Computing and Programming

Notes for CSC 100 - The Beauty and Joy of Computing
The University of North Carolina at Greensboro

Reminders: What you should be doing!

Before Lab on Friday:

- Review handout on pair programming tips (will get today)
- Watch video on pair programming (link on class web site)
- Read (and think about!) Pre-Lab reading for Lab 2
  - *Warning*: Parts of this lab are tricky - be prepared before lab!

Remember: No class on Monday (Labor Day)

Before ***Wednesday at 10:00am***:

Participate in on-line discussion of Blown-to-Bits, Chapter 1

Question to Start the Day...

Context: Computation takes place as a sequence of steps (can consider a 2 GHz computer to be taking 2 billion steps per second).

Question: What can a computer do in a single step?

Or… Which of the following can be done in a step?

1. Store the number 123 in a variable
2. Compute the sum of a variable plus 5
3. Compute the square root of a number
4. Compute the GCD of two numbers
5. Add together two 1000-digit numbers
6. Find the highest test score in a class of 10 students
7. Draw Alonzo on the screen
8. Find a collection of bank transactions that add up to a target value

A Basic Computational Step

Super-simplified CPU (Central Processing Unit):

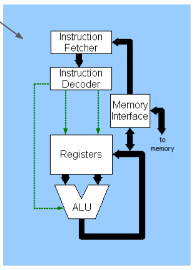


Diagram from Wikimedia Commons

Things to know:

- Data transferred around on wires (dark black arrows in diagram) - fixed number of wires limits how big (# digits) a number can be
- "ALU" is "Arithmetic Logic Unit" - this actually does the computations: two inputs, one output

So in general:
A *step* is an operation on two values from a limited range of numbers.

Can't operate on big numbers or more than two things in a single step.

Exercise: You identify the steps

Problem: Find the largest value in this list of numbers

338	773	655	72	882	80	533	278	626	689
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

Numbers in [brackets] are positions - use to refer to values (example: value [4])

What you're allowed to do:

- Set aside storage to remember values or positions (these are variables)
- Access a value based on its position
- Do any operation with one or two values (arithmetic, store in a variable, …)
- Make decisions based on comparing two values
- Control your "program" (repeating operations, etc.)

To do:

1. Decide on a general strategy (algorithm)
2. Go around the room executing: each person to identify the next "step" to take

A few more words about "steps"

Steps are very simplistic and not very powerful

But… do a few billion every second, and you can make some very complex, "smart-looking" behavior!

Note: While it works for computers, doing lots of stupid things at high speed is not the "smart-looking" behavior that you should aspire to as a college student.

If we had to specify every single step, programmers would not be very productive.

Goal: Make programming easier!

How Programming Languages Help

<u>How you want to describe an animation</u>

Move Alonzo 10 spaces to the right

<u>How the computer must do it</u>

```
Find what images are behind Alonzo
For each row of the background
    For each column of the background
        Redraw the background pixel
Add 10 to Alonzo's x coordinate
For each row of Alonzo
    For each column of Alonzo
        Draw the pixel of Alonzo
```

BYOB lets you do this

instead of this

["High Level" vs "Low Level"]

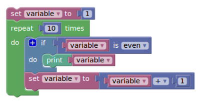
The goals of a good programming language:
a) is natural and understandable to people
b) can be efficiently (and unambiguously) turned into actual computer code

Different Languages, Different Styles

Basic structure of many languages is similar

Google Blockly lets you look at the *<u>same program</u>* in different languages:

As Blocks



In Python

```
variable = 1
for count in range(10):
  if variable % 2 == 0:
    print(variable)
  variable = variable + 1
```

In JavaScript

```
var variable;


variable = 1;
for (var count = 0; count < 10; count++) {
  if (variable % 2 == 0) {
    window.alert(variable);
  }
  variable = variable + 1;
}
```

<u>Note</u>: Don't get the wrong idea - some languages are *<u>very</u>* different!

See http://blockly-demo.appspot.com/static/apps/code/index.html

Some Quotes

These are by Alan Perlis (1922-1990): An early computer scientist and first winner of the Turing Award in 1966.

"A programming language is low level when its programs require attention to the irrelevant."

"A language that doesn't affect the way you think about programming, is not worth knowing."

"There will always be things we wish to say in our programs that in all known languages can only be said poorly."

And since programmers are never satisfied with the programming language they are using, one more quote - from Bjarne Stroustrup (inventor of C++):

"There are only two kinds of programming languages: those people always bitch about and those nobody uses."

Some Quotes

These are by Alan Perlis (1922-1990): An early computer scientist and first winner of the Turing Award in 1966.

"A programming lang
to the irrelevant."

"A language that doe
not worth knowing."

"There will always be
known languages ca

And since programmer
language they are usin
(inventor of C++):

"There are only two k
bitch about and those nobody uses."

The "Turing Award" is the top award given in the field of Computer Science - sometimes referred to as the "Nobel Prize of Computer Science."

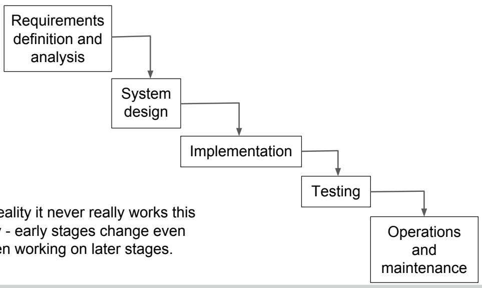
Named after Alan Turing (1912-1954), one of the most influential founders of the field of Computer Science.

- Designed computers in WWII to break German codes
- General model of computers named after him ("Turing machines")
- Also stated a test for Artificial Intelligence (the "Turing Test")

Software Engineering

How to control the complex process of creating software

Traditional "software lifecycle" has well defined phases that feed into each other one-way - called the "Waterfall Model"

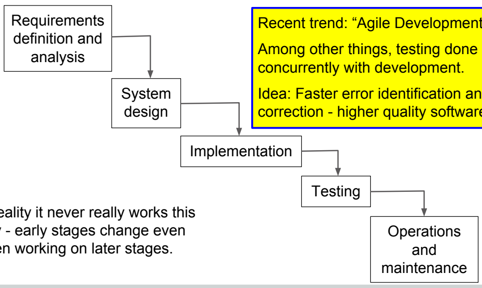

Requirements definition and analysis
System design
Implementation
Testing
Operations and maintenance

In reality it never really works this way - early stages change even when working on later stages.

Software Engineering

How to control the complex process of creating software

Traditional "software lifecycle" has well defined phases that feed into each other one-way - called the "Waterfall Model"


Requirements definition and analysis
System design
Implementation
Testing
Operations and maintenance

Recent trend: "Agile Development"

Among other things, testing done concurrently with development.

Idea: Faster error identification and correction - higher quality software?

In reality it never really works this way - early stages change even when working on later stages.

Peer Reviews and Pair Programming

Two ideas for software development...

Peer Reviews: At regular intervals, a developer presents and explains their code to co-workers, who critically review code. Similar to a writer going over drafts with an editor.

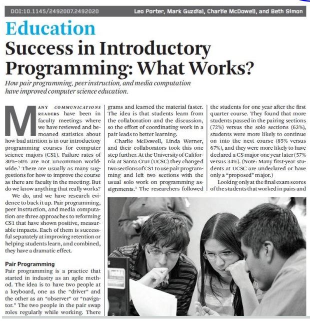
ALL OF THESE PARENTHESES AND SEMICOLONS TOTALLY RUIN THE FENG SHUI OF THE CODE.
SALLY LEARNS AN IMPORTANT LESSON ABOUT INVITING THE RIGHT PEOPLE TO HER CODE REVIEWS.

Pair Programming: Development is actually a collaborative activity - pushes "review" so far back that it is simultaneous with development!

Pair Programming for Learning

From August 2013 *Communications of the ACM*:

Plug: Don't forget our UNCG ACM chapter!



DOI:10.1145/2492007.2492020 Leo Porter, Mark Guzdial, Charlie McDowell, and Beth Simon

Education

Success in Introductory Programming: What Works?

How pair programming, peer instruction, and media computation have improved computer science education.

MANY *COMMUNICATIONS* READERS have been in faculty meetings where we have reviewed and bemoaned statistics about how bad attrition is in our introductory programming courses for computer science majors (CS1). Failure rates of 30%–50% are not uncommon worldwide.[1] There are usually as many suggestions for how to improve the course as there are faculty in the meeting. But do we know anything that really works?

We do, and we have research evidence to back it up. Pair programming, peer instruction, and media computation are three approaches to reforming CS1 that have shown positive, measurable impacts. Each of them is successful separately at improving retention or helping students learn, and combined, they have a dramatic effect.

Pair Programming

Pair programming is a practice that started in industry as an agile method. The idea is to have two people at a keyboard, one as the "driver" and the other as an "observer" or "navigator." The two people in the pair swap roles regularly while working. There

grams and learned the material faster. The idea is that students learn from the collaboration and the discussion, so the effort of coordinating work in a pair leads to better learning.

Charlie McDowell, Linda Werner, and their collaborators took this one step further. At the University of California at Santa Cruz (UCSC) they changed two sections of CS1 to use pair programming and left two sections with the usual solo work on programming assignments.[3] The researchers followed the students for one year after the first quarter course. They found that more students passed in the pairing sections (72%) versus the solo sections (63%), students were more likely to continue on into the next course (85% versus 67%), and they were more likely to have declared a CS major one year later (57% versus 34%). (Note: Many first-year students at UCSC are undeclared or have only a "proposed" major.)

Looking only at the final exam scores of the students that worked in pairs and

Surprise!

We don't do pair programming because we like to torture you and stick you with terrible partners.

We do it because it helps people learn!

Pair Programming: What to do

Really simple concept:

- Two students, one computer
  - Roles: "Driver" and "Navigator"
  - Driver has keyboard/mouse, but navigator describes how to build solution
- Both students always active
  - *Not "Driver" and "Sleepy passenger"*
- Switch roles regularly
  - At least once per lab activity, if not more often
- Be open and respectful
  - If you don't like a proposed solution, your job is to either explain why it' s not a good solution or to make the case for a better one - don't just dismiss it!

Handout: "Fun with Pair Programming!"

Details of Pair Programming in CSC 100

At the beginning of lab:

- Check partner/workstation information sheet posted on lab door
- Find your workstation
- If you don't know your partner, introduce yourself!
- Only one of you will log in to the workstation - decide which one (maybe first to sit down?)
- If your partner is a no-show by 10:02, let the instructor or lab assistant know

At the end of the lab:

- One of you logs in to Blackboard
- Attach lab files to the lab submission
- Add a comment saying who the partners were
- Submit!
- Everyone still does the quiz individually (before the next class)

Reminders

Before this Friday's lab:

- Watch the pair programming video
- Read (and understand!) Pre-Lab reading

Also: Remember the on-line discussion of Chapter 1 of *Blown to Bits* (participate before Wednesday's class!).