Part 1: Introduction to Computers and Programming

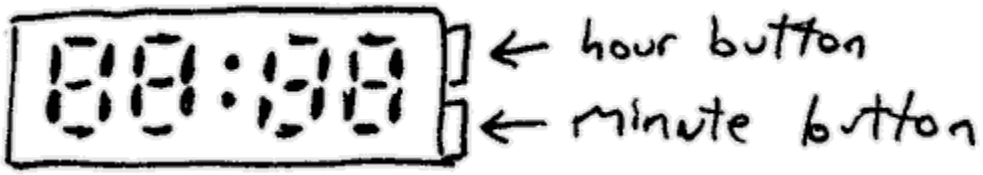
Chapter 1: Machines, Input/Output, Storage, and Logic

Introduction

This chapter is about computers and how they work. It is an overview of a great many concepts like input/output, logic, storage, instructions, machine language, etc. All of these concepts are written for novice computer users. There will be no questions, exams, or assignments at the end of this chapter. You are directed simply to read and *get into da groove!* Throughout programming you will come back to these things over and over again so there will be plenty of time for quizzing you about your knowledge of them. At this point I just want to break the ice.

Simple Example: Digital Clock

You know how to control a computer at a high level, user input. You click this, type that, or push this there and the computer responds to your action. But what is really happening when you do these things? A computer program is receiving your input as it requested and dealing with it as it sees fit. There are four things computers and any useful machines have: **input**, **output**, **storage**, and **logic**. These are the things that help them do the tasks they are meant for. To simplify things I'm going to simplify the machine we're speaking of. Rather than a computer, I'm going to use a digital clock:



I/O

Immediately you can recognize that the clock has input through it's hour and minute buttons. No input would be complete without some way of acknowledging it, **output**. In the case of our clock, we have a LED screen capable of displaying a 24-hour clock. Input/output seem to go hand in hand wherever you look. When something goes in, something comes out. In lew of this, the two things are usually globbed together under one term: "**i/o**", an abbreviation for

input/output.

Storage: Temporary and Permanent

We know that the clock has a **i/o**, but where does it keep the current time? The time seen is simply a display of the data where the clock keeps the current time. Internally this machine has some place to **store** this data. This is, obviously, known as **storage**.

When the power to the clock is shut off, the time resets. This means the data for the time is in **temporary storage**. In computery terms, our version of **temporary storage** might be defined as *data storage which is reset, blanked, or invalid when necessary power is lost or shut off*. Sounds complex, but just think: we take out the battery and the clock goes dead and when we put it back in the clock is at `00:00`. It has been reset due to lack of power. The time data is not kept in **permanent storage**.

Philosophically, nothing is permanent. In our terms, **permanent storage** would simply mean the opposite of **temporary storage**. I would define it as *data storage which can only be destroyed or change by explicit tampering or unforseen accidents*. Our clock has no **permanent storage**, but I think you get the idea. I'll explain a computer's version of this later on.

Any given **storage**, regardless of permanence, will have one of two different access privileges: **read-only** or **read/write** (I've never heard of **write-only**, but it *may* exist in some 3rd world country). Storage that is **read-only** can never be changed while **read/write storage** can. It's that simple.

And lastly the concept of **storage** could be described simply in terms of **i/o** from a **storage machine/device**. Say our clock's **storage** is a little teenie circuit inside. Reading data from that **storage** would be output from that circuit received as input; and vice versa when writing. I chose to keep **storage** a seperate concept for simplicities sake because it's going to be a major aspect for you in programming. You'll *always* need to store data, and it's best to think of it as a container of stuff rather than a bunch of hoses :).

Logic

Our machine can access **input**, display **output**, and keep a current time in its **storage**; but how does it bring all these things together

and control them? **Logic** of course. You can think of **logic** as *instructions that encompass reading input, writing output, and making decisions*.

Notice that I didn't mention **storage**. Again, that aspect is enveloped in **i/o**. Because **input** from **storage** and **output** to **storage** is simply implicit. Say, for example we have a jar of pennies for storage. I would rather say *add a penny to the jar* than *output a penny and feed it as input to the jar*. That two-way, give/take transaction is implied.

Logical instructions for machines of any kind are very explicit and finite. There is no creative meanduring. Let me give an analogy. Say you write some directions to get to your home from some known freeway. Now, if you gave these to a computer it would act them out one by one in perfect sequence and to the letter. A human, especially a man, might try to take short-cuts. That's what instructions are on a machine. A list of actions to be carried out … it's that simple! What you do when you *program* is write or alter those instructions for your specific purpose.

Now that I've rambled a bit, let's look at the logic for our clock.

- If the hour button is pressed, increase the current time by one hour.
- If the minute button is pressed, increase the current time by one minute.
- If a second has passed, update the current number of seconds.
- If the number of seconds is greater than or equal to 60, increase the current time by one minute and reset the number of seconds.
- Update the display with the current time.

Even though I can sum up the clock's logic in five points, implementing it can be fairly complex. First of all, how does the clock know when a second has passed? A lot of these lower-level details may seem superfluous, but sometimes you'll need to know them. In the context of simply warming you to programming, I'll gloss over super-specifics like these.