



# Debugging



# Debugging

Debugging is the process of **detecting and removing** of existing and potential errors (also called as '**bugs**') in a software code that can cause it to behave unexpectedly or crash



# Debugging: Semantic Errors

It does not mean syntax errors, with which the application cannot be compiled, but it means **logical or semantic** errors.



# Debugging: Working Example

Imagine You are a developer in a **Space Station**. Your job is to debug the code and fix any errors. Remember, the lives of the crew rest squarely upon your shoulders. Your **directions** are as follows:

- **Launch the shuttle** only if the fuel, crew and computer all check out **OK**.
- If a check fails, **print that information to the console** and scrub the launch (then scrub the deck).
- If all checks be successful, print a **countdown to the console**, then below **"Liftoff!"**



# Debugging: Working Example

Luckily, **Before Launching** the Shuttle, you have run your code in a simulation and your **Shuttle has crashed**.

Now, it's your responsibility to **find the error** in the code

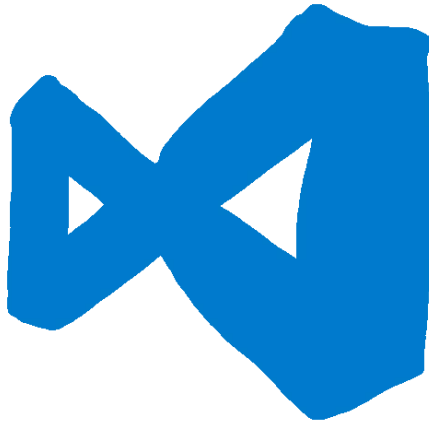


```
static bool shuttleLaunch(int fuelLevel, string computerStatus, bool crewStatus)
{
    bool launchReady = false;
    if (fuelLevel >= 20000)
    {
        Console.WriteLine("Fuel level cleared.");
        launchReady = true;
    }
    else
    {
        Console.WriteLine("WARNING: Insufficient fuel!");
        launchReady = false;
    }
    if (crewStatus && computerStatus == "green")
    {
        Console.WriteLine("Crew & computer cleared.");
        launchReady = true;
    }
    else
    {
        Console.WriteLine("WARNING: Crew or computer not ready!");
        launchReady = false;
    }
    return launchReady;
}
```

```
static void Main(string[] args)
{
    bool launchReady = shuttleLaunch(17000, "green", true);
    Console.WriteLine(launchReady);
    Console.ReadKey();
}
```

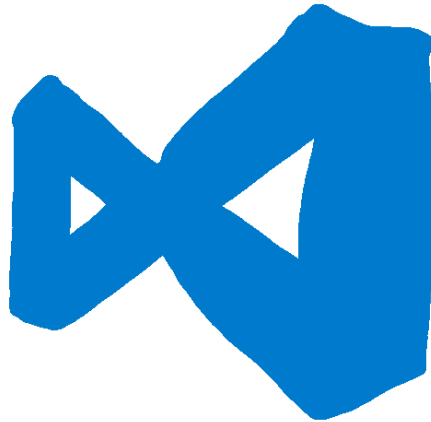
# Debugging: Working Example

Visual Studio gives us a **built-in debugger**, thanks to which we can place breakpoints at places we have chosen and stop the execution of the code to see the values in the variables.



# Debugging: Working Example

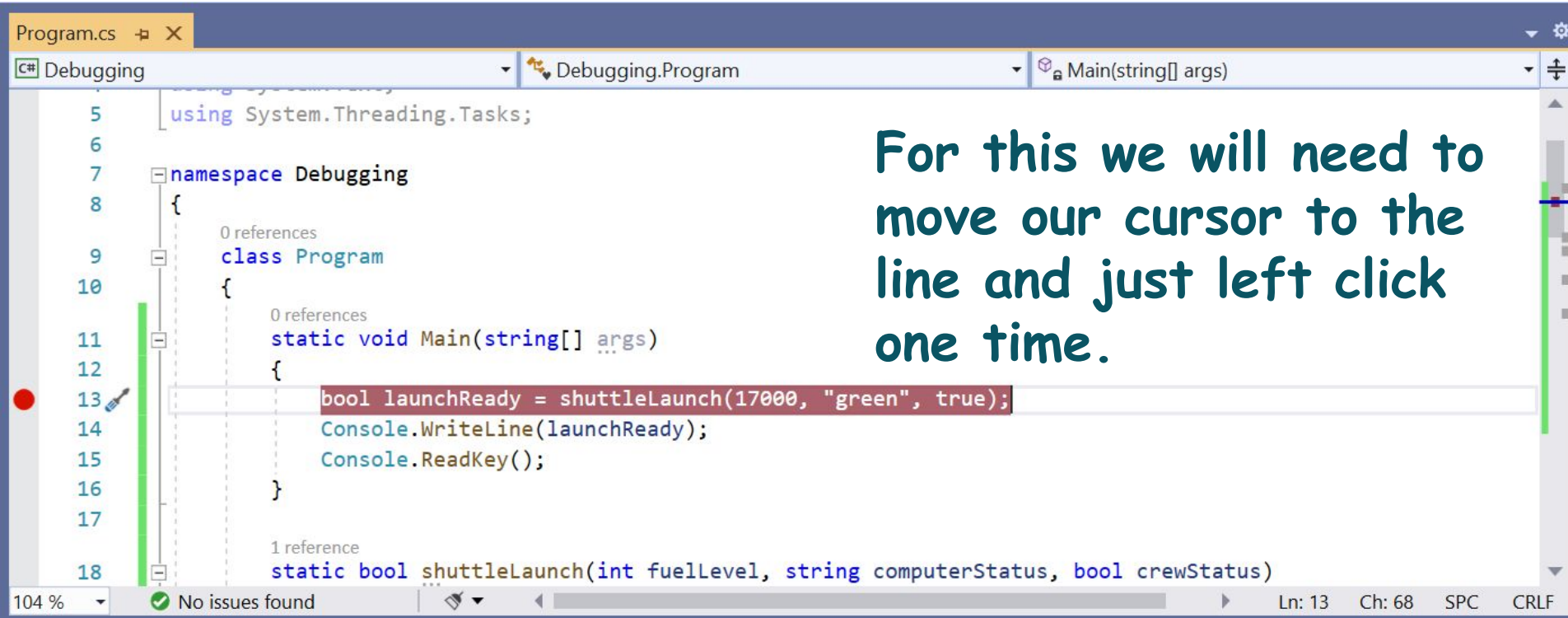
When it reaches a **breakpoint**, the program stops running and allows **step-by-step** running of the remaining lines.





# Debugging: Breakpoint

For this we will need to move our cursor to the line and just left click one time.



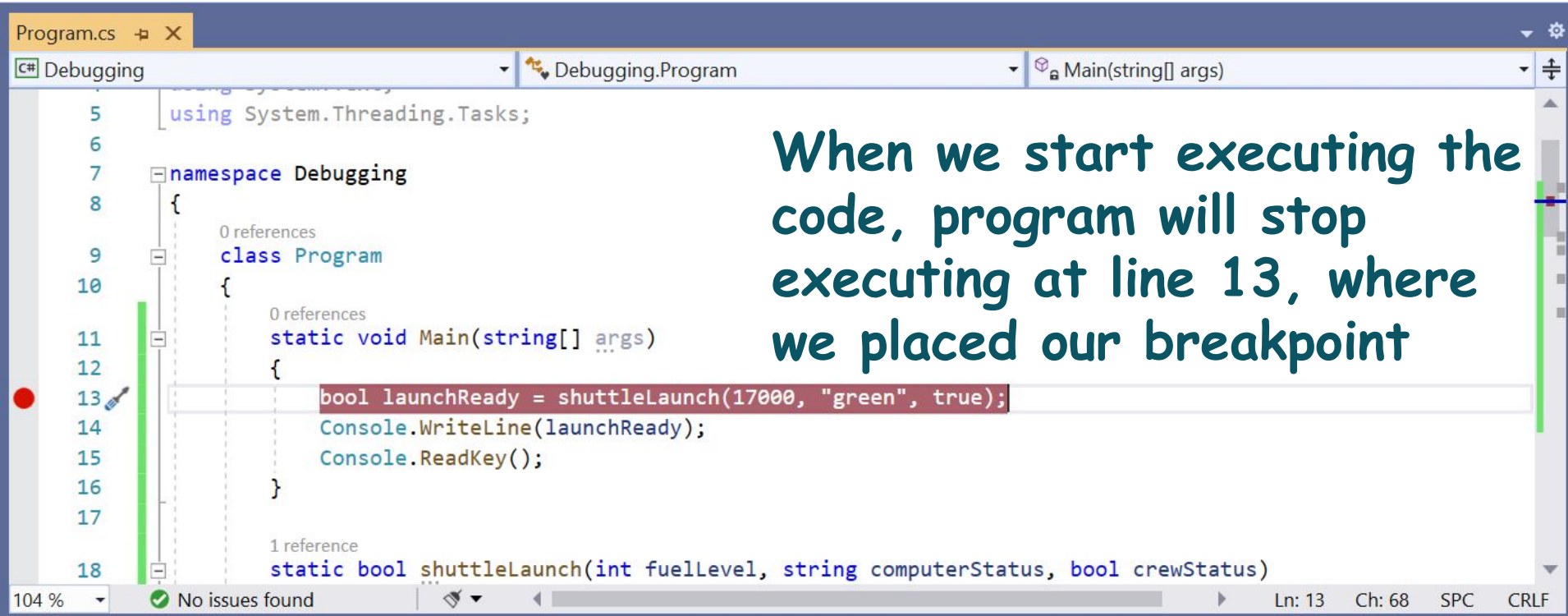
The screenshot shows the Visual Studio IDE with a C# file named Program.cs. The code is as follows:

```
5 using System.Threading.Tasks;
6
7 namespace Debugging
8 {
9     0 references
10     class Program
11     {
12         0 references
13         static void Main(string[] args)
14         {
15             bool launchReady = shuttleLaunch(17000, "green", true);
16             Console.WriteLine(launchReady);
17             Console.ReadKey();
18         }
19
20         1 reference
21         static bool shuttleLaunch(int fuelLevel, string computerStatus, bool crewStatus)
```

A breakpoint (red dot) is set on line 13, at the start of the line `bool launchReady = shuttleLaunch(17000, "green", true);`. The status bar at the bottom indicates "104 %", "No issues found", and "Ln: 13 Ch: 68 SPC CRLF".

# Debugging: Breakpoint

When we start executing the code, program will stop executing at line 13, where we placed our breakpoint



The screenshot shows the Visual Studio IDE with a C# file named Program.cs. The code is as follows:

```
5 using System.Threading.Tasks;
6
7 namespace Debugging
8 {
9     0 references
10     class Program
11     {
12         0 references
13         static void Main(string[] args)
14         {
15             bool launchReady = shuttleLaunch(17000, "green", true);
16             Console.WriteLine(launchReady);
17             Console.ReadKey();
18         }
19
20         1 reference
21         static bool shuttleLaunch(int fuelLevel, string computerStatus, bool crewStatus)
```

A red dot breakpoint is set on line 13. The status bar at the bottom indicates "104 %", "No issues found", and "Ln: 13 Ch: 68 SPC CRLF".

# Debugging: Navigating the Code

We have three options when navigating through our code: **Step Into**, **Step Over**, and **Step Out**.



# Debugging: Step Into

**Step Into** causes the debugger to go inside the method call on the current line and pause the execution there.



# Debugging: Step Over

**Step Over** advances the debugger to the next line without stepping into any method call.



# Debugging: Step Out

**Step Out** causes the debugger to continue executing the current function and pause the execution when it returns.



# Debugging: Find the value of a variable

The simplest way to find the value of a variable at any particular time of execution is to just hover over them.



```
17  if (fuelLevel >= 20000)
18  {
19      Console.WriteLine("Fuel level cleared.");
20      launchReady = true;
21  }
22  else
23  {
24      Console.WriteLine("WARNING: Insufficient fuel!");
25      launchReady = false;
```

fuelLevel 17000

# Debugging: Working Example

The value of `launchReady` assigned in the first if/else block got changed in the second if/else block.

Since the issue is with `launchReady`, ONE way to fix the logical error is to use a different variable to store the fuel check result. Update your code to do this.



# Debugging: Working Example

Add a final if/else block to print a countdown and “Liftoff!” if all the checks pass, or print “Launch scrubbed” if any check fails.

# Conclusion

- Debugging is the process of detecting and removing of existing and potential errors (also called as 'bugs') in a software code that can cause it to behave unexpectedly or crash
- A **breakpoint** is a point in the program where the code will stop executing.
- When it reaches a breakpoint, the program stops running and allows **step-by-step** running of the remaining lines.
- We have three options when navigating through our code: **Step Into**, **Step Over**, and **Step Out**.



# Learning Objective

Debug the Programs to find  
**Semantic errors** in the Code.

