Comp 170 Object Oriented Programming

A FIRST SERIOUS PROGRAMMING CLASS

DR. WILLIAM L HONIG

What will we do?

Learn Programming

Variables / Types: store information; create knowledge
Control Structures: what happens when; how to get job done
Object Oriented Programming: the most important kind of development (and why)

How?

Learn concepts: Have a tool box of approaches; pick and choose
Write code: Picky rules of syntax; know what to do when
Read code: Learn how others / experts do things
Work in teams: Share the learning

A lot of work! Really a lot of work!!

Many have trouble@#?

I can help you get it – do what I say!

Other Programming Courses….
maybe right for you? Let's talk….

Comp 125 — Core

Comp 150 — Core

Comp 180 — (for scientists)

Comp 170 — Starts the major

Some programming experience (any language)
Know variable, if statement, function / subroutine
…or good (very good) in math, puzzles, logic

New Graduate Students:
Combined 170/271
one semester

Comp 271

Can program in Java, C++, C# well
…incl create classes that interact

“Programming is a Contact Sport”

W L Honig August 2016

con·tact sport

NOUN
a sport in which the participants necessarily come into bodily contact with one another.

To learn to program:

Your eyes, hands, fingers, brain, and whole body need to be in regular and close contact with programming.

Anyone here suffer from Ergophobia?

ergophobia

NOUN
Ergophobia: An abnormal and persistent fear of work. Sufferers of ergophobia experience undue anxiety about the workplace environment even though they realize their fear is irrational. Their fear may actually be a combination of fears, such fear of failing at assigned tasks, fear of speaking before groups at work, or fear of socializing with co-workers.
"Ergophobia" is derived from the Greek "ergon" (work) and "phobos" (fear).

Let’s work on this together!

Classroom is a good place to get over it

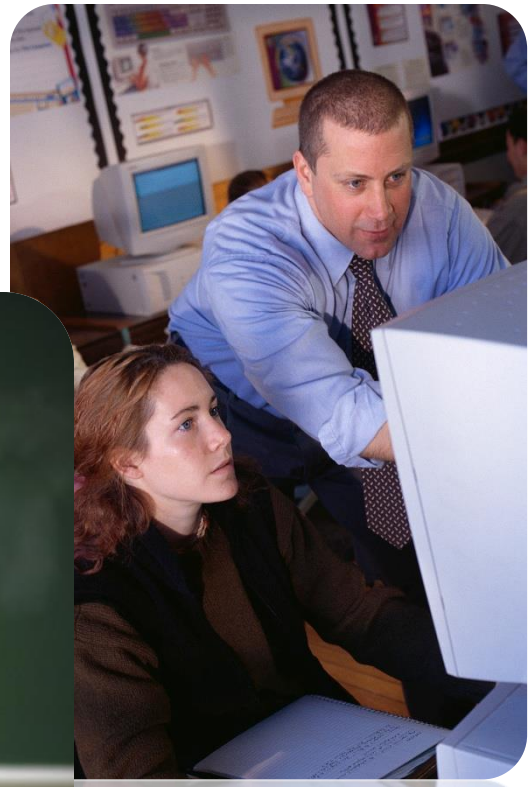
Applying Programming Principles …

Using reading…

Thinking…

Doing…

Getting help…



Germane (effective) cognitive load

LOYOLA UNIVERSITY CHICAGO

Preparing people to lead extraordinary lives

Class may not be what you expect…

Active learning, hand-on engagement, flipped classroom,..

To succeed in this class you must:

1. Prepare outside class
2. Participate fully in class work
   - Attendance required
3. Use online resources outside of class
   - Discussion board
   - MPL lab exercises
   - Programming

To begin this class you need:

1. Some exposure to programming
2. Good math knowledge (love of solving problems) e.g. Sudoku anyone?
3. Concerns? Not sure: let's talk now!

Introduction to Object Oriented Programming

LOYOLA UNIVERSITY CHICAGO

- Thinking as a Programmer
  - Concepts
  - Design Choices
  - Think before Do
- Programming in Java
  - Syntax – very carefully
  - Major, Serious, Programming Language
  - Write small and medium sized programs

Two Level Thinking for Programmers

CONCEPTS

Variables

Boolean Logic

Statements and Expressions

Types and Declarations (and Type Checking)

Control Flow; Control Structures (Conditionals, Loops)

Functions / Methods / Procedures

Class and Object (separate and related)

Memory Allocation and Usage

DETAILS

All those funny symbols

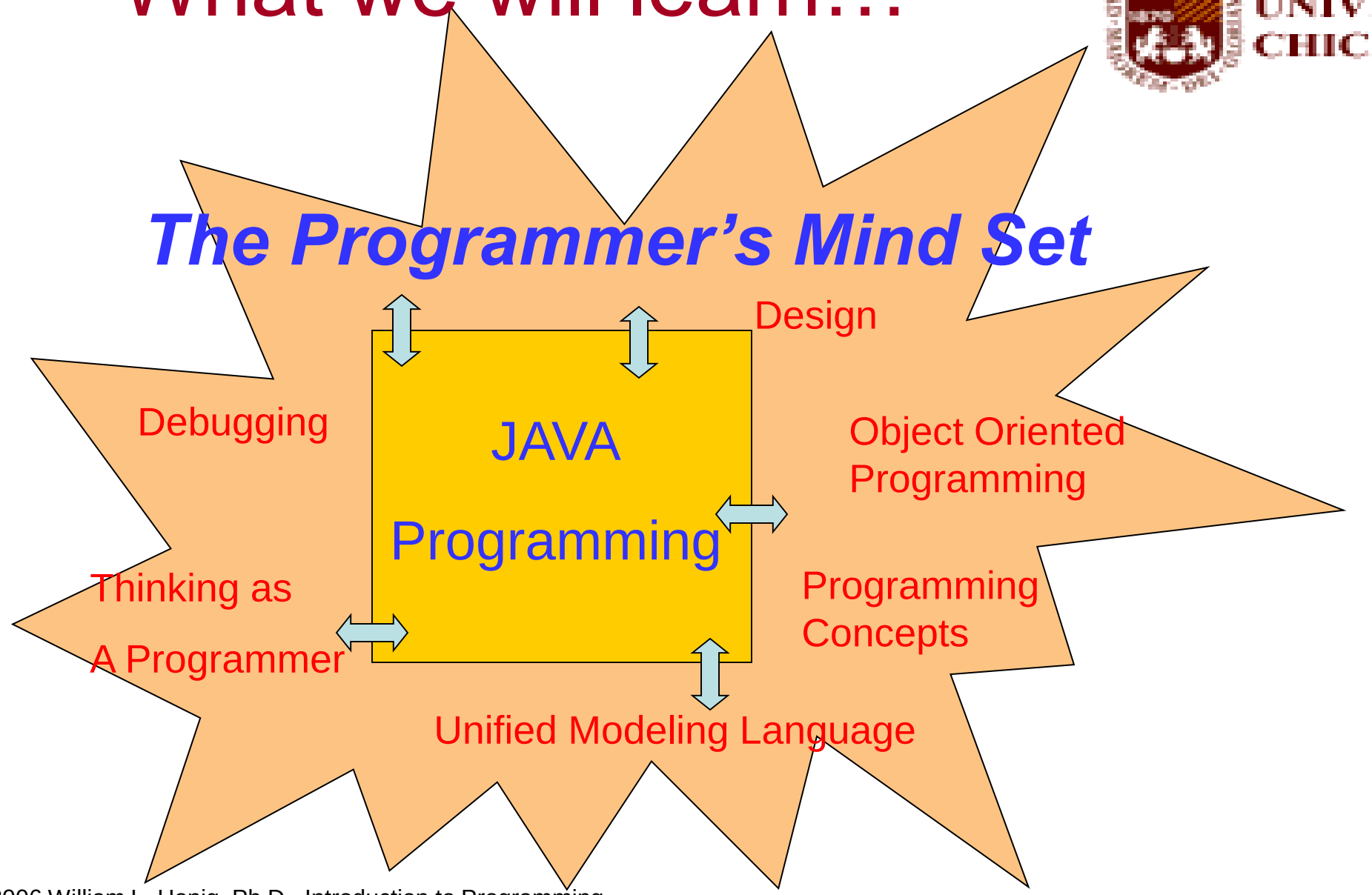
- { } ( ) [ ] , ;
- + - / * %
- || &&

All the gibberish

- void
- public, private
- If while do return break

What we will learn…

LOYOLA UNIVERSITY CHICAGO



A Lot of Things We Won’t Do (at first)

1. Use GUI such as Windows
2. Do OOP, except in passing
3. Write large programs

So programs may be a bit BORING, for a while

The Harkness Method
…old but new…

- Using discussion, Harkness table, all contribute, learn with others.
- Students make meaning of new information by talking, listening, and thinking together.
- Collaborative learning, student lead (teacher as guide)

Originated Phillips Exeter Academy, 1930, Edward Harkness

http://www.exeter.edu/exeter-difference/how-youll-learn

LOYOLA UNIVERSITY CHICAGO

Preparing people to lead extraordinary lives

Rules….Expectations…Fairness see details in syllabus

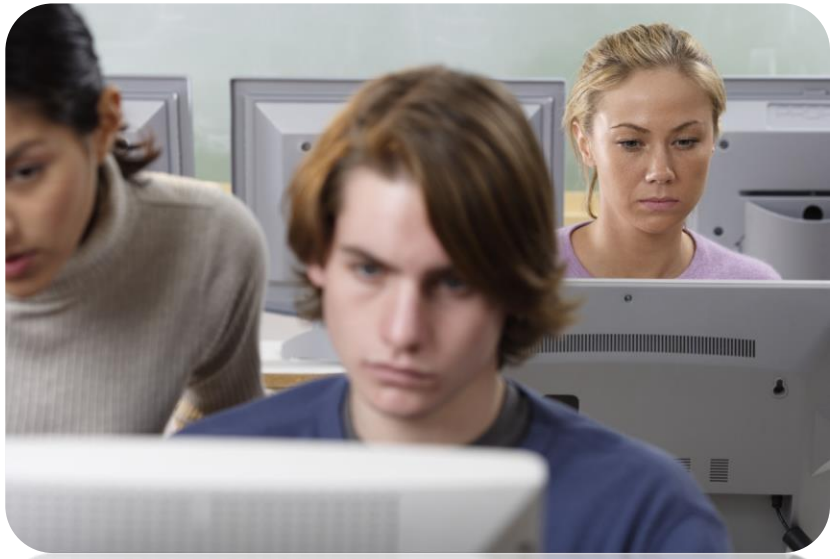
Required	Examples	How?
Preparation and Participation	1. *Come to class ready to work, discuss, question, explore* 2. *BRING KNOWLEDGE with you* 3. *Work outside class too* 4. *Attend all classes, full time*	Read Text + Forum Lab (3xclass time)
Assignments and Communication	1. *Correct assignment, correct files, right format, finished work* 2. *On time, ahead of time* 3. *Late Pass only exception (2 only)* 4. *Email is last choice for communication (last century tool, from '79 and before)*	Plan Ahead Use Schedule Checklist Accept consequences
Responsibility & Accountability Avoid needing excuses	*Don't even ask:* 1. *My internet was down, can I still turn in?* 2. *I forgot, sorry, can I make it up?* 3. *I was not in class, did I miss anything?* 4. *Can I make up something I did wrong?*	Don't Ask No No No No

Why? Programming is all about Precision and Correctness

LOYOLA UNIVERSITY CHICAGO 1870 AD · MAJOREM · DEI · GLORIAM

Preparing people to lead extraordinary lives

Learning to Program Properly…



Programming is about Precision and Quality / Correctness

Hacker Mindset:

- Program, program, program
- Work on whatever I know how to do first
- Big, complex programs
- Continue until get ok outputs for some inputs, or until time runs out
- Always about 80% done

Computer Scientist Mentality:

- Plan, consider alternatives
- What language features and concepts may help
- Coding is the last and easiest part of the work
- Design (pseudo code) first, then program
- Code that is small, well structured, easy to read
- Code that is obviously complete and correct
  - BEAUTIFUL CODE!

You CAN learn the difference by the end of Comp 170

If you DON'T may never get it

Trust me, I am one of the best!

LOYOLA UNIVERSITY CHICAGO

Preparing people to lead extraordinary lives

Assignment Submit Process

LOYOLA UNIVERSITY CHICAGO 1870 AD · MAJOREM · DEI · GLORIAM

Preparing people to lead extraordinary lives

...how to have high quality result

1. Select the correct Sakai Assignment
2. Complete your work, check it twice
3. Save your work; create files with correct file types and file names; you can save multiple times, replace old files, etc.
   - Examples A4mylucidHolyDigits.zip, Quiz2mylucid.doc
   - Files types may be .doc, .docx, .pdf
   - Java code in .java files or .zip of a whole project
   - You can submit multiple files; only .zip if indicated in assignment
   - No .pages files; no spaces in file names; must have file extension

4. Start early, at least 15minutes before due time
5. After you think you are done, close the assignment window, reopen it and download your files to check they are correct
6. Submit the assignment
7. Exit the assignment and come back in to be sure it shows submitted!

Etiquette

It Should Not Be Necessary To Say …

- No, you cannot listen to music during class
- No, you cannot use your phone during class (quiet mode please)
- No, you cannot get up and leave whenever you wish
- No, don't chat with your neighbor (except during work time)
- No, don't ask for extra credit at the end

LOYOLA UNIVERSITY CHICAGO

Preparing people to lead extraordinary lives

Tools = A Programmer Lives and Breathes

https://www.eclipse.org/home/index.php

An Open Source Project

Eclipse Integrated Development Environment (IDE)

IDE =

- Editor (enter and change text) with knowledge of language
- Compiler (check program for grammar and spelling rules)
- Execution Environment (run the program)

A real world tool, a bit complex…

Supports many languages and computers (e.g. Android development)

Tools = A Programmer Lives and Breathes

MyProgrammingLab™

http://www.turingscraft.com/

CodeLab a tool for learning programming

http://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/

MyProgrammingLab is bundled with online text (Peason packaged text with CodeLab)

Hands on, try out, guess answers, see solutions

Instant answers to program fragments

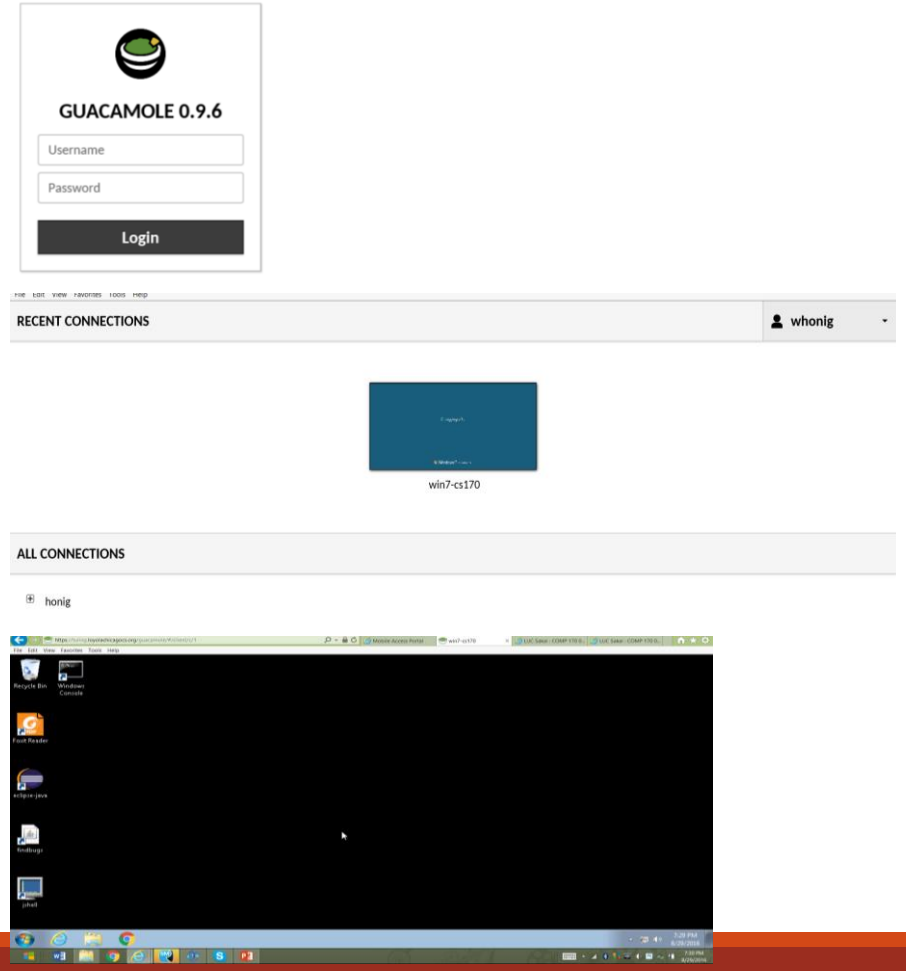
- See others solutions
- Easy to try, guess, experiment
- Easier than Eclipse (needs a full program)

Will use for some course labs (points)

One place for you to look while reading text or studing a concept

- Try, Try, Try
- Try again
- Keep trying…

Tools = A Programmer Lives and Breathes



Virtual Machine

Your own cloud computer

- You are the administrator
- Can add things, install,…
- DO NOT remove things that are there (you need them for class)

A sign of the future

- Pervasive Computing
- Mobile devices
- Internet of Things
- …

My Program Won't Work !?#$?

Three distinct kinds of errors while programming:

1. Syntax Error (Compile Error) Error messages in Eclipse; May give exact information or may be misleading, confusing. Often only the first one counts

   Approach: Compile often (at least every method); fix the first one; look above the error message too!

   Have some templates (main, stub of a function with parameters and that returns values, etc)

2. Runtime Error (Bomb, Exception,…)

   Approach: What input caused the error? Does it always happen, or only on some inputs?
   Should the program check to prevent bad inputs?
   Add debugging statements to see how far the program ran before the error

3. Logic Error (Runs and does the wrong thing)

   Approach: Play computer and follow through the program. Where did it go wrong (parameters, logic, ….)

KEY Ability to learn = What Kind of Error and How To Fix It!

Why Java??

1. Most widely used programming langauge
2. In the main serious programming family (C, C++, C#,…)
3. Java and the Web arose together
4. Java programs (largely) computer, machine, operating system independent

What we learn will apply to Java, C++, Javascript many other languages

Some Links to Java Documentation of Interest

1. Java 8 SE Reference for Developers

https://docs.oracle.com/javase/8/docs/api/

2. Java Tutorials

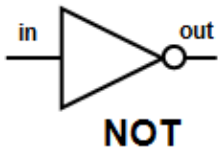
http://www.oracle.com/technetwork/topics/newtojava/overview/index.html

3. Java keywords (don't use for your names)

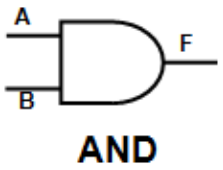
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Good to be able to see the reference to augment the text!
Real Programmers Use This all the TIME!

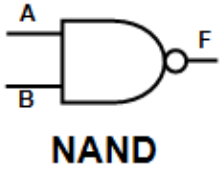
Truth Tables = Basis for EVERYTHING



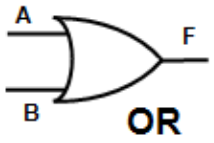
Input	Output
I	F
0	1
1	0



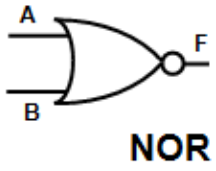
Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1



Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0



Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1



Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0

Why George Boole is famous

True / False On / Off 1 / 0 Hi / Lo

Any calculation can be done with And, Or, Not

Any calculation can be done with only NAND

Transistors are either On or Off

Computers use lots of transistors (to build logic gates)

Conjunctive Disjunctive Normal Form

ANY TRUTH TABLE CAN BE TURNED INTO A LOGIC FORMULA

FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

CDNF - I CALL IT AND OR NORMAL FORM

Mechanical way to convert any truth table into a logic formula using AND, OR and NOT

Conjunction = AND Disjunction = OR

How to do it: and together each row that has a one in the result. If the input is True , use it by itself, if the input is False, put a NOT in front. OR them all together. You look only at the rows of the truth table with a 1 in the result. Ignore those with 0 in the result

Example: (A and NOT B and C) or (NOT A and B and NOT C) or (NOT A and B and C)

How Many Truth Tables Are There with 2 Inputs?

GIVEN EVERY POSSIBLE COMBINATION OF INPUTS

Draw a truth table with two inputs A and B

Add every possible output you can think of…..

(a bit like program testing, thinking of all the possible values a variable or a return value can take)

HOW MANY CAN YOU WRITE IN C#?

Try it:

- Given variables a and b
- (What would their type be???)
- Make each one a function, give it a name
- (Return the result, what type???)

Why worry about programming style?

MOTIVATION

Programming languages are big, flexible
- Many options; alternative ways to express concepts
- NOT very prescriptive on appearance of program

Without direction, programmers deal with this complexity on their own
- Programmers develop own personal style
- Often ad hoc and haphazard
- Changes over time as programmer "learns"
- Trial and error process
  - Opportunity to speed up this learning

Programming is seldom a One Person Task
- Sooner or later, other's must read, use, review your code
- Team programming, software reuse, grading….
  - A shared or common style eases the next person's task

WHAT DOES IT DO?

Standard Rules for:
- identifiers
- indenting
- Naming different things to look differently
- Ways to write control structures

Goal: Familiar, easy to read, code that avoids common errors

Use Good Mnemonic Names (1 of 2)

Use names that imply their role in the program

- Use names that are reasonably long and descriptive
- Better than adding separate comments to the code!
- Identical in code size, run time, …
- It takes time to devise and make use of informative names
  - Aids readability (self and others)

```
x1=x2*x3+x4*x5;          grossPay = (wage * hours) + (overtimeWage * extraHours);
x6=x7*x1;                tax = taxRate * grossPay;
x=x1-x6;                 netPay = grossPay - tax;
```

Use Good Mnemonic Names (2 of 2)

Use style that helps to identify What is What

- No real cost if done consistently and from the beginning of development
  - Prevents compile errors
  - Reduces need to “look things up”

```
date my_birthday;
if (my_birthday.year() > too_old_year)
   cout << “Too old for this event”;
```

```
Date myBirthday;
if (myBirthday.Year() > TOO_OLD_YEAR)
   cout << “Too old for this event”;
```

Why impose a style guide?
Three Reasons

1. Help programmer avoid common errors
   - Common syntax or semantic errors
   - Experience shows programmers often make this error
2. Make programs more readable or reusable
   - All programmers to do things in similar ways
   - Standard ways to find things in a program
   - Helps original programmer and later “users”
3. Limit flexibility or options to a few well proven ones
   - Select among many ways to do the same thing
   - Experience in real world shows the best approach

Examples
Avoid errors and common problems

A1.6.1—The `if` Statement

Avoid the "`if` ... `if` ... `else`" trap. The code

```
if ( ... )
   if ( ... ) ...;
else ...;
```

will not do what the indentation level suggests, and it can take hours to find such a bug. Always use an extra pair of `{ ... }` when dealing with "`if` ... `if` ... `else`":

```
if ( ... )
{
   if ( ... ) ...;
} // {...} are necessary
else ...;

if ( ... )
{
   if ( ... ) ...;
   else ...;
} // {...} not necessary, but they keep you out of trouble
```

Studies have shown that the most acceptable forms of for loops (to avoid off-by-one errors) are these:

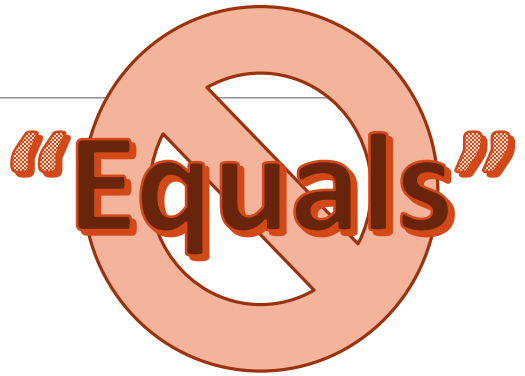
```
for ( i = 0; i <= max_index; i++ )

for ( i = 0; i < sizeof(array); i++ )

for ( i = max_index; i >= 0; i-- )
```

You need a really good reason to write anything different!

Please, avoid the “E word”
Three Things to Say Instead



1. Assignment Statement is “BECOMES”
   - myName = “Joe” + “ “ + “Jones”;
   - (expression value on the RHS stored into variable on the LHS)
2. Testing NUMBERS for the same value is “IS THE SAME AS”
   - myScore == 100;
   - Never test double or float variables for equality this way!
   - (Works for all primitive types)
   - And the related comparison operators are >= <= > < !=
3. Testing OBJECTS for the same value is “DOT EQUALS”
   - someStudent.equals(anotherStudent)
   - (Correct for all reference types)
   - Often also someName.compareTo(anotherName) to get ordering (< == >)
   - Your own class definition may need toString(), equals(), compareTo(), hashCode()

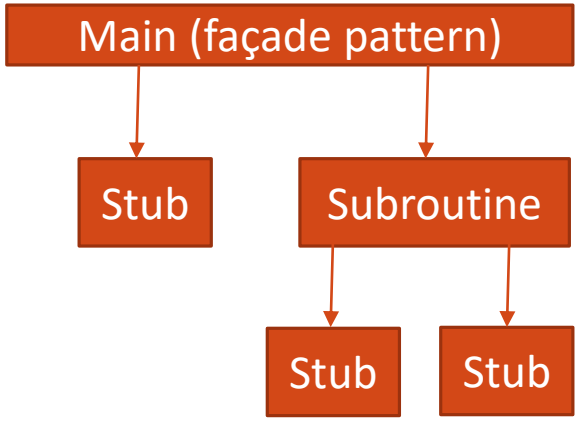
Top – Down Development

PROBLEM APPROACH: MAKE IT EASY TO SOLVE THE WHOLE THING AT ONCE!

System Thinking:

1. What is everything I need to do?
2. Divide into key chunks
   Just name the “hard” parts, e.g. “reverse the string”
3. Order the chunks (full program)
   Create the proper control flow for the whole app
4. Create stubs for the underlying parts
   They can do stupid things, e.g. always return 1
5. Order the chunks
   Create the proper control flow for the whole app

STEPWISE REFINEMENT



Get the whole program to run; keep it running!

Then refine stubs into more complete code

(any number of levels work)

Eventually solve the hard chunks when they are self contained

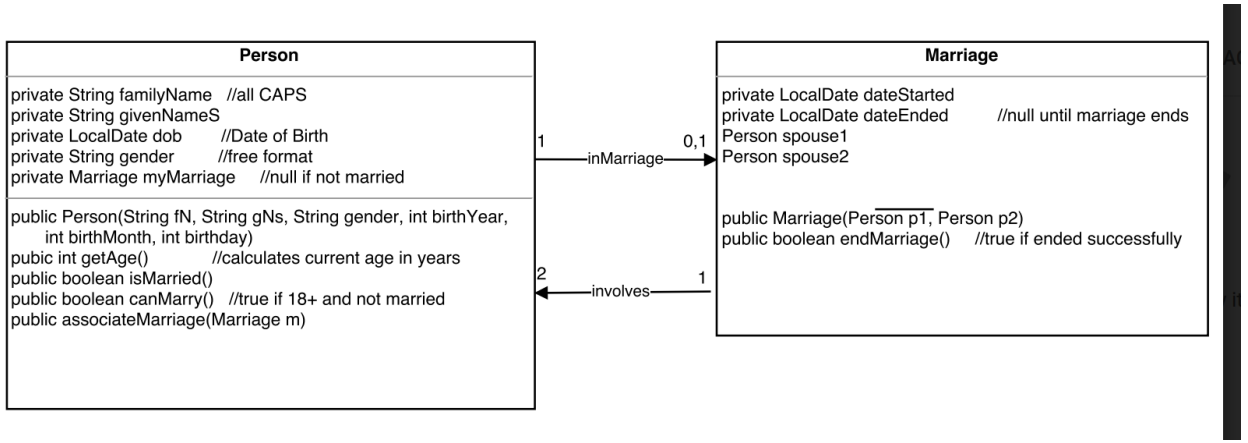
OOP Problem Solving

PROBLEM APPROACH: UNDERSTAND THE CLASSES AND RELATIONSHIPS FIRST

THINK before PROGRAMMING:

1. What classes do I need? Think about STATE
2. Find some key methods. Write signatures
   Not only get and set….
3. What's hard / complex / interesting to do?
   How do objects need to interact?
4. Test driven development (Advanced)
   Details of a few test cases and expected results
5. Done? Can you see how it all works?
   Recycle through the above, again, and again, and…
6. Code
   Recycle through the above, again, and again, and…

UML IS A TOOL FOR UNDERSTANDING



See the footsteps for the whole thing you need to do!

No mysteries; no magic

Remember: state is key (how to set it up at start, how to maintain it correctly, when it changes)

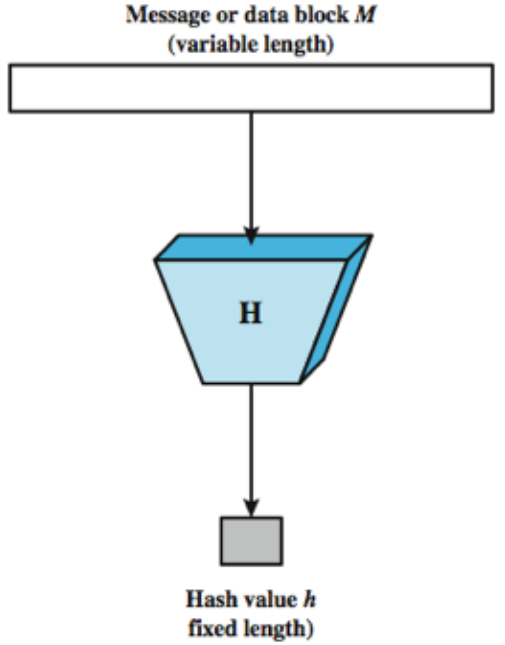
Hash Functions – Quickly…

A KEY PART OF SO MANY PARTS OF COMPUTING

Hash function:

1. Linear Algebra – a many-to-one function, may be onto; usually Many >> One.
2. Map or translate a large number of possible inputs to a smaller number of possible answers. A function that “compresses”
3. “Collisions” occur when more than one input maps to the same result. (All solutions deal with this)
4. E.g. SHA1 maps strings of characters to a 160bit number.

EASY TO DO (BADLY); HARD TO DO WELL



Simple example is modulo arithmetic

The 170 Top Ten

Maybe not everything you need to know to ace 170, but key things you **must** know for success.

We will come back often to this list.

Keep your own checklist of how you are doing!

Deprecated, update coming soon. You can still use for a general idea of key things to know!

170	Key Concept; Learning Outcomes; Critical Knowledge
1	**Be able to distinguish variable / type and class / instance**
2	**Know syntax rules and style with careful precision**
3	**Use conditional control structures fully and correctly**
4	**Construct loops with control structures**
5	**Use the primitive types and operators appropriately**
6	**Understand and create functions and methods**
7	**Learn how to create OO Classes**
8	**Understand the OO interface concept and usage**
9	**Use arrays and Collections to organize information**
10	**Perform simple Input / Output processing**

How can I help?

Comp 170 is a hard course, but there will be lots of help!

You don’t know how to do something until you can teach it!