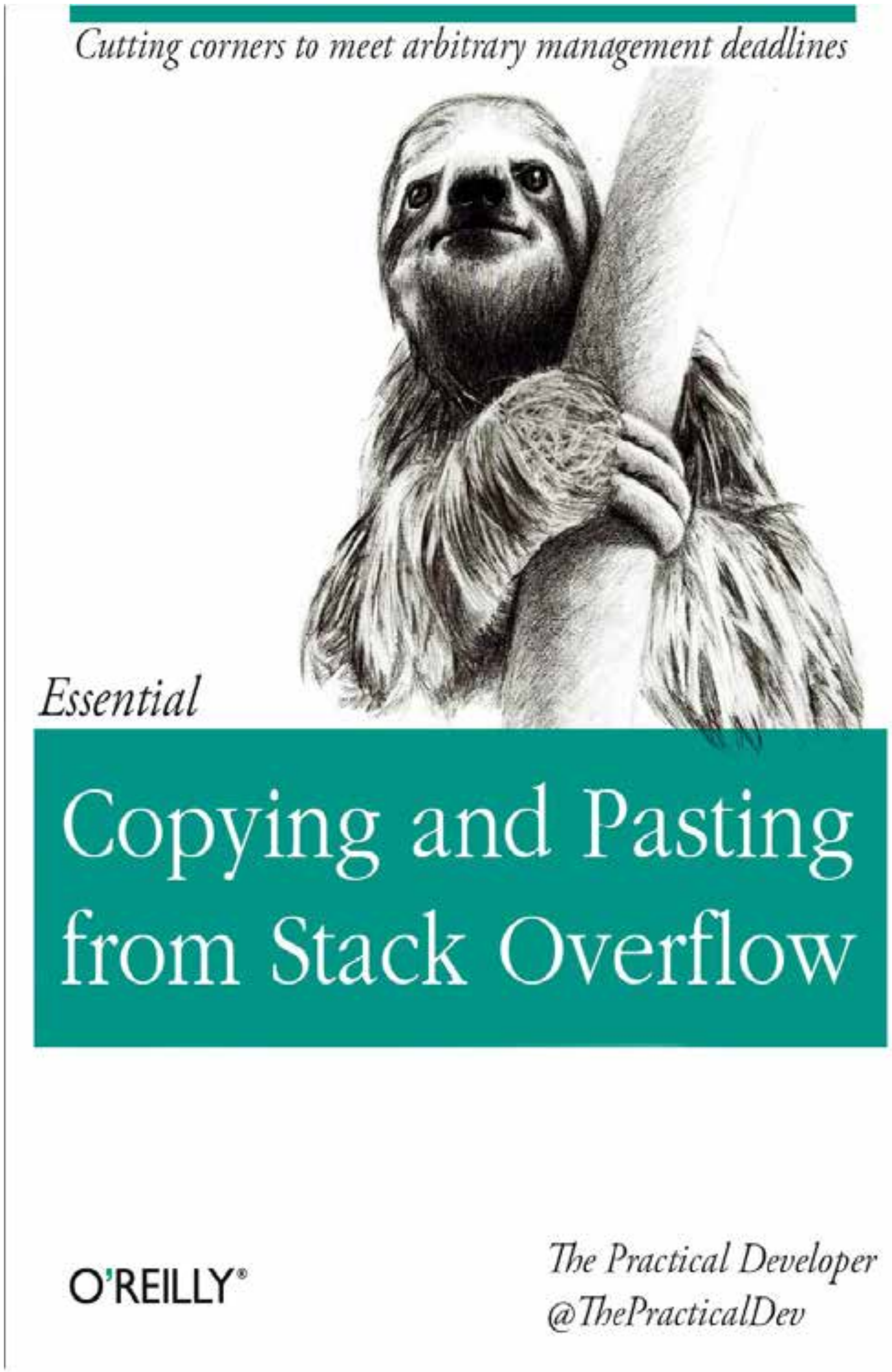
# Final Exam Review

**CS 16: Solving Problems with Computers I**
**Lecture #18**

Ziad Matni
Dept. of Computer Science, UCSB



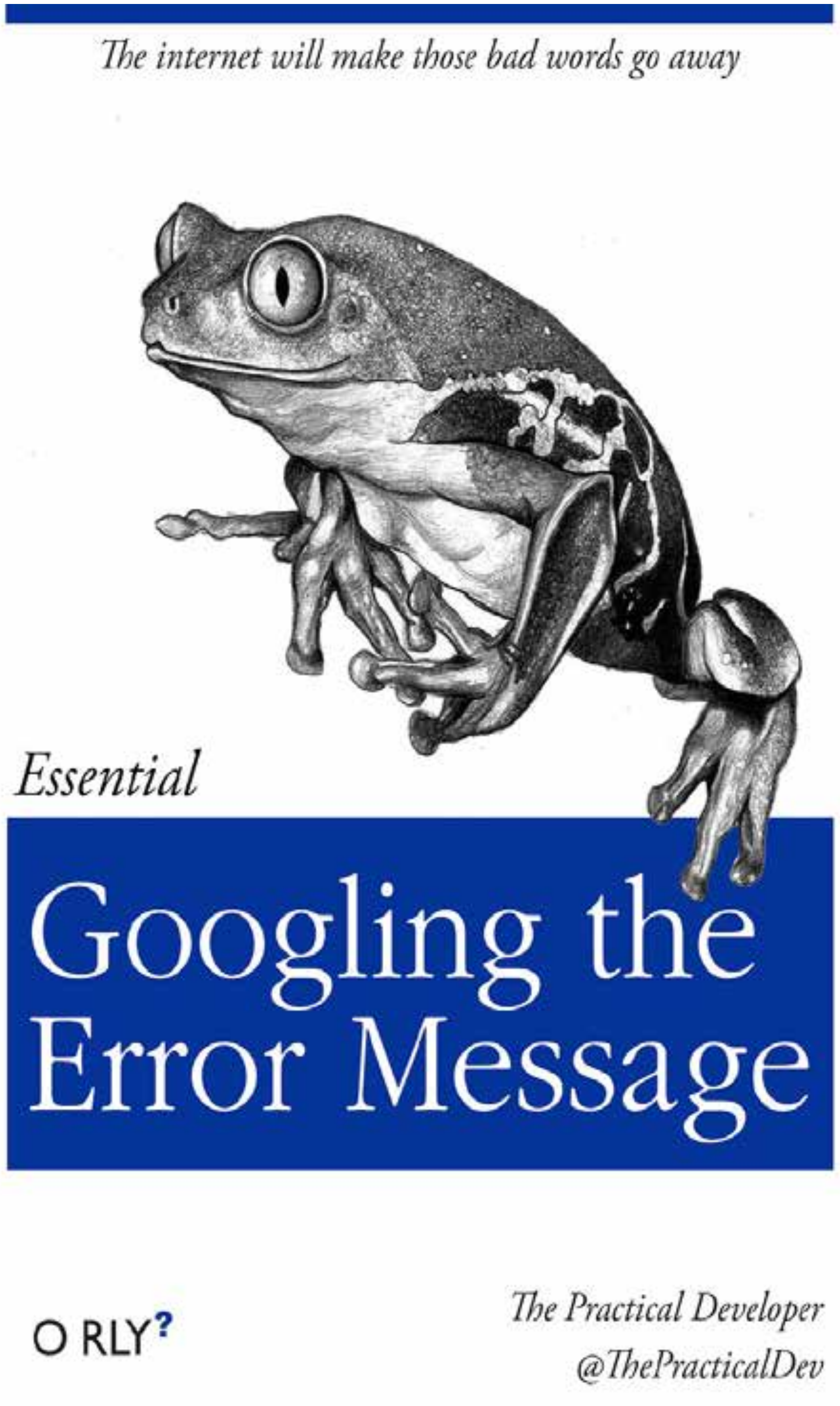
*Cutting corners to meet arbitrary management deadlines*

*Essential*

Copying and Pasting from Stack Overflow

O'REILLY®

*The Practical Developer*
*@ThePracticalDev*



*The internet will make those bad words go away*

*Essential*

Googling the Error Message

O RLY?

*The Practical Developer*
*@ThePracticalDev*

# FINAL IS COMING!

- Material: ***Everything*!**
- Homework, Labs, Lectures, Textbook
- **Monday, 6/12** in this classroom
- **Starts at 12:00pm **SHARP****
- **Seating will be assigned for you!**
  - ***BRING YOUR UCSB IDs PLEASE!***
    ***Arrive 10-15 minutes early***
- Duration: **3 hours long** (but really designed for 1.5 – 2 hours)
- Closed book: no calculators, no phones, no computers
- Only 1 sheet (***double***-sided is ok) of written notes
  - Must be no bigger than 8.5” x 11”
  - You have to turn it in with the exam
- **You will write your answers on the exam sheet itself.**

# Concepts You Will Have To Know
## *The Basics*

Lecture 2

- What does a CPU do?
- What does an OS do?
- What are compilers? Linkers?
- What's an algorithm and how is it different from a program?
- Variables and their operations in C++

# Concepts You Will Have To Know

## *Programming Basics*

Lectures 3-6

- **cin** and **cout**
- **if/else** statements
- Boolean operations and logic
- Rules and precedence of operations in C++
  - Including different ways to do increments

- Loops in C++
  - **while**, **do-while**, **for**
  - Controlling statements
  - Infinite loops
  - Multiway branches
- **switch/case** statements
- Global vs. local variables
- Type casting
- Random number generation

# Concepts You Will Have To Know

## *Functions*

Lectures 6, 7

- Function declaration
- Function definition
- Function calling
- Placing of all of these
- Return statements
- "Black Box" Abstraction
- Block scope of variables
- Overloading functions in C++
- **void** functions
- **main ( )** function in C++
- Call-by-value vs. Call-by-reference
- Functions calling functions
- How do we best design a program using functions?

# Concepts You Will Have To Know *Designing Loops and Debug*

Lecture 8, 9

- Designing loops
  - Exit on flag
- Debugging Loops/Functions (and programs in general)
  - Tracing using cout statements
- Testing Functions
  - Stubs, assert
  - Fundamental rules for testing functions
- Using good comments
  - Describing the Pre and Post conditions of a function

# Concepts You Will Have To Know
## *Number Conversions*

Lecture 9

- Positional Notation
- Binary to Hex
- Binary to Decimal
- Any-base to Decimal

# Concepts You Will Have To Know
## *I/O Streams and File I/O*

Lectures 9, 10

- Strings and C-Strings
- File I/O and Stream Variables
- **ifstream** and **ofstream** libraries
  - Variable/object declarations
  - Use of file names
  - Using **.open( )** and **.close( )** member functions
  - Use of the **>>** and **<<** operators
  - How to handle errors in File I/O: **fail( )** and **exit( )**
  - How to append data to an output file

# Concepts You Will Have To Know
## *More I/O Streams*

Lectures 10, 11

- Stream names as arguments in a function
- Detecting the end of an input file
  - Using **(in_stream.eof()**) vs. **(in_stream >> next)**
- Using **get()**, **getline()**, **put()**, **putback()**
- Formatting outputs
  - Using member functions like **.setf( )** and **.precision( )**
  - Using manipulators like **setw( )** and **setprecision( )**

# Concepts You Will Have To Know
## *Strings*

Lecture 11, 12

- Character functions
  - **toupper( ), tolower( ), isspace( ), isalpha( ), isdigit( )**
- Basics
  - The **+** , **+=** operators
  - The use of [ ] to look at one character in a string
- Built-in string manipulators
  - Search functions
    - **find**, **rfind**, **find_first_of**, **find_first_not_of**
  - Descriptor functions
    - **length**, **size**
  - Content changers
    - **substr**, **replace**, **append**, **insert**, **erase**

# Concepts You Will Have To Know

## *Combining Multiple Files* Lecture 13

- Why bother? (the 4 reasons)
- Compiling with g++
- Using make

# Concepts You Will Have To Know

## *Arrays*

Lectures 12, 13, 14

- Basics
  - What are arrays? What types can they be?
  - How do we declare them? Initialize them?
  - Indexing use and index vs. size
- Using arrays in loops
- Using arrays in functions
  - *Passing* an array
  - The **const** modifier
  - *Returning* an array

- How are arrays stored in computer memory?
- Partially-filled arrays
- Searching arrays
- Sorting arrays
- Multi-dimensional arrays

# Concepts You Will Have To Know
## *Vectors*

Lecture 14

- Basics
  - How to use them, initialize them
  - Accessing elements
- Using **push_back( )**
- Size of a vector
  - Using the **.size( )** member function
- Vector efficiency, capacity
  - And other advantages over arrays

# Concepts You Will Have To Know

## *Pointers*

Lecture 15, 16

- Basics
  - What are they? Why do we care?
  - How do we declare them? Initialize them?
- Use of the **&** and ***** operators
- The **new** and **delete** operators
- The freestore or heap
- Dangling pointers

- Automatic variables
- Using **typedef**
- Dynamic Arrays
  - Creating them and managing them
  - Multidimensional dynamic arrays
- Linked Lists
  - Definition

# Concepts You Will Have To Know *Structures and Linked Lists*

Lecture 15, 16

- Defining structures and classes
- Using structures
- Specifying member variables in structures
- Structures as arguments and return types
- Initializing structures

- Linked Lists
  - Implementing nodes and pointers
  - Heads and NULL (and nullptr)
  - The arrow operator
  - How do we delete and insert nodes in a linked list?

# Concepts You Will Have To Know

## *Recursive Functions*

Lecture 16

- Recursive functions
  - How to build them from a repeating series
- How to track them
- Ending recursive calls
  - The stopping case and why it's important
- Infinite recursion
- The "stack" concept and LIFO data structures
- Stack overflow
- Recursion vs. Loop Iteration
- Recursive functions that return something vs. void ones
- The 3 rules for thinking recursively & checking to see if it works
  - Check for infinite recursion; check stopping case; check all returned values

# Recursion in Poetry!

A child couldn't sleep, so her mother told a story about a little frog,<br>
who couldn't sleep, so the frog's mother told a story about a little bear,<br>
who couldn't sleep, so bear's mother told a story about a little weasel<br>
...who fell asleep.<br>
...and the little bear fell asleep;<br>
...and the little frog fell asleep;<br>
...and the child fell asleep.

# SAMPLE PROBLEMS

**What is the output of this C++ code?**

```
int x = 50;
while ( x > 1) {
    cout << x << “,” ; x /= 5;
}
```

50,10,2,

**What is the output of this C++ code?**

```
int x[3];
x[0] = 2;
x[x[0]] = 3;
x[x[2] - 2] = x[0] + x[2];
cout << x[0] << “, ”;
cout << x[1] << “, ”;
cout << x[2] << endl;
```

2, 5, 3

**What is the output of this C++ code?**

```
int y[2][4] = {{1,2,3,4}, {10,20,30,40}};
for (int n = 0; n < 1; n++)
    for (int m = 0; m < 2; m++)
        y[n][m] = y[n+1][m+1] + m;
cout << y[0][2];
```

3

**What is the output of this C++ code?**

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

int main () {
    int w = 0, v = 14, count = 0;
    string s="a";
    while ( (w < 10) && (v > 10) ) {
        s += s;
        for (int j = 0; j < 2; j++) {
            cout << "j = " << j << "; w = " << w << "; v = " << v << endl;
            count++;   }
        w += 5;
        v -= 3;        }
    cout << s << ++count << endl;
    return 0;
    }
```

```
j = 0; w = 0; v = 14
j = 1; w = 0; v = 14
j = 0; w = 5; v = 11
j = 1; w = 5; v = 11
aaaa5
```

**What is the output of this C++ code?**

```
vector<int> v;
v.push_back(5);
v.push_back(20);
v.push_back(v[0]*v[1]);
for (int k = v.size() - 1; k >= 0; k--)
    cout << v[k] << “,”;
cout << v.size() << endl;
```

**100,20,5,3**

**What would happen if I changed k-- to k++ in the for loop?**

**Show all the outputs of this C++ code:**

```
int *p1, *p2;
p1 = new int;
p2 = new int;

*p1 = 10;
*p2 = 20;
cout << *p1 << endl;    10
cout << *p2 << endl;    20

*p1 = *p2;
*p2 = 30;
cout << *p1 << endl;    20
cout << *p2 << endl;    30

p1 = p2;
cout << (*p1 + *p2) << endl;    60
```

Write a recursive function program to find the *n*th element in the following arithmetic numerical sequence: **3, 11, 27, 59, 123, …**

Hint: You first have to figure out what is the recursive pattern (try a linear combination, like **$a_n$ = C. $a_{n-1}$ + D** , where C and Dare constants). You also have to identify the base case. A correct example output would look like this:

*Which element of the sequence would you like to know?*
*4*
*Element number 4 in the sequence is 59.*

```
int main( ) {
    int number(0);
    cout << "Enter an integer number: ";
    cin >> number;
    cout << "Element #" << number << " is: "
        << formula(number) << endl;
    return 0;
}

int formula(int n) {
    if (n == 1) return 3;
    else return (2*formula(n - 1) + 5);
}
```

**WHAT IS THE SERIES DOING?**

$a_1 = 3$, $a_2 = 11$, $a_3 = 27$, $a_4 = 59$, etc…

Hint says to look for something in the form of: **$a_n$ = C. $a_{n-1}$ + D**

Note that if I make **C** = 2,
Then $a_2 = (2 \times 3) + \mathbf{5}$ and
Then $a_3 = (2 \times 11) + \mathbf{5}$, etc…

So **D** must be 5.

The recursive formula is thus:

$$\mathbf{a_n = 2\,a_{n-1} + 5}$$

**WHAT IS THE STOPPING CASE?**

**$a_1 = 3$**

</LECTURE>