Course Intro & Algorithms and Abstraction

15-110 – Monday 08/30

Learning Objectives

- Understand the **expectations**, **resources**, and **policies** associated with 15-110

- Define the essential components of computer science, **algorithms** and **abstraction**

- Construct **plain-language algorithms** to solve basic 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.

We'll start from the basics of programming and how computers work, then build up to how computers are used to support a variety of applications in different fields (including your own field of study, possibly).

Staff

Prof. Kelly Rivers

Prof. Franceska Xhakaj

30 Teaching Assistants!

How to Learn in 15-110

1. Attend lecture and recitation. If you can't attend live, watch the posted recording / review the recitation problems promptly.
2. Complete the **exercise** associated with the lecture.
3. Complete the **check-in/homework assignment** associated with the lecture content.
4. Demonstrate your knowledge on the **quiz** associated with the lecture.
5. Demonstrate all collected knowledge in the **final exam**.

The bolded items all contribute to your final grade; see syllabus for details.

More tips here: https://www.cs.cmu.edu/~110/syllabus.html#tips

Active Learning

Lectures will primarily present new content, but we'll frequently use **active learning** to give you a chance to practice new skills.

You do: turn to the person (or people) next to you and introduce yourself! Name, major, why you're taking 15-110.

Important Resources

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

Also:

- Piazza – announcements, questions
- OH Queue – OH questions
- Gradescope – homework handin, read feedback
- Canvas – watch recordings, see grades

Collaboration Policy

We encourage you to collaborate on the assignments! We'll release **collaboration forms** where you can ask to be paired up with other students in the class to find collaborators, or you can just pair up with people you already know.

When you collaborate, **all students** should contribute intellectually to the work, and **each student** must write up their solutions independently. Do not have one student solve a problem and present it to the rest of the group; instead, have all students solve the problem together.

The following actions count as cheating, not collaboration, and lead to penalties: copying, providing answers to others, comparing solutions, searching for answers online, collaborating during quizzes/the final exam.

Frequently Asked Questions - Placement

- **I have no prior programming experience. Can I succeed in this class?**
  - Most of your classmates (usually ~75%) have no prior experience as well. You can definitely succeed, and you're not alone!
- **Should I take 15-104, 15-110, or 15-112?**
  - **Content:** 104 focuses on creative applications of programming. 110 gives a broad overview of programming and computer science. 112 focuses more deeply on programming and problem solving.
  - **Pace:** 104 is paced a bit slower than 110. 110 is paced slower than 112 at the beginning (data, functions, conditionals, loops). 112 is fast-paced throughout.
  - Feel free to contact the professors if you want advice on your individual situation.

Frequently Asked Questions – Deadlines

- **What if I need to turn something in late?**
  - Each exercise and assignment has a regular deadline and a **revision deadline**. Submissions received between the regular deadline and the revision deadline are graded for a max of 90 points.
  - Submissions made before the regular deadline may also be resubmitted for a max of 90 points. Submit early and get feedback so you can fix your mistakes!
  - Students in exceptional situations (COVID/medical/family/personal emergencies) may reach out to the professors to arrange further extensions.

Frequently Asked Questions – Resources

- **I'm struggling with the homework/quizzes. What can I do?**
  - Homework: use **Piazza** to ask short questions and see questions others have asked. Use **office hours** to get one-on-one help. Consider **collaborating** with other students so you have someone to bounce ideas off of. Submit early and view your **feedback** after the regular deadline, then **revise and resubmit**. And above all, remember that **it's okay to ask for help!**
  - Quizzes: complete **practice problems** to get additional exposure to the material. Go to **small group sessions** for more guided review of specific concepts. Go to **drop-in tutoring** for one-on-one review of the topics you struggle with.

Take care of yourself!

Taking care of yourself is incredibly important, especially in tumultuous times. Your personal wellbeing is more important than academics.

Make sure you regularly eat healthy food, get enough sleep, exercise, socialize, and take some time to relax. You will be happier, and you will do better academically as a direct result.

We want everyone to feel welcomed and capable of learning in 15-110. If you feel that the course is negatively impacting your wellbeing, or you do not feel included, **reach out and let us know**.

Take-Home Tasks

Before Wednesday, do the following:

- Fill out the **pre-semester survey** to help us gather more information about your incoming knowledge and interests: https://bit.ly/110-f21-pre
- Read the course syllabus: https://www.cs.cmu.edu/~110/syllabus.html
  - It includes many details we did not cover here. Seriously, read it!
- Install the **Python programming language** and the **Pyzo IDE** onto your computer
  - Instructions can be found here: https://www.cs.cmu.edu/~110/syllabus.html#materials

Algorithms

What is Computer Science?

Computer science is the study of **computation,** and **computational devices**. This can be studied through many different lenses, including:

- Computational **theory** – what are the possibilities and limitations of computation?
- Computational **application** – how can we use computation to fulfill a specific need?
- Computational **discovery** – given data, can we find patterns and answer questions through computation?
- Computational **expression** – how can computation change the way we communicate and engage with others?
- **Critical** computing – how does computation affect our lives, and how should it be regulated?

What do we mean by 'computation'? We can reduce this to two core themes: **algorithms** and **abstraction**.

Read more: https://cacm.acm.org/blogs/blog-cacm/249787-what-liberal-arts-and-sciences-students-need-to-know-about-computing/fulltext

Algorithms and Abstraction

Algorithms are procedures that specify how to do a needed task or solve a problem. They are used to standardize processes and communicate them between different people.

Algorithms can be incredibly powerful, but they're still designed by humans, which means they're vulnerable to human flaws.

Algorithms are like recipes, tax codes, and sewing patterns. When you give someone directions to a location, you're communicating an algorithm.

Abstraction is a technique used to make complex systems manageable by changing the amount of detail used to represent or interact with the system.

This can be done by identifying the most important features of a system and generalizing away unessential features.

Abstraction shows up in many interactions – for example, you can pay for groceries through many modalities (cash, debit, credit, an app), and each is **implemented** slightly differently, but all are just different representations of money.

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.

Before you begin, consider what level of **abstraction** to use. Assume the user knows the ingredients and how to do basic actions, but has no cooking experience.

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

An Algorithm with Moderate Abstraction

1. Before starting: make sure you have a bag of bread, a jar of peanut butter, a jar of jelly, a plate, and a knife
2. Open bag of bread
3. Reach hand in and take out 2 slices of bread
4. Place each slice on a plate
5. Open jar of peanut butter
6. Pick up knife and stick sharp side of knife into open jar
7. Use knife to scoop out peanut butter
8. Wipe and spread peanut butter on one slice of bread
9. Repeat 5, 6, 7 until slice of bread is covered in peanut butter. Then close jar
10. Open jar of jelly
11. Pick up knife and stick sharp side of knife into open jar
12. Use knife to scoop out jelly
13. Wipe and spread jelly on non-PB slice of bread
14. Repeat 10, 11, 12 until the slice of bread is covered in jelly. Then close jar.
15. Put the peanut butter side of one slice of bread on the jelly side of the other.
16. Result: you now have a peanut butter and jelly sandwich on a plate

We assume that the user can identify the ingredients and tools, and knows basic actions, but does not know complex actions.

An Algorithm with Heavy Abstraction

1. Before starting: make sure you have bread, peanut butter, and jelly
2. Get two slices of bread
3. Spread peanut butter on one slice
4. Spread jelly on the other slice
5. Combine slices into a sandwich
6. Result: you now have a peanut butter and jelly sandwich

If we've already taught someone the basics of sandwich-making, teaching them to make a PB & J sandwich is a lot simpler!

Note that we don't define *how* to spread the peanut butter or jelly. Maybe the user will have a different approach to ours.

An Algorithm with Little Abstraction

1. Before starting: make sure you have [specific quantity and type of bread in plastic bag with tab], hand, plate...
2. Define bread as a grain-based substance that has been divided into 1 inch wide parts (slices). Bread is in a plastic container (bag)
3. Open bread bag by gently pulling a plastic tab away from the plastic wrap.
4. Define hand as the appendage at the end of your arm. Define fingers as the smaller appendages at the end of your hand
5. Define plate as a hard, flat, usually-circular surface
6. Move hand into the opening in the bread bad. Move fingers to close position around the top bread slice
7. Lift hand until it is outside of bread bag.
8. Move hand over the plate, then down so that it is touching plate. Open fingers around the bread slice.
9. Repeat steps 5-7 so that a second bread slice is on the plate.
10. …

If someone doesn't even know the basic assumptions (a toddler, or a robot), we'll need to define every item used and how to execute even the simplest steps. And we're still making assumptions here!

Designing Good Algorithms

Designing algorithms at the right level of abstraction is a large part of computer science. When we represent an algorithm as **program code**, we communicate with a computer to tell it how to do a specific task.

What are the core parts of an algorithm?

- It should specify what is needed at the beginning (**input**)
- It should specify what is produced at the end (**output**)
- It should specify how to get from the beginning to the end (**steps**)

Algorithms can be Measured Many Ways

It's not always good enough to make an algorithm that works. There are **many ways** to solve any given problem, and these different approaches can be compared to determine which is 'best'.

Here are some metrics we might use to assess an algorithm:

- 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**)
- It shouldn't be broken by unexpected behavior (**robustness**)
- And more!

Post-Lecture Feedback

We encourage all students to fill out the **post-lecture feedback form** after each lecture, to let us know what you're still confused about.

We'll review common confusion points at the beginning of the following lectures.

Link: https://bit.ly/110-f21-feedback

Learning Objectives

- Understand the **expectations**, **resources**, and **policies** associated with 15-110

- Define the essential components of computer science, **algorithms** and **abstraction**

- Construct **plain-language algorithms** to solve basic tasks