

Testing and Debugging

Lecture 15



Outline

1. Function Analysis
 - Preconditions, postconditions, return values
2. Testing/Debugging Culture via XKCD
 - <http://xkcd.com>
3. Testing Programs & Functions
4. Review of Debugging



Review Exercise

Write a function named **swap** that takes two integer arguments and swaps their values

- After the function is done the first argument should be what the second argument was and vice versa

Example:

```
int val1 = 5;
int val2 = 8;
swap( val1, val2 );
// val1 will be 8
// val2 will be 5
```



Answer

```
void swap(int& x, int& y)
{
    int temp = x;
    x = y;
    y = temp;
}
```



Preconditions and Postconditions

- Whenever you have a function, you should document any *assumptions* made by the function and what the *effects* of the function are
 - Called preconditions (assumptions) and postconditions (effects)
- This allows you to understand how to use a function without needing to look at the actual code for that function



Example

- The **swap** function uses both **x** and **y** without assigning any values, so **x** and **y** must both have *some* value before **swap** is called
- After **swap** is finished, the new value of **x** is the original value of **y**, and the new value of **y** is the original value of **x**

```
// Preconditions: x and y have been initialized with some values  
// Postconditions: the values of x and y have been swapped
```

```
void swap(int& x, int& y)  
{  
    int temp = x;  
    x = y;  
    y = temp;  
}
```



Return Values

- Return values are usually listed separately from the preconditions and postconditions
- Sometimes given as postconditions

```
// Preconditions: a and b are lengths of short sides of a triangle
```

```
// Postconditions: none
```

```
// Returns: the length of the hypotenuse of the triangle
```

```
double hypotenuse(double a, double b)
```

```
{
```

```
    return sqrt( a*a + b*b );
```

```
}
```

```
// Preconditions: a and b are lengths of short sides of a triangle
```

```
// Postconditions: returns the length of the hypotenuse of the triangle
```

```
double hypotenuse(double a, double b)
```

```
{
```

```
    return sqrt( a*a + b*b );
```

```
}
```



Conditions

- Not all functions have preconditions or postconditions
- Example:

```
// Preconditions: none
// Postconditions: val has been given a value (from the user)
void get_integer(int& val)
{
    cout << "Enter an integer\n";
    cin >> val;
}
```



Exercise

What are the preconditions, postconditions, and return values for this function?

```
double do_something(int& val1, int val2)
{
    double result;
    result = val1 * val2;
    val1++;
    val2--;
    return result;
}
```



Answer

```
// Preconditions: val1 and val2 have been initialized
// Postconditions: val1 has been incremented by one
// Returns: the original value of val1 multiplied by val2
double do_something(int& val1, int val2)
{
    double result;
    result = val1 * val2;
    val1++;
    val2--;
    return result;
}
```



Testing Programs and Functions

- When testing your code, always test a variety of input values
- Never test only one or two values because those samples may not catch some errors
- Always test "interesting" values
 - Values that show up in the code (e.g. boundary values that change loop or if/else behavior)



“Interesting” Values (1)



“Interesting” Values (2)

$$\sqrt{\heartsuit} = ?$$

$$\cos \heartsuit = ?$$

$$\frac{d}{dx} \heartsuit = ?$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \heartsuit = ?$$

$$F\{\heartsuit\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{it\heartsuit} dt = ?$$

My normal approach
is useless here.



Be Nice to Your Computer

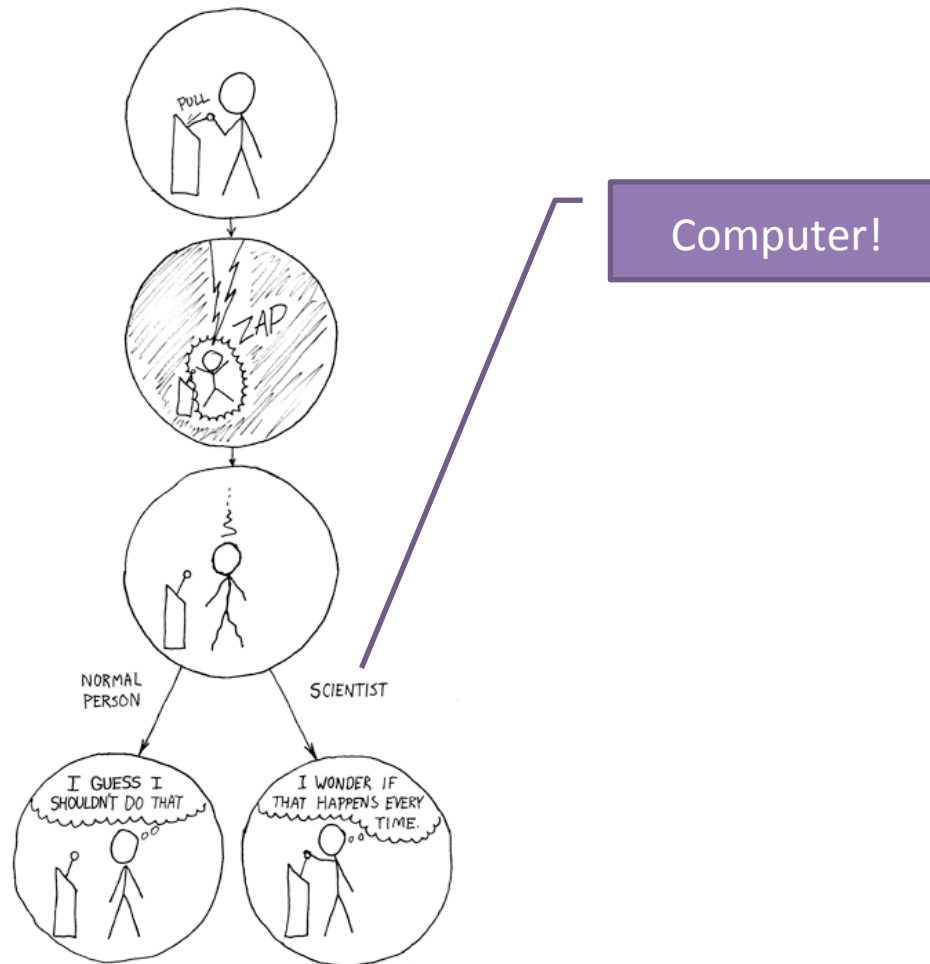
Crash



Everything



Be Diligent



An Unrelated Warning



Example Bug: Microsoft, 12/31/08

Early this morning we were alerted by our customers that there was a widespread issue affecting our 2006 model Zune 30GB devices (a large number of which are still actively being used). The technical team jumped on the problem immediately and isolated the issue: a bug in the internal clock driver related to the way the device handles a leap year. That being the case, the issue should be resolved over the next 24 hours as the time change moves to January 1, 2009. We expect the internal clock on the Zune 30GB devices will automatically reset tomorrow (noon, GMT). By tomorrow you should allow the battery to fully run out of power before the unit can restart successfully then simply ensure that your device is recharged, then turn it back on. If you're a Zune Pass subscriber, you may need to sync your device with your PC to refresh the rights to the subscription content you have downloaded to your device.



Zune Bug

```
// days is the number of days since 1/1/1980, e.g., 10592  
year = 1980;
```

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

What happens
on last day of
a leap year?

EPOCH FAIL!



Function Notes

- Always include comments describing all preconditions, postconditions, and return values for every function
 - Don't necessarily have all three for every function
- When testing your code, always test many values, including any interesting values that change the way the code behaves



Complicated Functions

- When you write a complicated function, you need to test it in isolation
- Copy the function to a new project/file and write a special `main()` function to test it
 - Called a *driver*
- Test a variety of inputs that cover the different possibilities in the function



Testing the Code

```
#include <iostream>
using namespace std;

// Preconditions: days is the number of days since Jan 1, 1980
// Postconditions: year is the year computed from days, and days
//                is the number of days in that year
void calc_date(int& year, int& days);

// Preconditions: year is initialized
// Postconditions: none
// Returns:       true if year is a leap year, false otherwise
bool IsLeapYear(int year);

int main()
{
    int y;
    int d=1000; // try 10592 and 10593
    calc_date( y, d );
    cout << "year is " << y << " and day is " << d << endl;
    return 0;
}
```



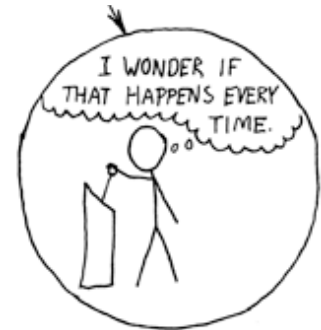
Common Errors

- Using = instead of ==
- Using > or < instead of <= or >=
- Other off-by-one errors
- Call by reference instead of call by value (or vice versa)
- Integer division



Localize Errors

- When you don't get the output you expect, **DO NOT** just change code randomly
- Narrow down where the problem is by checking values throughout the program
 - Use **cout** statements at key points to check variable values
 - Also use **cout** statements to verify which branch of **if/else** statements are taken, and how many iterations a loop goes through



Example

```
#include <iostream>
using namespace std;

double cone_volume(double radius, double height);

int main()
{
    cout << "cone_volume( 2.0, 5.0 )="
         << cone_volume( 2.0, 5.0 ) << endl;
    return 0;
}

double cone_volume(double radius, double height)
{
    const double PI=3.14159;
    double volume = 1 / 3 * PI * radius * radius * height;
    return volume;
}
```



Error Localized

```
double cone_volume(double radius, double height)
{
    const double PI=3.14159;
    cout << "cone_volume: PI=" << PI << endl;
    cout << "cone_volume: radius=" << radius << endl;
    cout << "cone_volume: height=" << height << endl;
    //double volume = 1 / 3 * PI * radius * radius * height;
    //double volume = 1 / 3 * PI;
    double volume = 1 / 3;
    cout << "cone_volume: volume=" << volume << endl;
    return volume;
}
```



Integer
division!



Example

```
#include <iostream>
using namespace std;
void do_choice(char choice, int val1, int val2, int answer);
int main()
{
    int answer;
    do_choice( '*', 10, 20, answer );
    cout << "main: answer=" << answer << endl;
    return 0;
}
void do_choice(char choice, int val1, int val2, int answer)
{
    if ( choice = '+' )
    {
        answer = val1 + val2;
    }
    else if ( choice = '-' )
    {
        answer = val1 - val2;
    }
    else if ( choice = '*' )
    {
        answer = val1 * val2;
    }
    else
    {
        answer = val1 / val2;
    }
}
```

Forgot call by reference!

Single equal sign!



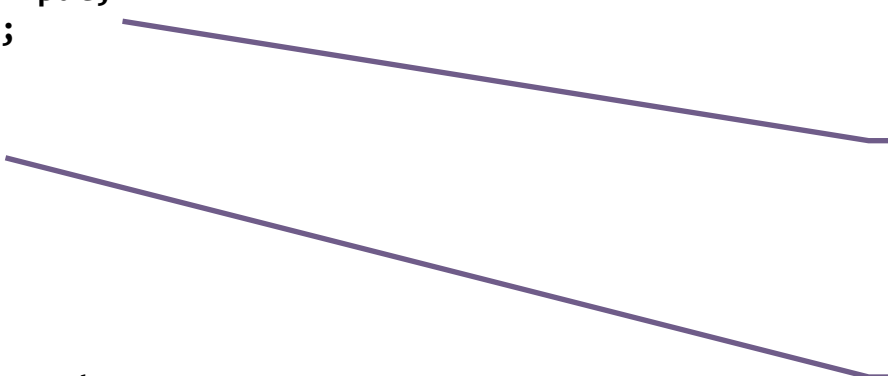
Example

```
#include <iostream>
using namespace std;

int main()
{
    double sum=0, avg=0, input;
    int count=0;

    cout << "Enter a set of positive numbers, and a negative number to stop." << endl;
    cin >> input;
    while ( input >= 0 )
    {
        sum = sum + input;
        cin >> input;
    }

    if ( count = 0 )
    {
        avg = 0;
    }
    else
    {
        avg = sum / count;
    }
    cout << "The average is " << avg << endl;
    return 0;
}
```



Forgot count increment!

Single equal sign!



Review of Debugging

- Debuggers help you to quickly find and identify errors in your code
- Allow you to:
 - *Step* through your code one line at a time
 - View *current values* of all variables as the program progresses
 - Set *breakpoints* that will stop the code at certain places in your code
- Should NOT be used *instead* of proper testing/analysis, but as an assistive tool



Visual Studio Debugger (1)

- Visual Studio has a built in debugger that has all the normal debugging functionality
- Set breakpoints by clicking on the left of a line of code in the light gray vertical bar
- Start the program with debugging (F5, or the play button), or go to the Debug menu and click Step Into (F11) or Step Over (F10)
 - If you use F5, the program will execute like normal until you come to a breakpoint
 - If you use F11 or F10, the program will enter execution mode like normal, but will stop and wait for you to do something on the first line of your main() function



Visual Studio (2)

- Once debugging you can see the value of all local variables in the bottom left window (look for Autos and Locals)
 - Locals is the list of all local variables in the current function
 - Autos is an auto-generated list that contains local variables, return values from functions, and other recently used variables
- Use the Play, Step Into, Step Over, and Step Out buttons to navigate through the code
 - Play starts the program running again, and it will continue until another breakpoint is encountered
 - Step Into executes the next line of code, and goes into a function if that line is a function call
 - Step Over executes the next line of code, and runs (but does not step through) a function if that line is a function call
 - Step Out starts the program running again, and it continues until the current function returns, then stops again



Example

```
#include <iostream>
using namespace std;

int main()
{
    double sum=0, avg=0, input;
    int count=0;

    cout << "Enter a set of positive numbers, and a negative number to stop." << endl;
    cin >> input;
    while ( input >= 0 )
    {
        sum = sum + input;
        cin >> input;
    }

    if ( count = 0 )
    {
        avg = 0;
    }
    else
    {
        avg = sum / count;
    }
    cout << "The average is " << avg << endl;
    return 0;
}
```



Steps (1)

- Start by putting a breakpoint at the **if** statement
 - Click in the gray bar on the left of the line
 - It should add a red dot to indicate that there is a breakpoint there now
 - You can click on the dot to remove the breakpoint
- Hit F5 to start debugging, and enter some values
- It will stop automatically at the **if** statement
- Examine the **count** variable and see that it is zero
- Click the Step Into button to execute the **if** statement



Steps (2)

- It jumps down to the **else** statement, even though count was equal to zero, so you then know to look very hard at the **if** statement
 - When an **if/else** does the "wrong" thing, it's usually a missing = sign
- Fix the **if** statement, then stop debugging and start the program over (with debugging)
- The **if** statement is correct now, but **count** is still zero, so you have to look at all the places count changes
 - It never changes!



Another Example

```
#include <iostream>
using namespace std;

int main()
{
    int num_vals, i;
    double next_val, sum = 0;

    cout << "Enter the number of values to follow: ";
    cin >> num_vals;
    cout << "Now enter " << num_vals << " values:" << endl;

    for ( i=0; i<=num_vals; i++ )
        cin >> next_val;
        sum = sum + next_val;

    cout << "The average is " << sum / num_vals << endl;
    return 0;
}
```



Steps (1)

- Run the program and find the following:
 - It asks for one too many numbers
 - The sum is wrong
- There must be something wrong with the loop, so put a breakpoint at the **for** loop
- Start with debugging and it will stop when it gets to the **for** loop for the first time
- Use the Step Over button to execute one statement at a time



Steps (2)

- Keep using Step Over, and you'll see that it never executes the **sum** statement
 - That means the sum statement is NOT in the **for** loop!
- So, stop debugging, then add the curly braces around the loop body
- Start it up again, and this time you'll see the **sum** update each loop iteration
- Now, count the number of times the loop executes and you'll see it is one too many, so start the loop at one instead of zero



Wrap Up

- Testing your code is vital, and it takes time to learn how to do it well
- Use simple testing and debugging techniques such as adding **cout** statements throughout your code
 - Of course, be sure to remove them once you have fixed any problems!
- Use debuggers, like the Visual Studio debugger, to trace through code in order to find errors
 - Set breakpoints near lines you want to check to skip ahead to those areas of the code

