ISSN 1754-3657 (Print)
ISSN 1754-3665 (Online)

CS4FN

Computer Science for Fun

Issue 28

Cunning Computational Contraptions

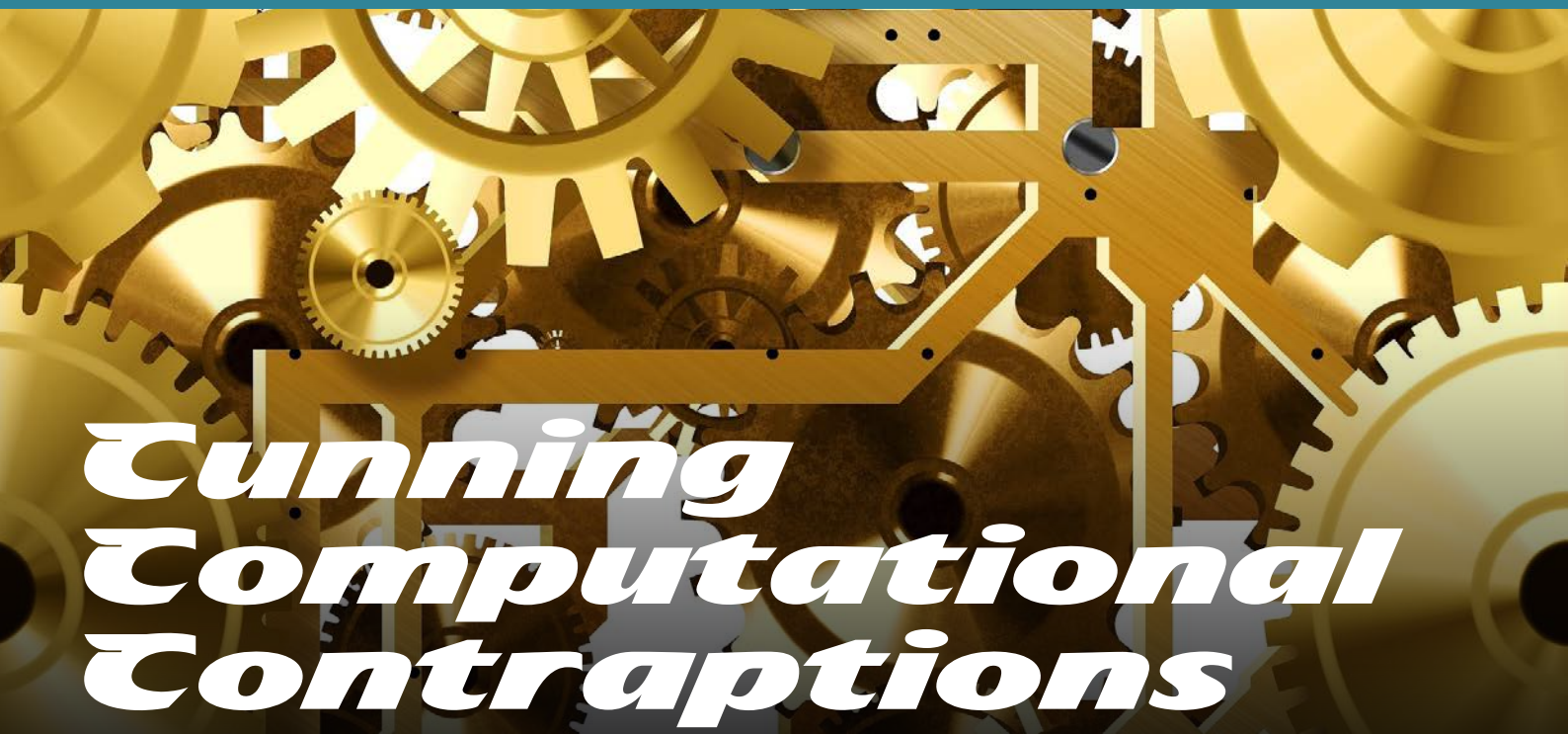
Babbage's adder

Custard computers

I Ching binary

cs4fn.blog/cunningcontraptions/

Queen Mary
University of London



Cunning Computational Contraptions

From the dawn of humanity people have created cunning computational contraptions.

Some have been bodged together, Wallace and Gromit style with little hope of working long term, others have been marvellous miracles of engineering. Some just demonstrated an idea, some entertained, the best have been immensely useful and completely changed the way we live. This issue is about contraptions with links to computation, if not always computers themselves. Charles Babbage gets a special place as, along with other computational inventions, he was the one who first worked out how a general-purpose computer might work and even designed one that would have worked if built despite it being the age of cogs and steam. We also explore how to do computation with marbles and custard, though not together…

Image by dawnydawny from Pixabay

An ode to technology

Cunning contraptions date back to ancient civilisations.

People have always been fascinated by automata: robot-style contraptions allowing inanimate animal and human figures to move, long before computers could take the place of a brain.

Records show they were created in ancient Egypt, China, and Greece. In the renaissance Leonardo designed them for entertainment, and more recently magicians have bedazzled audiences with them.

The island of Rhodes was a centre for mechanical engineering in Ancient Greek times, and the Greeks were great inventors who loved automata. According to an ode by Pindar the island was covered with automata:

The animated figures stand.
Adorning every public street.
And seem to breathe in stone, or
move their marble feet.

Image by Devanath from Pixabay

Core rope memory

by Jo Brodie and Paul Curzon, Queen Mary University of London

Weaving, in the form of the Jacquard loom, with its swappable punch cards controlling the loom's patterns inspired Charles Babbage. He intended to use the same kind of punch card to store programs in his Analytical Engine, which had it been built would have been the first computer. However, weaving had a much more direct use in computing history. Weaving helped get us to the Moon.

In the 1960s, NASA's Apollo moon mission needed really dependable computers. It was vital that the programs wouldn't be corrupted in space. The problem was solved using core rope memory.

Core rope memory was made of small 'eyelets' or beads of a metal called ferrite that can be magnetised and copper wire which was woven through some of the eyelets but not others. The ring-shaped magnets were known as *magnetic cores*. An electrical current passing through the wires made the whole thing work.

Representing binary

Both data and programs in computers are stored as binary: 1s and 0s. Those 1s and 0s can be represented by physical things in the world in lots of different ways. NASA used weaving. A wire that passed *through* an eyelet would be read as a binary 1 when the current was on but if it passed *around* the eyelet then it would be read as 0. This meant that a computer program, made up of sequences of 1s and 0s, could be permanently stored by the pattern that was woven. This gave read-only memory. Related techniques were used to create memory that the computer could change too, as the guidance computer needed both.

The memory was woven for NASA by women who were skilled textile workers. They worked in pairs using a special hollow needle to thread the copper wire through one magnetic core and then the other person would thread it back through a different one.

The program was first developed on a computer (the sort that took up a whole room back then) and then translated into instructions for a machine which told the weavers the correct positions for the wire threads. It was very difficult to undo a mistake so a great deal of care was taken to get things right the first time, especially as it could take up to two months to complete one block of memory. Some of the rope weavers were overseen by Margaret Hamilton, one of the women who developed the software used on board the spacecraft, and who went on to lead the Apollo software team.

The world's first portable computer?

Several of these pre-programmed core rope memory units were combined and installed in the guidance computers of the Apollo mission spacecraft that had to fly astronauts safely to the Moon and back. NASA needed on-board guidance systems to control the spacecraft independently of Mission Control back on Earth. They needed something that didn't take up too much room or weigh too much, that could survive the shaking and juddering of take-off and background radiation: core rope memory fitted the bill perfectly.

It packed a lot* of information into a small space and was very robust as it could only break if a wire came loose or one of the ferrite eyelets was damaged (which didn't happen). To make sure though, the guidance computer's electronics were sealed from the atmosphere for extra protection. They survived and worked well, guiding the Landing Modules safely onto the Moon.

One small step for man perhaps, but the Moon landings were certainly a giant leap for computing.

***well, not by modern standards! The guidance computer contained only around 70 kilobytes of memory.**

To weave your own core rope memory or for further reading about computing in space, go to **cs4fn.blog/cunningcontraptions/**

Image by WikiImages from Pixabay

Babbage's adders

by Paul Curzon, Queen Mary University of London and
Adrian Johnstone, Royal Holloway, University of London

Charles Babbage famously designed the first computer: a steam powered contraption that was never built. At its core was something very simple and elegant: a cunning contraption that allowed his machines to store numbers and do arithmetic, all made of Victorian tech: metal wheels and levers.

Babbage's Analytical Engine, had it been built, would have been the first working general-purpose computer. The size of a factory, and powered by a steam engine, it was the ultimate cunning contraption. A giant whirring, clanking, puffing mechanical brain. Babbage's first attempt at mechanical computation, though, was a simpler machine, the Difference Engine. It could only do a fixed, if very useful, kind of calculation. It computed what are called polynomials: patterns of additions and multiplications. It used a complicated adding mechanism. Later, whilst working on designs for his ambitious Analytical Engine he thought of a much simpler adder (described below). His second Difference Engine used this new adder and so needed about 16,000 fewer parts! The Science Museum built it in the twentieth century.

Representing Numbers

First he had to devise a way to represent numbers. Unlike modern computers which use binary, so only two digits, Babbage stuck to decimal. He was going to do all his calculations using our normal ten digits, 0 - 9. But how? His solution was to use metal cog-like wheels. His wheels had 40 teeth corresponding to the digits 0 to 9 repeated four times. To get the idea

Babbage's wheel from a 3D model by Adrian Johnstone.

of how they worked, imagine a wheel with only 10 teeth, each with a digit 0-9 in order, next to a tooth. The wheel lays flat and one digit faces you. That digit is the number that the wheel represents. Turn it one place to the left and it represents one digit higher. Turn it a place to the right and it represents one digit lower.

That is fine for numbers between 0 and 9. For larger numbers, just do as we do: use the decimal place system where the value of a digit changes with its position. That first wheel is in the ones row so stands for 0-9. Put a wheel above it in a 10s row and it stands for 10 times the value shown. If a 5 is facing you on that wheel it stands for 50. You can now represent numbers 0 - 99. Put more wheels on top and you can represent hundreds, thousands, and so on.

It's a neat way of representing numbers using the system we do (though our numbers run right to left not bottom to top). It makes it not only easy for a machine to manipulate the numbers by turning the wheels, but also easy for a human operator to read the numbers.

His Difference Engine used several stacks of them, but Babbage envisaged a gigantic room-sized data store of column after column of these number wheels as the memory of his analytical engine, each column storing one potentially very large number.

A machine that can count

We now have a way to represent numbers but it isn't yet enough to allow a machine to manipulate them automatically. As it stands it can't even count properly. We have seen only how to count on one wheel, so only up to 9: every time we turn a crank the 1s wheel turns one notch and so the number represented moves on one. However, we need the other wheels to turn too, but only when the wheel below turns from a 9 to a 0 (so should really become 10). We need a mechanism to carry up to the wheel above.

Babbage did this by adding a ridge (a 'nib') on the wheel that triggered the carry. When the wheel got to 0, the nib caught against a mechanism above and pushed it, before allowing it to spring back. That nudged the wheel above along one place as required. The 1s wheel was controlled by the crank. The 10s wheel was turned by the 1s wheel moving to 0. The 100s wheel was turned by the 10s wheel moving to 0, and so on. Babbage had a machine that could count!

A machine that can add

The next problem is how to add numbers stored on wheels. Imagine two wheels, interlocked by their teeth, When one is turned it turns the other the same amount. However if you lift one of the wheels they no longer interlock and move independently.

To do addition, the first wheel is used to hold the number to be added. The second holds the total so far: the answer. That answer wheel starts off set to 0. Now, with the wheels interlocked, turn the first wheel one position at a time counting up to the first number of the addition. It turns the answer wheel exactly the same number of positions transferring the number on to it.

Oops. When cogs interlock, the second wheel turns in the opposite direction to the first! Our machine is subtracting! To make it add, you need a connecting wheel between them. The middle wheel then turns backwards, turning the answer wheel forwards as required. With three wheels like this, any number on the first wheel is added to the answer wheel.

To add a second number, just lift the first wheel to disconnect it, spin it back to zero, drop it back in place and turn it to the second number. The answer wheel then holds the sum of both numbers. If you want to add more numbers, just keep doing this, loading one number at a time onto the first wheel. Each time the total passes 10, the carry mechanism passes the 10 onto the higher wheel and the full decimal total is stored up the stack of wheels.

We have a machine for doing addition!

A machine that can multiply

Babbage's machines could multiply as well. How do you do multiplication on wheels? Well, multiplication is just repeated addition. If you want to work out 5 x 3, then it can be calculated as 5 + 5 + 5. So multiplication can simply be done using the adder, adding the same number over and over again. A counter keeps track of how many times to add it. There are faster ways to multiply though. For the Analytical Engine, Babbage designed an efficient table-based multiplier that he was justifiably very proud of.

Putting it together

Put this together and you have both a number store (a computer memory) and temporary storage areas (registers). You can transfer numbers from one place to another in the machine, and you have the basics of an arithmetic unit that can do calculations. That is about as far as Babbage managed to build. However, he also envisioned programs on punch cards that determined what instruction to execute, mechanisms that allowed instructions to make decisions, and to repeat instructions…everything needed for a general-purpose computer.

Sadly, only parts of his Analytical Engine were ever built, the Victorians did not start the digital age, and we had to wait nearly a century for the first working computers.

Smoke image by Hanjörg Scherzer from Pixabay

Babbage's triumph over brutal reality

by Adrian Johnstone, Royal Holloway, University of London and Paul Curzon, Queen Mary University of London

Charles Babbage is famous for his amazing technical skills in designing a computer, but also infamous for his apparent spiky and obsessive personality.

He certainly seems to have had poor social skills in that he often immensely irritated the people who funded his work. Part of the reason he never managed to complete a working version of his computer is that his funders pulled the plug on him. If only he had had better people skills to complement his technical skills and creativity, perhaps we would have had computers a century earlier!

However, perhaps we should be less harsh. He wasn't a total social misfit: his salons (Victorian high society parties) were extremely popular, and attended by what would now be considered celebrity A-listers. They often centred around demonstrations of science and engineering wonders, so presumably he could be the life and soul of the party… as long as he had a technological wonder to talk about. The young Ada Lovelace attended one such salon and was enthralled by his machines. Encouraged by her mathematically trained mother, Lady Byron, she studied maths and in 1840 collaborated with Babbage on a description of his Analytical Engine.

More to the point, if you consider the context of Babbage's life, he suffered extremes of grief. In one year alone, 1827, he buried three of his children as well as his wife. Of his eight children only three survived beyond the age of ten. That was the brutal reality of the pre-antibiotic world.

In this context perhaps it is better to think about his work and achievements, as a response to adversity. That he achieved so much is a triumph of ambition over terrible loss.

Image by Goran Horvat from Pixabay

Quicksilver memory

by Jo Brodie and Paul Curzon, Queen Mary University of London and Adrian Johnstone, Royal Holloway, University of London

Some 1950s computers used tubes filled with mercury as a memory to store numbers. Mercury is a metal that is liquid at room temperature. It's also known as quicksilver as it flows very easily, but in computing it was actually used to trap information.

Early computers needed a way to store data that would survive indefinitely, even if the computer was stopped. 'Delay lines' provided the solution. Data arriving electronically at a mercury delay line struck a converter (called a 'transducer') which converted the information to a sound pulse in the mercury. The sound travelled through the tube at the speed of, yes, sound and when the pulse reached the other side it hit another transducer and was returned to its electronic form. That might not sound (sorry) like much of a delay but compared to the speed that an electrical signal moves through a wire (a fraction of the speed of light), it's like a gentle stroll. Once inside the mercury tube the sound pulses could be looped back and forth, safely 'parked' until needed. The computer would use its clock to help it count how many pulses had passed and a microphone listened for the right time to release it from the memory store back into the circuitry to do a calculation with it.

Think about tennis serving machines that shoot balls at you. If you put one in a squash court, then a ball being fired will bounce back and forth off the walls but quickly drop to the floor. A delay line works like having two machines facing each other. One fires a ball so that it hits a lever (the transducer) which tells the other machine to fire a ball back, which then hits a lever on the first machine... and so on. Now there is always a ball in flight (a pulse in the delay line) because the motion of the original ball is detected, and used to make a new ball (pulse) that is injected back into the system. Start the first machine by making it fire balls in an initial ball-no-ball pattern and the system stores that pattern, that information.

Using cunning contraptions, motion can keep information firmly in one place.

Mercury is expensive, so computer pioneer Alan Turing recommended using gin instead. He claimed its mix of alcohol and water gave perfect properties for the job while being so much cheaper.

Image by Roman Shashko from Pixabay

The taming of the screw

by Paul Curzon, Queen Mary University of London

Charles Babbage had an obsession for precision and high standards because if his machines were to work, they needed it. One of his indirect contributions to contraption construction the world over concerned the humble screw. We take screws pretty much for granted now, especially the idea that there are standard sizes. Lose one when putting together that flatpack furniture and you can easily get another that is identical. Before the 1800s though that was not the case. Screws made by different people were unlikely to be the same and might only fit the specific thing they were hand made for. Babbage's demands for precision helped change that.

The key person in the invention of the standard screw was Stockport engineer, Sir Joseph Whitworth. Having worked as a boy in his uncle's Derbyshire cotton mills, he was fascinated by the machinery there. He realised the accuracy of the workmanship in the machines was poor and needed to be better.

The Difference Engine was built by Joseph Clement in the years up to 1833, and who should be there helping him do so, having moved on to start a career as a skilled mechanic? None other than Whitworth. For Babbage's machines to work they needed precision engineering of lots and lots of identical parts and to levels of accuracy far greater than previously needed. For the Difference Engine Clement and Whitworth, with their shared passion for accuracy, were up to the challenge. This work showed the coming need for ways to engineer ever more precisely, and to be able to repeat that work…a challenge Whitworth pursued for the rest of his life.

Also famous for inventing the first ever truly accurate "sniper rifle" he went on to create a standard thread for screws that then became the world's first national screw thread standard: the British Standard Whitworth system. It suddenly meant screws could be made by mass production, bought from anywhere, and guaranteed to fit precisely for whatever job was needed. Whilst sadly the need to mass produce computers didn't materialise, the standard was adopted for building ships, trains… for industry throughout the nation, making Great Britain's industry more efficient and so more competitive. Now we rely on the idea of national and international standards like this not just for hardware but for software too. Standards help ensure our computers work but also keep us safe.

The equivalent of this engineering precision is still lacking in the development of software though, much of which is buggy and developed to poor standards by people hacking out software that may or may not work. High standards tend only to apply in safety-critical software, and then often poorly. We need the next generation of programmers to have the same obsession for precision of Babbage and Whitworth and apply it to the development of software, ending the age of buggy, poorly developed software.

Image by Alexas_Fotos from Pixabay

Mary Coombs, teashops and Leo the computer

by Jo Brodie, Queen Mary University of London

Tea shops played a surprisingly big role in the history of computing. It was all down to J. Lyons & Co., a forward thinking company that bought one of the first computers to use for things like payroll. Except they had a problem. Computers need programs, but no such programs existed, and neither did the job of programmer. How then to find people to program their new-fangled computer? One person they quickly found, Mary Coombs, suited the job to a T, becoming the first female commercial programmer.

J Lyons and Company, a catering company with a chain of over 200 tea shops in London, wanted to increase its sales and efficiency. With amazing foresight, they realised computers, then being used only for scientific research in a few universities, would help. They bought the technology from Cambridge University, built their own and called it LEO (the Lyons Electronic Office). They hoped it would do calculations much more quickly than the 1950s clerks could, using calculating machines. But it could only happen if Lyons could find people to program it. At the time there were only a handful of people in the world who were 'good at computers' (programmer didn't exist as a job yet) so instead they had to find people who could be good with computers and train them for the job. Lyons created a Computer Appreciation Course which involved a series of lectures and some homework, all designed to find staff within the company who could think logically and would learn how to write programs for LEO.

One of those was Mary Coombs. Born in 1929, she studied at Queen Mary University of London in the late 1940s. You might think, given that this is about a computer, that she studied computer science, but she actually studied French and History. She couldn't study computer science: what we'd call a computer science course didn't exist. There wasn't one anywhere until 1953, when the University of Cambridge opened a Diploma in Computer Science.

By then, Mary was already working at Lyons. She'd had a holiday job there in 1951, as a clerk (in the Ice Cream Sales department) as she finished her degree. A year later she returned to the company where her career changed direction. In addition to her language skills, she was good at maths so transferred to Lyon's Statistical Office. That's where she heard about LEO and the need for programmers to learn about it and help test it as it was being built and refined. Intrigued, she signed up for the company's first computer appreciation course, did well, and was one of only two people on the course then offered a job on the project. As a result she became the first woman to work on the LEO team as a programmer and the first female commercial programmer in the world.

LEO was an enormous computer, built from several thousand valves, and took up an entire room, though it could only store a few kilobytes of memory. It was also a little temperamental. It needed to be very reliable if it was going to be of any use, so it underwent months of testing and improvement, with Mary's help, before it was put to work on solving real problems, again with Mary and others on the team writing the programs for everything it did.

One of its first tasks was to make sure everyone got paid! LEO was able to calculate forty people's payslips in one minute (one every 1.5 seconds) where previously it would have taken one clerk six minutes to do one: a huge improvement in efficiency for Lyons.

LEO was both pioneering and a big success, but the real pioneers were the programmers like Mary. They turned computers, intended to help scientists win Nobel prizes, into ones that helped businesses run efficiently, ensuring people got paid. Obvious now, but remarkable in the 1950s.

Image by Pexels from Pixabay

Predicting the future

Marble runs, binary and the I Ching

by Paul Curzon, Queen Mary University of London

Binary is at the core of the digital world, underpinning everything computers do. The mathematics behind binary numbers was worked out by the great German mathematician Gottfried Wilhelm Leibniz in the 17th century. He even imagined a computing machine a century before Babbage, and two centuries before the first actual computers. He was driven in part by an ancient Chinese text used for divination: predicting the future.

I Ching

Leibniz was interested in the ancient Chinese text, the I Ching because he noticed it contained an intriguing mathematical pattern. The pattern was in the set of 64 symbols it used for predicting the future. The pattern corresponded to the counting sequence we now know of as binary numbers (see I Ching binary, overleaf).

Leibniz was obviously intrigued by the patterns as a sequence of numbers. Already an admirer of the great Chinese philosopher, Confucius, he thought that the I Ching showed that Chinese philosophers had long ago thought through the same ideas he was working on. Building on the work of others who had explored non-decimal mathematics, he worked out the maths of how to do calculations with binary: addition, subtraction, multiplication, and division as well as logical operations such as 'and', 'or' and 'not' that underpin modern computers.

Algorithms embedded in machines

Having worked out the mathematics and algorithms for doing arithmetic using binary, he then went further. He imagined how machines might use binary to do calculations. He also created very successful gadgets for doing arithmetic in decimal, but saw the potential of the simplicity of the binary system for machines.

He realised that binary numbers could easily be represented physically as patterns of marbles and the absence of marbles. These marbles could flow along channels under the power of gravity round a machine that manipulated them following the maths he had worked out.

His computer would have been a giant marble run!

A container would hold marbles at the top of a machine. Then, by opening holes in its base above different channels, a binary number could be input into the machine. An open hole corresponded to a 1 (so a marble released) and a closed hole corresponded to a 0 (no marble). The marbles rolled down the channels with each channel corresponding to a column in a binary number. The marbles travelled to different parts of the machine where they could be manipulated to do arithmetic. They would only move from one column to another as a result of calculations, such as carry operations.

Addition of digits in binary is fairly simple: 0+0 = 0 (if no marbles arrive then none leave); 0+1 = 1+0 = 1 (if only one marble arrives then one marble leaves; (1+1 = 2 = 10) if two marbles arrive in a channel then none leave that channel, but one is passed to the next channel (a carry operation). The first rules are trivial, open the holes and either a marble will or will not continue. Adding two ones is a little more difficult but Leibniz envisioned a gadget that did this, so that whenever two marbles arrived in a channel together, one was discarded, but the other passed to the adjacent channel. By inputting two binary numbers into the same set of channels connected to a mechanical gadget doing this addition on each channel, the number emerging is a new binary number corresponding to the sum.

Multiplication by two can be done by shifting the tray holding the number along a place to the left. In decimal, to multiply a number by ten, we just put a 0 on the end. This is equivalent to shifting the number to the left: 123 becomes 1230. In binary the same thing happens when we multiply by 2: put a 0 on the end so shift to the left and the binary number is twice as big (11 meaning 2+1 = 3 becomes 110 meaning 4+2+0 = 6). Multiplication of two numbers could therefore be done by a combination of repeated shifts of the marbles of one number, releasing copies of it or not based on the positions of the 1s in the other number. The series of shifted numbers were then just added together. This is just a simplified version of how we do long multiplication.

```
   110
 X 101
 ------
   110
  0000
+ 11000
 -------
  11110
```

To multiply 110 by 101, you multiply 110 by each digit of 101 in turn. This just involves shifts, and then discarding or keeping numbers. First multiply 110 by 1 (the ones digit of 101) giving 110. Write it down. Now shift 110 one place to give 1100 and multiply by 0 (the twos digit of 101). This just gives 0000. Write it below the previous number. Shift 110 by another place to give 11000. Multiply it by 1 (the fours digit of 101). That gives 11000. Write it down below the others. Add all three numbers to get the answer.

The basics of a computer

Leibniz had not only worked out binary arithmetic, the basis of a computer's arithmetic-logical unit (ALU), he had also seen how binary numbers could flow around a machine doing calculations. Our computers use pulses of electricity instead of marbles, but the basic principles he imagined are pretty close to how modern computers work: binary numbers being manipulated as they flow from one computational unit to the next.

Leibniz didn't make his machine, it was more a thought experiment. However, helped by I Ching, a book for divining the future, he did essentially predict how future computers would work.

Image by Sergio Camacho Camacho S.A. from Pixabay

Image by Sushruti from Pixabay

I Ching binary

by Paul Curzon, Queen Mary University of London

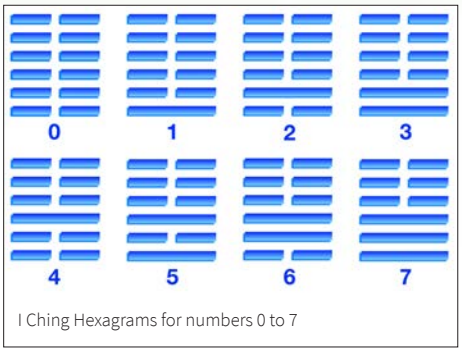
I Ching the ancient Chinese divination text, several thousand years old, is based on a binary pattern…

I Ching is one of the oldest Chinese texts. The earliest copies date from around 3000 years ago. It uses 64 hexagrams: symbols consisting of six rows of lines (see right). Each row is either a solid horizontal line or two shorter lines with a gap in the middle. The 64 hexagrams are all the possible symbols that can be made from six rows of lines in this way. In I Ching, they each represent possible predictions about the future (a bit like horoscopes). To use I Ching, a series of hexagrams were chosen. This was done in some unknown but random way, using stalks of the Yarrow plant, standing in for dice. The chosen hexagrams then told the person something about their future.

In the earliest versions of I Ching, the order of the hexagrams suggests that they were not thought of as numbers as such. However, in a later version, from around 1000 AD the order in which they appear is different. Thought to be written by a Chinese scholar, Shào Yōng, it is this version that Leibniz was given and that aroused his interest because the order of the hexagrams follows the pattern we know of as binary (see 'Predicting the future', previous page). Shào Yōng had apparently picked the sequence deliberately because of the binary pattern, so understood it as a counting sequence, if not necessarily how to do maths with it.

How do the hexagrams correspond to binary? It is not in the lines themselves but the pattern of line breaks down the middle that matters. Think of a break in the lines as a 0 (yin) and no break as a 1 (yang). The order, as Leibniz realised, is a counting system, equivalent to our decimals but where you only have two digits (0 and 1) rather than our ten digits (0...9).

Whereas in decimal each column of a number like 123 represents a power of 10 (ones, tens, hundreds, …) in binary each column represents a power of 2 (ones, twos, fours, eights, …). To work out the value that the number represents you multiply each digit by its column value and add the results. So in decimal, 123 represents one hundred plus two tens plus three ones (one hundred and twenty three). 1011 in decimal represents one thousand plus no hundreds plus one ten plus a one (one thousand and eleven). In binary, however, 1011 represents instead one eight plus no fours plus one two plus a one (8+0+2+1) so eleven. It is just a different way of writing down numbers.



I Ching Hexagrams for numbers 0 to 7

Investigating the I Ching pattern helped Leibniz to work out the mathematics of binary arithmetic and on to thinking about machines to do calculations using it.

Ada and the music machine

by Paul Curzon, Queen Mary University of London

Charles Babbage found barrel organs so incredibly irritating that he waged a campaign to clear them from the streets, even trying to organise an act of parliament to have them banned. Presumably, it wasn't the machine Babbage hated but the irritating noise preventing him from concentrating: the buskers in the streets outside his house constantly playing music was the equivalent to listening to next door's music through the walls. His hatred, however, may have led to Ada Lovelace's greatest idea.

It seems rather ironic his ire was directed at the barrel organ as they share a crucial component with his idea for a general purpose computer - a program. Anyone (even monkeys) can be organ grinders, and so play the instrument, because they are just the power source, turning the crank to wind the barrel. Babbage's first calculating machine, the Difference Engine was similarly powered by cranking a handle.

The barrel itself is like a program. Pins sticking out from the barrel encode the series of notes to be played. These push levers up and down, which in turn switch valves on and off, allowing air from bellows into the different pipes that make the sounds. As such it is a binary system of switches with pins and no pins round the barrel giving instructions meaning on or off for the notes. Swap the barrel with one with pins in different positions and you play different music, just as changing the program in a computer changes what it does.

Babbage's hate of these music machines potentially puts a different twist on Ada Lovelace's most visionary idea. Babbage saw his machines as ways to do important calculations with great accuracy, such as for working out the navigation tables ships needed to travel the world. Lovelace, by contrast, suggested that they could do much more and specifically that one day they would be able to compose music. The idea is perhaps her most significant, and certainly a prediction that came true.

We can never know, but perhaps the idea arose from her teasing Babbage. She was essentially saying that his great invention would become the greatest ever music machine…the thing he detested more than anything. And it did.

Image by Holger Schué from Pixabay

Babbage's barrels

by Adrian Johnstone, Royal Holloway, University of London and Paul Curzon, Queen Mary University of London

Despite his hatred of Barrel organs, Babbage used barrels with relocatable pins in his machines. They worked in a similar way to a music box, where the pins flip the teeth of a metal comb to sound a note and by moving the pins you get a different tune. In Babbage's version the barrel's pins push levers that send information round the machine, determining what it does.

By programming the positions of the pins, different overall operations are created from the combinations of lever pushes. This is a similar idea to what later became called microcoding, in modern computers, where very simple low level instructions are used to program the operations available in a computer's instruction set.

Image by ♥Monika ♥ ♥ Schröder ♥ from Pixabay

Transistor symbol by CIker-Free-Vector-Images from Pixabay

A custard computer

by Paul Curzon, Queen Mary University of London

This simplistic custard contraption is inspired by a more sophisticated custard computer invented by Adrian Johnstone, Royal Holloway, University of London.

Imagine a room-sized vat of custard suspended from the ceiling. Below are pipes, valves and reservoirs of custard. At the bottom is a vast lake where the custard collects as it splurges out of the pipes. A pump sucks custard back up to the vat on the ceiling once more. Custard flows, sits, splurges…all the while doing computation.

Babbage worked out how to make a computer using wheels. How might you make a general purpose digital computer out of custard? (It sounds more fun!) Adrian Johnstone at Royal Holloway has designed one and if built it would look something like our above description, like something from Willie Wonka's chocolate factory.

Here we give a slightly simplistic version. The first step is to have something to represent 0 and 1. That's easy with custard: no custard in a storage tank is a 0 and custard is a 1. Out of that you can represent numbers using collections of such tanks: lots of tanks containing custard or no custard, with a code (binary) giving them meaning as numbers.

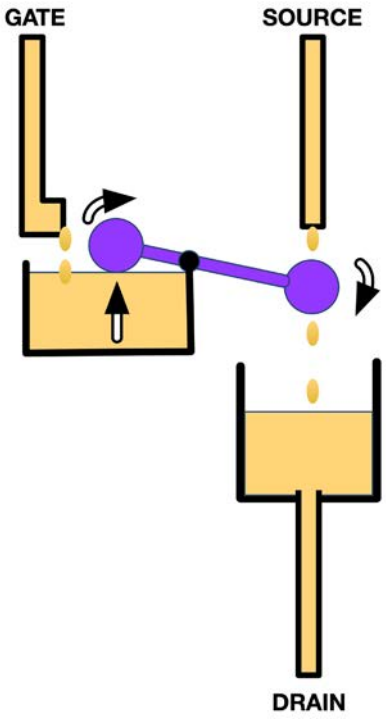
Once you have a way to represent numbers, the next step to making a computer is to make the equivalent of transistors. Transistors are just switches, but ones that revolutionised electronics to the point that they have been hailed as one of the greatest inventions ever. Starting with humble transistors, computers (and lots more) can be built.

Transistors have three inputs. One acts as the data input, or source, connected ultimately to the source of the current (in our case the vat of custard). Another, the drain, connects ultimately to the place the current drains to (in our case the lake of custard). A third input is the gate. It switches the transistor on and off, either allowing current (custard) to flow towards the drain or not. The gate thus acts as the switch to allow custard to flow.

One way to make a custard transistor would be to use a contraption based on your toilet but full of custard (don't think about that too much). Look in your toilet cistern and see how it switches water on and off when you flush the toilet to get the idea. For a (custard) transistor, have a small tank of custard with a ball floating on the surface. It acts as the gate. Fastened to the end of the ball is a lever The lever's other end can push up against the end of the pipe that runs from source to drain, blocking the flow. When the tank is full of custard it pushes the other end of the lever down, letting custard flow. If the tank empties then the ball drops, so the other end of the lever rises and blocks the flow.

Transistors have been hailed as one of the greatest inventions ever.

Droplet image by freddy urbina from Pixabay



A custard transistor: when there is custard at the gate, custard flows from source to drain. When no custard is at the gate, the floating ball drops and closes the link.

There are two kinds of transistors. They differ in that the gate just operates in opposite fashions. With one kind, custard can flow from source to drain only when there is current (custard) at the gate (as above). In the other, custard flows only when there is no custard at the gate.

Once you have (custard) transistors, you can make (custard) logic gates (NOT, AND, OR,...). A (custard) NOT, for example, would need to let custard into its out pipe only if there were no custard on its input pipe (and vice versa). We can do this using a transistor with the NOT circuit's input connected to the gate, and where custard flows only when the gate has no custard. The drain of the transistor becomes the output of the NOT circuit. The source of the transistor connects to the vat of custard to provide the custard that will flow when the transistor switches on. When custard arrives at the gate which is acting as a switch, it stops custard flowing to the drain, and vice versa, as required.

AND logic needs to let custard out only when there is custard at both its input pipes. OR logic allows custard through when there is custard at either of its input pipes. This can be done with appropriate plumbing together of the transistors as follows.

A (custard) AND uses two transistors It allows custard to flow when there is custard at both gates which are the input pipes of the AND circuit. Connect one input of the AND circuit to the gate of the first transistor with the source connected to the vat of custard. Connect its output to the source of the second transistor. The gate of this second transistor is linked to the second input of the AND. Custard will flow from the vat down towards the drain only when there is custard at both gates. If either gate has no custard, then the custard will not flow, just as required for custard AND logic. We will leave you to work out how to make (custard) OR logic.

Once you can create gadgets that do (custard) NOT, AND and OR, you can then start to build more interesting circuits by combining them: building up the components of a computer like (custard) adders and (custard) multipliers, circuits that compare numbers and ones that trigger custard to be moved about… put it together in the right way and you can build a computer with control unit, arithmetic logic unit, memory unit and so on… (as long as you have enough custard).

Out of the glooping vat of custard, computation emerges….Would it really work? You would have to build it to find out!

The Wood Computer

by Jo Brodie, Queen Mary University of London

Punch cards inspired Babbage as he invented the first Victorian computer, and were a way the first computers stored data a hundred years later. Variations, called edge-notched cards, had their uses before the first working computers, though. They provided an efficient way to look up information. One use was to help identify timber: Oxford's human-operated 'wood computer' was used in forests world-wide.

Interested in nature and enjoying a nice walk, you come across an unfamiliar tree, and want to identify it. How do you do it? You might work through a set of questions, first looking at the leaves: what shape are they, what colour, do they have stalks, do they sit opposite each other on a twig or are they diagonally placed, and so on. You then move to questions about the bark… Gradually, you narrow it down to one tree.

What, though, if your job is to check that your company is buying the right timber and the tree is cut up into logs (no leaves or bark)? The task is the same, going through a checklist of questions, just harder unless you are an experienced botanist. Now you consider things like the pattern of the grain, the hardness, the colour and any scent from the tree's oils.

Historically, one way of working out which piece of timber was in front of you was to use a wood identification kit or 'wood computer'. This was prepared (programmed!) from a pack of index cards with 60 or more features of timber printed on them. However, they weren't just cards to read but cards to compute with.

Holes and notches

The cards were special because they had regularly placed holes round all four sides. Each card had notches cut into different holes. Each feature of the timber was linked to one or more holes. The features were grouped together around related properties: so, for example, all the possible colours of timber might be grouped together on one section of the card. Properties about how fine-grained the timber is would be grouped in another section.

Each card represented one type of wood and the 'programmer' of the cards would notch the holes next to the features that defined it. If a particular type of timber was fine-grained you would add the notch to the hole next to "fine-grained", if it wasn't that hole would be left un-notched. Notches were added for all relevant timber properties making each card unique, with a slightly different pattern of notches, uniquely describing the features of the tree it represents. (See an example of an edge-notched card on page 17.)

How it works

To use a wood computer, take the pile of cards, pick a feature of the timber in front of you and insert a thin knitting needle into the hole linked to the feature. Then lifting the pile up shake out any cards with notches in that hole. All of the cards for timber that don't have that feature will have an un-notched hole and will hang from your knitting needle. All cards representing timber that does have that feature are now sitting on the table. Yours is somewhere amongst them. If your timber is NOT fine-grained then instead, when you put the knitting needle in the fine-grained hole, keep those left on the knitting needle.

You repeat the process several times to whittle (sorry!) your cards down, each time choosing another feature of the timber in front of you. Eventually you have only one card left and have identified your timber.

Just the cards for the job

The cards are incredibly low tech, requiring no electricity or phone signal and are very easy to use even without specialist botanical knowledge. All the knowledge is programmed into them. You also find the answer very quickly. Margaret Chattaway, a botanist at the Imperial Forestry Institute, Oxford, in the 1930s realised that was exactly what was needed for their team inspecting timber and so the original wood computer was created.

So next time you are out for a walk, make sure you have your knitting needle and a suitable pile of cards with you. Then identifying trees, birds, fungi or even animal poo will be so much quicker and simpler.

Image by Peter H from Pixabay

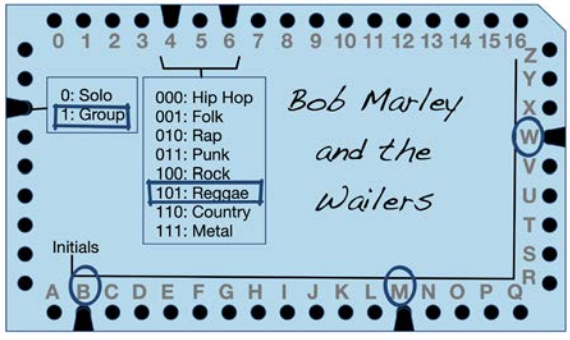
Edge-notched cards and Relational Databases

by Paul Curzon, Queen Mary University of London

Edge-notched cards implement a physical, but still powerful, version of a database: an organised way of storing data.

Databases consist of lots of records each storing the information about one thing like one kind of timber. Each card in our pack corresponds to a record. A whole pack of records about one thing (like our pack of cards) is a database table. Records consist of fields with each field describing some aspect of the data like what the grain of the timber is like. Each group of related notches therefore acts as a database field.

In a relational database you do not have one gigantic set of records, so not just one gigantic pack of cards. You have a series of different sets of records/cards. Each has fewer fields so fewer holes as they no longer need to store details of every possible feature. Each smaller pack of cards is a table describing a specific thing (like leaves or bark). There is also still a master pack describing trees as a whole. The tree cards no longer have to include all the details of leaves and bark, however. Instead each table includes a unique identifier field. Leaf cards include a leaf identifier that is also on the tree's card. Bark cards similarly include a bark identifier. Once you have identified the leaf, you can use the leaf identifier on the tree cards to find any trees with that set of properties of leaf, then narrow it down further once you know the bark identifier. The smaller packs of cards still do the job but in a much more convenient way.



An edge-notched card for Bob Marley and the Wailers notched as a Reggae Group with initials B-M-W: those three letters are notched, the other letters are un-notched.

Why not create an edge notched card system for something you are interested in, for identifying birds perhaps or quickly finding details of films, or music or of something you collect?

The beach, the missionary and my origin myth

by Paul Curzon, Queen Mary University of London

Superheroes always have an origin myth that describes how they emerged as a hero. Spider-man has his spider bite and death of his uncle; Batman, his fall into a cave full of bats and the murder of his parents; Captain Marvel was exposed to an alien energy source…
Why not work out your own origin myth? Everyone should have one. Mine involves a beach, a book of programs, and before that a missionary. It is the backstory of how I became a computer scientist.

Image by Daniel Nebreda from Pixabay

The beach

My origin myth usually begins with a beach and a book containing some programs, some articles about computers and some computing cartoons. The articles were vaguely interesting, some of the cartoons were funny, but the program listings were fascinating: a whole new language that made computers tick.

At that point computers were way too expensive for me to dream of owning one (and in any case back then there were no mobile computers so unlike now, you certainly couldn't take one to the beach). All I had was my imagination, but that was enough to get me started.

With nothing else to do on the beach (it was too hot to move), I spent my time lying in the sun reading programs and trying to work out what they did and how they did it. With no computer, all I could do was pretend to be the computer myself, stepping through the listings line by line with paper and pen, writing out the changing numbers stored (their variables) and what they printed. I then moved on to writing some simple game programs myself, like a cricket game. I wrote them in my notebook and again pretended I was the computer to make them work. By the end of the holiday I could program.

Ada Lovelace

I didn't discover this till decades later but Ada Lovelace, the famous Victorian computer pioneer working with Charles Babbage was in a similar position (well sort of … she was a rich countess, I wasn't). She also had no computer as Babbage hadn't managed to fully build his. She had no programming language either to write programs in, or for that matter any actual programs to read. However, she had some algorithms written by Babbage that he intended his machine's programs would be based on. Just like me, she stepped through the algorithms on paper, working out what they would do (should Babbage ever build his computer), step by step. As a result she learnt about the machine and as it happens also found a mistake in one algorithm. The table she drew of the computer working is often taken as proof she was the 'first programmer', though as Ursula Martin, who studied her papers, has pointed out, it is not a program. It is an execution trace or 'dry run table'. She was actually the first 'execution tracer' or 'dry-runner'.

The importance of dry running code

Dry running programs like this on paper is not just a useful thing for people with no computer, it is also a critical thing for anyone learning to program to do - a way of actively reading programs. You didn't learn to write English (or any other language) by just writing, you read lots too and the same goes for programming. It turns out that the way I taught myself to program is a really, really powerful way to do so.

Just as importantly dry running programs on paper in this way is also important as a way of checking that programs work as Lovelace found. The modern version is the code walkthrough - a powerful technique that complements testing programs as a way of discovering problems.

The missionary

While that is the point in time when I learnt to program, there was someone earlier who originally inspired me about computation: a missionary. Sadly I don't know his name, but he came to our school to talk about his life as a missionary in Papua New Guinea. He told us that one of the problems of travelling there was that communities were isolated from each other and each village spoke its own language. That meant that, as he travelled around, he had a big problem speaking to anyone. Every time he moved on he had a new language to learn or an old one of many to remember. It wasn't the idea of converting people, or travelling to exotic places that means I remember him more than 40 years later. It was what he showed us next: how he solved the problem. He pulled out a massive pile of cards with holes punched round the edges, labelled with letters. Each had a word written on it in English as well as words in other languages from different places. He spelled out a word a letter at a time (pig was the example he used) by putting a knitting needle through a hole in all the cards next to the letter. Those that fell out were used for the next letter and so on. After three rounds just the card for pig fell out, as if by magic. It wasn't magic though, it was computation. On the pig card he had cut notches in the holes for p, i and g. As that was the only word with those letters, it was the only card that could fall out for all three letters and then he could read off the translation for the village he was in..

Bitten by a bug

What the missionary was showing us was an edge-notched card system (see the 'Wood computer', page 16). I was fascinated and have been ever since about computation, especially when it's done physically. It was that general fascination for algorithms that led me to want to learn to program.

In my origin story, I was bitten by a bug: the missionary converted me… to computer science.

Image by StockSnap from Pixabay

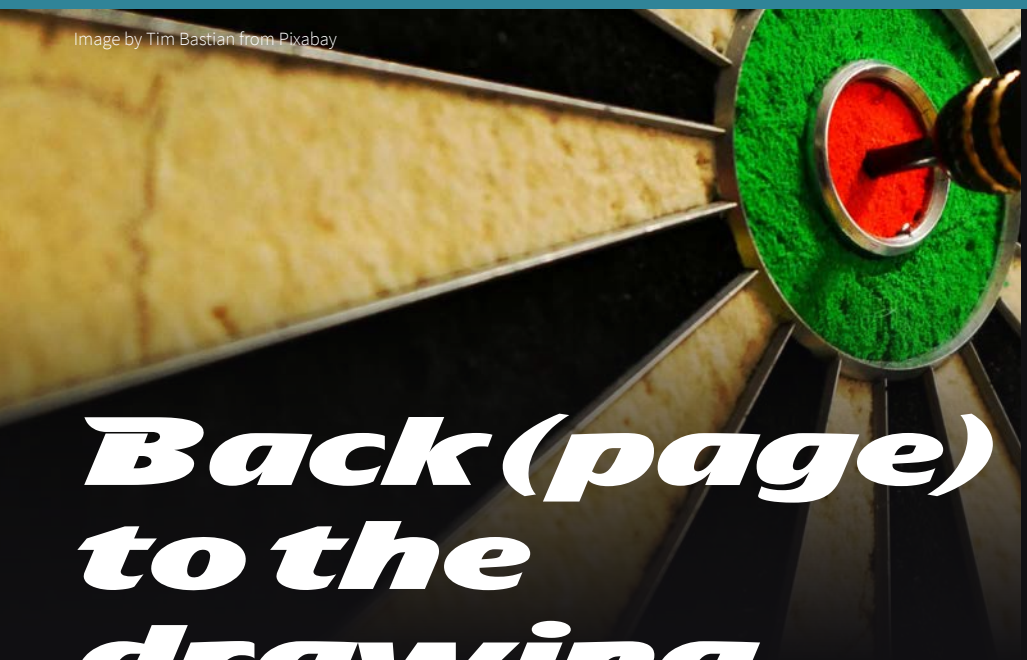


Image by Tim Bastian from Pixabay

Back(page) to the drawing board

by Jo Brodie, Queen Mary University of London

Here are some more cunning contraptions, with and without a purpose…

Bullseye!

Mark Rober, an engineer and YouTuber who worked for NASA, has created a dartboard that jumps in front of your dart to land you the best score. Throw a dart at his board and infra-red motion capture cameras track its path, and, software (and some maths) predicts where it will land. Motors then move the dartboard into a better position to up the score in real time!

Search for: Rober dartboard

You switch me on…

In 1952 computer scientist and playful inventor, Marvin Minsky, designed a machine which did one thing, and one thing only. It switched itself off. It was just a box with a motor, switch and something to flip (toggle) the switch off again after someone turned it on. Science fiction writer Arthur C. Clarke thought there was something 'unspeakably sinister' about a machine that exists just to switch itself off and hobbyist makers continue to create their own variations today.

Search for: Ultimate Machine

Image by Drpixie from Wikipedia (CC BY-SA 4.0)

Tie a knot in it

Quipu (the Quechua word for 'knot') are knotted, and sometimes differently coloured, strings, made from the hair fibres of llamas or alpacas. They were used by people, such as the Incas, living hundreds of years ago in Andean South America. They used the quipu to keep numeric trade or military records. A 'database' was formed of several of the strings tied together at one end. Each string stored numbers as different kinds of knots at different positions along the strings, with positions for ones, tens, hundreds, etc. It worked a bit like an abacus, but with much less danger of losing your work if you turn it upside down.

The number '1' was represented as a figure-of-eight knot in the ones position and '40' could be indicated by four simple knots in the tens position. Not many quipu survive and even fewer have been decoded, but anthropologists have begun to find evidence that they might contain not just numbers but a written (well, a tactile) form too.

Search for: quipu knots

Look out, leeches!

When we leave our homes we might check a weather app to give us predictions from number-crunching computers, to see if we'll need an umbrella, but in the mid-1800s the appropriately named George Merryweather thought he'd make use of the alleged weather-predicting properties of leeches to create a 'leech barometer' to measure the weather. His notion relied on the belief that leeches, kept inside small glass bottles, would try and escape when a storm was due (because they might be more sensitive to subtle changes in electrical conditions in the air that we humans would miss). The escaping leeches would trigger a small hammer placed above the bottles which would strike a bell and alert everyone in earshot that a storm might be imminent and also that your living room was about to be overrun with leeches. Not surprisingly it wasn't very popular, though Merryweather claimed to have great success with it.

Search for: Tempest Prognosticator

A pat on the shoulder

In lockdown, during the Covid-19 pandemic, inventor and roboticist Simone Giertz created a coin-operated 'proud parent machine' which, for 25 US cents, would pat her on the shoulder and give a few words of encouragement. Putting in a coin sent a signal to a microcontroller that turned a motor on which lowered a 3D-printed arm (to pat her shoulder), then played a pre-recorded audio file telling her how proud of her the automaton was. Making the machine involved skills in woodworking, computer-aided design, mechanics and electronics. She also gave a TED Talk called "Why you should make useless things".

Search for: well done Simone Giertz

cs4fn is edited by Paul Curzon and Jo Brodie of Queen Mary University of London. Spring 2022. Thanks to Sue White and Jane Waite for proof reading. Ursula Martin and Adrian Johnstone have provided advice and explanations. This magazine was funded by UKRI, through grant EP/K040251/2 held by Professor Ursula Martin and forms part of a broader project on the development and impact of computing, as well as grant EP/W033615/1. Further funding was provided by Paul Curzon. Cover Image by Oleg Gamulinskiy from Pixabay. Paul Curzon writes and edits cs4fn in his own time. Magazine design by Kelly Burrows, kellyburrows@gmail.com