CSE11 Spring 2015

Lecture 1

Who Am I?

- UCSD Undergrad 1981-1985, Revelle
- UCSB, PhD, Electrical Engineering, 1993
- Current
  - Chief Technology Officer, San Diego Supercomputer Center
  - Areas of Interest
    - Large-scale Cluster Computing
    - High-speed Networking
    - High-performance storage systems
  - Part-time lecturer in the CSE Department

Is CSE11 right for you?

- This is an accelerated pace class
- Previous programming experience is helpful, but not strictly necessary.
    - See: http://cse.ucsd.edu/node/43 for departmental placement advice

Prerequisites:

Recommended prep: High school algebra and a course in programming in a compiled language. See CSE Course Placement Advice.

- **This is a time-intensive class**. 3-4 hours of reading/week + 4 – 6 hours of programming
    - Don't get behind.
    - Don't think you can "catch up" with a few days of "cramming".

Topics Covered

- Is CSE11 the right class for you?
- Overview of Grading, Online Book

- A short history of programming languages
- Computer organization
- What is a procedural language?

Overview of Class

- https://sites.google.com/a/eng.ucsd.edu/cse11-spring15-a00/home
- Grading
  - 15% Reading Assignments. Progress is checked electronically. In any week, complete 85% of the reading to get full credit. Due Every Monday for previous week's lectures.
  - 25% Programming Assignments – Seven Programs. Final program is a two-week assignment.
  - 25% Midterm - Tuesday, 6th week of classes, in class
  - 35% Final – Tuesday, March 17th, 8-11am
- Late Assignments
  - No late reading assignments. Feel Free to read ahead
  - No late programs. But you have two 24 hour slip passes, if you need them

Reading Assignments

- This class is using an online textbook from zybooks.com. It is REQUIRED. Cost is $40.00
  - Reading are short sections with interactive “activities” that reinforce concepts.
  - There are also “Challenge Programming” assignments. These are optional and are not graded.
  - You are graded on the percentage of interactive activities that you complete.
  - You must complete the reading for current week by Monday (11:59pm) of the following week.
  - Goto http://www.zybooks.com. Book Code: **UCSDCS11PapadopoulosSpring2015**

Programs

- This is a programming class and you need to write code every week
- Programs are to be your own work.
  - Don't copy. Don't "split" the assignment with your buddy.
  - DO talk to the tutors, TAs, Professor.
  - DO discuss general approach with your classmates
  - DO start early. Programming has a way of going completely wrong at 2 AM.
- All Code must work on the Lab machines. This is a Linux environment. Test before you turn in.
  - **Code that does not compile, will receive a 0 for the assignment**

Midterm/Final/Extra Credit

- Midterm is in class
  - Closed Book. Closed Notes. Closed Neighbor. Closed Device. Bring some extra pencils.
  - Multiple choice, True False, Short-answer, and writing code (with your pencil, not your laptop)
- Final
  - Same as Midterm, only longer and more difficult.
- Bonus Quizzes – In Class
  - Two quizzes, each worth 25 points. Dates determined in the future. Will be given notice the lecture prior. Added to your program score.

Grading Standards

- Graded relative to class, but if your total weighted score is
  - >= 90%. A- or better, guaranteed
  - >= 80%. B- or better, guaranteed
  - >= 70%. C- or better, guaranteed
- Cutoffs for A,B,C may be lower than the above cutoffs, but will not be higher
- If your weighted score is < 50%, you will receive an F, no matter how the class as a whole scores.
- Greater than 50% DOES NOT mean you will get at least a D.

LAB

- You are given an account on a UCSD-managed system. To find your account for THIS class, go to
  - https://sdacs.ucsd.edu/~icc/index.php
- Tutors will be on 260B (CSE Building Basement)
- Your code must work in Linux, and on the version of Java used in the Lab.
  - Windows formats text file (programs) differently from Linux.
  - If you write code on your Windows laptop, verify that it works as expected in the Lab BEFORE you turn it in.
- All code turn in is electronic. Usually, the turn-in program is available about 48 hours before the program is due.

Programming Environment

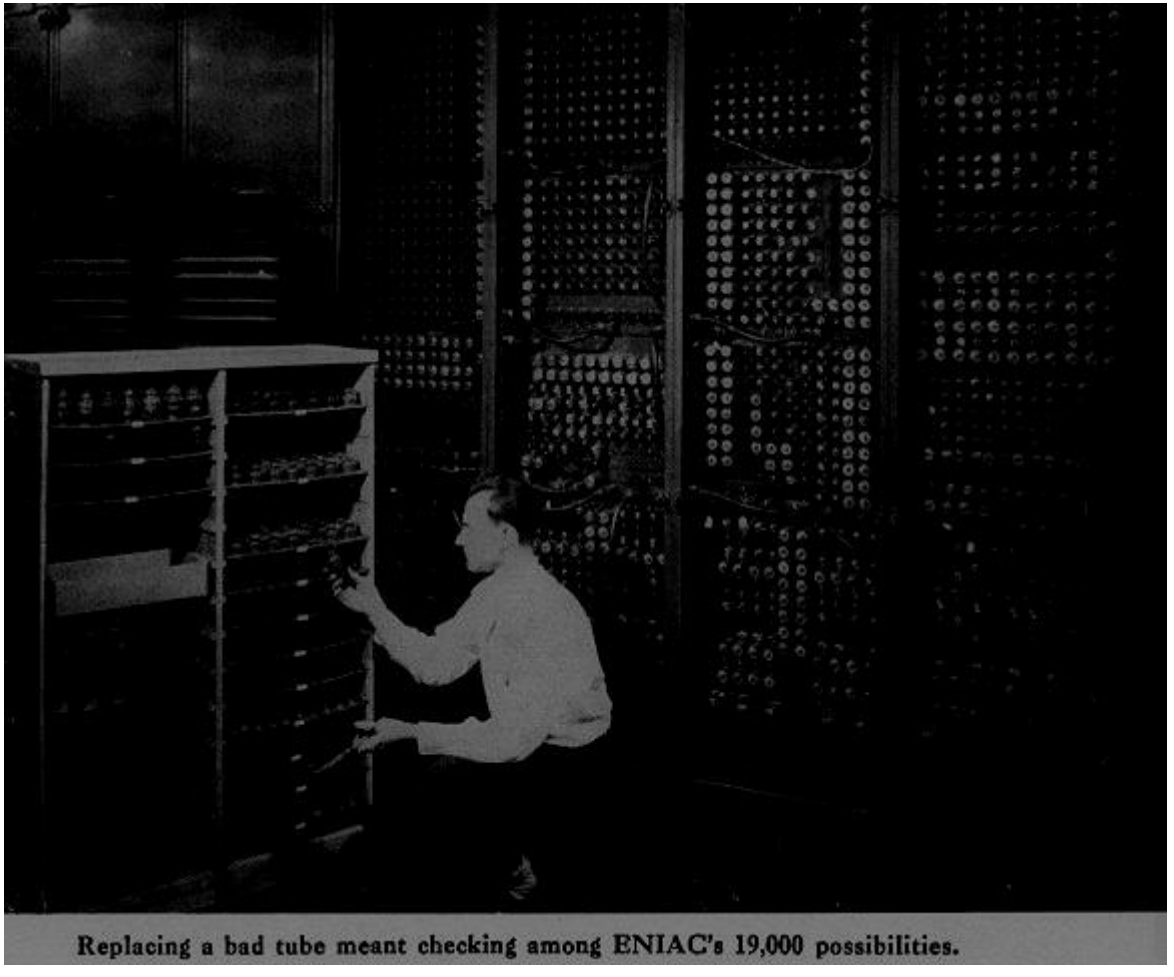
- UNIX command line, e.g.,
  - `javac –cp .:'*' HelloWorld.java`
  - `java HelloWorld`
- We are NOT using an IDE like Dr. Java, Eclipse or IntelliJ.
- You need to learn how to code in one of the two well-established editors
  - VIM – VI Improved
  - emacs
- Get comfortable with unix directories/commands
  - `mkdir, cd, cp, mv, pwd, ls, …`

Structure of Lectures

- Spend 60-80% lecture covering concepts
- Spend balance of time with interactive programming, chalk-board, debugging, looking at program assignments.
  - **Questions are encouraged at any time.**
- There will be material covered in lecture, that is not covered in the book. You are responsible for all of it.

A Short History of Programming

- 1943 – The ENIAC. University of Pennsylvania
- Used for Artillery Projectile Calculations 19,000 Vacuum Tubes
- Programmed in Assembly Language (Machine Code).



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Assembly Language

- This is what CPU “understands”
- Here's an Intel x86 code snippet

```
1       pushl %ebp
2       movl %esp, %ebp
3       subl $4, %esp
4       movl $10, -4(%ebp)
5       leal -4(%ebp), %eax
6       addl $66, (%eax)
7       leave
8       ret
```

- Not readable without comments
- Specific to brand particular hardware (Intel, ARM (what's in many smartphones), PowerPC all different)
- Prone to errors. Slow to program.

1950s, 1960s

- 1955: FORTRAN (Formula Translation)
- 1958: LISP (List Processing)
- 1959: COBOL (Common Business Oriented Language)
- 1964: BASIC (Beginner's All Purpose Symbolic Instruction Code)
- These were critical advances in programmability of computers.
  - English-like constructions
    - Compare, Test, Jump → If
  - Compiler/Interpreters translated automatically to machine code

1970s,1980s

- 1970 – Pascal
  - UCSD Pascal, and the p-System in 1978 made Pascal portable
- 1972 – C
  - This made the UNIX operating System practical
  - Most of Linux kernel is in C
- 1978 – SQL (Structured Query Language)
  - Program databases
- 1980 - C++ (C with “classes”)
- 1984 – MATLAB (Matrix Linear Algebra)
- 1987 – PERL (Practical Extraction and Reporting Language)
- Proliferation of Specialized Languages, Greatly improved the ease with which complex algorithms could be expressed.

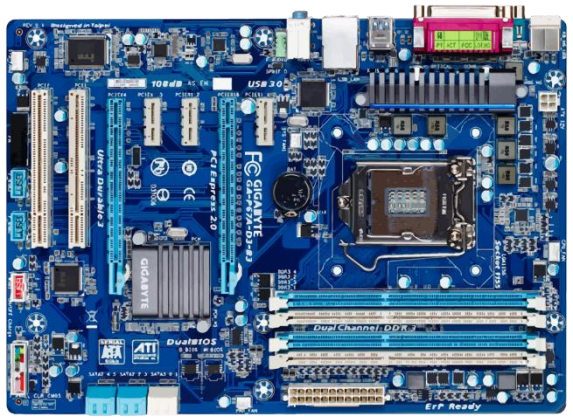
1990s

- 1991 – Python, Visual Basic
- 1991 – HTML (Hypertext Markup language)
  - Websites
- 1995 – Java
  - Rapid Application Development
  - Object Oriented
  - Loosely based upon C
  - Simplified inheritance versus C++
  - Portable (more on this later)
- Programmers uses the Language that is most suitable the problem at hand.
- Even “Old” languages persist. e.g., FORTRAN is used on all modern supercomputers. C is critical for *NIX operating systems

Computer Organization

- Basic Computer operation is very straightforward
- CPU – Central Processing Unit
- Memory
  - Instructions
  - Data

- Input/Output Devices (files, display)



The basic CPU “Loop“

CPU

1 Program Counter

2

3

Memory

Load A into Register A (RA)
Load B into Register B (RB)
Add RA,RB place in RC
Store RC into Location C

Memory Location 0

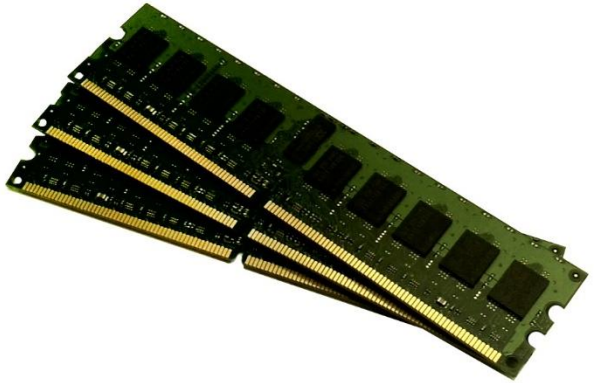
Memory Location 4

Memory Location 1036

Memory Location 1040

Memory Location 1044

- Program Counter Starts at Location 0
- Reads Instruction (1)
- Executes Instruction
- Advances to Loc 4
- Read Instruction (2)
- Executes instruction
- Advances to Loc 8
- Continues (3)



Computer Memory
1's and 0's

0	1	0	1	1	0	1	1

= 91 (decimal), 5B (hex), '[' (ascii)

Byte = 8 Bits

0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	1

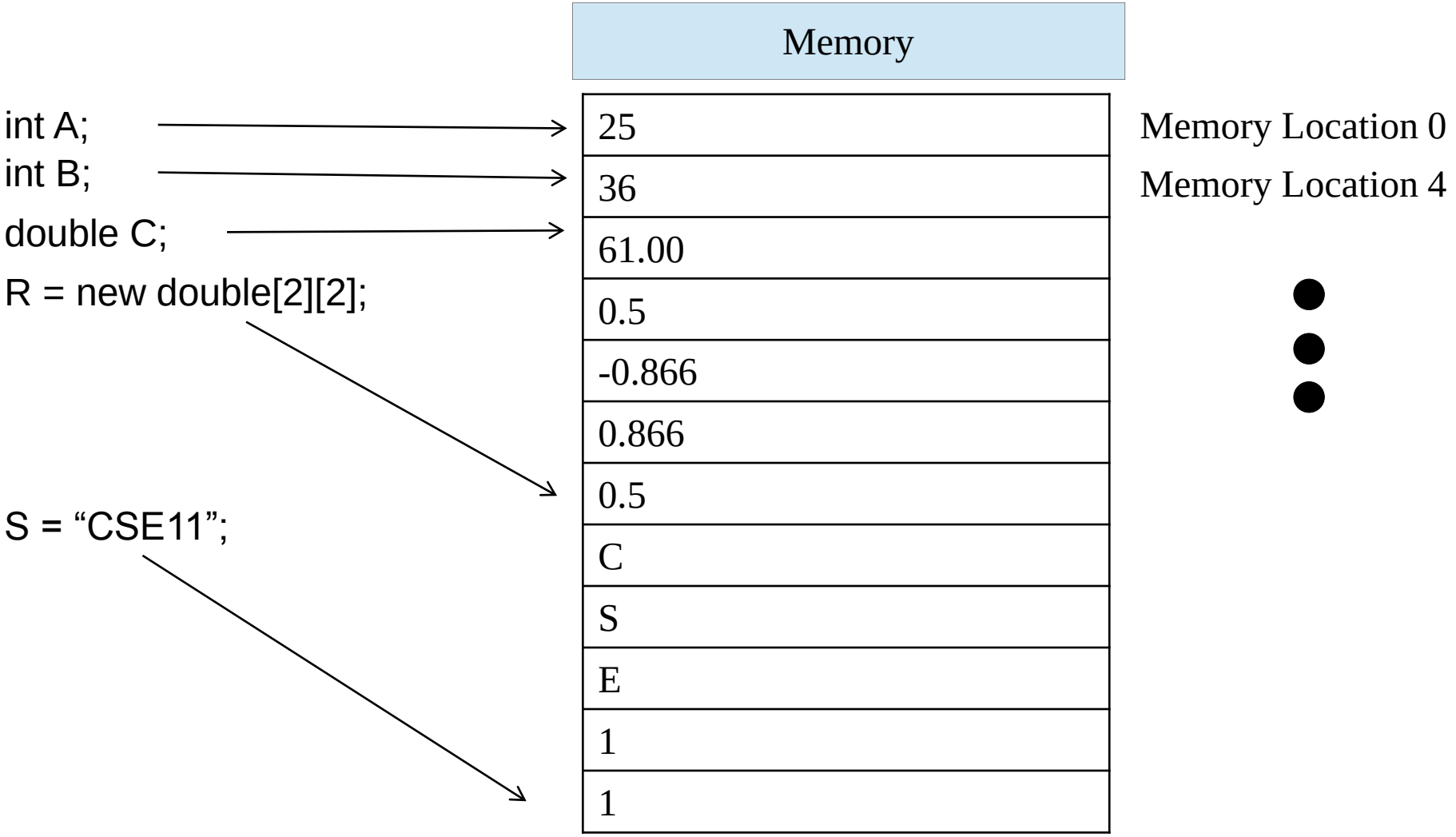
Word = 2 Bytes

0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	1

Double Word = 4 Bytes

Quad Word = 8 Bytes

Memory that Stores Data



Memory is <u>Universal</u> Storage.

What is stored on Location N is different for every program

Different Types of Data Consume Different Amounts of Memory

- | <type> | to indicate size of memory used
- Java is very specific of the sizes of different primitive types
- | byte | < | char | = | short int | < | int | = | float |
- | float | < | double |

Abstraction of Memory

- Data is stored in a linear set of memory locations on the computer itself
- Languages let us declare the **names and types** of variables that we want to use
- The compiler and runtime systems keeps track of exactly where in memory a variable is located
  - Nothing in the computer itself protects us from storing a double in location N and then reading it back as if it were an integer
- The same ideas apply to the “code” part of the program.

Programming Languages

- Convert high-level, human-understandable instructions into a form that the computer can execute
- Different kinds of languages make it easier (or harder) to express these high-level instructions

What came before Object-Oriented Programming?

- FORTRAN introduced in 1955
- 25 years later, the first popular object-oriented language appeared (C++)
- The basic abstraction is called *procedural* programming
  - It is the bedrock of computer programming and you use elements of it all the time (even in object oriented programming)

What is procedural programming?

- A program is organized as data and a set of procedures (also called subprograms).
- For the program to do its job, the procedures are called in the correct order
- This very much mirrors how data and code are defined/organized in the computer

```
#include <stdio.h>
int square(int iA) { return iA * iA; }
int cube (int iA) { return iA * iA * iA; }
void main() {
   int A, squaredA, sixthA;
   A = 3;
   // Calculate A^6
   squaredA = square(A);
   sixthA = cube(squaredA);
   printf("%d\n", sixthA);
}
```

What are some of the “complaints” of procedural programming

- Code was separated from the actual data
- Suppose you have
  - Integer A
  - 2 x 2 Matrix B
- You use one procedure to square an integer A (it's just one multiplication)
- You use a different procedure to square matrix B (8 multiplies, 4 additions)
- In procedural programming, the author must explicitly track whether he/she is squaring a number or a matrix and the call the right piece of code to get the job done

Object-Oriented Programming

- Revisit integer A and Matrix B
- Suppose A and B knew “how to square themselves”?
- Instead of the programmer explicitly figuring out which “squaring subprogram to call”, he/she would rely on these variables (objects) to know how to perform the operation properly on themselves.
- This would be called the squaring method

Function Calls vs Method Invocations

- code: `B = square(A)`
  - A *function* called "square" has an *argument* A.
  - Whatever square does, it does it on A (the argument)
  - Whatever square does, it returns something and stores it in B
  - Square is a *procedure* or *function*

- code: `B = A.square()`
  - An object called "A" has *method* called square
  - Whatever square does, it does it to A.
    - If A and B are the same kind of objects, then B.square() is valid
  - Whatever square does, it returns and object and stores it in B
  - Square is called a *method*