CS103 Lecture 1 Slides

Introduction
Mark Redekopp
What is Computer Science

• All science is computer science
  – It is very interdisciplinary: Math, Engineering, Medicine, Natural sciences, Art, Linguistics, etc.

• Computer science is no more about computers than astronomy is about telescopes

• It is about solving information-based problems using computers as the main tool
  – How do I recognize objects in a photograph
  – How do I determine the best web page to return given a search query
  – Identify the function of this protein given it structure
What Computer Scientist Do...

• Observe, organize, transform and discover useful information from data
• Use math and logic to solve problems
• Work in groups
• Innovate and improve
• Program
Computer Science Is...

• Essential to economic growth and development
• Dealing with society’s problems
  – Health and E-Science
  – Big Data
    • Conservation & the environment
    • Developing personalized learning
  – Who you might want to date 😊
• A great way to make a living
  – Maria Klawe, et. al. - To the age-old question -- "What do you want to do when you grow up?" -- children today give many modern answers: "Help feed hungry families." "Prevent and cure diseases." "Find sources of renewable energy." "Understand the universe."
  One clear path leads to each of these aspirations: the study of computer science.
  http://www.huffingtonpost.com/maria-klawe/computing-our-childrens-f_b_388874.html
What Is this Course About

• Introduction to Programming
  – Introduction: Don't require prior programming experience
    • Experience says about half of you have not programmed
    • However, we will move fast so you must be prepared to put in some extra time if you've never coded before
    • Students who want/need a slower on-ramp may consider first taking CSCI 101 or ITP 165, 109, or 115
  – Programming
    • We'll try to teach good coding practices and how to find efficient solutions (not just any solution)
    • We'll focus on concepts present in most languages using C/C++ as the primary language (not Java)
High Level Languages

Mother Tongues
Tracing the roots of computer languages through the ages

Just like half of the world’s spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouse languages C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic linguists. If you will aim to save, or at least document the lingua of classic software. They’re combing the globe’s 9 million developers in search of coders still fluent in those nearly forgotten “tongues”.

Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-caker Grady Booch, Rational Software’s chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grep the code. Why bother? “They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time,” Booch explains. “They’ll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure.” Here’s a peek at the strongest branches of programming’s family tree. For a nearly exhaustive rundown, check out the Language List at http://www.informatik.uni-freiburg.de/java/misc/lang_list.html. - Michael Mendee

Key
1944 Year Introduced
Active: Thousands of users
Protected: Great at universities; compilers available
Dormant: Large dropping off
Extinct: No known active users or aging compilers
Lineage continues

Sources: Paul Boutin, Brent Hallperr, associate director of computer science at IBM Research; The Recomputing Museum; Todd Probsting, senior researcher at Microsoft; Gis Wiederhold, computer scientist, Stanford University

http://www.digibarn.com/collections/posters/tongues/ComputerLanguagesChart-med.png
Why C/C++

- One of the most popular languages in industry
- C/C++ is close to the actual hardware
  - Makes it fast & flexible (Near direct control of the HW)
  - Makes it dangerous (Near direct control of the HW)
  - Most common in embedded devices
  - C became popular because it was the language used to implement Unix & then Linux (all Unix/Linux distributions came with a C / C++ compiler)
- C/C++ is ubiquitous
  - Used everywhere, even to implement other programming languages (i.e. Python, Matlab, etc.)
- Principles learned in C/C++ will allow you to quickly learn other programming languages
- Not Java
Syllabus
Course Advice

• Catch the wave!
  – Overestimate the time you will need and your ability to get your work done
  – Limit extracurricular activities in the 1st semester
  – Don’t let shame or embarrassment keep you from the help you need

• Experiment and fail

• Computer science requires practice
  – It's like learning a musical instrument
Research at USC

• Integrated Media Systems Center
  – Sound, video, online collaboration, streaming media research

• Information Sciences Institute
  – AI, Internet, Advanced Processing Systems research

• Institute for Creative Technologies
  – Virtual Reality, Graphics, Animation, Games
Media

• Robotics
  – http://www.isi.edu/robots/superbot/movies/FoxNews.swf

• Virtual Reality
  – http://www.youtube.com/uscict#p/u/13/Fh9gIswxbvU
  – http://www.youtube.com/uscict#p/u/0/0U7-q_9YV5c
20-Second Timeout

• Who Am I?
  – Teaching faculty in CENG
  – Undergrad at USC in CECS
  – Grad at USC in EE
  – Work(ed) at Raytheon
  – Learning Spanish (and Chinese?)
  – Sports enthusiast!
    • Basketball
    • Baseball
    • Ultimate Frisbee?
THINK LIKE A COMPUTER
Path Planning

• Find shortest path from S to F
Path Planning

- Find shortest path from S to F
Path Planning

• A computer usually can only process (or "see") one or two data items (a square) at a time

May just compute a straight line path from ‘S’ to ‘F’
Path Planning
Path Planning
Path Planning

• What if I don’t know where the Finish square is? Can you devise a general search order to find the shortest path to ‘F’ while examining the minimum number of squares as possible.
Path Planning

- Examine all closer squares one at a time before progressing to further squares.

If you don’t know where F is and want to find the shortest path, you have to do it this way.

Uninformed search for shortest path: **Breadth-first**
Path Planning

- Now I’ll tell you where F is
- Can that help you reduce the number of squares explored?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S</td>
<td>3</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select a square to explore with minimum distance to the finish.
Path Planning

- Now I’ll tell you where F is
- Can that help you reduce the number of squares explored?

Select a square to explore with minimum distance to the finish

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S</td>
<td>3</td>
<td>2</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select a square to explore with minimum distance to the finish
Path Planning

• But what if we run into a blockage?
  – Now we would pick the best among the remainder.

Select a square to explore with minimum distance to the finish
Path Planning

- But what if we run into a blockage?
  - Now we would pick the best among the remainder.

Select a square to explore with minimum distance to the finish.
But Why?

• Why can’t computer just “look” at the image
  – Computer store information as numbers
  – These numbers are stored as units of 8-, 32- or 64-bits and the processor is only capable to looking at 1 or 2 numbers simultaneously
  – Each pixel of the image is a separate piece of data
Memory

- Set of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
  - Used to reference the value in that location
Memory Operations

- Memories perform 2 operations
  - Read: retrieves data value in a particular location (specified using the address)
  - Write: changes data in a location to a new value
- To perform these operations a set of address, data, and control inputs/outputs are used
  - Note: A group of wires/signals is referred to as a ‘bus’
  - Thus, we say that memories have an address, data, and control bus.
Programming vs. Algorithms

- Programming entails converting an algorithm into a specific process that a computer can execute.
20-Second Timeout: CS/CENG True or False

• Control Question: USC basketball will win the NCAA championship this year

• True or False: The following achievements were performed here at USC in CS and EE depts.
  – Algorithmic basis of JPG, MPG, and MP3 formats developed here
  – A CS faculty won an Academy Award in 2010
  – THX audio was partly developed here
  – CD’s and DVD’s use error-correcting codes developed here at USC
  – Internet security has its roots in the research of a professor at USC
Your Environment

GETTING STARTED
Development Environment

• To write and run software programs in C you will need
  – A text editor to write the code
  – A ‘C/C++’ compiler, linker, etc. to convert the code to a program

• Different OS and platform combinations have different compilers and produce “different version” of a program that can only run on that given OS/platform.
  – Mac XCode (Mac only)
  – MS Visual Studio (Windows only)
  – CodeBlocks (cross-platform)
Ubuntu VM Image

• We are providing a virtual machine appliance (An Ubuntu Linux image that you can run on your Mac or Windows PC)
  – Requires installation of Oracle VirtualBox and download of the Ubuntu Image

• Video walkthrough
C OVERVIEW AND DEMO
C Program Format/Structure

• Comments
  – Anywhere in the code
  – C-Style => “/[*” and “*/”
  – C++ Style => “//”

• Compiler Directives
  – #includes tell compiler what other library functions you plan on using
  – ’using namespace std;’ -- Just do it for now!

• Global variables (more on this later)

• main() function
  – Starting point of execution for the program
  – Variable declarations often appear at the start of a function
  – All code/statements in C must be inside a function
  – Statements execute one after the next
  – Ends with a ‘return’ statement

• Other functions
# Software Process

## Std C++ & Other Libraries

```c++
#include <iostream>
using namespace std;

int main()
{
    int x = 5;
    cout << "Hello"
         << endl;
    cout << "x=" << x;
    return 0;
}
```

## Compiler

- `g++` = Enable Debugging
- `-Wall` = Show all warnings
- `-o test` = Specify Output executable name

## Executable Binary Image ("test")

```
1110 0010 0101 1001
0110 1011 0000 1100
0100 1101 0111 1111
1010 1100 0010 1011
0001 0110 0011 1000
```

## Load & Execute

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```

### 1. Edit & write code

```
$ gedit test.cpp &
```

### 2. Compile & fix compiler errors

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test
```

### 3. Load & run the executable program

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```
Software Process

#include <iostream>
using namespace std;

int main()
{
  int x = 5;
  cout << "Hello" << endl;
  cout << "x=" << x;
  return 0;
}

C++ file(s) (test.cpp)

$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test

Fix compile-time errors w/ a debugger

Executable Binary Image (test)

1110 0010 0101 1001
0110 1011 0000 1100
0100 1101 0111 1111
1010 1100 0010 1011
0001 0110 0011 1000

1  Edit & write code
2  Compile & errors
3  Load & run the executable program

- g = Enable Debugging
- Wall = Show all warnings
- o test = Specify Output executable name

Fix run-time errors w/ a debugger