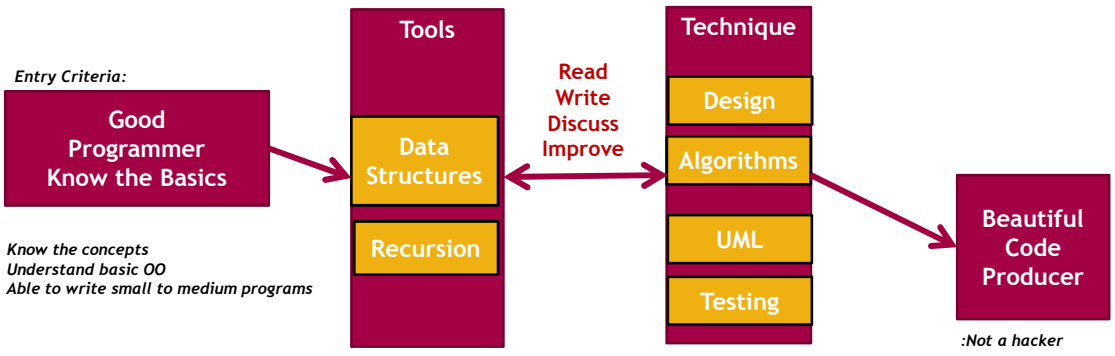
Comp 271
Data Structures

LEARN TO WRITE BEAUTIFUL CODE

DR. WILLIAM L HONIG

Course Summary
Key Things and Process



Which do you want to be:
Hacker: Write lots of code, get it to run, test it until you get reasonable outputs for some inputs.
Alternative: Write concise correct code, constructed so that it is likely correct, uses best practices and looks good.

Applying Programming Principles …

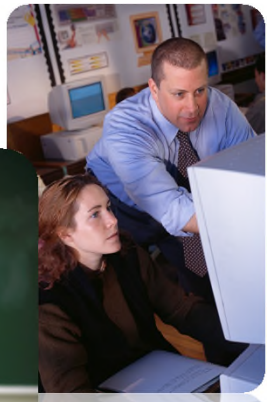
Using reading…

Thinking…

Doing…

Getting help…

Germane (effective) cognitive load



Preparing people to lead extraordinary lives

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

Preparing people to lead extraordinary lives

Class may not be what you expect…

Active learning, hands-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
   - Programming
4. Bring ideas, questions, mysteries to class

To begin this class you need:

1. Adequate preparation
2. Willingness to work hard
3. Concerns? Not sure: let's talk now!

Learning to Program Properly…

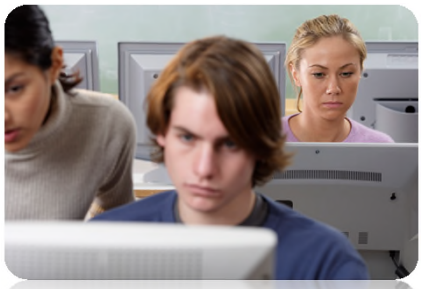
Programming is about Precision and Quality / Correctness

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 271

If you DON'T may never get it

Trust me, I am one of the best!

LOYOLA UNIVERSITY CHICAGO 1870 LUD · MAJOREM · DEI · GLORIAM

Preparing people to lead extraordinary lives

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

"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.

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

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

January 13, 2015

LOYOLA UNIVERSITY CHICAGO
Preparing people to lead extraordinary lives

Tools = A Programmer Lives and Breathes

eclipse

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

An Open Source Project

Eclipse Integrated Development Environment (IDE)

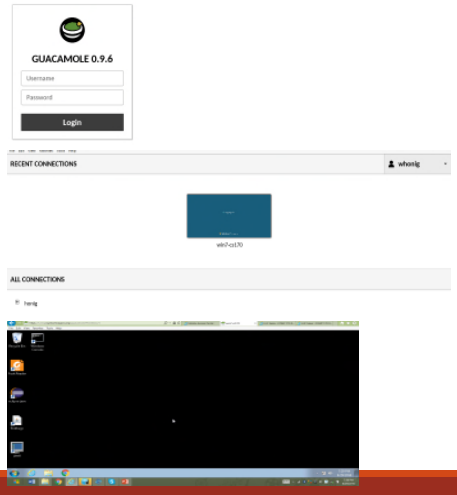
IDE =

- 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



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
- …

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

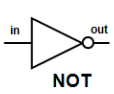
http://www.oracle.com/technetwork/topics/newtojava/overview/index.html

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

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

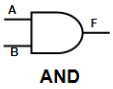
Good to be able to see the reference to augment the text!
Real Programmers Use This all the TIME!

Truth Tables = Basis for EVERYTHING



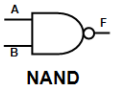
NOT

Input	Output
I	F
0	1
1	0



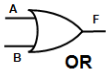
AND

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



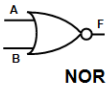
NAND

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



OR

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



NOR

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!

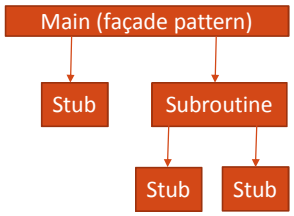
Top – Down Development

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

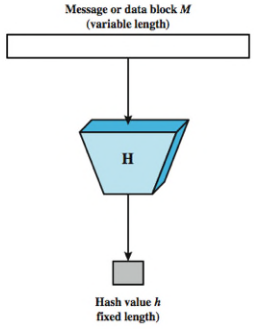
Hash Functions – Quickly…

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

Big O Notation – Quickly…

STANDARD MEASURE OF ALGORITHM EFFICIENCY OR COMPLEXITY

1. Time Complexity – how long it takes to run
   Count basic operations
   Ignore initializations, and other minor stuff
2. Typical Time Complexity Growth Rate Relationships (n>10)
   $1 < \log(\log n) < \log n < \log^2 n < n < n^2 < n^3 < 2^n < n!$
3. Also: Space Complexity – how much memory it needs to run

BIG O = "ORDER OF AT MOST … "

1. Function f(n) is of order at most g(n):

   A positive real number c and positive integer N exist such that f(n) <= c*g(n) for n>=N
2. Complexity of control structures:
   O(sequence S1; S2; S3…) = maximum of O(Si)
   O(if then else) = O(condition) + max of O(then) O(else)
   O(loop) = times it runs * O(body)
3. Math helps:
   Sum of 0 or 1 to n = n * (n+1) / n
   Sum of 0 or 1 to n-1 = n * (n-1) / n

Heuristic: look at loops; highest order in equation

Exceptions, Brief Style
Use exceptions for unexpected events

STANDARD PATTERN FOR RUNNING CODE THAT MAY CAUSE AND EXCEPTION

1. Use try { }; catch { }; catch { }; finally { };
2. Required to catch checked exceptions. Correct the problem or add information and re throw
3. Generally, detect problems and throw close to the source; catch and correct as far up as possible (more information)

YOUR OWN CHECKED EXCEPTIONS

1. Usually extend Exception
2. throws in the method header in which it may arise

```
public void openFile(){
        FileReader reader = null;
        try {
            reader = new FileReader("someFile");
            int i=0;
            while(i != -1){
                i = reader.read();
                System.out.println((char) i );
            }
        } catch (IOException e) {
            //do something clever with the exception
        } finally {
            if(reader != null){
                try {
                    reader.close();
                } catch (IOException e) {
                    //do something clever with the exception
                }
            }
            System.out.println("--- File End ---");
        }
    }
```

Preconditions; Postconditions
Programming by Contract; Design by Contract

Making assumptions and commitments explicit. What can code count on?

Precondition: A condition which must be true in order for the code to begin and produce desired results.

Use present tense and passive voice (describe states and relationships that are true at the moment)

"courseSection is offering of courseTitle in current semester"

Precondition: A condition which must be true after the code has run to completion. It describes what changes should occur in the system during the execution of the code.

Use past tense and passive voice (emphasize the results, not how they were achieved)

"roomNumber and meetingTime assigned to courseSection"

Design by Contract (Bertrand Meyer)

Contract: description of a behavior system component commits itself to carry out. Emphasis on what will be accomplished (not how).

Preconditions; Postconditions
Example and How To's

Coding practices:

At entry: Check anything that your code relies on to run correctly if it NOT given as a precondition. You may, but need not, check preconditions.

Your code decides what to do with any problems above: throw exception, return error indication, return default values,…

At exit: if you return, you are guaranteeing the postcondition holds. Check with an assert if you wish. You CANNOT return otherwise (must throw exception).

Advanced topic: This all applies to exception catching and handling too! (e.g. how you cause, catch, re-throw exceptions (or not)).

```
/**Method to return an integer data value between two specified end points (inclusive)
 *
 * @param someNumbers Any collection of suitable numbers (satisfying List interface)
 * @param minN Smallest value in desired range
 * @param maxN Largest value in desired range
 * @return the first data value that is in the range (inclusive)
 *
 * @pre someNumbers exists, but may be empty
 * @post minN <= returned value <= maxN
 * @throws IllegalArgumentException
 */
public static int findInt(List<Object> someNumbers, int minN, int maxN)
```