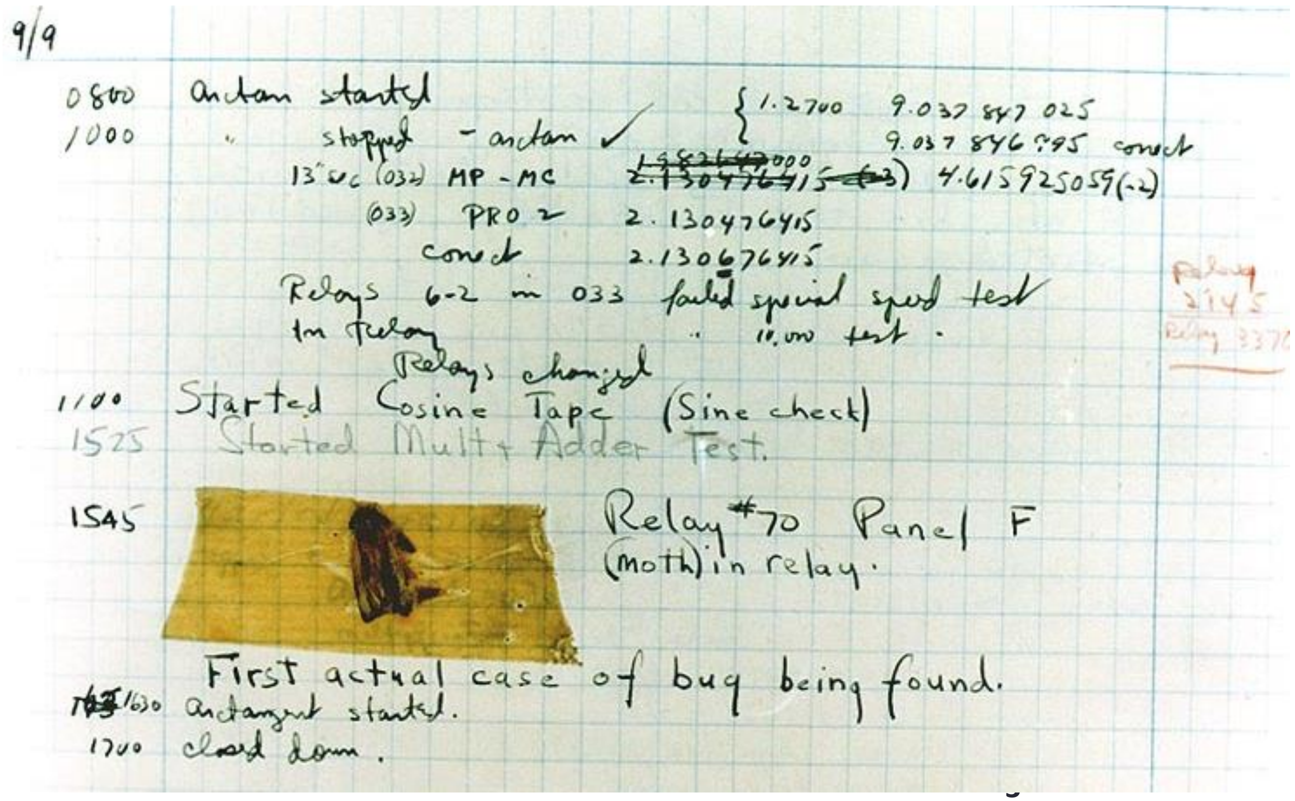
TESTING AND DEBUGGING

Outline

- Debugging
  - Types of Errors
    - Syntax Errors
    - Semantic Errors
    - Logic Errors
- Preventing Bugs
  - Have a plan before coding, use good style
  - Learn to trace execution
    - On paper, with print statements, using the debugger
  - Explain it to a teddy bear
  - Incremental development

Debugging

- Majority of program development time:
  - Finding and fixing mistakes! a.k.a. bugs
  - It's not just you: bugs happen to all programmers



9/9

0800 Anctan started
1000 " stopped - anctan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In Relay " 11,000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Anctangent started.
1700 closed down.

Relay 2145
Relay 3370

Debugging

- Computers can help find bugs
  - But: computer can't automatically find all bugs!
- Computers do exactly what you ask
  - Not necessarily what you want
- There is always a logical explanation!
  - Make sure you saved & compiled last change

"As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."
-Maurice Wilkes

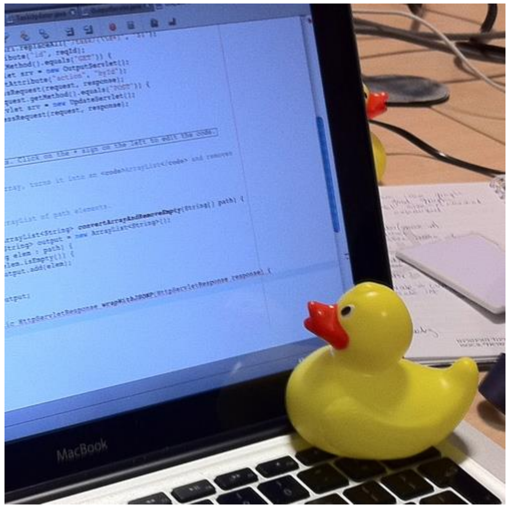
"There has never been an unexpectedly short debugging period in the history of computers."
-Steven Levy

Preventing Bugs

- Have a plan
  - Write out steps in English before you code
  - Write comments first particularly before tricky bits
- Use good coding style
  - Good variable names
    - "Name variables as if your first born child"
    - If variable is called `area` it should hold an area!
  - Split complicated stuff into manageable steps
  - ()'s are free, force order of operations you want
- Carefully consider loop bounds
- Listen to Eclipse (IDE) feedback

Finding Bugs

- How to find bugs
  - Add debug print statements
    - Print out state of variables, loop values, etc.
    - Remove before submitting
  - Use debugger in your IDE
  - Talk through program line-by-line
    - Explain it to a:
      - Programming novice
      - Rubber duckie
      - Teddy bear
      - Potted plant
      - …



Debugging Example

- Problem:
  - For integer N > 1, compute its *prime factorization*
    - $98 = 2 \times 7^2$
    - $17 = 17$
    - $154 = 2 \times 7 \times 11$
    - $16{,}562 = 2 \times 7^2 \times 13^2$
    - $3{,}757{,}208 = 2^3 \times 7\ 13^2 \times 397$
    - $11{,}111{,}111{,}111{,}111 = 2{,}071{,}723 \times 5{,}363{,}222{,}357$
  - Possible application: Break RSA encryption
    - Factor 200-digit numbers
    - Used to secure Internet commerce

A Simple Algorithm

- Problem:
  - For integer N > 1, compute its *prime factorization*
- Algorithm:
  - Starting with i=2
    - Repeatedly divide N by i as long as it evenly divides, output i every time it divides
  - Increment i
  - Repeat

Example Run

i	N	Output
2	16562	2
3	8281	
4	8281	
5	8281	
6	8281	
7	8281	7 7
8	169	
9	169	
10	169	
11	169	
12	169	
13	169	13 13
14	1	
…	1	

Buggy Factorization Program

```
public class Factors
{
    public static void main(String [] args)
    {
        long n = Long.parseLong(args[0])
        for (i = 0; i < n; i++)
        {
            while (n % i == 0)
                System.out.print(i + " ")
                n = n / i
        }
    }
}
```

This program has many bugs!

Debugging: Syntax Errors

```
public class Factors
{
    public static void main(String [] args)
    {
        long n = Long.parseLong(args[0]);
        for (int i = 0; i < n; i++)
        {
            while (n % i == 0)
                System.out.print(i + " ");
                n = n / i;
        }
    }
}
```

- Syntax errors
  - Illegal Java program
  - Usually easily found and fixed

Debugging: Semantic Errors

```
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (int i = 0; i < n; i++)
      {
         while (n % i == 0)
            System.out.print(i + " ");
            n = n / i;
      }
   }
}
```

Need to start at 2 since 0 and 1 cannot be factors.

```
% java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
        at Factors.main(Factors.java:8)
```

- Semantic error
  - Legal but wrong Java program
  - Run program to identify problem

Debugging: Semantic Errors

```
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (int i = 2; i < n; i++)
      {
         while (n % i == 0)
            System.out.print(i + " ");
            n = n / i;
      }
   }
}
```

Indentation implies a block of code, but without braces, it's not.

```
% java Factors 98
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ...
```

Debugging: Even More Problems

```
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (int i = 2; i < n; i++)
      {
         while (n % i == 0)
         {
            System.out.print(i + " ");
            n = n / i;
         }
      }
   }
}
```

```
% java Factors 98
2 7 7 %
```

Need newline

```
% java Factors 5
```

No output???

```
% java Factors 6
2 %
```

Missing the 3???

Debugging: Adding Trace Print Statement

```java
public class Factors
{
    public static void main(String [] args)
    {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i < n; i++)
        {
            while (n % i == 0)
            {
                System.out.println(i + " ");
                n = n / i;
            }
            System.out.println("TRACE " + i + " " + n);
        }
    }
}
```

```
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5

% java Factors 6
2
TRACE 2 3
```

i for-loop should go up to n!

Success?

```
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (int i = 2; i <= n; i++)
      {
         while (n % i == 0)
         {
            System.out.print(i + " ");
            n = n / i;
         }
      }
      System.out.println();
   }
}
```

Fixes the "off-by-one" error in the loop bounds.

Fixes the lack of line feed problem.

```
% java Factors 5
5

% java Factors 6
2 3

% java Factors 98
2 7 7

% java Factors 3757208
2 2 2 7 13 13 397
```

Except for Really Big Numbers...

```java
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (int i = 2; i <= n; i++)
      {
         while (n % i == 0)
         {
            System.out.print(i + " ");
            n = n / i;
         }
      }
      System.out.println();
   }
}
```

```
% java Factors 11111111
11 73 101 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723 -1 -1 -1 -1 -1 -1 -1 ...
```

Correct, But Too Slow

```java
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (long i = 2; i <= n; i++)
      {
         while (n % i == 0)
         {
            System.out.print(i + " ");
            n = n / i;
         }
      }
      System.out.println();
   }
}
```

```
% java Factors 11111111
11 73 101 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723 5363222357
```

Faster Version

```
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (long i = 2; i*i <= n; i++)
      {
         while (n % i == 0)
         {
            System.out.print(i + " ");
            n = n / i;
         }
      }
      System.out.println();
   }
}
```

Missing last factor

```
% java Factors 98
2 7 7

% java Factors 11111111
11 73 101

% java Factors 11111111111
21649

% java Factors 11111111111111
11 239 4649

% java Factors 11111111111111111
2071723
```

Fixed Faster Version

```java
public class Factors
{
   public static void main(String [] args)
   {
      long n = Long.parseLong(args[0]);
      for (long i = 2; i*i <= n; i++)
      {
         while (n % i == 0)
         {
            System.out.print(i + " ");
            n = n / i;
         }
      }
      if (n > 1)
         System.out.println(n);
      else
         System.out.println();
   }
}
```

Handles the "corner case"

Need special case to print biggest factor (unless it occurs more than once)

```
% java Factors 98
2 7 7

% java Factors 11111111
11 73 101 137

% java Factors 11111111111
21649 513239

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723 5363222357
```

Factors: Analysis

- How large an integer can I factor?

```
% java Factors 3757208
2 2 2 7 13 13 397

% java Factors 9201111169755555703
9201111169755555703
```

digits	(i <= n)	(i*i <= n)
3	instant	instant
6	0.15 seconds	instant
9	77 seconds	instant
12	21 hours *	0.16 seconds
15	2.4 years *	2.7 seconds
18	2.4 millennia *	92 seconds

* estimated

Incremental Development

- Split development into stages:
  - Test thoroughly after each stage
    - Don't move on until it's working!
    - Bugs are (more) isolated to the part you've just been working on
    - Prevents confusion caused by simultaneous bugs in several parts

Summary

- Debugging
  - Types of Errors
    - Syntax Errors
    - Semantic Errors
    - Logic Errors
- Preventing Bugs
  - Have a plan before coding, use good style
  - Learn to trace execution
    - On paper, with print statements, using the debugger
  - Explain it to a teddy bear
  - Incremental development
- Test, Test, Test!!