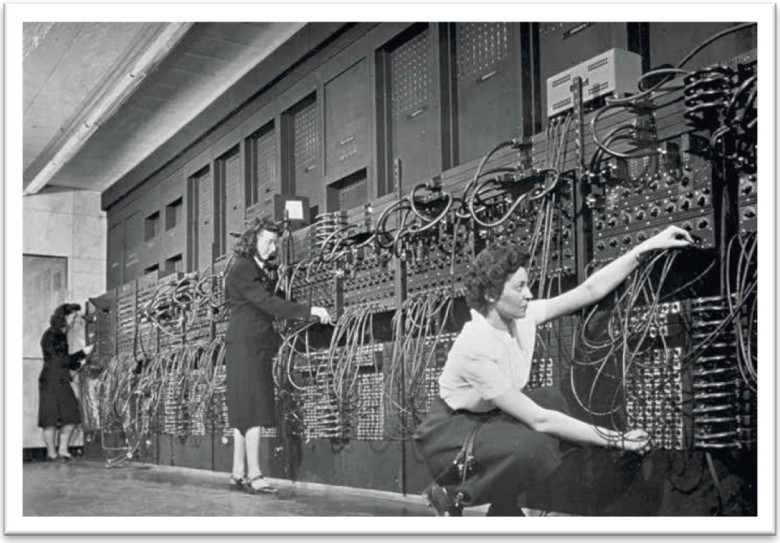
A QUICK GUIDE TO PROGRAMMING YOUR COMPUTER

STEVE HARRIS

PART I. The Computer. Almost everything that you do on the computer isn't real: it's a metaphor. There's no such thing as a desktop. There are no folders or directories, neither icons nor music, neither pictures nor emoji. They are all metaphors. The only real parts of the computer are the circuit boards, the memory, the display, and the peripheral devices (mouse, keyboard, printer, speakers, etc.). So, learning to use and program a computer is all about learning the metaphors.

Let's start with what's real. In 75 years, it hasn't changed substantially.

During the Second World War, the US government developed the computer. It was part of the effort to break enemy codes and it looked like this:



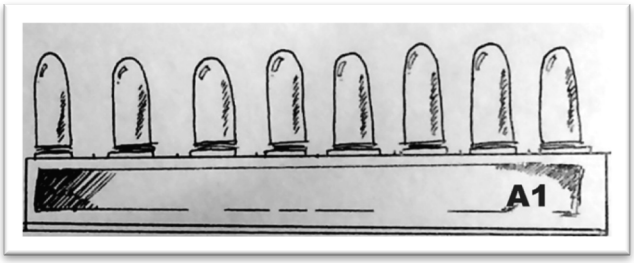
It was called ENIAC. And it was made up of two basic parts. The first is the **memory**. And the second is the electro-mechanical circuit board that does the actual computing—called the **motherboard**.

Here are four of ENIAC's six programmers holding the main components:

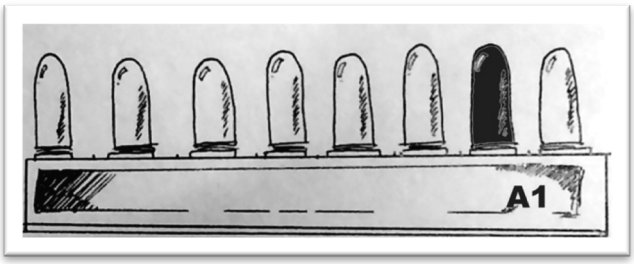
Second from left, the unidentified programmer is holding a byte of memory, here made up of eight vacuum tubes. A tube can be either on or off. Programmers invented conventions to represent numbers and letters. They needed eight tubes to do the job. As an example, the letter A is represented by off-on-off-off-off-off-off-on. To make it easier to understand, they represented off as "0" and on as "1" which is called **binary code**. So, "A" is 01000001. You could wire up a tube in order to discover

whether it was on or off. If it was on, the electric current continued through the wires. If it was off, the current stopped. You could also wire up the tubes to turn them off or on. That's what's meant by **programmable**. Putting data into ENIAC was a matter of electronic circuitry.

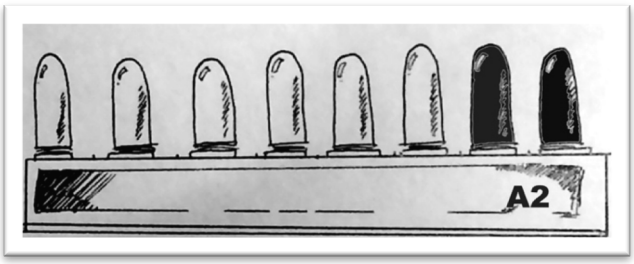
Let's say you wanted to add 2 and 3. First, you choose a **memory address**, which is another way of saying where in the wiring diagram a particular set of eight vacuum tubes sits. Let's say you choose the byte at position A1. It has eight tubes wired from right to left.



Now, you turn on some tubes in order to store the number two (which in binary code is 00000010). Starting at the right, you count leftwards. It's like the columns in base 10: on the right is the tens column, then the hundreds, then the thousands, and so on. In binary, it's the one column, the two column, the four column, and so forth. The number two has no one's (so the first tube is off), and one two (so the second tube is on, which I've indicated by coloring it black).



You can see now that the tubes are off-off-off-off-off-off-on-off, representing in binary the number 2. Choose another memory address, say A2. Wire it to store the number three, which is made up in binary of one and two (00000011).

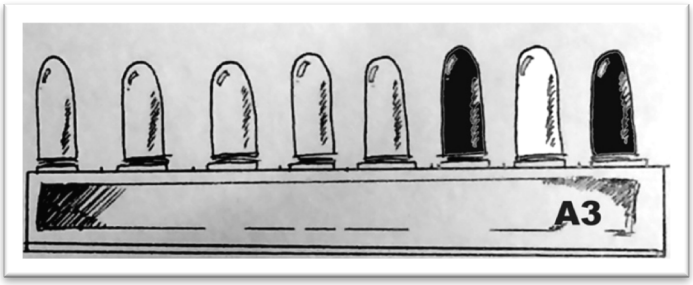


Now you have two numbers in storage. As long as the power doesn't go out and the tubes don't fail, you can store those numbers for a long time. Storage of data was one of the great breakthroughs in computing.

Next, you need to add these numbers. Luckily, they're not in base-10, but in base-2, which is called binary. So 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10. Remember, there are no two's in binary, only ones and zeros! So you can wire your motherboard to send an electronic signal to A1, position 1 (on the far right), to see if it's on or off. If it's on, then you are dealing with the number 1. Then send the current to A2, position 1 and see whether it's on or off. Through a series of switches in

your wires, you can "add" the positions. So an off tube (A1-1) that connects to another off tube (A2-1) yields a third off tube because off and off equal off, or 0 + 0 = 0. Put that answer into a third memory location, say A3, position 1.

Now you can go to the second position of A1 and the second position of A2 and put the answer into A3-2. And so on. When you're finished with all eight tubes, your answer is waiting for you in A3:



Because you read binary, you know it equals five.

And that is how the first electronic calculators were made.

By this time, the big rack of tubes had been replaced by integrated circuits. They contained transistors that could turn current off and on. Integrated circuits were invented by Jack Kilby of Texas Instruments and were introduced to the world in 1958. They changed radios, televisions, and computers forever.

Figure 1. Casio calculator, 1966

Since then, computers have been getting smaller and smaller. Remember the ENIAC programmer holding the byte? That byte could now fit thousands of times inside the dot on the letter *i*. But its basic function hasn't changed. And neither has the basic function of the motherboard. It still adds, subtracts, divides, multiplies, and moves data around.

Until fairly recently, people programmed computers with switches and wires. It wasn't obvious how to use letters on a keyboard to connect circuits to each other. Computers were programmed in **machine language**. It looks like this: "169 1 160 0 153 0 128 153 0 129 153 130 153 0 131 200 208 241 96." That program prints the letter *A* one thousand times on an ATARI computer. Machine language used substantially less memory, which mattered when memory was expensive. The first Macintosh computer had 216K bytes of memory (K means 1000). The system (called the **operating system** or **OS**) was only a part of that. Today, the Mac OS is many hundreds of gigabytes. A gigabyte is one billion bytes.

Memory got cheaper. Chips got faster. And the wires that carried data got more efficient. So rather than programming in machine language, people wrote **programming languages**. In November of 1954, FORTRAN was introduced. Now, instead of wiring vacuum tubes to store a number, you could program using your language rather than the computer's. It worked not with a keyboard, but

with punch cards. A programmer punched out little holes corresponding to data, instructions, or comments. (This is how I started programming!) So what does a program *do*?

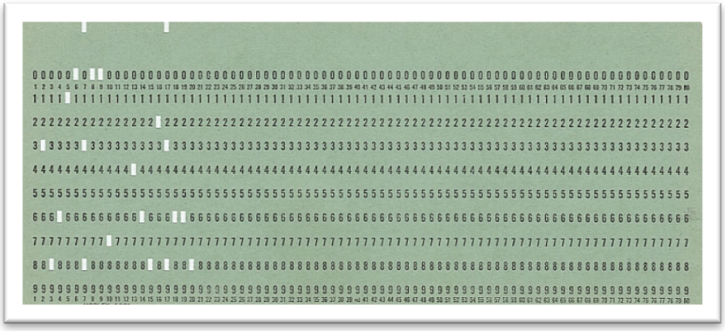


Figure 2. Fortran input card

First, you must tell the computer how much memory you will need. Let's say you want to add 2 to 3. Well, how much room will the computer need to store the number 2? We saw that on ENIAC it only needed eight tubes. But what if you want to store a number like 3.1416. How does computer store decimals?

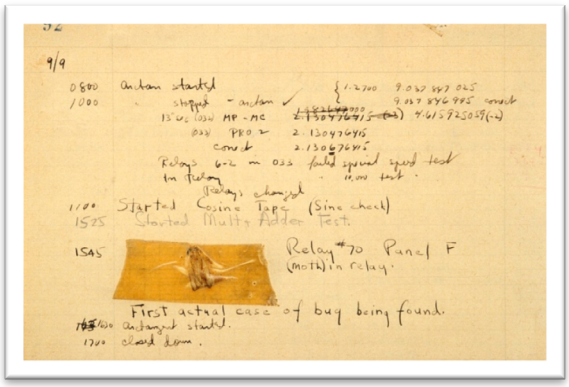
All of that had to be figured out and conventions established. A simple integer like 2 required eight bits (just as it did on ENIAC). Bigger numbers like two billion required more—sixteen or thirty-two bits.

So, you **declared** your variable. "Computer," you said in FORTRAN, "I need an integer of eight bytes. Let's call it *x*." You would type INTEGER X. The computer would then read your code and set aside one byte of data for *x*, then wait for you to tell it what value to put into *x*'s memory address.

Picture a mailbox. That's like a memory address. Instead of a mailman reading "Smith" on the side, a computer reads "A1"; and it sets aside one mailbox for your variable called *x*. Just like ENIAC. Now you can start to see what I meant by the claim that most of what you do on a computer is metaphor. A variable is just a metaphor for a memory address.

Now you need another mailbox, INTEGER Y, and a third to put your answer into, INTEGER Z. Then you put **data** into them. X = 2. Y = 3. And instruct the computer to add them together: Z = X + Y. Tell the computer to write the answer on the printer: WRITE Z.

Obviously, there are lots of ways to accomplish the same goal. You could declare a single integer and tell the computer that Z = 2 + 3, then tell it to write Z. Much simpler. And simplifying code was very, very important then, because memory was expensive and it took longer to run instructions through a chip. So, programmers would **bum** their code. And they would try to get rid of **bugs**.



Just so you know, in the beginning, these were actual bugs. On September 9, 1947, the massive IBM computer at Harvard shut down. The lead programmer, Grace Hopper, and her team looked

through the valves and wires and on Relay #70 of Panel F found a moth. The noun turned into a verb, and now you **debug** code.

Fortran was a great language. You can still learn it and run it on your computer. If you use a Macintosh, you already have a fully-functioning UNIX operating system underneath the Mac OS. On Windows, you can download a fully-functioning version of LINUX. Both UNIX and LINUX understand FORTRAN and will execute your program.

But FORTRAN was difficult for novices and students to learn. So other languages were written. A **programming language** is a set of instructions that *you* can understand. Those instructions are then turned into machine language behind the scenes. Today, there are thousands of programming languages. It's not easy to know which one to choose to learn.

OPERATING SYSTEMS?

Does it matter which operating system you have? Yes. The **operating system** or **OS** is a program that lets you interact with the machine. When the computer starts up, it runs a few programs quickly behind the scenes to determine if there is a keyboard, a monitor, a printer, and so forth. Then it looks for a program called the OS. The OS is really just a very large program, like MS Word or Firefox.

That means that your computer is not the same thing as your OS. If you want to change the OS on your computer, you can. You can even run two or more OS's on the same computer at once. For example, I run LINUX on a Macintosh computer as well as on a Dell computer. Windows 10 lets you download LINUX and run it while you are running Windows. You can run your Windows programs just as always, then switch over to LINUX and run some of the thousands of free programs it offers. Click this link to learn how or search Microsoft.com for *linux*.

On a Macintosh, open your Applications folder (Check the *Go* menu). Then open the Utilities folder. In there, you will find the application **Terminal**. Open it. Now you are running unix.