My Computing Life

My Computing Life

By

Norman Sanders

Cambridge
Scholars
Publishing

My Computing Life

By Norman Sanders

This book first published 2021

Cambridge Scholars Publishing

Lady Stephenson Library, Newcastle upon Tyne, NE6 2PA, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Copyright © 2021 by Norman Sanders

All rights for this book reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owner.

ISBN (10): 1-5275-6216-6
ISBN (13): 978-1-5275-6216-5

Table of Contents

FOREWORD AND ACKNOWLEDGEMENTS

This book arose from an initiative by Helen Carter and Mark Jones of Archives of IT to record my voice as part of a project to preserve an account of the early years of Computing; there aren't many of us left. As of mid-2020, I seem to be one of the last of the pioneers of the digital Industrial Revolution, and as Helen and Mark started to ask me to talk about what I'd done I began to recall more and more of the detail. Eventually my rough notes evolved into chapters, which I hope have interest for readers of a technological and historical bent.

I should say right away that I didn't invent a single computer. By the time I hit the road, computers had been invented, and there were hotspots around the planet where general agreement as to their technical philosophy had come about. Even the press had become aware of them and had started to call them electronic brains. But little use had yet been made of them; computer builders simply continued building computers, each an improvement over its predecessors, as a succession of technological problems was solved; they were not out there actually using them. Instead they were creating the technological substrate of an utterly unpredictable future–all based on the arrival of machinery capable of doing high-speed arithmetic; who on earth was interested in carrying out high-speed arithmetic? Corner tea shops in Britain's towns, perhaps?

I got the hang of how computers worked from the Mathematical Laboratory in Cambridge, and when I felt I'd mastered enough to call myself a programmer (whatever that may be) I left academia to start to beat the bushes. Show me your computers, I said to the worlds of industry and commerce. But there were no computers; no jobs advertised in the media, so I saw it as my task to go out there and start a new profession as a second-generation computer nerd.

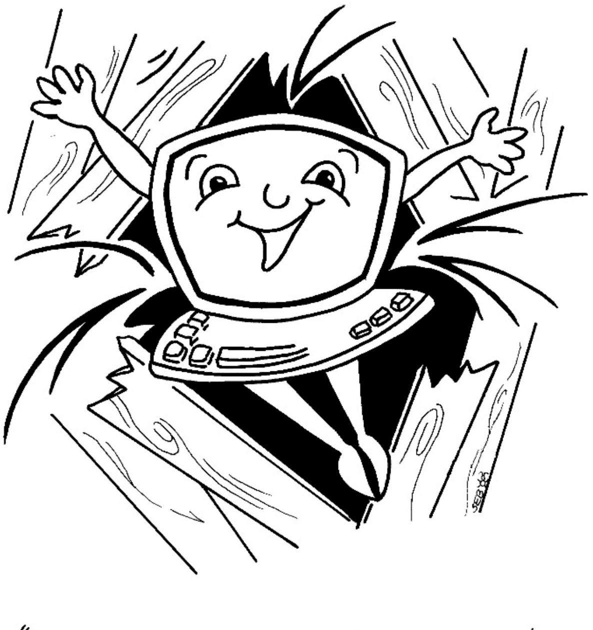
It was a thoroughly exciting career, and I hope that the excitement shows in these chapters. Throughout a working lifetime of some forty years, the phone was constantly ringing with offers of new problems to solve; new potential applications of this revolutionary technology. It opened up opportunities for going places, meeting people and discovering how the world works. It brought me into university life, the manufacturing industry, global project management and political life, including the United Nations. I started out armed with a working knowledge of mathematics, but soon discovered that maths in itself doesn't tell you much about the real world. It's peopled with experience of it who introduce you to reality, and there's probably little needed beyond reproducing that reality in a computer to understand how it all works; all that computers need to function profitably is the ability of its programmers to ask questions and do something with the answers.

So, I take this opportunity of very belatedly thanking everyone for allowing me to take part in this global experiment, and in helping me enjoy a long career as one of the team who did it. Firstly, Helen and Mark who told me to remember it and write it down, then to the computer builders who gave us the tools with which to do it, many of whom I knew so well and are named in this book, and my erstwhile colleagues thrown up by the cosmic dice in organisations of wide variety in various parts of the world. All starting from a need to do arithmetic accurately and quickly. And finally a very special thank you to Sonya Burrows for encapsulating it in her wonderful cartoons, continuing a tradition in my computer books started fifty years ago by Tony Hart.

INTRODUCTION

This book covers the history of a narrow window of the computer era from about 1950 to 1970. The automatic, programmable electronic computer was already up and running in fairly isolated hot spots around the planet at the onset of this era, and had become, if not all-pervading, at least globally familiar by the end. It was the window that opened up the second industrial/social revolution of modern life, following the industrialisation of steam, the creation of the factories and the explosion of the cities.

The computer era was never ordained or proclaimed; no one ever said, "There shall be a computer". I suppose the computer appeared to most people to have just popped out of the woodwork, but it didn't, and if you read chapter 1, you'll find out why. But one thing was to create the computer in the first place, while it was another thing to actually use it to do anything. It wasn't at all obvious to the originators that it was going to be put to use. Indeed, very few, if any, of the originators had anything to do with its application. That was an almost water-tight generation altogether; the generation from which I emerged - via the back door.



"the computer appeared to most people to have just popped out of the woodwork"

But, why this book? Surely others have written about the computer in the mid-twentieth century. Yes, they certainly have, and many of them have recorded their voices via Archives of IT, who got me started on this project. But Archives, in the person of Helen Carter, tell me that I'm one of the oldest survivors of the early days of the era, heading very rapidly to a ripe old age. And as I've thought about what to record, I've been reminded of much that had happened. So, a few words on the record became a full book. Thus, I claim no more than pure ***anno domini*** as my ***raison d'etre***, as outlined in chapter 1.

During the first eighty years of the computer era an enriching range of specialities has evolved from the computer design and fabrication by mathematician and engineer, via the maintenance engineer to the computer scientist and entrepreneur. But in there somewhere emerged the computer programmer, which is what I profess to be. Of course, the original computer designers were also programmers. Their job was to enable programming to take place; programming, explained mainly in chapter 2, is the key to the phase of evolution that the book is about.

Because of the initial ***laissez-faire*** scattered culture of the early computer builders, the early energy and creativity was spread too thinly to constitute a viable global entity. Consequently, different local enterprises created different versions of the same thing, which led to a lack of standards; standard machine language codes and standard programming languages. And how standards came about was a very hit and miss affair, as exemplified in chapter 10. I don't suppose it's much better today.

However, this isn't a technical book. It contains technical terms, which have hopefully been sufficiently explained, but much of it is about the people, some of whom are still around - for verification purposes. And one of the facts that I hope has emerged clearly is that the pioneering life that we lived was enriched by a wealth of non-computing side-shows, very little of it appearing in our academic foreplay.

To entice you to continue reading, I'll briefly mention one or two. There's the Norwegian, Ralph, who skied across Greenland, skied to the South Pole and climbed Mount Everest in his copious spare time, then, at the age of 80 got his PhD in mathematics; the Pole, Janusz, who was accused by the United States of being a spy and forbidden to enter the US, who got there anyway, bringing his programming skills from Lech Wałęsa's Gdansk Shipyard to Seattle–you don't often get the chance of beating the American Government at its own game; the American, Grant, who volunteered for the RAF, and spent a chunk of the war as a Japanese POW; another American, Bill (Gates), who was a schoolboy across town in my time; the Brit, Harold–you don't often get the chance of making a prime minister famous; the playwright, Sir Antony, famous in his own right, who wrote a play called “Yes Minister”; another Tony who, with another Norwegian, Einar, decorated my books with cartoons, plus countless others who enriched my life, and saved me from being a schoolmaster–which I became after retiring anyway. They are all here in the book, including some of the grandees I knew from yesteryear.

And fundamental to everything we do with the computer lies the solution to the problem of how a digital machine manages to live and function in an analogue world? This is explained in chapter 15, where you will also find a few non-scary words about mathematics.

There are other chapters that I haven’t mentioned in this one. How typical they are of the experiences of others of my generation is difficult to say, but between us all we, perhaps inadvertently, got the whole edifice going, and today’s new generations are living in a very different world than we did.

CHAPTER 1

AN INTRODUCTION TO THE COMPUTER

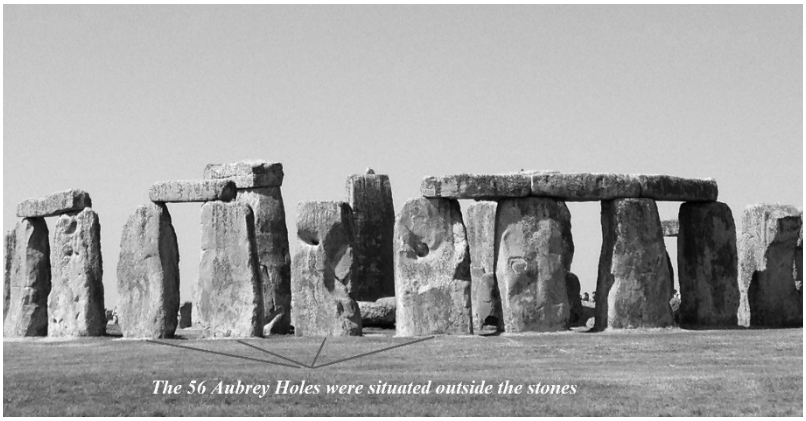
The full story of the computer starts way back before the beginning of time. We don't know when this was nor what people did to carry out arithmetic, which is what computers were first meant to do. These days it seems that computers do little actual computing, at least at the visible level. The task I have before me is to try to remember what happened during the early days of the computer revolution. There aren't many of us left, certainly of the people I knew around the world, but we all had our own story to tell. I think my memory still works and I luckily–and unbelievably–kept quite a bit of the paperwork as evidence. So, between the two aides memoir I think I can tell a fairly accurate yarn.

The computer brought about an industrial and social revolution. But why write about it? It's going on before our very eyes; who could possibly be interested? On the other hand, we've all heard about the historical Industrial Revolutions, the harnessing of steam, the discovery of electromagnetism, the availability of coal etc., and the migration of the country folk to the factories and cities; we all got it at school. It's central to our National Story. And we learnt about the people who brought it about; James Watt, Eli Whitney, George Stephenson and Richard Arkwright. But who today has heard of Charles Babbage, Ada Lovelace, J Presper Eckert, John Mauchley, Maurice Wilkes and Alan Turing? These names, along with their exploits, should be equally well remembered.

I think it's important to start out by saying that there is an enormous gap between what I mean by a computer, and what today's (2020's) computer-user (which seems to be almost everybody) understands. I have tried to establish some sort of numerical measuring device to signal the difference. For example, to me a computer occupies at least three dimensions in anyone's ballroom, whereas to the kids throwing beer cans at my neighbour's cat a computer can be kept in their pocket and is easily lost. On the other hand, my first computer was limited to a mere one K of memory, while theirs is so large that numbers have become meaningless to anyone except the designer and would have been absolutely unbelievable to computer people fifty years ago.

This attempt to illustrate the world in which we now dwell cannot cope with how we display the previous industrial revolutions - we have no parallel with the increases in the speeds of locomotives, the lengths of canals or the human densities of towns. The differences between yesterday's history and today's actuality make history almost impossible to discuss. Nevertheless, I shall discuss it.

Probably the first numerical problem that ***Homo sapiens*** had to deal with was *time*. Solutions were found using water, dripping in even periods, and pendulums that swung in even periods, at one extreme, and the sun and stars at the other. One of the functions of Stonehenge might have been the prediction of eclipses, not least the 56 Aubrey holes, precisely matching the 56-year moon-cycle. The idea of Stonehenge as a computer was advanced in 1966 by Professor Gerald Hawkins in his book, ***Stonehenge Decoded***. The moon's orbit around the earth is periodically changing such that it returns to any particular position every 56 years. Using small stone "indicators" placed in the Aubrey holes, and moving them one hole each year, "all the extremes of the seasonal moon, and eclipses of the sun and moon at the solstices and equinoxes could have been foreseen. If six stones, spaced 9, 9, 10, 9, 9, 10 Aubrey holes apart, were used, each of them moved one hole counter clockwise each year, astonishing powers of prediction could have been achieved".



The Aubrey holes provided an up-front clue as to the purpose of Stonehenge, but a wealth of detailed mathematical analysis, carried out on an IBM 704 computer already in 1961, and published in Hawkins's book, makes it quite clear that the alignments of the sun, moon and the planets were a major preoccupation of prehistoric life.

Eclipses were certainly meaningful events five thousand years ago, but we have no idea specifically what they meant; did they forecast doom and remind the populace of the power of the priesthood? But more important probably was Stonehenge's guide to agriculture; when to start sowing seed, and when to expect crops to ripen.

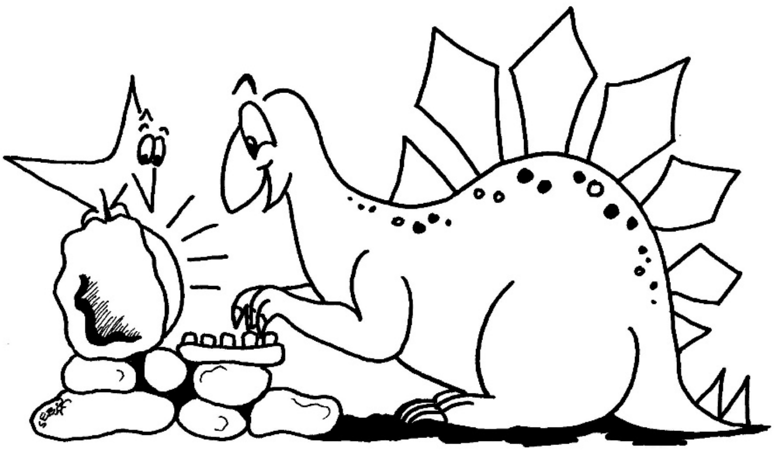
As an aside, although Hawkins describes himself as a non-computer man, his predictions of today's use of the computer are quite mind-boggling; he predicts the credit card as a precursor of a cashless society. I think his writings should be recognised in any computer-history publication.

Eventually we got to watches, possibly started by the Antikythera, built in 100 to 150 years BC and discovered in 1902 in a sunken Greek ship off a small island located between Kythera and Crete, that might have been used to predict astronomical positions and eclipses for calendar and astrological purposes.

From vast stone edifices down to watches we underwent a major decrease in magnitude similar to the decrease from the first electronic computers to today's pocket devices. We could say that the Antikythera led us to regard today's watches and clocks as computers; they are special-purpose devices that compute the commodity known as ***time***. According to the archaeologists, the Antikythera is Greek, but there have been no similar devices discovered, therefore my guess is that it is a Chinese invention. The

Chinese were good at manufacturing small metal devices, and the Silk Road brought all manner of objects from east to west–and vice-versa.

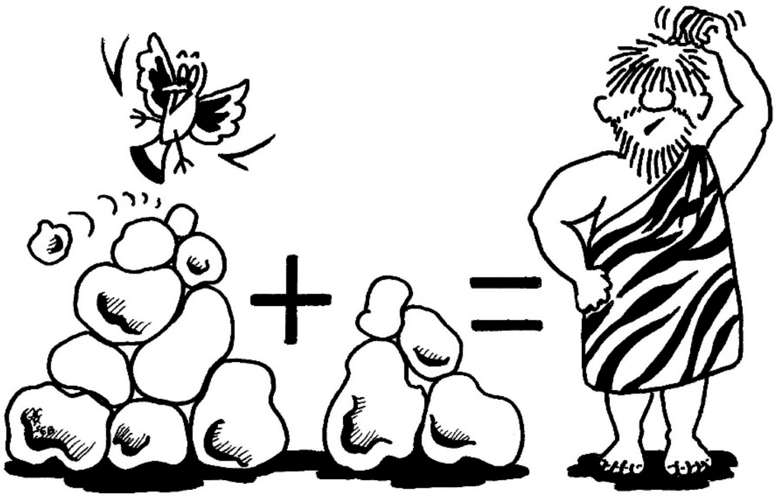
But from the idea of a watch being a computer we have the idea of measuring and counting devices through the ages; scales and thermometers as well as desk-top calculators. Thus, one way and another, early computing was around very early.



"Early computing was around very early"

But if we are discussing counting and measuring stuff and things, we are required to discuss the concept of number, together with simple arithmetic.

At rock bottom maths is about numbers. So where did they come from? Indeed, what came first, mathematics or the numbers? Did some very early Pythagoras or Newton have an idea that required the concept of number, something hitherto totally unthought-of? Or were the numbers already in existence? For non-mathematical reasons? The first question may well strike you as nonsense. What was Oink the cave-dweller doing thinking about infinity, or where two straight lines might intersect, when his job was either to hunt or to gather, as the rest of the tribe were doing? You mustn't think that just because Neolithic people lived in caves they weren't as intelligent as we are. Who do you suppose got us out of the cave?



"At rock bottom maths is about numbers"

In reality, of course, it was ***trade*** that got us into numbers. Wherever ***Homo Sapiens*** found himself on this planet he made things; boats, houses, axes, arrowheads, roads and so on. What we made was always the result of the impact of what we found at hand and our ability to invent things to do with it. And perhaps the best-known historically traded substance was silk. Not everybody was able to make silk early on–they didn't have the worms. But once the Chinese had started producing it in large quantities, their immediate neighbours wanted some, hence the Silk Roads. Hence roads in general; roads for silk, roads for cattle, roads for people, roads for armies. Roads stretching from China to Persia. Persians buying silk from unknown factories in the mysterious East, wherever that might be.

But the Chinese didn't just load up their camels and donkeys with months of production in the hope that they might get something in return at some imponderable date. And what might they have wanted? Spices, leather, honey, paper? Hence barter, and the problem of how many shoes might a unit bundle of silk be worth. Hence money. Hence contracts. Hence numbers. But if you want really convincing evidence that numbers were around back in Neolithic times, say, visit Grimes Graves in East Anglia (see colour centrefold). The picture shows you a large number of shallow holes.

Despite its name, it is not a grave, or burial place, but a flint mine worked between about 2200 to 2500 BC, just about the time of Stonehenge. Each hole is the top of a shaft dug with bone shovels to mine flint. The earth from any particular shaft was thrown into the adjacent, now empty, shafts, while the flint was used to make axes, knives and spear heads. I don't know how many shafts you can count in the picture, but you can clearly see that Grimes Graves was a Neolithic factory, and you can imagine the mountains of implements that it produced and exported, and the networks of flint roads that came about to export them–and the numbers it must have needed to handle the consequent business. When you look at the picture you see one of the birthplaces of numbers on his planet, though you'll find no mention of this if you go there on a visit. While at the other end of the Silk Road someone was exchanging a bundle of silk for half a dozen arrow heads.

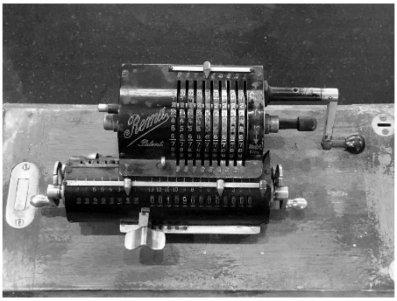
So, Grimes Graves helped us to get us started with counting things. But numbers were also needed for ***measuring*** things, for example in building pyramids; angles and lengths of sides. This in turn meant that we had to invent standards, standard lengths and distances as well as standard angles (degrees of some sort).

And following Grimes Graves we got agriculture and astronomy, exemplified by Stonehenge. Numbers were everywhere you looked. But could you write them down? One problem the Romans had was that their system of writing numbers, which consisted of I's, V's, X's, L's, C's and M's, as you know, was no good for doing mathematics; ***the Romans couldn't have invented the computer***, even if they'd had the electronics. They had the abacus for doing arithmetic with small numbers–but ironically couldn't handle the smallest; they didn't have a zero. What a computer needs is a way of representing numbers by their positions in ***space***–on a page or in a word of computer memory; a ***place-value*** system. A place-value system needs a ***base,*** ten for most cultures because we count on our fingers, (but amazingly sixty in South America, something they got from the Sumerians). It also needs a "point" to separate the integer (whole number) part from the fraction, and a row of places left and right of the point, each a "place" in which to write a digit (1 to 9) and the late arrival digit, the zero, as a placeholder. We don't consciously think of the decimal system, the one we use in Tesco's, very often because we're brought up with it.

The decimal system originated in China in the following way. Each item to be counted was represented by a stick placed in a wooden box–the *ones* box. When the contents of a box numbered ten the box was emptied and a stick was placed in the box to the left–the tens box, and so on. We don't know what they called the boxes, but they exported the idea westwards until it got to India, where they were placed and numbered on paper. The idea moved further west to the Arabs, where in 1200 they bumped into Fibonacci (1170 to 1250), who had come over from Italy to immerse himself in the latest mathematics. He brought the place-value system to Italy at around 1200, getting to Napier in Edinburgh in about 1600.

The decimal system took much time to gain acceptance in Britain, but once it had become understood and accepted, the first desk-top calculator emerged as early as 1623. This led to an explosion in experimentation in desk-top calculators–by such well-known scientists as Napier, Pascal, Dalton, Leibnitz, Odhner, Brunsviga, Friden, Monroe and Marchant.

I think the sudden appearance of these machines demonstrates the latent aspiration for mathematical machinery that had been the elephant in the room, perhaps for generations. Note that they were all *western* inventions. For some reason desk-top machinery did not arrive with the Silk Road. Why didn't the Chinese, the Indians or the Arabs invent the desk-top mechanical calculator?



I was driving some years ago in an almost impenetrable forest in the deep Swedish outback and for some reason stopped by an ancient shack, on the wall of which was a sign stating that Carl Friden had invented his desk-top calculator there. Ingenuity can pop up anywhere at any time and automatic arithmetic is no exception.

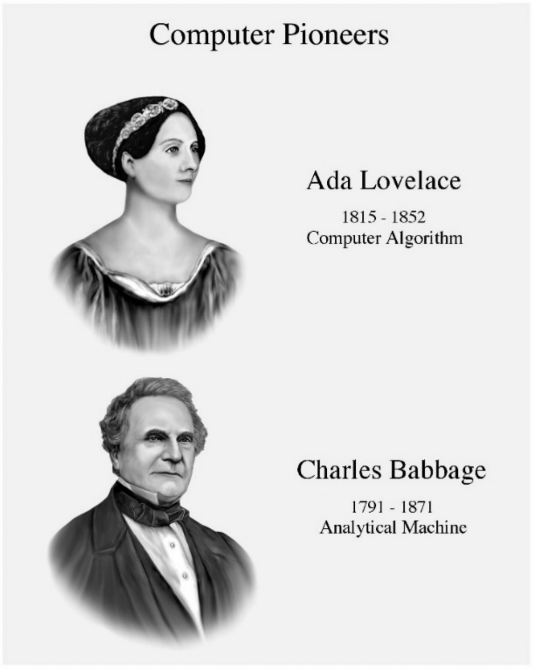
Perhaps the first automatic computer was a machine that didn't yet print paper; it produced patterned fabrics. It was a mechanical loom from the year 1840, under the control of punched cards - a technology invented by Jacquard (1752 to 1834) in 1840, and a trick adopted by Herman Hollerith, 1860 to 1929, to control mechanical accounting machines (see colour centrefold).

The problem to be solved was that of the American 1890 census. Until then, computing the data produced by each ten-year census took less than ten years, but it was predicted that the 1890 census would take longer if it were done by hand, and Hollerith was given the contract to use mechanical methods. This was successful and led to the establishment of a range of punched-card equipment that kept going until around 1950, when taken over by electronic computers.

"I wish to God these calculations had been executed by steam". So said the Astronomer Royal, Sir George Airy (1801 to 1892) to Charles Babbage (1791 to 1871) as they walked around a room full of human computers, people performing arithmetical calculations on astronomical observations–and creating errors, as we all do. Airy's goal was to make calculations accurately rather than quickly. And thus, the final step was taken in the evolution from the abacus to the digital computer.

Babbage designed two devices, the Difference Engine and the (theoretical) Analytical Machine. They were intended to be built in brass, but neither was completed in his lifetime. However, Ada, Lady Lovelace, Byron's daughter, became a very knowledgeable programmer trying to get the Analytical Machine (theoretically) operating.

Her contribution to the history of computing was writing papers on programming a digital computer; she must be regarded as the world's first programmer. But what is a programmer? Keep reading. All will be explained.



But who am I to write this book? What right do I claim for taking up space on the bookshop shelves? Or taking up your reading time? Not much really. I suppose a necessary condition, though not a sufficient one, is that I was around when the whole enterprise began to take off. But when was that? It certainly wasn't at the creation of the first computer, whenever that might have been. You can't build a machine and then be the one who actually puts it to use. But can that be true? You could make a telephone, but you couldn't use it before you'd made another one. And for it to be of viable use you'd have to make a lot of telephones, as well as an exchange–which would in turn require some training. Moreover, my experience was that people who understood electronics and logic were never happy far from making a better one, an improvement over the computer they were just finishing. But this is understandable. They'd discovered so much with the last machine that they just had to put that experience into the next one; no time to put it to use somewhere.

So, computer creators were by definition out of the running as computer ***programmers,*** which is what this book is about; what was needed was people who basically didn't really know anything much–beyond perhaps a working knowledge of arithmetic. But even that needed some additional personal traits; an unbounded thirst for knowing how things worked and the ability to ask relevant questions–in a comprehensible way. Also, what a sentence is, spoken or written–leading over time to a successful string of resulting tools for the use of someone–an organisation of some sort, be it commercial, industrial, educational, medical, or what.

Beyond that, at least in my case, was the pioneering instinct; the willingness–indeed even the urge–to go elsewhere when the current job was done; green fields. This lifestyle, though, has its sad side. Even though you may be working with computers, you are working with people too. Computers, in principal at least, were much of a muchness–unlike people–whom you inevitably left behind. So, the ability to accommodate yourself to new specimens of humanity was also a major factor–including their language, though English has very much become the ***lingua franca*** of the computer world; words like Google, on-line and e-mail.

But there are other reasons for claiming competence. Promotion–or rather a lack of it; I was never promoted. I stayed most of the time close to the details. There's a pernicious movement in companies to in some way measure a person's worth in proportion to their position on the organisation chart, which could be much at variance with their technical skills. We all need salary, but more weight should be given to technical competence than seems to be the standard of our commercial culture. What we should not do is take top quality technical people and turn them into bottom quality managers; but that’s what we’re forced to do to get the money.

I have taken a significant cut in salary to move to more interesting work elsewhere, and created new departments, leaving them when the time came to move out. In short, I stayed with the fun, not with the long-term responsibility. Of course, whenever you start something you are bound to find yourself running it; the balancing act is to be aware of the point in time when a decision has to be made between you and a replacement. I did fairly well at getting this point right. Your responsibility is to find a reliable replacement, presenting him or her with a career.

CHAPTER 2

AN INTRODUCTION TO PROGRAMMING

The first step in bringing about the computing era was building the computer; once that got started the idea spread around this planet like wildfire. But when I look back at some of the initiatives, I think there were far too many, but who then was to know? In my lectures in Trondheim (chapter 9) I did what I could to dissuade the students from redesigning what was already a *fait accompli*–with little eventual effect.

The next step was to get the computers that had already got their feet in the door of industry and academia actually working; actually earning their keep. That step consisted of writing programmes–the series of instructions required to operate the machine; to tell it what to do. The generic term for programmes is, of course, software, but even the concepts and rules around software had to be created; there was much to get done between the arrival of the gaily painted shiny hardware and bringing about the chatter of thousand-lines-a-minute printers.

But who was supposed to do the programming? The computer manufacturers couldn't be expected to know their customers' business at the level of detail required by their products; knowing how to build a computer doesn't tell you how to make or sell sausages. One application that brought this to bear early on was writing a programme to schedule the flow of petroleum through a 2,000-mile pipeline from Edmonton to Toronto in Canada. (See chapter 6.) By the time I showed up, there had been two attempts at writing the programme, one by a professor of physics–who knew how the computer worked but nothing about scheduling–and one by a petroleum consulting company who knew nothing about computers. I knew nothing about petroleum and nothing about scheduling, but they assigned someone who knew about both to tell me what to do. And after a couple of days of scheduling talk it took me just a single day to do the actual coding. Thus, the next question answers itself. Even the clever chaps who had built the EDSAC would have had to attend Petroleum 101 before they could have done it.

What does it take to be a computer programmer? Should you be an engineer, a mathematician, a logician? I know a chap who is none of these.

Indeed, he has no paper qualifications at all–apart from bog-standard school exam pass grades. Yet he's one of the best programmers I have ever known. I got him started, but I could safely do that because I had been around programmers for about thirty years and had got to know some of their characteristics reasonably well. Anyway, I was one myself and recognised something I didn't really understand, but that I relied on. Call it intuition or artificial intelligence if you like; but it doesn't matter. It's probably something that isn't recognised by the Proclaimers of Things, so doesn't have a name.

Indeed, it could be better if a programmer isn't any of the usual suspects. Primarily, a programmer must be a ***questioner***. By his very title, his job is to turn a tangle of thoughts and expressions into a device for handling precise logic; an indescribable process that needs a lot of second-guessing.

But if this is to be about the life of a programmer perhaps it needs an introductory bit about what a programme is. When we talk about computers in this day and age, we mean a ***digital*** computer, a machine that handles numbers in exactly the same way as we write numbers on sheets of paper, in order to do arithmetic with them; we add, subtract, multiply and divide them, all to some purpose–we suppose, however, there are differences between paper and electronics. The primary difference is that computers are faster than people. That's what most of the fuss was about; it's why we had computers in the first place. In fact, the very first computers were ***people***. Another difference is that you can have as many digits as you need on paper; there's always another sheet of paper, whereas a computer has a basic "word" length, the maximum number of digits a number can consist of; both integral and fractional part.

I think that we have now arrived at the point where we should usefully discuss the term "binary". Everyone knows it exists, but not everyone understands how it works. However, it's easy to understand, and the explanation starts with understanding the system we've been brought up with, the ***decimal*** system. We are born with ten fingers and thumbs, and over the millennia our ancestors used them to do their arithmetic. They almost certainly had a word for each of the numbers 1 to 10, and probably had one for zero (nothing)–though the Romans themselves didn't. But in the really early days of his existence mankind almost certainly didn't have names for the individual digits.

The key to understanding the decimal system is the ***decimal point***, that separates the integer part of a number from its fraction, as seen, for example, in the number 624.5. The 624 is the integer, while the .5 is the fraction. Furthermore, as we all know, the value of the 4 is 4x1 which equals

4, while the value of the 2 is 2x10=twenty and the value of the 6 is 6x100=six hundred. We are so used to the system that we rarely give it a second thought, but there's plenty to think about.

Each place in the integer part has a value depending on its distance from the decimal point; 1, 10, 100, 1000 and so on. We call them powers of ten, written 10^1, 10^2, 10^3 and so on; the ***decimal system.*** And we say that the system has the ***base*** ten. But ten isn't the only possible base. If we'd been born with ***twelve*** fingers and thumbs, we would have naturally created a system consisting of twelve unique characters, and each position would have a value of twelve to the something or other; powers of twelve.

Any integer can serve as a base, though I don't suggest that you mess around with any system other than the one we've got–except the number 2. Hidden away in the computer we've got that too; the memory of the computer has two "fingers", 0 and 1. It doesn't have a decimal point–but it does have a ***binary*** point–which serves the same function, separating the integral and fractional components of each number.

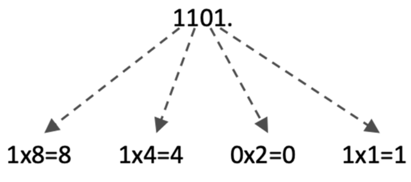
The reason why base 2 lends itself to the electronic computer is that central to electronics lies the fact that the number 2 is found all over the place; magnets have two poles, north and south; circuits can be conducting a current, or not. Stuff like that. But in addition, dealing with binary electronic components is simpler than any other arrangement.

So, what do numbers look like when dressed up in binary clothing? The answer is that we simply replace the tens in the decimal definition with twos. And this is what it looks like.

The decimal system, base ten, requires ten unique characters, while binary, base 2, tells us immediately that all we need are two, 0 and 1. That really simplifies matters; the value of each position in a number is then just a power of 2; $2^1= 2$, $2^2=4$, $2^3=8$ and so on plus, believe it or not, $2^0=1$. Thus, the position immediately to the left of the binary point has the value one, then 2, 4, 8, 16 and so on as we move to the left.

So, what does a representative number look like in binary? Consider the number: 1101.1

The integer part gives us:



Total: 8+4+0+1=13

This leaves us with the 1 to the right of the binary point. Note that along the integer row, the value of each position is half the value of the place to the left. It's the same with the fractional part, so the first position to the right of the point is one half, which =.5 in decimal. So, the complete conversion is: 1101.1 = 13.5.

So, the world around us is clothed in decimal notation, while the inner world of the computer is in binary. However, for most use of the computer we aren't bothered about this. We input numbers to the computer in decimal and allow a programme to convert it to binary for us, converting the other way on the way out.

Within any particular machine the words are of equal length, typically 32 "positions" or "places" in which you can store 32 binary digits - ***bits***. Word size was one of the attributes of a computer which influenced its evolution. At first glance, 32 zeros and ones may not sound like much, but they allow you to handle numbers in the area of a thousand million. That's a big number, and you might be journeying under computer control far away to outer space, so you do need such numbers.

But that's only talking about integers. Probably most numbers in the "memory" of a computer contain fractions; the stuff to the right of the decimal point. Probably not universally known is that most written fractions contain ***error***, and you don't want to take that error with you if you're off to outer space, if you can possibly help it.

But back to the programme. A built-in adjective applying to today's computer is that it is ***general purpose***. That means that you can do whatever you like with it within the realm of numbers. Right at the start, say in the 1940s, computers were invented to serve a single purpose, required mostly by national governments. In the US, John Mauchly and J. Presper Eckert built a machine called the ENIAC, with the sole purpose of helping field gunners manage their artillery–even though it had the fancy name of Electronic Numerical Integrator and Calculator. That was in 1942, and it was a ***special-purpose*** machine. At the same time the British Government brought about special-purpose machines to assist the code-breaking activities in Bletchley Park. Nevertheless, the experience gained with electronic valves and logical design provided much of the know-how to move on to computers that would allow you to change their function from moment to moment. This is what the programme does. One period of the day a computer could be calculating a company's payroll and the next it could be providing a report on the number of sausages sold. That's what we mean by ***general purpose***, and it is this that got the computer going, to the extent that it brought about an industrial-social revolution on this planet.

The visible machinery that you saw in the workplace was called the ***hardware***, while the programmes that produced the payroll or counted the sausages was called the ***software***. While the hardware consisted of aluminium, mercury and electricity, the software comprised ***instructions***, such as “add the number in location 25 to the number in location 143 and store the result in location 97.”

My job in life was to write programmes; I was a so-called software engineer, although I preferred to call it software creator. I never wrote software for myself, except at weekends, always for other people. I always worked in existing organisations, university departments, engineering/manufacturing, transportation, food-supply; at the operations levels as well as the corporate. I wrote a series of books trying to explain it all, and wrote numerous articles in the professional media. I worked as an advisor to the United Nations and had a stint working (unpaid) for the British government. If I were asked to summarise my career, I would say my job was to help computing get going - in all its aspects. My active period stretched from the 1950s to the late 1990s, from the very early, very large mainframe computers, which did little more than help you do the thing you had always been doing, but faster and with fewer mistakes (possibly), to pocket-sized computers that allowed you to telephone anywhere in the world, take pictures, guide you through the traffic and the geography, measure your pulse and your blood-pressure and help the architect, engineer and doctor do his or her job, all this in a single generation.

As I said earlier, programming a computer requires you above all else to be good at asking questions. There weren’t many academic studies that helped you write programmes in the early years, although the mathematical topic of ***numerical analysis*** (see Chapter 15) was a required tool, required right from the start, playing a significant role in the design of the hardware–they are all-pervading today. After all, whatever the major theme of any programme, down in the engine room (of any computer), there lies a pantheon of numbers, and today a cloud of “subroutines” that will handle them for you. I suppose the actual programming consists of understanding what the problem or function is all about and reducing it to a problem in algebra. In a sense we call this analogue-to-digital (A to D) conversion (see chapter 15). Whatever you do, at the end of the day all is ***digital***. But don’t worry, engineers will always answer your questions if you tell them you aren’t an engineer - which I’m not.

There weren’t many pioneers during those early years, but I had the privilege of knowing and working with quite a few of them. Those were energetic years and I am most grateful for fate landing me right in the middle of it. I will name some of them as I go along, and they stand out as stars in the firmament of computer people.

I'm not an engineer but I've written plenty of programmes for engineers. I've had no education in the petroleum industry, yet I put the scheduling of oil on a computer, enabling it to master-mind the flow of oil in a 2,000-mile pipeline (across Canada), and hence replace a department of ten people. Some years later I was given the job of establishing a university offshore technology department, without any academic qualifications at all in offshore technology.

Perhaps there's a clue there. Perhaps programmer paper qualifications are of no value in industry (however don't try it on). Though between you and me I have known of computing positions in which academically qualified applicants would have meant a disaster if selected. Academia has its industrial components, and I think this should be recognised. I think the reverse is also true, though it needs to be properly arranged; industry should welcome academics, but they should be deeply embedded in the employing companies. I worked in the computing department at Boeing - who created a separate ***scientific*** department, which on paper looked like a good idea. But we very rarely met them; they might just as well have been up at the University of Washington lecturing the students on something or other. My guess is that Boeing, like any other large industrial company, didn't know how to handle a scientific department and didn't tell it what its terms of reference really were.

Basic physics, accompanied by mathematics, is everywhere in flying. We already had aeronautical engineers sitting amongst the other engineers; they lived where their field of expertise was needed. There was little gained in creating a purple palace on their side of the tracks - where we never wandered naturally during the working day.

More generally in programming, because by definition you are an outsider–at least to begin with in any situation–you inevitably get drawn into the warm-up stage, the rather frenzied brainstorming that typifies programming start-ups. You will need a broad range of human attributes at your disposal. And it often leads to shouting and aspersions cast upon your ancestry. You've got to be able to take it. A stint of National Service was a most valuable component of a person's character. I had been in the RAF, and I could hold my own.

But back to the idea of programming itself. Pioneering this revolution was a vastly more complicated and frustrating undertaking than it is today; it needed a whole raft of human traits–not least tolerance of the Great Unwashed, i.e. almost everyone. There were as yet no standard accepted ways of discussing anything, though much has happened during the intervening fifty years; computer programming has become an understood study at almost every academic level around this planet. You were in a hospital, or a steelworks, or a bank or in an office of an insurance company, and you hadn't got a clue what the organisation did nor how they do it. Your first step was the interrogative stage. You need to start asking the questions, the first one being question zero; what are the questions? The organisation's champion will know. Discovering who the organisation's champion is will be our way of finding out whether you are the man or woman for the job. On the organisation chart it will tell you that it's the Managing Director. That could be true, but don't bet on it.

In the Boeing Company in my day the spiritual leader of the unofficial organisation was Dave Redhed. No meeting was held or a decision made without Dave's participation. He stoically refused every offer of promotion to a managerial position, which meant that his salary was always well below his value to the company. Everyone knew this, and when we spent evening time in everlasting meetings, we could see Dave outside playing tennis.

To get an idea of what the questions might be you need to lounge in a deep chair facing the organisation's champion–just the one–letting him or her give you the idiot's guide to it all.

So, you sit there in comfort for half an hour until you crystallise out questions 1, 2 and 3, which you then start to ask. Don't expect answers at this stage, but do expect names, those of the departments down the corporate bush who are expected to know the answers or know who knows. If there are more than two departments at any level, somebody might be too widely spread. Just discussing the expectations of a computer can lead to changes in how the organisation shall be structured.

So that was Day Zero, the day on which no programming took place. Day One will be devoted to your meeting with one of the five components of the corporate structure. They will tell you which department enters first. And day 2 will be like unto day 1, but something new will now have appeared. The day 1 picture will look like a hedgehog. It will have spikes sticking out of it labelled with numbers, all from 1 to 5, while picture 1 will have some new spikes sticking into it–from picture 2. And by this time, you will have conjured up the full 5-dimensional picture on a 2-dimensional sheet of paper, aided and abetted by judicially selected code numbers. Let the department managers do their own collective coding; don't do it for them. It will lead them to a clearer understanding of how the ball of wax works; just defining and agreeing on the codes will teach them something.

By the start of week 2 you'll have brought about the first programming step. Indeed, you will have created the corporate picture of the programme, no coding yet, but an emergent programme stands before you.

And at each level of the logical design you will have a picture of activities connected by links (constraints), the links consisting mostly of coding; statements in the corporate language being used, with an occasional activity where extra coding is required. Thus, you see that the programme consists, at all times, of a cacophony of activities and code, the latter eventually replacing the former as the level of work sinks to the bottom, the level where the programmers are happy with what they've done - their professional reward, and the customers happy with a tool with which to run the company.

Ah, but it's not over yet. Throughout the implementation of the programme, as each activity-writer claims that his activity works as described in the specification, it needs testing. We call this step Quality Control (QC), carried on by the QC department, staffed by people who really know how the company works. The healthiest fight a company can have is that between the programmers and the QC department, ***of which the MD should be a member***.

But even that isn't the whole story. The ***functions*** of a programme need to be described for everyone to see. To do this a ***documentation*** group needs to be set up already in week 1, and though the document writers need to know how to write, the bottom level detail can only come from the programmers.

And when you've landed yourself a reliable, working programme, thoroughly tested and documented, if you're going to use it you are going to have to teach the users how to use it.

At all of these junctures there has to be provision made for changing the whole caboodle - even up to the level of changing the company to fit in with the programme. And the QC function needs to check the user interface with the specification; does the programme conform to the specification? And does the documentation provide a comprehensive description?

With any computer project you should always arrange a final meeting of all who took part; the Come-to-God session, an honest discussion of every facet of the enterprise, with everyone present, will always be a healthy exercise, from which both the corporation and the computer team will learn–perhaps learn a lot. I have seen too many projects end with the project team wandering off taking expensively earned experience with them.

In the case of any organisation having contact with the general public, the ultimate question is, can the general public effortlessly use it? Indeed, can the company MD himself use it; is the MD a customer of his own company? The MD of Ryanair certainly isn't. Whenever he wants to book a flight, he gets his EA to do it for him. Thus, the MD must be a member, the ultimate member, of the QC committee. ***Nowhere in Britain is this the case***. Then, and only then, will I be able to buy my own airline ticket, and not have to bother my grandchildren. This planet is enshrouded in two spheres, the atmosphere and the infosphere.

The last sentence from the managing director has to be: "Ok, ladies and gentlemen, the show's over. Onwards and upwards to ever greater achievements".

It might be helpful to throw in a thought here concerning programmer qualification. When is the best time to start writing a programme? I always ask this at an interview, but no one produces the right answer. You get all sorts of views and wonderful suggestions, but my answer to the question is–when you've already written it; only then will you fully understand what it's all about. You can now make it run in half the time and the errors will have been expunged. And hopefully your longed-for customers can painlessly understand how to use it.