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.
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
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, spend time reviewing and practicing even when you don't have to
• Respect each other in how you speak, how you listen, and how you act.
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                                                                                                                   |
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'
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 2,360-plus computer programming languages are either endangered or extinct. As powerhouses C++, Visual Basic, Cobol and Java and other modern dialects of assembly language, 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 lingua of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten linguistic frangais. Among the most endangered are Ada, API, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-writer 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 times," Booch explains. "They'll provide the raw materials 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.TU-BREMEN.DE/JAVA/MISC/LANG_LIST.HTML - Michael Mendonco

Survival of the Fittest
Reasons a language endures, with examples of some classic tongues

Appeals to a wide audience
Get a job done
Delivers new functionality
Fills a niche
Offers a modicum of elegance
Has a powerful user base or backing
Has a charismatic leader

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
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 an exceptional case
- **Structure**: Code for some default action (i.e. the rule) is followed by code for an exceptional case

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

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

```cpp
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
Computer Operations

• **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.

• 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
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
  – \( P_{\text{new}} = C \cdot P_{\text{old}} + k \)
    • \( C > 1 \) = 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
SURVEY
TAKE A TOUR
Computer Organization

- Three primary sets of components
  - Processor
  - Memory
  - I/O (everything else)
Computer Components

• Processor
  – Executes the program and performs all the operations

• Main Memory
  – Temporary storage of data and program (instructions)
  – RAM = read and write but volatile (lose values when power off)
  – FLASH = non-volatile (maintains values when power off) but much slower

• Input / Output Devices
  – Generate and consume data from the system

• Processor carries out instruction from the program
  – Who generates the program?
# Software Process

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

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

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