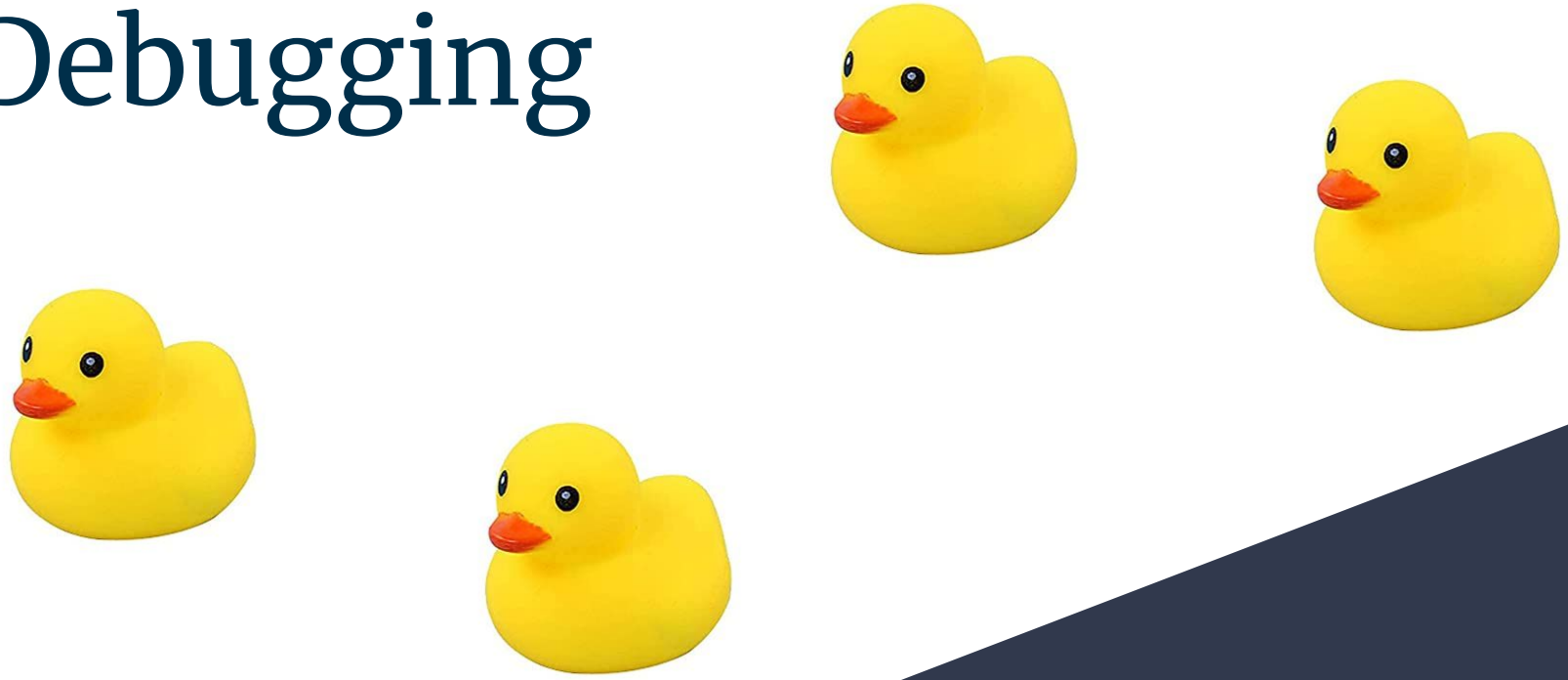
Debugging



Bugs

Errors in your program as it runs

Sometimes called *logic errors*

- Although these aren't the only bugs

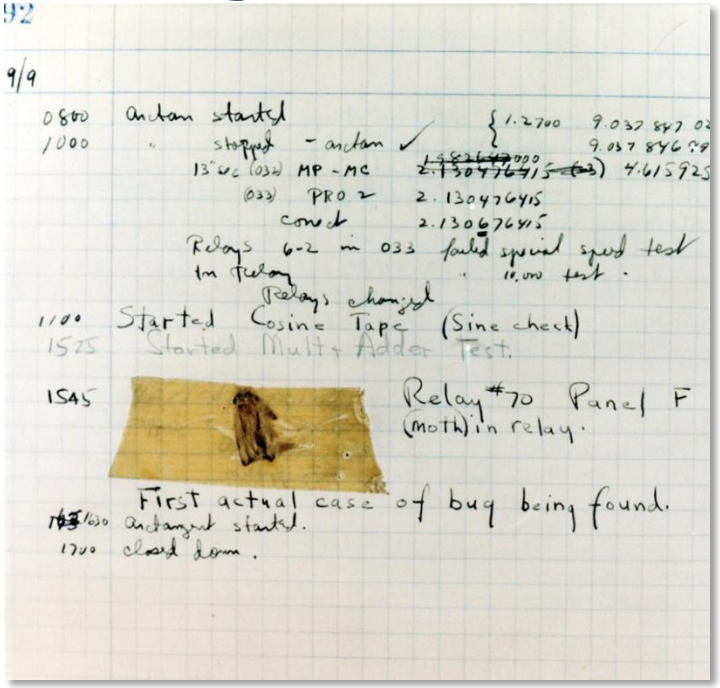
Not to be confused with *syntax errors*

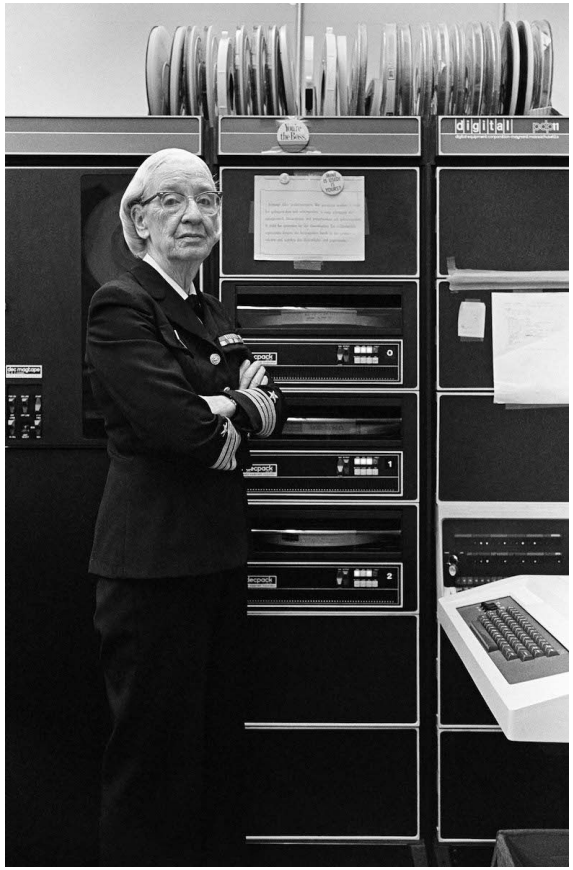
- Problems which prevent compilation

Why are errors called bugs?

The very first computer bug

An actual moth found in the Harvard Mark II





Made famous by computer-language pioneer Admiral Grace Hopper.

Bugs in your own code

You're going to have bugs at some point

Sometimes they're simple:

- Forgetting to alter a variable somewhere
- Using the wrong variable
- Copy/pasting incorrectly
- Not realizing a number could go negative
- Forgetting to sanitize user input
- Off by one in a loop (e.g. < vs <=)

Complex bugs

Sometimes the bugs are more complex

- Due to multiple small errors adding up

Especially as we start using more of C#

- If statements
- Loops
- Methods
- Properties
- Etc.

Finding bugs

Some bugs may crash your program

- Usually easy to find
- Can see the line number
- Note: That line might not be the root cause!

Some bugs are hard to track down

- Reason might not be obvious
- Or they might happen very rarely

The nastiest bugs are often the ones that DON'T crash the program!

Debugging Strategies

Where to Start

When our software doesn't behave as we expect it to, we ask a series of questions to narrow down the source of the problem:

1. What should have happened?
2. What sequence of events should have led to the desired result?
3. What actually happened?

Various resources and techniques help answer these questions

1. **What should have happened?**
   Requirements (e.g. PE and HW write-ups)
2. **What sequence of events should have led to the desired result?**
   Design, coding todo's pseudo-code, etc.
3. **What actually happened?**
   Test results
   Output logs (e.g. WriteLines of interim data to track progress)
   Manual execution trace
   Use the debugger!!!

Most IDEs include tools to help debug a project

Capture information about a crash

Allow you to step through the code, line-by-line and see the state of the program

- I.e. an programmatic execution trace!

Watch the values of certain variables

Pause execution at specific points

These are powerful tools worth learning now.

You'll need them for the rest of your careers.

Bug hunting in Visual Studio

Bug hunting with Visual Studio

Visual Studio is more than a fancy text editor

Running a program with “F5” is *debugging*

- Not Ctrl+F5

While debugging:

- Your program is connected to Visual Studio
- VS monitors for crashes (run-time errors)
- When one is found, it visually shows you the error

Debugging example

This program was run with just “F5”

```
9   class Program
10  {
11      static void Main(string[] args)
12      {
13          int number = 100;
14          int probablyAnError = number / 0;
15
16
17
18
```

DivideByZeroException was unhandled

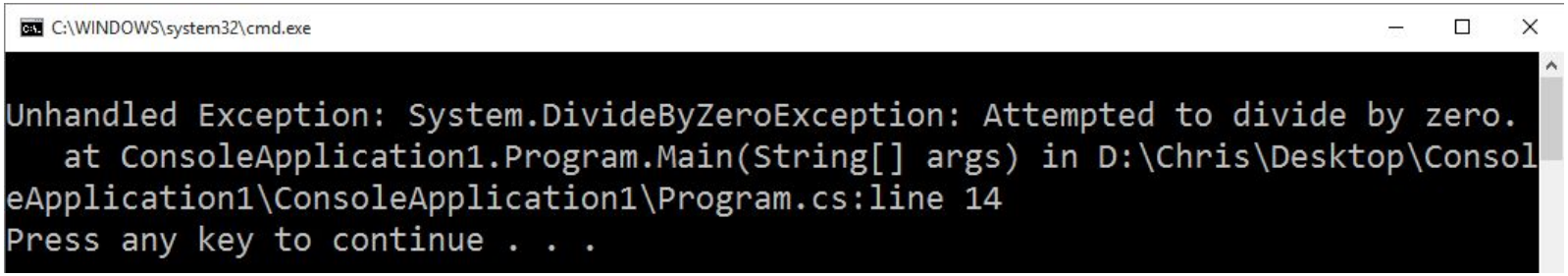
An unhandled exception of type 'System.DivideByZeroException' occurred in ConsoleApplication1.exe

Additional information: Attempted to divide by zero.

What about Ctrl+F5?

Ctrl+F5 is “Run without Debugging”

- Your program is not connected to Visual Studio
- So it can’t detect the crash



Notice much of the same info as before

- Although a bit harder to read

Finding crashes is easy

Finding or fixing the root cause of the crash is often harder

```
// Get a name from the user
String name = Console.ReadLine();

// Determine how many copies of that
// name will fit in 100 characters
int howManyFit = 100 / name.Length;
```

This line could potentially crash

Fixing it requires changing something other than this line

What about finding logic errors?

You could use code to find errors

Put in extra Console.WriteLine( ) statements

- Print out variables before you alter them
- And again afterwards
- See if the changes are what you expect

This is a quick and useful trick

But this can lead to WriteLine overload

Logic errors – A better solution

A quick print statement can *sometimes* work

But littering your code with print statements quickly becomes a chore

Instead, we can use Visual Studio itself to help figure out exactly what our code is doing

Breakpoints

Visual Studio allows a programmer to set specific "stopping points" in their code

These are called *breakpoints*

- Not to be confused with a "`break;`" statement!

While debugging, if a breakpoint is hit:

- The execution of the program pauses
- Visual Studio pops into focus
- You now have line-by-line control

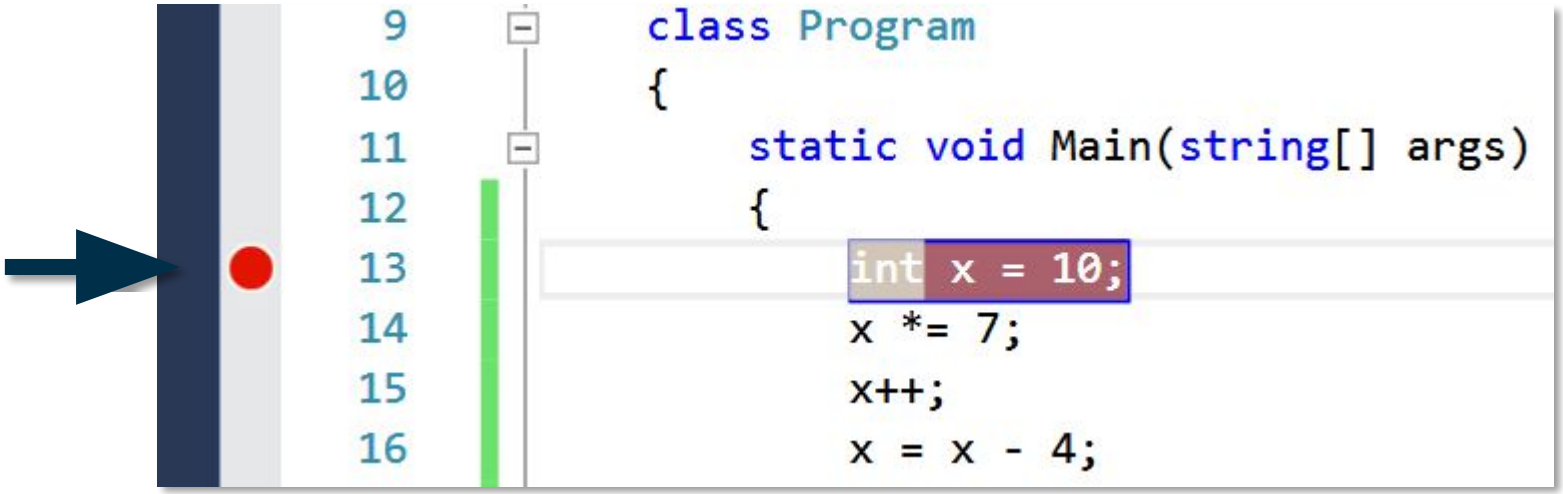
Setting a breakpoint

A breakpoint can be set on any statement

- But not whitespace, comments, etc.

To set one, click the grey column to the left

- The whole statement will turn red



Setting & unsetting breakpoints

You can continue clicking to set multiple

- Or click on the red dot to unset a breakpoint

```
9   class Program
10  {
11      static void Main(string[] args)
12      {
13          int x = 10;
14          x *= 7;
15          x++;
16          x = x - 4;
```

Using breakpoints

When you run with “F5”:

- Your program will execute as normal
- Until a breakpoint is encountered

The program will then pause *before* the line

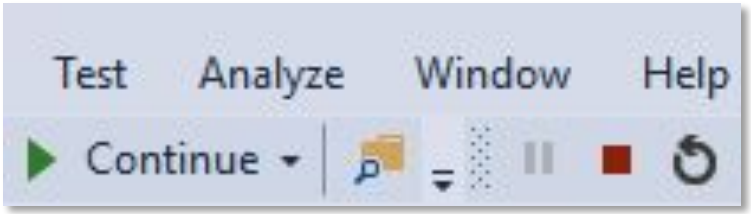
- The line highlights yellow – it has not run yet
- The dot will have a yellow arrow on it

```
9   class Program
10  {
11      static void Main(string[] args)
12      {
13          int x = 10;
14          x *= 7;
```

We’ve hit a breakpoint – Now what?

Our program is paused

- The VS menu bar is different while debugging
- Here are a few of the options:



Keep running code until
the next breakpoint.
Pressing F5 will do this too.

Immediately
end the program

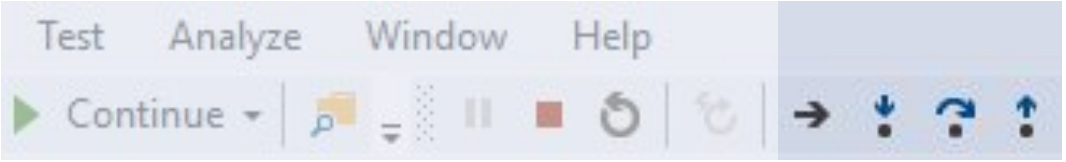
Restart the program

Like hitting stop and
start again

Stepping through

You can also step through your code exactly one line at a time

- Rather than skipping from breakpoint to breakpoint



Step Out – Finishes the current method and pauses at the next line encountered

More on this later

Step Into – Pauses at the first line inside the next method called on this line

More on this later

Step Over (F10) – Executes exactly one line in the current file and pauses at the next one

Step over – F10

Pressing F10 while paused in the debugger will run the current line and pause again

- If the next line is inside an *if block*, we pause there
- If the *if block* is skipped, stepping skips it too

We can “see” the path our code takes

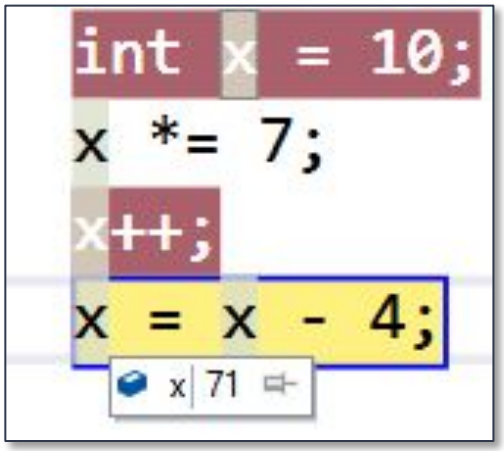
- This is hugely helpful when debugging

What else can we do while paused?

Hover the mouse over a variable to see the *current* value

- Not the value it had at that line

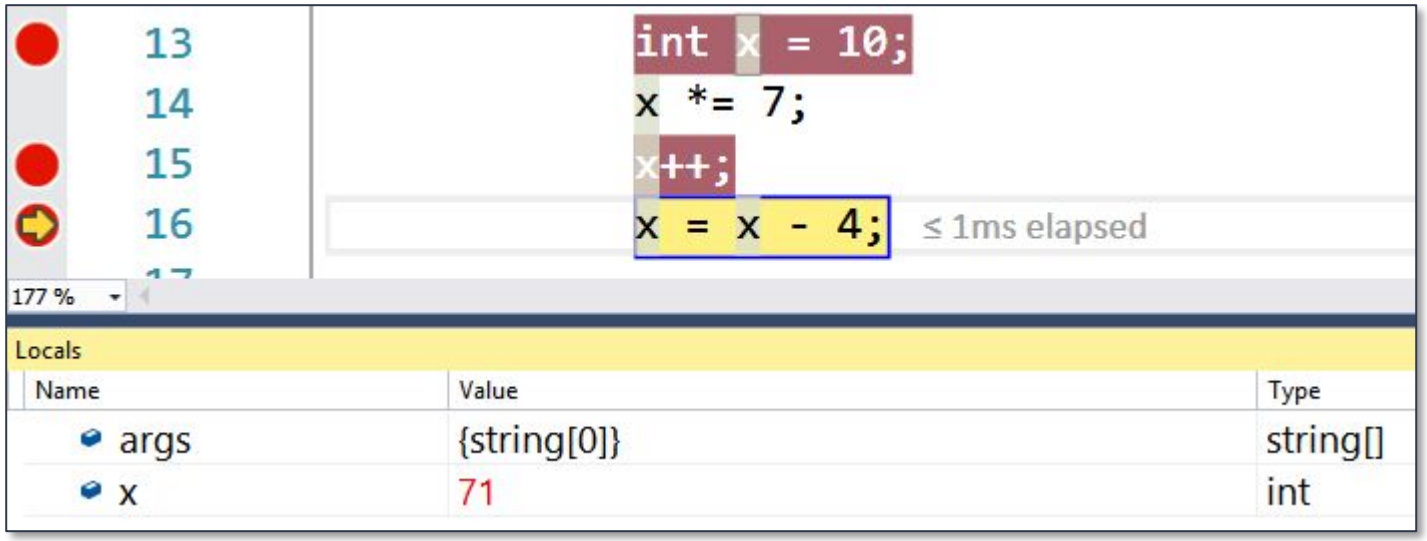
You can even pin the little pop-up so it stays open



Locals window

Shows all local variables in the current scope

- As well as their values and their types
- Values highlight red when they change



Opening the Locals window

Some windows only open during debugging

- Like the locals window
- It doesn’t have a use at any other time

You can open it yourself if it’s closed

Debug > Windows > Locals

- “Windows” is the top option under “Debug”
- “Locals” is about halfway down the “Windows” menu

Breakpoints & Ctrl+F5

Ctrl+F5 means “Run without Debugging”

- Visual Studio isn’t connected to your program
- So it can’t pause at breakpoints
- But it does auto-add a “Press to continue” at the end of your program

Sometimes this is useful

- And sometimes not

Just keep it in mind when debugging

In-class exercise

I'm going to share a very broken program with you in Slack...

With a partner, debug the 3 sub-programs in Main().

- ONE person should "drive" (run, take notes in comments, etc)
- The other "navigates" (make suggestions, look up info needed, etc.).
- *Whoever is the more experienced/confident programmer should be the navigator!*

<u>Together, for each section:</u>

1. Talk through the code together.
2. Add comments explaining the parts you understand *(this code is intentionally UNDER commented)*
3. Add comments with questions for parts you're unsure about
4. Make a hypothesis about where the problem is
5. Use the debugger to test your hypothesis *(do NOT just hack at the code blindly until it works!)*
6. Take notes about what you look at in the debugger and how that helped narrow down the problem *(also in comments)*