CSE 1310 - Introduction to Computers & Programming

Introduction

Alex Dillhoff

University of Texas at Arlington

What this course covers

- UNIX Basics
  - Basic shell commands
  - Compiling and running code
- Learn to think algorithmically
  - What must be solved?
  - What steps are involved?
  - How much detail is needed for each step?

What this course covers

- Learn programming
  - What is programming?
  - Basic definitions and concepts
  - Writing code that works
  - Writing code that is easy to understand
  - Debugging
  - Testing
- Programming is not dependent on the language choice

What happens after this course?

- Advanced concepts of C
  - Memory management
  - Pointers
  - String tokenization
  - Structures
- Advanced programming paradigms and concepts
  - Recursion
  - Object-Oriented Programming (OOP)
  - Event-driven
  - Functional
  - Logic Programming
  - Differentiable
  - ...

What happens after this course?

- Theory
  - Algorithm analysis
  - Computational complexity
- How computers work
  - Operating Systems
  - Architecture & Organization
  - Networks
  - Compilers

Layers of Computing – Abstraction

- **High:** Play a game, load a web page, play video
- **Middle:** Render a frame, process login, decode audio stream
- **Low:** Execute specific lines of code, call a function
- **Lower:** Process assembly code translated from higher level code
- **Lowest:** Process electrical signals on the hardware level

What is programming?

- Giving detailed and specific instructions to a computer
  - Computers cannot infer much
- Communicating those instructions to others
  - Documentation
  - If you want others to use your code, it should be well documented
  - You may forget the details of your own code

How do we program?

Command: "Drink some water"
This is a loaded command. Why?

- **High:** Is there water in front of you?
- **Middle:** Does a container need to be fetched?
- **Low:** Turn on the faucet
- **Lower:** Bend joint
- **Lowest:** Wetware

Programming Languages

- Many languages exist
  - C, C++, Objective-C, C#, Java, Python, Haskell, Perl, Ruby, ...
- Why use one over the other?
  - Project requirements
  - Target platform
  - Hardware limitations
  - Third-party support
  - Features

Programming Languages

- The first language takes the longest to learn
- Shorter time to get used to new syntax and features
- The more important skill is learning how to program vs. learning a language

Algorithmic Thinking

An **algorithm** is a sequence of instructions that perform a specific computation.

Algorithms can be expressed in a few ways:

- as **pseudocode**: a high level description of instructions
- as a series of mathematical functions
- as formal code

Algorithmic Thinking – Example

Design an algorithm that determines if a number x is even or odd.

First, define an even number.

Algorithmic Thinking – Example

Design an algorithm that determines if a number x is even or odd.

First, define an even number.

- An integer is **even** if it is divisible by 2
  - If x is an integer, then $2n$ is even
- An integer is **odd** if it is not divisible by 2
  - If x is an integer, then $2n + 1$ is odd

Algorithm 1 Is x even or odd?

1: divide x by 2
2: store the remainder as y
3: **if** y is 0 **then**
4: x is EVEN
5: **else**
6: x is ODD
7: **end if**

Algorithm vs. Program

- An algorithm describes how to compute a task or solve some problem.
- A program is a specific implementation of an algorithm or set of algorithms.
- Algorithms can be formulated and analyzed independent of a programming language.

Succeeding in this Course

- Practice often
- Fail often
- Start small
- Do not rush
- Ask questions
- Value knowledge

Practice often

Programming is a skill like any other, and it requires practice to improve.

Be mindful when practicing. Aim to understand what you are doing and why.

Fail often

- Failure often brings about the best outcomes.
- If you are not failing, you must already know everything.
- Nobody is expecting you to be an expert.

Start small

- Break down problems into smaller pieces.
- Use functions to modularize small solutions.
- Use the smaller solutions as building blocks for larger solutions.

Do not rush

- Programming is just a tool to implement a solution.
- The solution **must** be understood before it can be implemented.
- Rushing towards an implementation without a plan will lead to more problems.

Ask Questions

I genuinely want you to succeed in this course!

- If you are not sure, just ask.
- If you are stuck, just ask.
- If you want a code review, just ask.

Getting help with code

If you need help with your code, you must be able to answer the following questions:

- What is the problem you are trying to solve?
- What is your approach to solving the problem?
- What is the code you have written so far?
- What is the error you are getting?

Getting help with code

The easiest way to show your code is to push it to your repository.

From there, either the TA or myself and provide feedback and push our comments back to your repository.

Value Knowledge

One can often spot your shortcomings by drilling into your solution.

You may recognize and be able to hum a melody, but would you be able to write it down?

Value Knowledge

Take pride in knowing the smallest details, but do not beat yourself up if you do not.

Rubber duck debugging