Course Overview
Key ideas behind a simple program.
Class learning goals

- To provide an overview of some of the big ideas and concepts of computer science
- To give an understanding of what motivated the development of these ideas
- Provide a sense of exciting current developments, and where the future lies

What this class is not:

- Not a programming course (try CMPS 5J, CMPS 5C, or CMPS 5P)
  - Computer science is far more than just programming languages
- Not a computer literacy course (I assume you know how to use a computer)
Class grades

- 15% - in-class quizzes
  - These will be unannounced pop-quizzes, designed so that you will do well on them if you have been attending class
- 25% - homework assignments
- 40% - midterm exams
  - Two exams, each worth 20%
- 20% - final exam

- The final exam is scheduled for Tuesday, June 7, 8am-11am
  - This class will have a final exam. Please plan on this.
Class website

- http://www.soe.ucsc.edu/classes/cmps010/Spring11/

- Please write this down, and bookmark it

- Holds:
  - Syllabus (including homework due dates)
  - Homework assignment descriptions
  - Description of course readings
  - Links to class lecture notes
Why study computers?

- The computer is a profoundly important technology
  - Broadly impactful
  - Occasionally disruptive

- What are some of the impacts the computer has had the way we live, the way we think, and the way we do business?

- We are perhaps only 1/3 to 1/2 of the way through the process of absorbing the impact of computing in our lives.

- Computers will have a substantial influence on any area of study you choose at UC Santa Cruz. Understanding computers is important.
What is Computer Science?

- Broadly, computer science is the study of
  - **what** can be accomplished using computers, and
  - **how** to construct software to do these things.

- Many views on the field
  - Mathematical vs empirical
    - Mathematical: views computers as a device with precise, formal meaning to all operations. Hence, mathematics should be capable of describing everything a computer can do.
    - Empirical: views computers as complex systems where there are often multiple ways to accomplish a task, and tradeoffs among them. Understanding is only possible via a process of building programs and exploring their properties.
What is Computer Science? (cont’d)

- **Domain-specific**
  - Information-centered
    - Computer science is the study of information, including its representation, storage, transmission, and processing (e.g., data structures, databases)
  - Program-centered
    - Computer science is the study of programming, including appropriate choice of language for a problem, language design, compiler construction, program verification and correctness.
  - Algorithm-centered
    - Computer science is the study of algorithms. Study of algorithm design, characteristics of algorithms, what is computable, tradeoffs of different algorithms.
What is Computer Science? (Edge cases)

- Human-computer interaction
  - The study of the user interfaces of software, and how users interact with software.
  - Expands the focus of inquiry to be a system comprised of the computer plus humans interacting with it.

- Software engineering
  - The study of the construction of large-scale software systems.
  - Requirements engineering – determining what a software system should do is a social process, and is very challenging to get right.
  - Large software code bases are so complex that formal approaches are insufficient to model software behavior. Mathematical view of computer science is insufficient.

- Digital arts
  - A field that explores what computers can accomplish, but with the aim of creating an artistic experience.
Let's tease apart some of the intellectual roots that make this simple program possible...
static void Main(string[] args)

- string[] args
  - Holds input data passed to the program
    - These inputs are called “arguments”, or args
  - A mathematical use of the term argument:
    - “The angle, arc, or other mathematical quantity, from which another required quantity may be deduced, or on which its calculation depends.” OED

- Derives from console-based input
  - program_name arg1 arg2 arg3…
  - A convenient way to pass data into programs when using a teletype or early console
  - Limited communication speed between console and computer meant full screen interfaces were slow (and with paper teletypes, not even possible)
The first line (cont’d)…

- **Main**
  - The name of the *method (also called a procedure or function in other languages)*
  - A convention for the name of the entry point into a program
  - Sense of main as “primary”
  - Mainframe (primary computer), main memory (primary memory), etc.

- **Void**
  - Indicates that Main does not return any data (the data returned is void, null, empty, etc.)
  - But, why should a method return data?
  - The *underlying model is that of a mathematical function*
  - Output = function(inputs) (e.g., $y=f(x)$)
  - Computing mathematical functions was an early strong goal of computer programming
### int playerGuess = 0

- **int playerGuess = 0**
- The *variable* named playerGuess has a *type* of integer, and is *assigned* the initial value of 0.

#### Variable
- A named memory location in the computer that can hold a value of a given type.
- It is analogous to a variable in mathematics, but with subtle, yet important differences.
- In mathematics, a variable is a symbol that represents a value, but is often just left symbolic:
  - E.g., in \( y=x^2 \) one could pick a specific \( x \) of 2 to get \( y \) of 4, but it’s also OK to never bind variables to values and just work with the symbolic expression.
  - Computer variables are *always* bound to a value.
- Computer variables take up space in memory while mathematical variables exist outside of a physical implementation.
- Computer variables have a specific type, which indicates how they are physically represented inside the computer. Mathematical variables are not concerned with specific representation, though can be typed in some instances.
- In mathematics, an expression like \( y=x^2 \) can be evaluated in both directions unless one of the variables is explicitly stated as being dependent on the others:
  - Given a \( y \) of 4, you can determine that \( x \) is 2.
  - With computers, there is a fixed order of computation, and some variables are always dependent on others.
Variables

- The earliest computers were developed to calculate tables of numbers and decipher coded messages
  - That is, *mathematical computation* was a primary early goal for developing computer technology.
  - As a consequence, mathematical terminology, and mathematical ideas were very influential in the early development of computers, and of programming languages.
  - The early pioneers of the field were all mathematically sophisticated.
- The earliest computer is Charles Babbage’s Analytical Engine, which built on his Difference Engine
  - In the 1840s, conceived of the idea of a variable, which abstracts away from a specific numerical value to a location that can hold a number of values.
  - The relationships the variable has with other variables can be expressed abstractly, using equations
int playerGuess = 0

- `int` indicates that `playerGuess` is an integer
  - Specifically, a *signed, 32-bit* integer

- An integer is a number that doesn’t have a fractional part
  - -1, 0, 1, 2, 3...

- Integers can be very easily mapped into binary numbers
  - Binary = using only 0 and 1. One such digit is a “bit”
    - 000 = 0  001 = 1  010 = 2  011 = 3  100 = 4
    - 001 = 0*2^2 + 0*2^1 + 0*2^0 = 1
    - 011 = 0*2^2 + 1*2^1 + 1*2^0 = 3

- 32 bits indicates that a maximum of 32 binary digits are used to represent a number

- The *signed* part indicates that one bit is used to indicate whether the number is positive or negative
  - 1 bit – sign
  - 31 bits – integer value (range of values is -2,147,483,648 to 2,147,483,647)
Why bits?

- Electrical circuits can easily have two states
  - Current flowing, current not flowing
  - Or, high voltage, low voltage
  - These are physically easy to create and work with
  - The early telegraph sent messages using current flowing (short = dot, long = dash), current not flowing (long or short break).

- Claude Shannon coined the term bit in 1948, in a landmark paper on information theory
  - But, earlier, in 1937, had abstracted away the idea of a circuit being on/off (current flowing, not flowing) into the information content this carried (1 or 0).

- When engineers started developing computer memory, they used the idea of a physical mapping from voltages and currents into a logical 1/0, and then to representing numbers with multiple digits
Console.WriteLine("Let's play a game…

- Displays text to the console
- But, how is that text represented? Use a code…

- Basic idea: map each letter to a number…
  - L = 76, e = 101, t = 116, and so on
  - But, why these specific numbers?

- Goes all the way back to the telegraph
Evolution of character codes

- In the 1870s, there was a desire to send Roman characters across a telegraph without requiring understanding of Morse code.
  - Let to creation of Baudot code. Each character is 5 bits long
- In the 1960s, two competing standards, ASCII (open standard) and EBCDIC (IBM platform-specific)
  - ASCII won out in the end (government mandate to only buy ASCII computers came out in 1968), a 7-bit standard
- Need to represent accented characters led to ISO/IEC 646, which added an 8th bit to represent accented characters
- Unicode effort started in 1980’s to create single universal character code for all alphabets
  - A C# string is Unicode (which builds on ASCII)
if (playerGuess > computerNumber)

- This is known as a *conditional*
  - A logical condition is evaluated, and if true, some action is taken
  - The “playerGuess > computerNumber” is a logical condition
    - If the player’s guess is greater than the random computer number, the condition is *true* (1) otherwise it is false (0)

- This approach is borrowed from *mathematical logic*
  - The original goal of logic is to create a *precise language for reasoning and argumentation*.
  - That is, if something is said to be true, how do we know it really actually is?
  - Would like to have a set of rules for starting from known good facts and deriving new correct knowledge from them.
George Boole

- It was George Boole (and simultaneously Augustus De Morgan), in the 1840’s who joined logic with mathematics
  - “The respective interpretation of the symbols 0 and 1 in the system of logic are Nothing and Universe.”
  - I.e., not true (false), or universally correct (true)
  - We sometimes call this Boolean logic in honor

- By the invention of the computer, Bertrand Russell and Alfred North Whitehead had used logic to try and unify all of mathematics, in their *Principia Mathematica (1910-1913)*
  - Hence, any serious mathematician would have been familiar with mathematical logic by the time of the invention of computer languages in the 1950s
Intellectual roots of computer science

- So, just by looking “under the hood” of a simple program we see connections back to:
  - Telegraphy and early computer communication
  - Mathematical functions, mathematical logic
  - The earliest computers
- Gives us a strong sense of the deep influence mathematics has had on computers
- But, also gives us the sense that to deeply understand the present, one must also understand the past
- In coming lectures, will aim to:
  - Provide **historical context** on a subject
  - Provide **key ideas and concepts** about a subject
  - Provide some sense of where the **future** is going