4. Start debug.
5. End debug.
6. Step into.
7. Step Over.
8. Step Out.
9. Continue.
10. Locals.
11. Edit Value.
12. Watch.
13. Memory.
14. Call Stack.

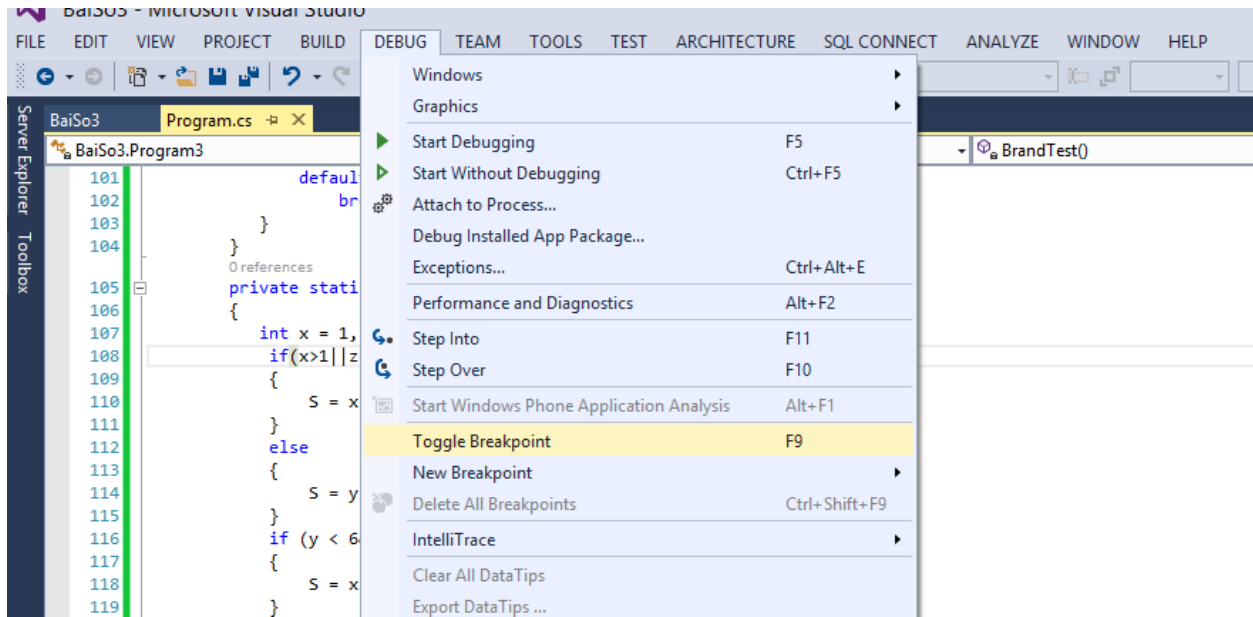
1. Thêm/xóa một điểm ngắt

Có 3 phương pháp để thêm một điểm ngắt như sau:

- Di chuyển chuột đến vị trí bên lề trái, trước các số thứ tự của dòng muốn đặt ngắt, Click chuột trái.
- Di chuyển chuột đến vị trí bất kì của dòng muốn đặt ngắt. Click chuột phải rồi chọn mục "Breakpoint" -> "Insert Breakpoint".

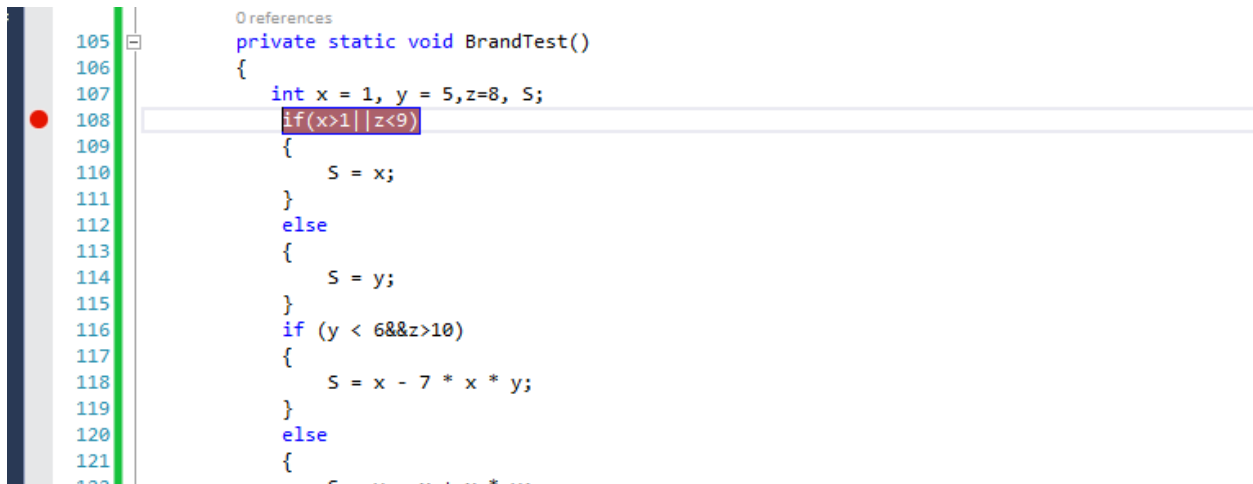


- Di chuyển dấu nháy đến dòng muốn đặt ngắt, rồi bấm vào biểu tượng Debug -> *Toggle Breakpoint (F9)*



Hình 2: Biểu tượng “Toggle Breakpoint” trên thanh công cụ.

Sau khi thêm thành công một điểm ngắt, đầu bên trái của dòng đó sẽ xuất hiện biểu tượng hình tròn (màu đỏ) và dòng đó sẽ sáng lên (màu hồng đậm).

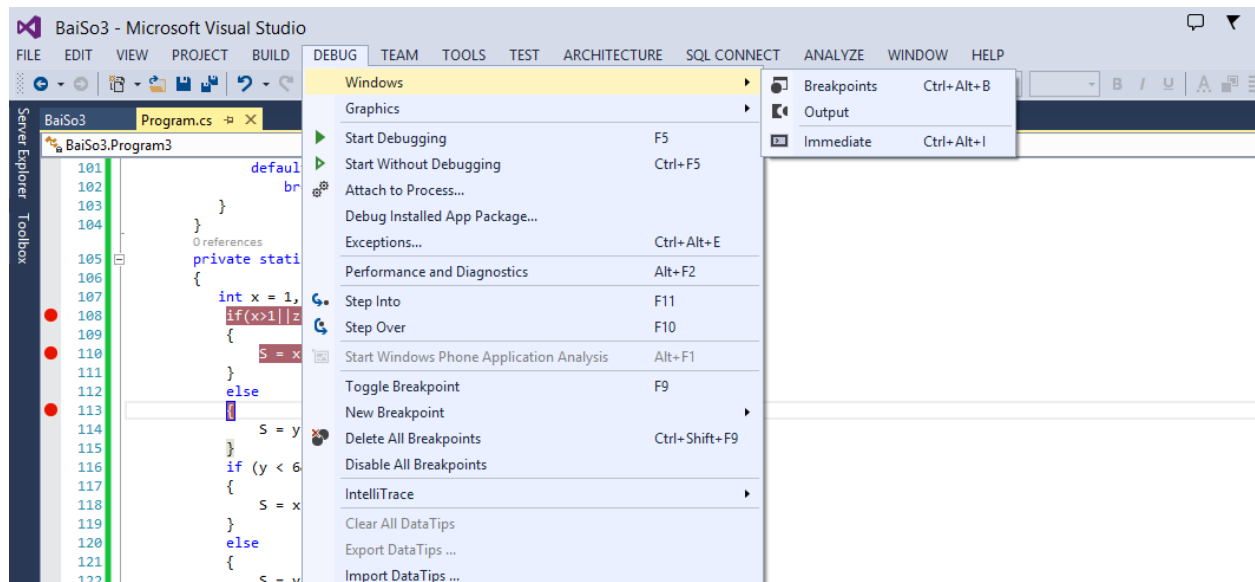


Hình 3: Dòng chứa điểm ngắt sáng lên.

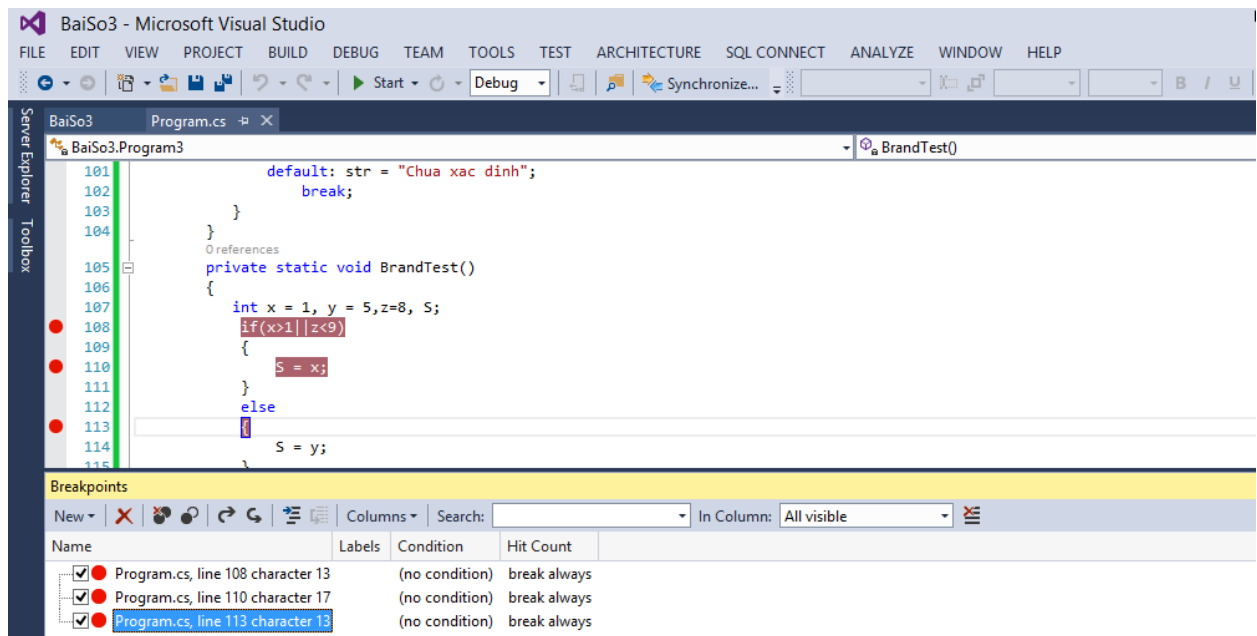
Để gỡ bỏ một điểm dừng thì chỉ cần tiến hành lại các thao tác ở trên tại dòng tương ứng và chọn Delete Breakpoint thay chọn Insert Breakpoint.

2. Quản lí các điểm ngắt

Bạn mở cửa sổ Breakpoints để quản lý các breakpoint đang có bằng cách: chọn *Debug* → *Windows* → *Breakpoints* . Cửa sổ Breakpoints sẽ hiện ra như Hình 4.1



Hình 4: Hộp thoại “Breakpoints”.



Hình 4.1 :Hiện cửa sổ Breakpoint.

- Các thông tin hiển thị trong bảng:

- + Trường File: hiển thị đường dẫn đầy đủ đến file mã nguồn chứa điểm ngắt.
- + Trường Line: hiển thị số thứ tự của dòng chứa điểm ngắt (so với dòng đầu tiên, tính từ 1)
- + Trường Condition: hiển thị điều kiện của điểm ngắt.

- Các thao tác với bảng:

+Delete Breakpoint checked: loại bỏ nút ngắt đang chọn.

+Delete All Breakpoint: loại bỏ toàn bộ nút ngắt.

3. Điểm ngắt điều kiện

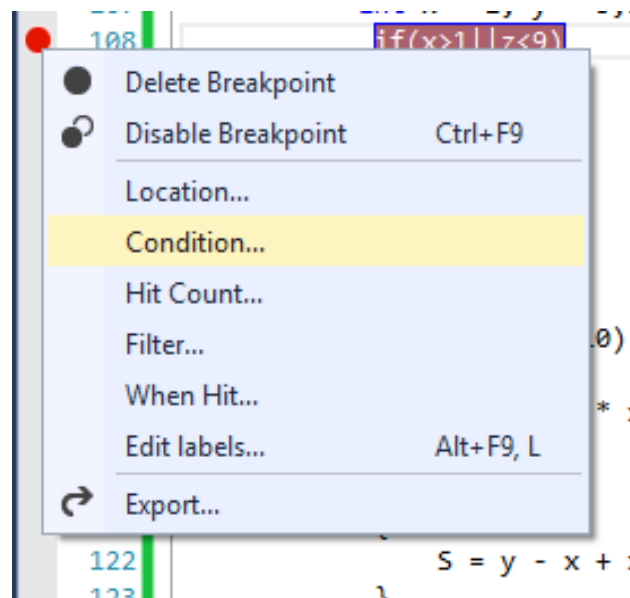
Thông thường ,Chương trình sẽ dừng lại hàng rào có đặt Breakpoint. Tuy nhiên nếu hàng này nằm trong một vòng lặp (một hàm đệ quy) và chúng ta chỉ muốn

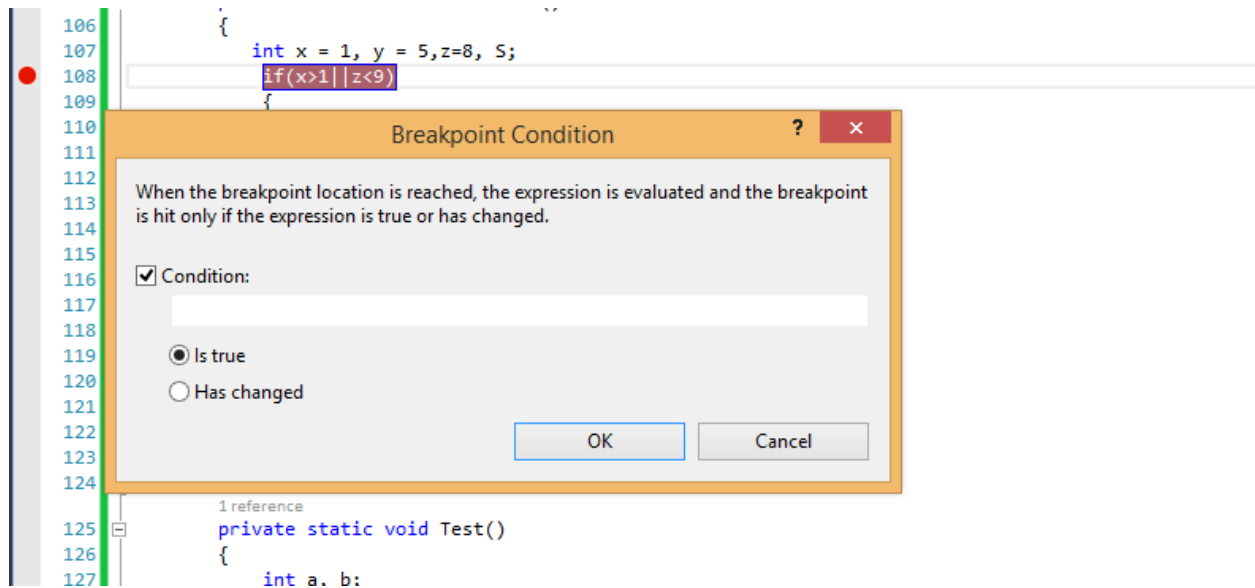
dừng lại trong một vòng lặp(hay một lần gọi đệ quy) cụ thể nào đó thì có hai cách làm điều này.

Cách 1.

Tạo 1 Breakpoint tại hàng rào muốn dừng sau đó cứ nhấn F5 cho đến khi chạy đến vòng lặp (hay lần gọi đệ quy) mong muốn. Cách này mất công rất nhiều. VD ta muốn dừng ở lần gọi đệ quy thứ 99 thì phải nhấn F5 ít nhất 98 lần.

Cách 2: Tạo 1 Breakpoint tại hàng rào muốn dừng sau đó kích phải tạo Breakpoint sau đó click chuột phải vào Breakpoint này chọn Condition(hình 5) .Cửa sổ Breakpoint Condition xuất hiện (hình 5),sau đó bạn nhập điều kiện vào vùng condition



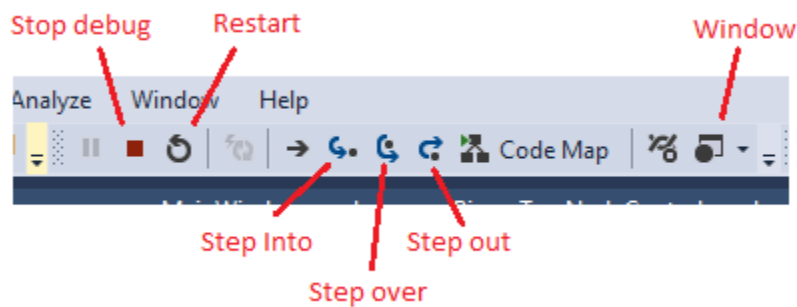


Hình 5: Cách đặt điều kiện cho ngắt.

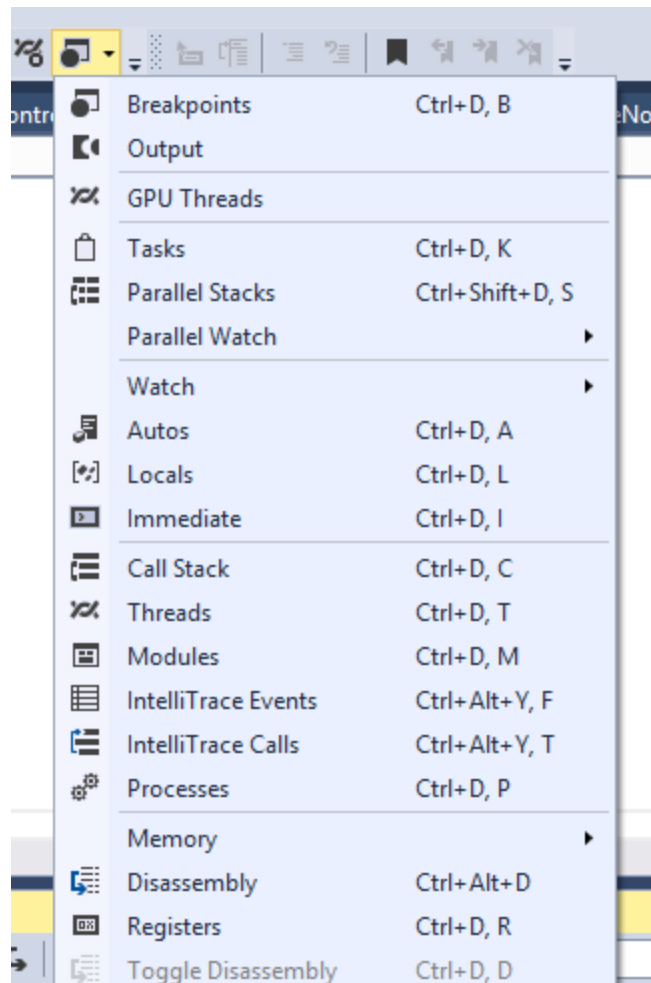
4. Start debug

Click vào "Start debugging" trong menu debug hoặc ấn vào biểu tượng "Start" trên thanh công cụ, hoặc ấn F5. Chương trình bắt đầu được gỡ rối.

Lúc này, thanh công cụ gỡ rối xuất hiện.



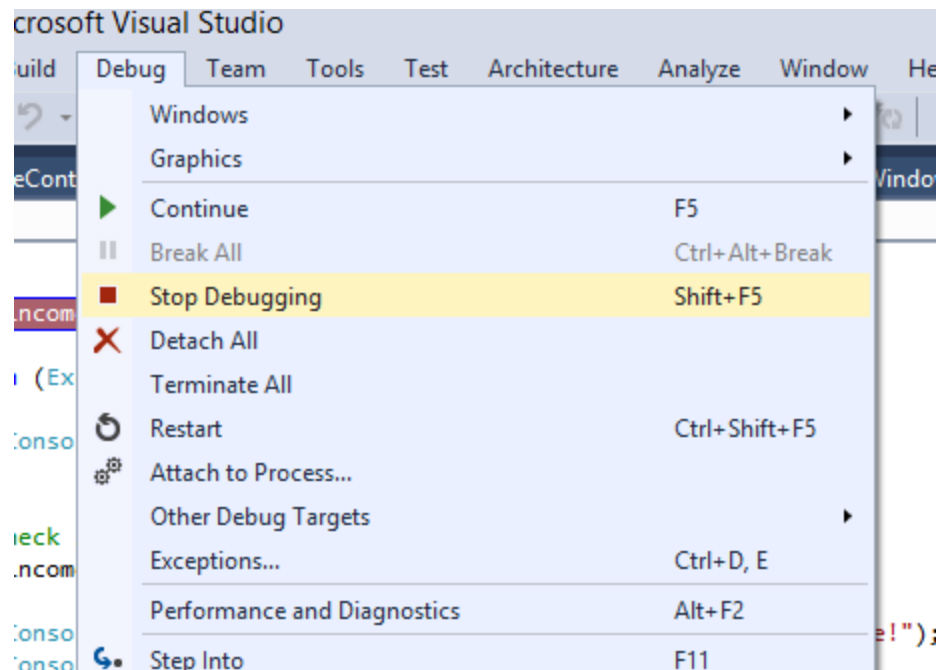
Các lựa chọn trong mục Window:



Hình 6: Thanh công cụ gỡ rối (debuging).

5. Stop debug

Khi chương trình đang gỡ rối, bạn có thể click vào "Stop Debugging" trong menu debug hoặc ấn vào biểu tượng "Stop debugging" trên thanh công cụ, hoặc bấm tổ hợp phím "Shift+F5" để kết thúc quá trình gỡ rối.

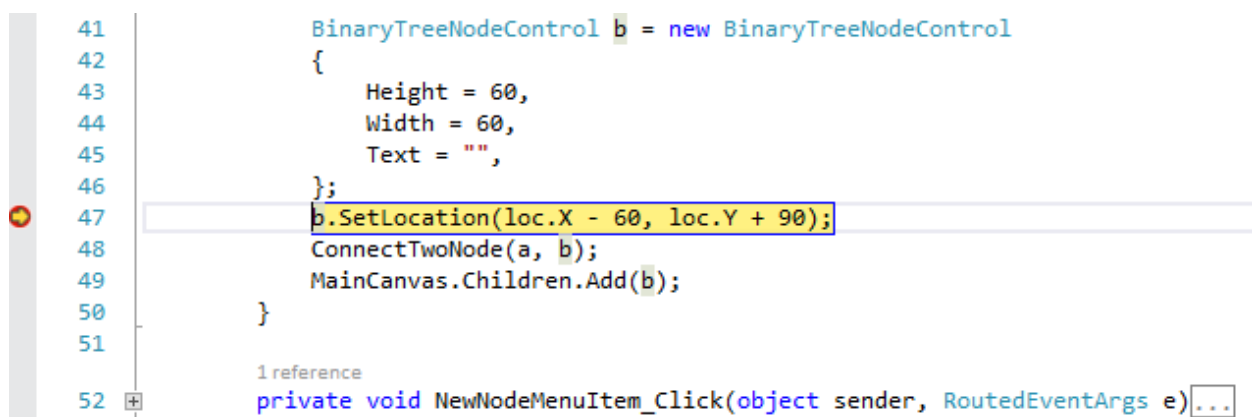


Hình 7: End debugging.

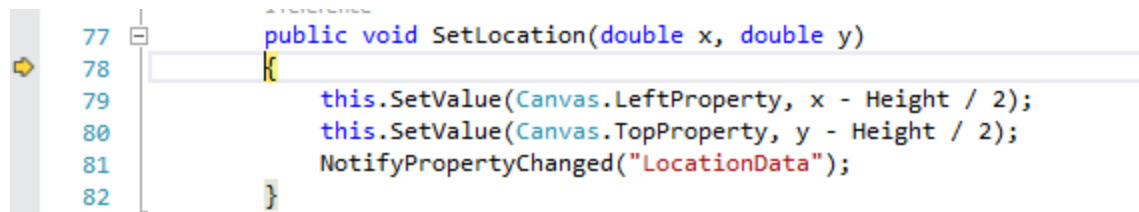
6. Step into

Khi chương trình ngắt tại một lời gọi hàm, ta có thể click vào nút "Step Into" trên thanh công cụ debug để chạy vào trong thân hàm.

Dùng Step into nếu như ta quan tâm đến cấu trúc bên trong thân hàm.



Hình 8: Chương trình bị ngắt tại lời gọi hàm SetLocation.

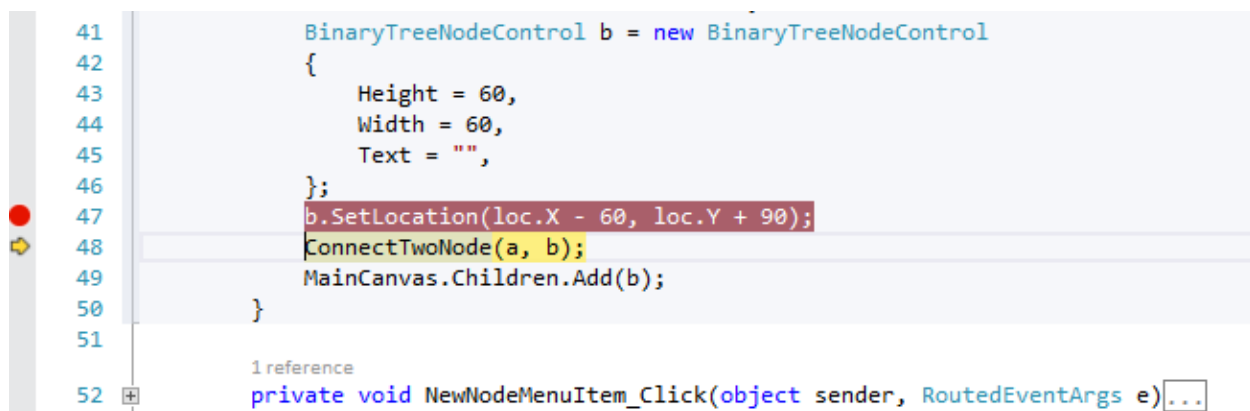


Hình 9: Chạy vào thân hàm sau khi ấn “Step Into”.

7. Step Over

Khi chương trình bị ngắt, bấm vào “Step Over” trên thanh công cụ debug, chương trình chạy đến câu lệnh tiếp theo.

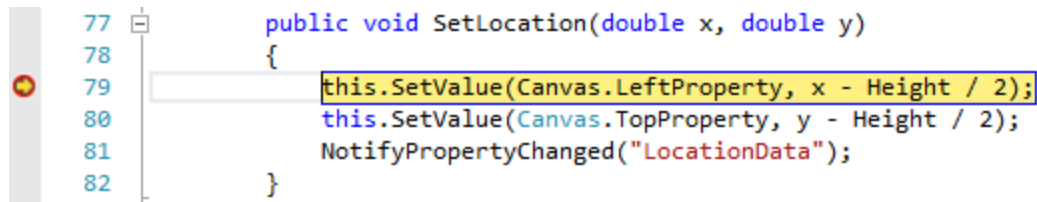
Nếu chương trình bị ngắt tại câu lệnh chứa lời gọi hàm thì chức năng “Step Over” sẽ không chạy vào trong thân hàm như “Step Into” mà sẽ lấy ngay kết quả của hàm rồi chạy đến lệnh tiếp theo.



Hình 10: Chạy đến lệnh tiếp theo khi ấn “Step Over”.

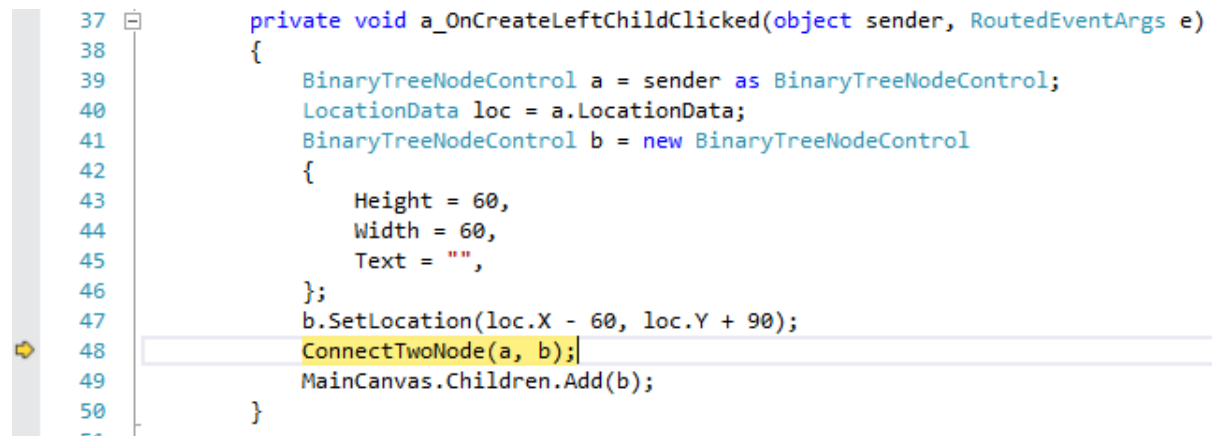
8. Step Out

Khi chương trình bị ngắt trong một thân hàm, bấm vào “Step Out” trên thanh công cụ debug, chương trình sẽ thoát khỏi thân hàm và tiếp tục thực hiện dòng lệnh tiếp theo sau khi trả lại giá trị của hàm.



```
77 public void SetLocation(double x, double y)
78 {
79     this.SetValue(Canvas.LeftProperty, x - Height / 2);
80     this.SetValue(Canvas.TopProperty, y - Height / 2);
81     NotifyPropertyChanged("LocationData");
82 }
```

Hình 11: Chương trình bị ngắt trong thân hàm.



```
37 private void a_OnCreateLeftChildClicked(object sender, RoutedEventArgs e)
38 {
39     BinaryTreeNodeControl a = sender as BinaryTreeNodeControl;
40     LocationData loc = a.LocationData;
41     BinaryTreeNodeControl b = new BinaryTreeNodeControl
42     {
43         Height = 60,
44         Width = 60,
45         Text = "",
46     };
47     b.SetLocation(loc.X - 60, loc.Y + 90);
48     ConnectTwoNode(a, b);
49     MainCanvas.Children.Add(b);
50 }
```

Hình 12: Chương trình thoát khỏi hàm và thực hiện lệnh tiếp theo.

9. Continue

Click vào nút “Continue”, chương trình sẽ chạy đến dòng chứa điểm ngắt tiếp theo.

```

70     Line line = new Line()
71     {
72         Stroke = Brushes.Black,
73     };
74     MultiBinding[] binding = new MultiBinding[4];
75     Binding bind1 = new Binding
76     {
77         Source = a,
78         Path = new PropertyPath("LocationData"),
79     };
80     Binding bind2 = new Binding
81     {
82         Source = b,
83         Path = new PropertyPath("LocationData"),
84     };
85     string[] parameter = { "X1", "Y1", "X2", "Y2" };
86     for (int i = 0; i < 4; i++)
87     {
88         binding[i] = new MultiBinding();
89         binding[i].Bindings.Add(bind1);
90         binding[i].Bindings.Add(bind2);
91         binding[i].Converter = new TwoNodeLocationToLineLocation();
92         binding[i].ConverterParameter = parameter[i];
93     }
94     line.SetBinding(Line.X1Property, binding[0]);
95     line.SetBinding(Line.Y1Property, binding[1]);
96     line.SetBinding(Line.X2Property, binding[2]);
97     line.SetBinding(Line.Y2Property, binding[3]);
98     MainCanvas.Children.Add(line);
99 }

```

Hình 13: Bị ngắt ở dòng 74, sau khi ấn “Continue” chương trình chạy đến ngắt tại dòng 94.

10.Locals

Khi chương trình đang bị ngắt ở trạng thái gỡ lỗi, của sổ Locals sẽ hiển thị các giá trị của tất cả các biến địa phương ở thời điểm hiện tại. Hình ảnh dưới đây cho ta thấy các biến địa phương và trường giá trị tương ứng.

Locals	
Name	Value
this	{BinaryTree.Models.TwoNodeLocationToLineLocation}
values	{object[2]}
targetType	{Name = "Double" FullName = "System.Double"}
parameter	"X1"
culture	{en-US}
loc1	{BinaryTree.Models.LocationData}
loc2	{BinaryTree.Models.LocationData}
z	310.82149217838844

Hình 14: Cửa sổ Locals.

Ta cũng có thể thay đổi giá trị cho các biến địa phương này bằng cách dùng chức năng “Edit Value” được trình bày ở phần 11.

11. Edit Value

Click chuột phải vào một mục trong cửa sổ Locals và chọn “Edit Value”. Ta sẽ được chỉnh sửa giá trị của biến đó

loc1	{BinaryTree.Models.LocationData}
loc2	{BinaryTree.Models.LocationData}
z	333.325342

Hình 15: Chỉnh sửa giá trị tại cửa sổ Locals.

Nhập giá trị mới cho biến rồi ấn Enter, chương trình sẽ chạy với giá trị biến mới gán.

12. Watch

Chọn Window/Watch/Watch 1(2,3,4) trên thanh công cụ Debug, một cửa sổ Watch 1(2,3,4) mở ra. Ta có thể theo dõi một biểu thức bằng cách chọn hàng cuối cùng không chứa giá trị nào trong cửa sổ Watch rồi nhập biểu thức, rồi gõ Enter

Watch 1	
Name	Value
loc1	{BinaryTree.Models.LocationData}
Math.Sqrt(z)	

Watch 1	
Name	Value
loc1	{BinaryTree.Models.LocationData}
Math.Sqrt(z)	17.630130237136321

Hình 16: Thêm biểu thức vào cửa sổ Watch 1 và sau khi thêm

Có thể click chuột vào một mục trong cửa sổ locals để hiển thị hộp thoại trên.

13.Memory

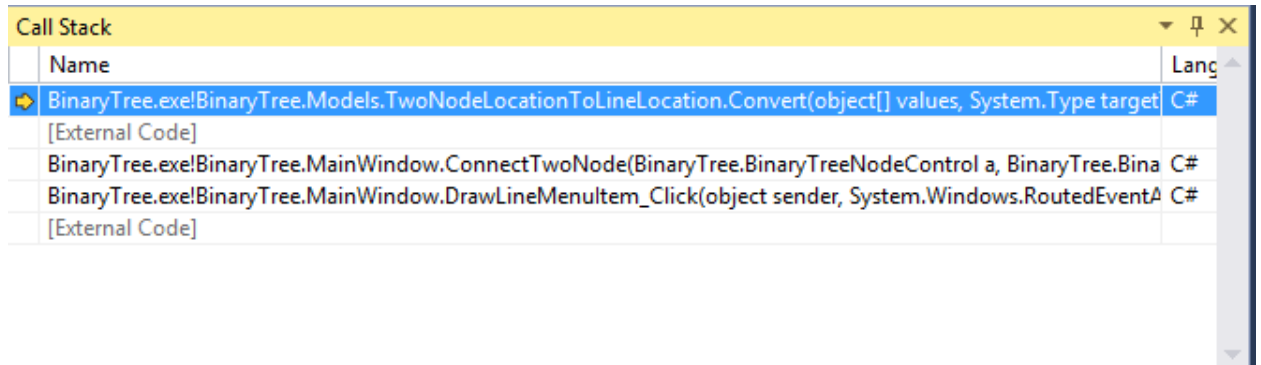
Chọn “Window/Memory/Memory 1(2,3,4)” trên thanh công cụ debug, một cửa sổ hiện ra, nhập &a (hoặc &i) và ấn enter, ta có thể xem nội dung của biến trong bối cảnh bộ nhớ. (ở đây, sau khi ấn enter, dòng “Address” sẽ chuyển &a (hoặc &i) thành địa chỉ thực trong bộ nhớ)

Memory 1	
Address: 0x0550E458	Columns: Auto
0x0550E458	e1 5f e0 91 67 14 73 40 54 df 10 02 04 db 10 02 18 cd 10 á_à'g.s@TB...Ô...í.
0x0550E46B	02 70 c6 10 02 00 00 00 00 00 00 00 00 00 00 00 00 .pÆ.....
0x0550E47E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0550E491	00 00 00 00 00 00 00 00 00 00 00 f4 81 fc 02 f0 c5 10 02ô.ü.ôÄ..
0x0550E4A4	70 c6 10 02 18 e5 50 05 9e 9f de 5f 7c 5e fc 01 1c c2 10 pÆ...âP.žŸp_ ^ü..Â.
0x0550E4B7	02 e0 1d ff 01 dc e4 50 05 e0 1d ff 01 18 cd 10 02 7c 5e .à.ÿ.ÜäP.à.ÿ..í.. ^
0x0550E4CA	fc 01 18 cd 10 02 6c c2 10 02 a4 cc 10 02 00 00 00 00 ü..í...lÄ..đİ.....
0x0550E4DD	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0550E4F0	02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0550E503	00 00 00 00 00 00 e8 39 ff 01 6c c2 10 02 a4 cc 10 02 40 e5è9ÿ.lÄ..đİ...@ă

Hình 17: Cửa sổ “memory”.

14.Call Stack

Chọn “Window/Call Stack” thanh công cụ debug, cửa sổ “Call Stack” hiện ra như hình dưới. Cửa sổ cho ta một cái nhìn trực quan hoạt động của stack khi chạy chương trình.



Hình 18: Cửa sổ “Call Stack”.

TÀI LIỆU THAM KHẢO

[1] *Code Complete*, The second Editor, Steve McConnell.