Syllabus & Algorithms

15-110 – Monday 01/13

Learning Objectives

- Understand the expectations, resources, and policies associated with 15-110
- Understand the difference between an algorithm and a program
- Follow steps provided by an algorithm to perform specific tasks

Course Introduction

Purpose of 15-110

The goal of this course is to introduce you to the field of **computer science**. This includes both programming and more general algorithmic concepts.

The course is broken into five units:

- Programming Skills and Computer Organization
- Data Structures and Efficiency
- Scaling Up Computing
- CS as a Tool
- CS in the World

Staff

Kelly Rivers

Margaret Reid-Miller

26 Teaching Assistants!

Important Resources

Course website: www.cs.cmu.edu/~110/

Also:

- Piazza
- OH Queue
- Gradescope
- Canvas

Syllabus Overview

Read the whole syllabus on your own time!

Course Expectations: unit-hour breakdown, grading scale

Course Resources: office hours, collaboration policy

Course Policies: extensions, grace days, late submissions

And most importantly, take care of yourself!

Take-Home Task

Before Wednesday, you should install the **Python programming language** and the **Pyzo IDE** onto your computer.

If you don't have a computer, that's okay- the cluster machines already have this software installed.

Instructions can be found here.

Algorithms

What is Computer Science?

Video: Computer Science is Changing Everything

https://www.youtube.com/watch?v=QvyTEx1wyOY

Computers are impacting many different aspects of the world - medicine, engineering, shopping, safety, politics, friendships, and so much more.

The core of computer science boils down to two main ideas: **algorithms** and **abstraction**.

Algorithms and Abstraction

Algorithms are procedures that specify how to do a needed task or solve a problem.

Algorithms are used to standardize processes and communicate them between different people.

Algorithms are like recipes, tax codes, and sewing patterns.

Abstraction is the concept of representing ideas at different levels of detail by identifying what is essential.

This can be done by making something more general, or by encoding information at multiple levels.

For example, consider that a textbook is also a book, or how words can be represented as braille or written text.

Activity – Make a PB & J Sandwich

You do: work with a partner to write a list of instructions (an algorithm!) on how to make a peanut butter and jelly sandwich.

We'll test your instructions in a few minutes...

Making a Peanut Butter and Jelly Sandwich

1. Open bag of bread
2. Reach hand in and take out 2 slices of bread
3. Place each slice on a plate
4. Open jar of peanut butter
5. Pick up knife and stick sharp side of knife into open jar
6. Use knife to scoop out peanut butter
7. Wipe and spread peanut butter on one slice of bread
8. Repeat 5, 6, 7 until slice of bread is covered in peanut butter. Close jar
9. Open jar of jelly
10. Pick up knife and stick sharp side of knife into open jar
11. Use knife to scoop out jelly
12. Wipe and spread jelly on other (non-PB) slice of bread
13. Repeat 10, 11, 12 until the slice of bread is covered in jelly. Close jar.
14. Put the peanut butter side of one slice of bread on the jelly side of the other.
15. Pick up and eat sandwich.

Designing Good Algorithms

Designing good algorithms is a large part of computer science. When we represent algorithms as **program code**, we are communicating with a computer to tell it how to do a specific task.

What makes an algorithm 'good'?

- It should specify what is needed at the beginning (**input**)
- It should specify what is produced at the end (**output**)
- It should produce the right output for all given inputs (**correctness**)
- It shouldn't take too long to finish (**efficiency**)
- Others should be able to understand and modify it as needed (**clarity**)

Algorithm Example – Calculating Age

Let's say we want to write a basic algorithm that calculates someone's age based on their birth date.

What's the **input**?

Their birth date, and the current day.

What's the **output**?

A number (their age).

Algorithm Example – Calculating Age

1. Subtract the birth date year from the current year.

Is there an example that won't work with just Step #1?
What if someone was born on December 31st?

1. Subtract the birth date year from the current year.
2. If the month/day of the birth date comes after today's date, subtract 1 from the result.

We only take the action in Step #2 **if** the assumption is true. This is a common algorithmic technique.

Algorithm Example – New Year's Countdown

Now let's say we want to write an algorithm that simulates a New Year's Eve countdown (from 10 to 1).

What is the **input**?

There is no input!

What is the **output**?

The numbers from 10 to 1, backwards.

Algorithm Example – New Year's Countdown

We could write out ten output lines, but that would be tedious.

Alternatively, we could just tell the reader to **repeat** steps!

1. Set a value **number** to be 10
2. If **number** is greater than 0, do the following. Otherwise, go to Step 3.
   a) Output **number**
   b) Subtract 1 from **number**
   c) Repeat Step 2 with new values
3. Output the text "Happy New Year!"

Algorithm Example – Following Steps

Now let's try walking through the steps of the New Year's algorithm, to see if it produces the correct result.

1. Set a value **number** to be 10
2. If **number** is greater than 0, do the following. Otherwise, go to Step 3.
    a) Output **number**
    b) Subtract 1 from **number**
    c) Repeat Step 2 with new values
3. Output the text "Happy New Year!"

Learning Objectives

- Understand the expectations, resources, and policies associated with 15-110
- Understand the difference between an algorithm and a program
- Follow steps provided by an algorithm to perform specific tasks