CST0006 – Computer Programming Foundations

INTRODUCTION to WEEK03

In this lecture and lab section we are going to learn about algorithms, initial programming concepts and a high-level look at solving problems. Today's topics are difficult to cover in only one lecture. The topics we're covering today have taken over 300 years to evolve and cover an insane amount of information, so we're only going to be able to scratch the surface. We will also start using the book as a guideline, and will be covering two chapters per week. So this week, read at least the first two chapters.

People

Many people can be accredited for the development of the computer and computing science. However, there are some key people that have made huge leaps that we all benefit from to this day. Here are five of them.

Gottfried Leibniz (1646 – 1716) Inventor of mechanical calculators, developed logic in a formal, mathematical sense with his writings on the binary numeral system.

George Boole (1815 – 1864) Among other published material his writings on Boolean algebra in 1854 is said to have laid the foundations for the information age.

Charles Babbage (1822 – 1837) Originated the concept of a programmable computer. Designed the idea of the Analytical Engine in 1837, but due to lack of funding was unable to build it.

Ada Lovelass (1815 – 1852) Worked with Charles Babbage and created the first algorithm intended to be carried out by his machine. She is considered to be the world's first programmer.

Alan Turing (1912 – 1954) Considered to be the father of computer science and artificial intelligence. Built the first (re)programmable computer in 1936, which he called the a-machine (automatic machine) with which the world now has the term "Turing Complete". Turing completeness is the ability for a system of instructions to simulate a Turing machine.

Computing Science

A programming language that is Turing complete is theoretically capable of expressing all tasks that can be accomplished by computers; nearly all programming languages are Turing complete if the limitations of finite memory are ignored.

What it basically boils down to is with only three things you can solve any problem.

Repetition (For loops, While loops, Recursion)

Conditional Branching (IF/THEN/ELSE, CASE and GOTO)

Memory to hold a value in memory and change it

Automata Theory

Automata theory is the study of computational problems that can be solved by abstract machines and automation. The easiest way to think of this is to break it into two categories, States and Transitions. For each state, if given valid input, will accept and respond by transitioning into a new state or reject the input and wait for the next instruction. An elevator is a perfect example of this. If you are on floor 1, and you push 2 it will read the 2, accept the input, it will jump to the next state by closing the doors, and moving to floor 2. Now if you're on floor 2 and you press 2, it will ignore this input as there is nothing for the elevator to do, and wait for the next instruction.

Algorithms

To understanding programming theory it is important to understand what a program actually is and what it does. In simple terms, a program a sequence of written instructions (known as statements) to solve a problem or set of problems.

It is important to identify the problem you want to solve, then formulate a solution by breaking down a solution into a series of smaller steps that will allow you to solve that problem.

So then, what is an algorithm?

An algorithm is the set of steps you need to preform to solve a problem. This algorithm can be an advanced mathematical equation to solve a problem, or automating some mundane that you have to do everyday.

A perfect example to explain an algorithm is, say you're in a strange new City on a business trip. It's late and you're craving a slice of pizza. You see a drunk guy coming out of a bar and decide to ask him how to get to the nearest pizza place. Unfortunately he is a close-talker and has bad breath, but he manages to give you a set of directions on how to get to the nearest pizza place. These steps are a perfect example of an algorithm that will solve your problem. If you follow the algorithm, you will get to eat. If you don't follow the algorithm you'll end up being the victim of a crime scene. Much like this scenario, an algorithm is a sequence of steps that you can easily program to solve a problem.