Lecture Notes (week 1)

Introduction(s)

My name is Evan Drake, I am a CSU Chico graduate and I have lived in Chico my whole life. This is my first year teaching my own class of any subject or level. My interests include:

- Compiler research
- AI research
- Reading Books (all genre, though I'm not a sci-fy fan funnily enough)

About the class

I will explain the way I give lectures here.

I gravitate towards the mathematical / logical side of computer science so I will tend to teach the theory more heavily. If this scares you, do not worry I will do my best to treat all learning styles within lecture and lab. If you do not understand what I am doing at anytime during the class, please pose a question of simply say that you do not understand. I plan on giving homework every week and random quizzes[1]. I plan on having two tests throughout the semester: A midterm containing the first half of the course; A final containing the second half of the course.

Homework

All Homework will have two portions: a written portion where you will put your theory; a code portion where you will prove your theory. It is worth noting that your theory is more important to me than your actual code. Under no circumstances will I except late work[2].

Extra Credit

I am a fan of extra credit, so I will be excepting it throughout the semester. Note that there will be three ways of gaining extra credit:

1. Bring me a peice of computer science research I have not seen (the easiest).
2. Bring me an extensive personal project of some kind within the feild of comp-sci.
3. There will be various challenges throughout the semeseter for your completion.

Good study habits (small tangent), notes, practice, types of learning.

Lecture One

This lecture will introduce you too programming and what it takes to begin[3].

What do you need to program?

What is programming really? Programming is a kind of instruction creation technique for machines utilizing a translation mechanism. In order to program one must first understand that machines do not understand information the same way a human does. The second step here to analyze how a machine does understand logic. This use of "understand" is by definition, hyperbole as the machine never understands logic it merely follows it. This is akin to the way your car does not understand the physics of its own movement it merely exists in the definable state.

Hardware

RAM - random access memory provides the addressable space that allows us to design to programs. RAM can be thought of as a large grid like structure, composed of boxes in which will be our data.

---

[1] note the quizzes are for me not your grade

[2] if you have an emergency please come talk to me

[3] more specifically what you will need for this class

see more info RAM is where all the programs one writes will live once they are running, as such it is important to understand the basic structure of it.

The second piece of hardware that any good programmer should be aware of is a CPU or **central processing unit**. While this is not the only form a processing unit can take it is the most common. A CPU is where all our logic will eventually be acted on.

Software
Recall when I explained the CPU I used the word **eventually**. The reason I said that is because the language of the CPU and the language we will be writing in are different. One can imagine that the CPU speaks some form of Latin and we as humans speak English. In this situation there is a couple of ways for us to communicate, the easiest is to use a translator. In our case the translator is called a **compiler** (or **interpreter** in our case). The translator we will be using in this class is called Python3.

In the event you find this subject interesting please ask me about it as this is an area of interest for me.

Logic
The last and possible most important piece of the puzzle is the logic we wish for the CPU to enact. This comes in all kinds of flavours but the most common is a (sequential) programming language (such as python). Note that there are many kinds of programming languages in the world.

Logic: pseudocode (activity)
Brief introduction to pseudocode and peanut butter sandwich activity.

Editors
For this class we will be using VSCode, however I will show most of my code in (Neo)VIM or Zed. No matter what editor you are using I will do my best to assist you in getting it setup in the first lab.

Lecture two
Here we will discuss pseudocode further introducing Variables and Data types in a brief manner.

Why pseudocode
The reason we need pseudocode is that it can more descriptive than any programming language and it is easy to mock up quickly. As pseudocode is essentially math, one can depict allot of information in relatively few symbols. Here we will discuss some basic notation and logic. Note that at the bottom of this document there is a **nomenclature** section. Here we will introduce pseudocode, representing only simple bindings and utilizing basic arithmetic.

Bindings
First we define **binding**: A binding is a statement that connects some **name** to some **value**.

$$A \leftarrow B$$

In Eq. 0 we have *defined* the name A to be equal to the value B. The below is a more common example of a simple binding.

$$x \leftarrow 0$$

Here we have bound the value of zero to the name x. It is important to note that the left side of the binding operator is always a name while the right side is more complex.

Expressions
Expressions are more complex than bindings but are more common in mathematical notation. A most basic expression can be seen below.

$$x + (6 - 7)$$

Recall Eq. 0, we can see this equation utilizing both the addition operator and the subtraction operator. Eq. 0 allows us to expose a key trait of expressions, they can be evaluated (or solved).

$$x + (-1)$$

Looking at Eq. 0 we can see that $(6 - 7)$ has been evaluated as (-1), and the overall expression has become more well defined. This exposes another trait of expressions, namely that if variables are not well defined at evaluation time expression can not be evaluated. Different languages have different strategies for solving this phenomenon but that is beyond our scope.

Statements

Recall Eq. 0 and the basic structure of a bindings. Bindings in the theory of computation are regularly referred to as **statements**. We define statements here as: *A name, expression binding where the expression is partially evaluated at binding time.*

Variables

Here we can think of a **variable** as synonymous with binding. Note there is differences but we can ignore them for the scope of this class.

Numbers

While data types are mostly specific to computer science we find their root in mathematics.

$$z \in \mathbb{Z} \tag{1}$$

$$r \in \mathbb{R} \tag{2}$$

Let us use two numbers from different rings[4] for our example. See that Eq. 1 & Eq. 2 declare variables z & r from the groups $\mathbb{Z}$ & $\mathbb{R}$.

$$z + 1 \tag{3}$$

$$r + 1 \tag{4}$$

Note that for both Eq. 3 & Eq. 4 the logic is sound as the number one is an integer but also a real number, so both z & r have precise values on evaluations. What if we were to use the same expression but with an irrational number?

$$z + \sqrt{2} \tag{5}$$

$$r + \sqrt{2} \tag{6}$$

See that Eq. 6 the logic is still sound as r is a real number. Note that the operation $+$ is well defined for real numbers as the addition of both expressions resulting in a real number. See that Eq. 5 is not logically sound as z is defined as an element of *integer*. Note that the operation $+$ defined for integers evaluates the addition of both expressions resulting in an integer[5].

Data Types

- Numbers: Integers, Floats (Real numbers), Complex numbers ...
- Characters
- Booleans
- Collections

[4] A *ring* in mathematics is a group (or set) of numbers.

[5] It is important to note that this expression (while not mathematically valid) will work fine on a computer, as the operation will compose the *floor* function with that of the addition function.

Characters
These are single letters (for our purposes english letters). Not a complicated data type just a single ascii letter. https://www.techtarget.com/whatis/definition/ASCII-American-Standard-Code-for-Information-Interchange

Boolean
Booleans are a true or false logical statement with complex interactions. For now we will simple use these as simple single boolean values (later we will define operations for booleans though).

Collections
Collections can simply be understand as Collections of the other Base (primitive) types.

Nomenclature
$A \longleftarrow B$ - denoting a binding where A will represent the value of B
$A \longrightarrow B$ - read A **implies** B, denoting the that the statement A being true implies that B is also true
[ ] - denoting a set
$\mathbb{Z}$ - set of all integers (real numbers or their negatives)
$\mathbb{R}$ - set of all real numbers (rational numbers/irrational numbers, including fractions and their negatives)
$i \in I$ - denotes that i is an element of the set I