University of Strathclyde Engineering

Lecture 4:
Testing

With thanks to Greig Paul and Alisdair McDiarmid

Overview

University of Strathclyde Engineering

- When you write software, how do you know that it's working properly?
- You have to test it!
- Learning how to test is as important as learning how to program
- Because if you don't test your software…

Zune Crash

- On December 31st 2008, all Microsoft Zune players crashed for a day
- Users were unable to play their music or video
- Why did this happen?

Zune Crash

University of Strathclyde Engineering

- Lack of proper testing!
- 2008 was a leap year, so there were 366 days
- The Zune software was not tested for leap years, and a bug in the code made it crash
- See if you can spot the bug…

Zune Crash

University of Strathclyde Engineering

Find the year, given the number of days since 1980

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

Zune Crash

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

days = 10593;
year = 1980;

Zune Crash

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

days = 10227;
year = 1981;

Zune Crash

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

days = 9862;
year = 1982;

Zune Crash

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

days = 9497;
year = 1983;

Zune Crash

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

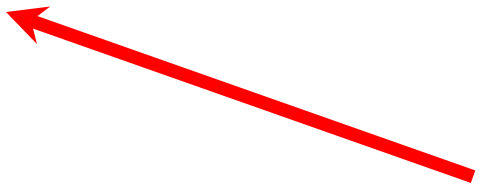
days = 366;
year = 2008;

Zune Crash

```
year = ORIGINYEAR; // = 1980

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

days = 366;
year = 2008;



Can't leave the while loop!

Let’s Make an Altimeter

```
#include<stdio.h>
int main()
{
  short altitude = 0;
  short ascent_rate = 3000;
  for (int i=0; i < 25; i++)
  {
    altitude += ascent_rate;
    printf("Altitude: %d\n", altitude);
  }
  return 0;
}
```

Let’s Make an Altimeter

```
jamesi@wellbeck:~/tests$ gcc -Os altimeter.c -o altimeter
jamesi@wellbeck:~/tests$ ./altimeter
Altitude: 3000
Altitude: 6000
Altitude: 9000
Altitude: 12000
Altitude: 15000
Altitude: 18000
Altitude: 21000
Altitude: 24000
Altitude: 27000
Altitude: 30000
```

Let's Make an Altimeter

```
jamesi@wellbeck:~/tests$ gcc -Os altimeter.c -o altimeter
jamesi@wellbeck:~/tests$ ./altimeter
Altitude: 3000
Altitude: 6000
Altitude: 9000
Altitude: 12000
Altitude: 15000
Altitude: 18000
Altitude: 21000
Altitude: 24000
Altitude: 27000
Altitude: 30000
Altitude: -32536
```

Integer Overflow → Short = 16 bits

Why Test Your Code?

University of Strathclyde Engineering

- Feel sure that your program is working before you get it marked
- Important skills for future software engineering or group projects
- Tests help you catch bugs when you're working on a large project
- You'll get more marks for it!

Why Test Your Code?

University of Strathclyde Engineering

- When working in agile/remote teams, we develop software to a specification
- How can we work effectively in parallel?
- How can we avoid sitting and waiting for someone else to build their bit of code?
- How do we know it will fit together?

Why Test Your Code?

University of Strathclyde Engineering

- Design your tests and specify how the code will behave (BEFORE you write the code!)
- Then you know what to expect of each other, and can build each part separately (e.g. agree function header definition – return type, inputs) and “definition of good”)
- But you’ll need to find a way to validate the code you write, without testing the full thing (as others’ work won’t be ready for testing)

Altimeter System Design

(QNH is a Q code for the pressure at sea level)

A

```
int32_t currentAltitude_ft(uint32_t press_mbar, double QNH) {
  // Developer A (logic implementer) does this
  return altitude_ft;
}
```

B

```
// QNH is entered via "spinny dial" on the cockpit
double QNHfromDial(CustomStruct OptoEncoderData) {
  // Developer B (instrument input) turns this into a decimal
  return QNH;
}
```

C

```
// Pressure comes from analog sensors
uint32_t pressureFromPitot_mbar(int32_t pitot_ADC_input) {
  // Developer C (ADC input) converts ADC input into air pressure
  return airPressure_mBar;
}
```

Things to notice…

- All variables appended with _units – helps you to sanity check and remind yourself!
- Developer A can't test their code without Developer B and C both finishing their bit?
- Or.... Can they....?
- We know the output Developer B and C should give us... So Developer A can test their code using "test values"!
- (Simplified example, real avionics systems won't pass around uint32_t's – use custom types!!)

Altimeter System Design Stubs

A

```
int32_t currentAltitude_ft(uint32_t press_mbar, double QNH) {
  // Developer A (logic implementer) does this
  return altitude_ft;
}
```

A

```
// QNH is entered via "spinny dial" on the cockpit
double QNHfromDial(void) {
  return 29.92;
}
```

A

```
// Pressure comes from analog sensors
uint32_t pressureFromPitot_mbar(void) {
  return 1013;
}
```

How to Test Code

- Manual testing is a good start!
- Driver testing is better
- Automated test suites are best

Manual Testing

- Write and compile your program
- Run the program
- Manually create correct input-output pairs
- Test that the input leads to the output

Manual Testing Issues

- Can only test the whole program at once
- Time consuming to enter the test data at the keyboard and check the results
- Tedious to run the tests multiple times—so you probably won't!

Driver Testing

University of Strathclyde Engineering

- Write your program using functions
- Create a separate driver program for testing those functions
- Use the driver to run the functions for your test input-output pairs

- Use simulator and terminal I/O to access printf and see results on screen

Driver Testing Example

```
int abs(int a)
{
  if (a < 0)
    return 0 - a;
  else
    return a;
}
```

Driver Testing Example

```
int abs(int a)
{
  if (a < 0)
    return 0 - a;
  else
    return a;
}

void main(void)
{
  printf("abs(10) should be 10, is %d", abs(10));

}
```

Driver Testing Example

```
int abs(int a)
{
  if (a < 0)
    return 0 - a;
  else
    return a;
}

void main(void)
{
  printf("abs(10) should be 10, is %d", abs(10));
  printf("abs(-4) should be 4, is %d", abs(-4));

}
```

Driver Testing Example

```
int abs(int a)
{
  if (a < 0)
    return 0 - a;
  else
    return a;
}

void main(void)
{
  printf("abs(10) should be 10, is %d", abs(10));
  printf("abs(-4) should be 4, is %d", abs(-4));
  printf("abs(0) should be 0, is %d", abs(0));
}
```

Driver Testing Issues

- Have to compile and run the test program separately from the real program
- Examine the test output carefully to check that it is correct
- Lots of time typing printf!

Automated Testing

University of Strathclyde Engineering

- Use a dedicated test library to help
- For example, for Java a good option is JUnit, which is supported by Eclipse

- C is low level, but can still use assertions
- Write your tests as a series of assertions
- Run the tests easily and check for bugs

Automated Testing

University of Strathclyde Engineering

- void assert (int expression)
- Need `#include <assert.h>` to use

- Checks an expression.
  - If it is false it prints a message to stderr and aborts
- The message has the following format:
  - File *name*; line *num* # Assertion failure "*expression*"
- To ignore assert calls
  - Put a `#define NDEBUG` statement *before* the `#include <assert.h>` statement.

Asserts

```
#include<assert.h>
#include<stdio.h>
int main()
{
int i=0;
  while (true)
  {
    i++;
    assert(i < 128);
    printf("%02x\n", i);
  }
}
```

Asserts

```
jamesi@wellbeck:~/tests$ gcc -Os assert.c -o assert
jamesi@wellbeck:~/tests$ ./assert
01
02
03
...
7c
7d
7e
7f
assert: assert.c:9: main: Assertion `i < 128' failed.
Aborted
jamesi@wellbeck:~/tests$
```

Automated Testing Issues

University of Strathclyde Engineering

- Need to learn how to use assertions
- Lots of effort up front writing the test suite

  …but your code will be much better for it!

- Remember that if an assert() fails “in the field”, your program will fail (although at least a bit predictably)
  - Hope you didn’t disable the watchdog!
- You want to use this pre-deployment only!

What To Test

- Edge cases (and boundaries)
  - Negative, when positive expected
  - Far too large a number, etc.
- Branches and loops
  - Test every branch condition (unlike Zune!)
- Calculations
  - Pen and paper, calculator, Excel, etc

Edge Cases

University of Strathclyde Engineering

- What are the valid limits for your variables or parameters?
- Test:
  - Just before the edge
  - Exactly on the edge
  - Over the edge
  - Absurdly beyond normal values

Branches and Loops

- Try to cover all of your code
- Set inputs to test both sides of each if (…) else (…) statement
- Test your loops! Iterate zero, one, and more times

Calculations

- Use a calculator or other software to derive correct input/output data
- Again, test edge cases and branches
- Check for error handling: what if inputs are invalid?

Testing for Security

- If you build something taking input from outside of your program, security is an issue
  - That means effectively anything on an embedded system!
  - Input could be buttons, ADC, I2C, SPI etc.
- Be careful of assumptions
  - What if that integer is negative?
  - What if the ADC value is out-of-range? 0? 255? 65535?
  - What if you get back garbage over I2C or SPI?
- We need to detect these kinds of scenarios!
- Use tests/asserts to detect and avoid issues

Testing for Security

- Be careful of signed versus unsigned ints, and size
- Rather than “int”, better to use explicit size and signage – `int16_t` = signed 16 bit. `uint8_t` = unsigned 8 bit
- `stdint.h` defines clearly-named types
- `uint8_t, uint16_t, uint32_t, int8_t, int16_t, int32_t` etc.
- Beware of integer overflows, underflows etc.
  - 255+1 = 256? Unless it’s an 8-bit integer
  - 127+1 = 128? Unless it’s 8-bit signed, then it’s -128!
  - (127+1) < 128 for an 8-bit signed integer!!

Arrays

- An array is a group of a given variable type, all put together in memory
  - `int score [5];`
  - This makes us 5 integers, one after the other
  - The first is at `score[0],` and the last is at `score[4]`
- Don't go outside the array, or you'll reach the next variable!
- You won't get any errors/warnings about this!
- Be careful accessing memory by variable:
  - `score[offset] = 10;`
  - Is offset in the allowed range?

```
#include<stdio.h>
int main()
{
  int lengthOfData = 5;
  int ourData [5];
  int a = 100;
  for (int i=0; i < lengthOfData + 6; i++)
    {
      if (i == 8)
        ourData[i] = 0;
      else if (i == 9)
        ourData[i] = 40;
      else
        ourData[i] = i;
    }
  for (int i=0; i < lengthOfData; i++)
    printf("%d\n", ourData[i]);
  printf("a = %d\n", a);
  return 0;
}
```

What is the Value of a?

Duh? 100 surely? – we didn't change it

```
jamesi@wellbeck:~/tests$ ./overflow
0
1
2
3
4
5
6
7
0
10
a = 0
Segmentation fault
jamesi@wellbeck:~/tests$
```

Actually we did…

- `ourData[8] = 0;` // overwrote a
- `ourData[9] = 40;`
- We changed the “for” loop max iteration counter too with ourData[9] – that’s why it printed more than 5 values out!
- Memory managed operating systems will segfault on access outwith your memory
- Microcontrollers don’t - be careful!

- Note if you try the code example, your actual results may differ depending on the compiler & processor architecture

Test-Driven Development

University of Strathclyde Engineering

- Write tests first, code second, then fix everything up after that
- Tests define valid behaviour
- Use testing to formalise requirements
- Highly successful methodology

Summary

- Does your software work? Test your code
- Manual, driver, and automated testing

- Edge cases, branches/loops, calculations
- Lots more to learn if you're interested!

University of Strathclyde Glasgow

DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING