Testing

- Unit Testing
- How to Test each Function
“To err is human, but to really foul things up you need a computer”

-- Paul Ehrlich
Software errors cost the U.S. economy $60 billion annually in rework, lost productivity and actual damages.
We all know software bugs can be annoying,

But faulty software can also be expensive, embarrassing, destructive and deadly.
Cost: $125 million
Disaster: After a 286-day journey from Earth, the Mars Climate Orbiter fired its engines to push into orbit around Mars. The engines fired, but the spacecraft fell too far into the planet’s atmosphere, likely causing it to crash on Mars.
Mars Climate Orbiter (1999)

**Cost:** $125 million

**Disaster:** After a 286-day journey from Earth, the Mars Climate Orbiter fired its engines to push into orbit around Mars. The engines fired, but the spacecraft fell too far into the planet’s atmosphere, likely causing it to crash on Mars.

**Cause:** Failure to convert between English and Metric Units.
LAX Flights Grounded (2007)

- 17,000 planes grounded at LAX
- Software problem hit US Customs and Border Protection agency
- Network care: instead of shutting down, sent out incorrect data across network
- Data cascaded and hit entire network bring it to standstill
- Nobody could be authorized to leave or enter US for 8 hours
Mars Polar Lander (1999)
few months after Orbiter

**Cost:** $165 million

**Disaster:** after descent phase was expected to be complete, lander failed to reestablish communication with Earth.

**Cause:** improperly ceased engine firing prior to lander touching surface, lander impacted at 50 mph. The inappropriate response of its engines was attributed to software glitches.
due to “...input and Division by ‘0’. ”

\[ X / 0 = \text{undefined} \ldots \]

i.e., division by 0
Cost: $500 million
Disaster: Ariane 5, Europe’s newest unmanned rocket, was intentionally destroyed seconds after launch on its maiden flight. Also destroyed was its cargo of four scientific satellites to study how the Earth’s magnetic field interacts with solar winds.
Ariane Rocket Goes Boom (1996)

Cost: $500 million

Disaster: Ariane 5, Europe’s newest unmanned rocket, was intentionally destroyed seconds after launch on its maiden flight. Also destroyed was its cargo of four scientific satellites to study how the Earth’s magnetic field interacts with solar winds.

Cause: Shutdown occurred when the guidance computer tried to convert the sideways rocket velocity from 64-bits to a 16-bit format. The number was too big, and an overflow error resulted.
Patriot Fails Soldiers (1991)

Cost: 28 soldiers dead, 100 injured
Disaster: During the first Gulf War, an American Patriot Missile system in Saudi Arabia failed to intercept an incoming Iraqi Scud missile. The missile destroyed an American Army barracks.
Patriot Fails Soldiers (1991)

Cost: 28 soldiers dead, 100 injured

Disaster: During the first Gulf War, an American Patriot Missile system in Saudi Arabia failed to intercept an incoming Iraqi Scud missile. The missile destroyed an American Army barracks.

Cause: A software rounding error incorrectly calculated the time, causing the Patriot system to ignore the incoming Scud missile.
AT&T Lines Go Dead (1990)

Cost: 75 million phone calls missed, 200 thousand airline reservations lost

Disaster: A single switch at one of AT&T’s 114 switching centers suffered a minor mechanical problem and shut down the center. When the center came back up, it sent a message to other switching centers, which in turn caused them to shut down and brought down the entire AT&T network for 9 hours.
**AT&T Lines Go Dead (1990)**

**Cost:** 75 million phone calls missed, 200 thousand airline reservations lost

**Disaster:** A single switch at one of AT&T’s 114 switching centers suffered a minor mechanical problem and shut down the center. When the center came back up, it sent a message to other switching centers, which in turn caused them to shut down and brought down the entire AT&T network for 9 hours.

**Cause:** A single line of buggy code in a complex software upgrade implemented to speed up calling caused a ripple effect that shut down the network.
Cost: Nearly all of humanity

Disaster: The Soviet early warning system falsely indicated the United States had launched five ballistic missiles. Fortunately the Soviet duty officer had a “funny feeling in my gut” and reasoned if the U.S. was really attacking they would launch more than five missiles, so he reported the apparent attack as a false alarm.
Cost: Nearly all of humanity

Disaster: The Soviet early warning system falsely indicated the United States had launched five ballistic missiles. Fortunately the Soviet duty officer had a “funny feeling in my gut” and reasoned if the U.S. was really attacking they would launch more than five missiles, so he reported the apparent attack as a false alarm.

Cause: A bug in the Soviet software failed to filter out false missile detections caused by sunlight reflecting off cloud-tops.
Mariner Bugs Out (1962)

The Mariner 1 rocket with a space probe headed for Venus diverted from its intended flight path shortly after launch. Mission Control destroyed the rocket 293 seconds after liftoff.
The Mariner 1 rocket with a space probe headed for Venus diverted from its intended flight path shortly after launch. Mission Control destroyed the rocket 293 seconds after liftoff.

**Cause:** due to “..period instead of comma in FORTRAN DO-Loop...”
Do Any Of These Come To Mind?

- It works for me
- We tried that once and it didn’t work
- I like to code, not test
- It is not my job to test
- Testing takes too much time
- But we don’t know if the answer is right
- The autograder will tell us if there is a problem
- If it already works why test it?
- If we design it right, then it should work
WEIRD — MY CODE'S CRASHING WHEN GIVEN PRE-1970 DATES.

EPOCH FAIL!
How users see the programmers

π D / Z
How programmers see the users

Ugh!

OOGH!

TOC TOC!!!
So Let's Talk About Design

bool equalsDigit(int num, int digit);

//Requires: num >= 0; 0 <= digit <= 9
//Effects: returns true if 'digit' occurs within 'num'
//         false otherwise

How should equalsDigit work?

<table>
<thead>
<tr>
<th></th>
<th>Input(s)</th>
<th>Expected Output(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0, and 0</td>
<td>true</td>
</tr>
<tr>
<td>b</td>
<td>0, and 1</td>
<td>false</td>
</tr>
<tr>
<td>c</td>
<td>12, and 2</td>
<td>true</td>
</tr>
<tr>
<td>d</td>
<td>25, and 2</td>
<td>true</td>
</tr>
<tr>
<td>e</td>
<td>125, and 1</td>
<td>true</td>
</tr>
<tr>
<td>f</td>
<td>12345, and 6</td>
<td>false</td>
</tr>
</tbody>
</table>
So Let's Talk About Design

bool equalsDigit(int num, int digit);

//Requires: num >= 0; 0 <= digit <= 9
//Effects: returns true if 'digit' occurs within 'num'
//false otherwise

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
<th>Test Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a 0, and 0</td>
<td>true</td>
<td>cout &lt;&lt; equalsDigit(0, 0);</td>
</tr>
<tr>
<td>b 0, and 1</td>
<td>false</td>
<td>cout &lt;&lt; equalsDigit(0, 1);</td>
</tr>
<tr>
<td>c 12, and 2</td>
<td>true</td>
<td>cout &lt;&lt; equalsDigit(12, 2);</td>
</tr>
<tr>
<td>d 25, and 2</td>
<td>true</td>
<td>cout &lt;&lt; equalsDigit(25, 2);</td>
</tr>
<tr>
<td>e 125, and 1</td>
<td>true</td>
<td>cout &lt;&lt; equalsDigit(125, 1);</td>
</tr>
<tr>
<td>f 12345, and 6</td>
<td>false</td>
<td>cout &lt;&lt; equalsDigit(12345, 6);</td>
</tr>
</tbody>
</table>
int main()
{
    void test_equalsDigit()
    {
        cout << equalsDigit(0, 0);
        cout << equalsDigit(0, 1);
        cout << equalsDigit(12, 2);
        cout << equalsDigit(25, 2);
        cout << equalsDigit(125, 1);
        cout << equalsDigit(12345, 6);
    }
    ...
}

```cpp
int reverseNumber (int n);

//Requires: n >= 0
//Modifies: nothing
//Effects: returns the int formed by reversing the digits of n
```

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>42</td>
</tr>
<tr>
<td>d</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>1234</td>
</tr>
</tbody>
</table>

Test Code
```
cout << reverseNumber(0);
cout << reverseNumber(1);
cout << reverseNumber(42);
cout << reverseNumber(100);
cout << reverseNumber(1234);
```
int main()
{
    test_equalsDigit();
    test_reverseNumber();
    ...
}

void test_reverseNumber()
{
    cout << reverseNumber(0);
    cout << reverseNumber(1);
    cout << reverseNumber(42);
    cout << reverseNumber(100);
    cout << reverseNumber(1234);
}
It Will Take Too Long

- What do you normally do?
  - A little analysis
  - A little design
  - A manageable chunk of coding
  - Some quick tests
  - Debugging
  - And then some documentation

- One Alternative
  - 15 mins think hard
  - 15 mins draw diagrams
  - 30 mins write code
  - 03 mins run code
  - 30 mins debug
  - 15 mins add readable variable names
It Will Take Too Long

- **Test Driven**
  - 20 mins define tests
  - 30 mins write tests
  - 07 mins write drivers
  - 30 mins write code
  - 03 mins run all tests
  - 15 mins simplify code
  - 03 mins run all test

- **One Alternative**
  - 15 mins think hard
  - 15 mins draw diagrams
  - 30 mins write code
  - 03 mins run code
  - 30 mins debug
  - 15 mins add readable variable names
It Will Take Too Long

- **Test Driven**
  - 20 mins define tests
  - 30 mins write tests
  - 07 mins write drivers
  - 30 mins write code
  - 03 mins run all tests
  - 15 mins simplify code
  - 03 mins run all tests
  - **108 minutes**

- **One Alternative**
  - 15 mins think hard
  - 15 mins draw diagrams
  - 30 mins write code
  - 03 mins run code
  - 30 mins debug
  - 15 mins add comments
  - **108 minutes**
It Will Take Too Long

- **Test Driven**
  - 20 mins define tests
  - 30 mins write tests
  - 07 mins write CRC
  - 30 mins write code
  - 03 mins run all tests
  - 15 mins simplify code
  - 03 mins run all tests
  - **108 minutes**

- **One Alternative**
  - 15 mins think hard
  - 15 mins draw diagrams
  - 30 mins write code
  - 03 mins run code
  - 30 mins debug
  - 15 mins add comments
  - **108 minutes**
Want to save time?

- Test as you go
- One function at a time
  - write
  - test
- Learn the debugger
  - watch your code execute
  - one line at a time
  - watch your variables change values
  - obvious when it goes wrong