The Mechanical Mind
Lecture 1: Computers

Lecturer for the course: Wylie Breckenridge, Room W6A 733, ext. 8859.
Reading for the course: Tim Crane (2003), *The Mechanical Mind*, 2nd ed. (London: Routledge), pp. 42-148.

1. In these three lectures we are interested in the relation between *computers* and *minds*, and between *computing* and *thinking*. Specifically:

   a. Can computers think?
   b. Do we think by computing?

2. We shall assume that we have such things as minds, that the mind is that by which we think (have beliefs, desires, and so on), and that thoughts are states of the mind (i.e. mental states).

3. We shall also assume that our thoughts, at least some of them, are *representational* – they are *about* the world or things in the world. My belief that Sofia is the capital of Bulgaria, for example, is *about* Sofia and Bulgaria; it *represents* that Sofia is the capital of Bulgaria.

4. We will not take a stand on the mind-body problem – the problem of how the mind and body are connected. (We know that they *are* connected – doing things to one's body can affect one's mind.) Are our minds identical with our brains? On the one hand, it seems obvious that we are entirely matter; but on the other it seems that we are more than that (love is more than just a complex chemical reaction). Here are some possible positions that one might take:

   Materialism/physicalism: The mind is just the matter of the brain organized in a complex way.
   Dualism: There are two kinds of things, the material and the mental.
   Idealism: Everything is mental.

   The questions we are interested in arise independently of this (very difficult) issue.

5. In this lecture we will talk about computers, in the next we will talk about minds, and in the third we will talk about the connection between computers and minds.

6. What is a computer?

7. According to Crane, a computer is a device which, given a representation of the argument of a function, generates a representation of the value of the function, by following an algorithm. It doesn't matter what a computer is *made of*, just what it *does*.

8. We need to clarify two things here: what a *function* is, and what an *algorithm* is.

9. What is a function? Given a set of things, called the *domain*, and another set of things, called the *codomain*, a function is a pairing up of things in the domain with exactly one thing in the codomain. If f is a function that pairs a in the domain with b in the codomain we say that $f(a) = b$; or that when the *argument* of f is a, the *value* of f is b. Sometimes we specify how f pairs things up by listing a set of pairs: {(1, 2), (2, 3), (3, 4), …}. Sometimes we do it by giving a rule: $f(x) = x + 1$.

10. The domain of a function may consist of *pairs* of objects. For example, the addition function: $f((x, y)) = x + y$ (more simply: $f(x, y) = x + y$).

11. What is an algorithm? It is a method for finding the value of a function for any given argument, such that (a) at each step it is clear what to do, (b) it takes only finitely many steps. That is, it is a method for *computing* the function.

12. When there is such a method for a given function, we say that the function is *computable*. Here are two computable functions: the prime factor function; the division function.

13. Note:
    a. There may be more than one algorithm for computing a given function.
    b. An algorithm might appeal to other algorithms along the way.
    c. Functions and algorithms are different kinds of things: an algorithm is a method for computing the value of a function; a function is *not* a method for computing the value of an algorithm.

14. Early last century, mathematicians tried to get more precise about what an algorithm is. In the 1930s, Alan Turing proposed that an algorithm is what can be performed by a *Turing machine*.

15. What is a Turing machine? It is a machine with (a) a tape divided into squares, (b) a device for reading and writing symbols on the tape (which we can take to be restricted to '0' and '1'), (c) the ability to move the tape left or right, one square at a time, (d) a set of internal states, and (e) a machine table – a set of instructions for what to do given the symbol currently on the tape and the internal state the machine is currently in (we can think of this as the Turing machine's *program*). (See Penrose, *The Emperor's New Mind*.)

16. Turing machines can compute functions. If a Turing machine has two internal states, A and B, and the following machine table, it will compute the function $f(x) = x + 1$:

		Symbol on tape	
		'1'	'0'
Internal state	A	Change to state B Write a '1' Move tape to right	Stay in A Write a '0' Move tape to right
	B	Stay in B Write a '1' Move tape to right	Change to state A Write a '1' Stop

17. A very interesting fact: a Turing machine's machine table can be encoded into a string of bits, which can then be printed onto tape and fed into another Turing machine. We can thus do away with all but one Turing machine, a *universal Turing machine*, which can mimic all others.

18. Turing (and, independently, Alonzo Church) proposed that: to be an algorithm is to be capable of being executed on a Turing machine. This is called the *Church-Turing thesis*. It is supposed to makes the idea of an algorithm unmysterious. (Question: how can algorithms such as that for baking a cake be performed by a Turing machine?)

19. If the Church-Turing thesis is correct then we can take a computer to be a Turing machine (perhaps a *universal* Turing machine). The symbols initially on the tape represent the argument of a function, the symbols finally on the tape represent the value of the function for that argument, and the computer generates the latter from the former by following an algorithm (according to its machine table). This gives us a nice concrete way to think about computers.

20. Note that a computer is a causal mechanism that operates on representations.