CS102 Unit 0

Introduction
Introduction

• This is how we often see and interact with software
  – In truth we interact with it far more than we think
  – We are interacting with software when we drive, fly, turn on the lights, watch TV, go to the bank, or buy something with our credit card

• So what is it really?
Introduction

• This is how the movies think computers see software
  – The far right picture is reasonably accurate

• While all programs eventually end up as 1s and 0s, we generally program using some form of "high-level" or scripting language
Computer Abstractions

- Computer systems can be viewed as a layered stack of abstractions from basic HW to complex SW
- Assembly and machine code are the fundamental instructions a computer processor can execute
  - Too low level
- Enter high level languages
  - More powerful and succinct descriptive abilities
- Because of how the hardware works, our software must be written using certain structures
  - This class is intended to teach you those programming structures.

<table>
<thead>
<tr>
<th>Compilers / Interpreters</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Level Languages: Python / Java / C++</td>
</tr>
<tr>
<td>Assembly / Machine Code</td>
</tr>
<tr>
<td><strong>Applications</strong></td>
</tr>
<tr>
<td>OS</td>
</tr>
<tr>
<td>Libraries</td>
</tr>
<tr>
<td>Processor</td>
</tr>
<tr>
<td>Memory (RAM)</td>
</tr>
<tr>
<td>I/O (Disk, Net, Keyboard, Graphics)</td>
</tr>
<tr>
<td>Digital Circuits (Transistors)</td>
</tr>
<tr>
<td>Voltage / Currents</td>
</tr>
</tbody>
</table>
This Class

• The goal of this class is two-fold
  – Teach you the basics of programming
  – Develop mathematical and algorithmic thinking skills needed to excel in future courses

http://climbingla.blogspot.com/2010/05/walk-6-hermon-and-highland-park.html
Syllabus
Expectations

• Attend lectures!
• Be engaged
  – Ask questions (in Zoom chat or just unmute and talk!)
  – Do your best to keep the web-cams on!
  – We're a team...I need you!
• Catch the wave!
  – Start assignments early, complete them on time
Programming Languages 1

• Declarative Languages
  – Describe the what but not the how
  – Examples: HTML, CSS
Imperative/Structured Languages

- Describe the what (data) and how (instructions/algorithm)
- Examples: C/C++, Java, Javascript, Python (which I'll use today)
- The focus of most programming courses
- Programs are like a recipe for how to operate on data

```
import math

a = int(input("Enter a: "))
b = int(input("Enter b: "))
c = int(input("Enter c: "))

det = b*b - 4*a*c
if(det >= 0):
    r1 = (-b - math.sqrt(det)) / (2*a)
    r2 = (-b + math.sqrt(det)) / (2*a)
    print("Roots are {r1} and {r2}")
else:
    print("Imaginary roots")
```
High Level Languages

Just like half of the world’s spoken tongues, most of the 2,360-plus computer programming languages are either endangered or extinct. As powerhouses 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 lexicographers, if you will—aim to save, or at least document the lingo of classical software. They’re compiling the globe’s 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, API, 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 grok 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 Mendes

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

Appeals to a wide audience C (boasted by the popularity of Linux)

Gets a job done Cobol (designed for business-report writing)

Delivers new functionality Java (runs on any hardware platform)

Fills a niche Mathematica (speeds up complex computations)

Offers a modicum of elegance Icon (has friendly, line-oriented syntax)

Has a powerful user base or backing C# (developed by Microsoft for .Net)

Has a charismatic leader Perl (programmer-author Larry Wall)

Key

Year Introduced
Active
Persisted

1954
1964
1974
1984
1994
2004
1990
2000
2001

Source: Paul Budnitz; Brent Hofacker, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Probstinger, senior researcher at Microsoft; Giso Wiederhold, computer scientist, Stanford University

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

• C++ is used widely
• C++ is "close" to the hardware (HW)
  – Makes it fast
  – Makes it flexible (Near direct control of the HW)
  – Makes it dangerous (Near direct control of the HW)
  – In fact, many other languages are themselves written in C/C++
• Because if you learn C++ you can likely learn MOST languages very quickly
• Because that's what we use in CS 103
What Language Aspects Will We Learn?

• Programming skills in C/C++
  – Overlaps with the first third of CS 103
  – Data Representation
  – Basics of discrete mathematics
  – Expressions
  – Conditional Statements
  – Iterative Statements (Loops)
  – Functions
  – Arrays

• Problem solving using common programming 'idioms'
More than just "Coding"...

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>• A precise problem statement to capture what the application requires (often requires the designer to make choices)</td>
</tr>
</tbody>
</table>
| Problem Solving | • Understanding specification  
• Planning, especially partitioning into sub-problems  
• Identifying and using appropriate idioms  
• Solving difficult sub-problems  
• Writing "glue code" to tie everything together |
| Idioms          | • Simple programming patterns/templates for solving specific tasks that can be used to connect your problem solving approach to actual code |
| Semantics       | • Meaning of a program or any of its parts                                                                                                  |
| Syntax          | • Rules of the language                                                                                                                   |
Problem Solving Idioms

• An idiom is a colloquial or common mode of expression
  – Example: "raining cats and dogs"

• Programming has common modes of expression that are used quite often to solve problems algorithmically

• We have developed a repository of these common programming idioms. We STRONGLY suggest you...
  – Reference them when attempting to solve programming problems
  – Familiarize yourself with them and their structure as we cite them until you feel comfortable identifying them

Rule / Exception Idiom

- **Name**: Rule/Exception
- **Description**: Perform a default action and then use an `if` to correct for an exceptional case
- **Structure**: Code for some default action (i.e., the rule) is followed by code for an exceptional case

```latex
// Default action
if( /* Exceptional Case */ )
{
    // Code for exceptional case
}
```

- **Example(s):**
  - Base pay plus bonus for certain exceptional employees

```latex
bool earnedBonus = /* set somehow */;
int bonus = /* set somehow */;

int basePay = 100;
if( earnedBonus == true )
{
    basePay += bonus;
}
```

- **Notes**: This can be implemented with an `if/else` where an `else` implements the other.
20-Second Timeout

• Who Am I?
  – Teaching faculty in EE and CS
  – Undergrad at USC in CECS
  – Grad at USC in EE
  – Work(ed) at Raytheon
  – Learning Spanish (and Chinese?)
  – Sports enthusiast!
    • Basketball
    • Baseball
    • Ultimate Frisbee?
STARTING TO THINK LIKE A COMPUTER
It’s A Numbers Game

• **Fact 1:** Everything in a computer is a number
  – Sure. Things like 102 and 3.9 are numbers
  – But what about text and images and sound?
  – Everything!

• **Fact 2:** Computers can only work with or "see" 1 or 2 numbers at a time (i.e. they can only do 1 thing at a time)

• Humans process information differently
  – Therein lies some of the difficulty of learning programming
Example (1)

• What do you see?
  – The letter 'a'!

• What does the computer see?
  – A number; each text character is coded to a number
    • Example: Character map / Insert symbol
Text Representation

• Most common character code is ASCII (UTF-8)
• Every character, even non-printing, characters have a corresponding numbers
  – Decimal (base 10) / Hexadecimal (base 16)

https://www.commfront.com/pages/ascii-chart
Example (2)

• What do you see?
  – A circle!

• What does the computer see?
  – Coordinate pairs of each "pixel"
  – ...or...
  – \( r = 120; \) origin = \((10, 14)\)
  – Computer has to enumerate and visit each location and color it black

\[(x,y)=(56,103)\]
\[(x,y)=(57,102)\]
\[(x,y)=(59,101)\]
\[(x,y)=(60,100)\]
Example (3)

• What do you see?
  – A man's face!

• What does the computer see?
  – Many numbers (aka pixels)
  – Value corresponds to color

Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston
The Connection with Mathematics

• Brightness
  - Each pixel value is increased/decreased by a constant amount
  - \( P_{\text{new}} = P_{\text{old}} + B \)
    - \( B > 0 \) = brighter
    - \( B < 0 \) = less bright

• Contrast
  - Each pixel value is multiplied by a constant amount
  - \( P_{\text{new}} = C*P_{\text{old}} + k \)
    - \( C > 1 \) = more contrast
    - \( 0 < C < 1 \) = less contrast

• Same operations performed on all pixels
"Enough" is NOT enough

• As we program we must be explicit
  – Example: drawing the circle on the screen

• Being general is not sufficient; we must be explicit!
  – Imagine a recipe for cinnamon rolls that simply read:
    • Mix and bake the following: butter, that white powdery baking substance, eggs, just *enough* sugar, and cinnamon. Enjoy!
    – How much of each, how much is "*enough*", how long, in what order?

• We will try to work on some of discrete math skills that help us explicitly define and analyze our programs
PROCESS OF WRITING SOFTWARE
Software Process

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

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

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