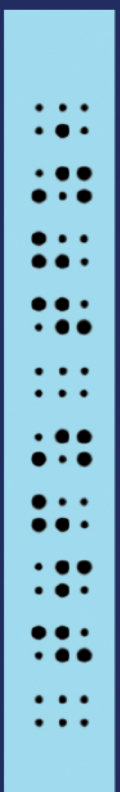
BC

BEFORE COMPUTERS

On Information Technology from Writing to the Age of Digital Data

Stephen Robertson



https://www.openbookpublishers.com

© 2020 Stephen Robertson

This work is licensed under a Creative Commons Attribution 4.0 International license (CC BY 4.0). This license allows you to share, copy, distribute and transmit the text; to adapt the text and to make commercial use of the text providing attribution is made to the authors (but not in any way that suggests that they endorse you or your use of the work).

Some of the material in this book has been reproduced according to the fair use principle which allows use of copyrighted material for scholarly purposes.

Attribution should include the following information:

© Stephen Robertson, *B C, Before Computers: On Information Technology from Writing to the Age of Digital Data*. Cambridge, UK: Open Book Publishers, 2020, https://doi.org/10.11647/OBP.0225

In order to access detailed and updated information on the license, please visit https://www.openbookpublishers.com/product/1232#copyright

Further details about CC BY licenses are available at https://creativecommons.org/licenses/by/4.0/

All external links were active at the time of publication unless otherwise stated and have been archived via the Internet Archive Wayback Machine at https://archive.org/web

ISBN Paperback: 978-1-80064-029-0
ISBN Hardback: 978-1-80064-030-6
ISBN Digital (PDF): 978-1-80064-031-3
ISBN Digital ebook (epub): 978-1-80064-104-4
ISBN Digital ebook (mobi): 978-1-80064-105-1
ISBN Digital (XML): 978-1-80064-106-8
DOI: 10.11647/OBP.0225

Cover image: Katsushika Hokusai (1760-1849), A merchant making up the account. Wikimedia https://commons.wikimedia.org/wiki/File:A_merchant_making_up_the_account.jpg Public Domain.
Cover design: Anna Gatti.

5. More about the alphabet

We have seen how important the alphabet was to many later developments in information technology. Once we have a small alphabet supplying the basic unit from which we can construct text, many other things become much easier or simply possible. Out of the inventions I have discussed so far, the most outstanding examples are movable-type printing and telegraphy.

In this chapter, I will look at four further aspects of the alphabet and alphabetic writing. The first is the way in which we separate our words when we write, mainly with spaces. The second comes directly out of telegraphy, and is the idea of encoding letters or characters with electrical pulses. Eventually, abstracted somewhat from the specific medium, the pulses will become what we now know as bits (the basic units of digital data, as we will see later). The third we have seen briefly already, on the Hughes printing telegraph: it is the keyboard, with each key representing a letter or other character. Finally, I will discuss the abstract notion of a 'character', which both I and the histories I have described have contrived to over-simplify. But first, a small anecdote.

My father was an intellectual, academic, writer, born in 1911. He wrote all his life—books, articles, reviews, letters, poems. His handwriting was all but unreadable; his main method of writing was the mechanical typewriter. He was a competent if sometimes inaccurate typist—I believe he taught himself. Despite being not at all mechanically minded, he came to some accommodation with his typewriters—he even learnt to change the ribbon, which readers of a certain age might just remember as a tricky operation. As a writer, he tended to do a lot of drafting and rewriting. So when, rather late in his life, relatively cheap word processors became available, he eventually acquired one. He was probably in his late seventies or eighty.

As with the typewriter, he became quite good at making the word processor do what he wanted. I don't remember the make, but it had a

© Stephen Robertson, CC BY 4.0 https://doi.org/10.11647/OBP.0225.05

monochrome green text-only screen, a floppy disk, and a more-or-less conventional keyboard. No mouse; no windows: special combinations of control keys would do things like move the cursor around the screen, go to the end of the text, delete whole words, save a document. He would typically open a document on which he had already done a lot of work, go to the end, and start adding to it or editing the previous days' work. But at some point he mentioned to me a small problem that he had. When opening such a document and going to the end, he would often find that the cursor was not actually at the end of the visible text, but considerably further down and to the right. He would have to use the cursor control keys to get back to where he wanted to be.

I worked out that, when deleting words at the end of the existing text (something he did quite often), he would not delete the spaces between them, or the newline characters—so they would accumulate at the end of his text. They were of course invisible—as far as he was concerned, all he had below the text was blank paper. The idea that this apparently empty space was actually part-full of invisible characters is really a very strange one—no wonder he had difficulty recognising it. But I only had to point out to him that he could delete them as well.

Spacing the words

In written English today, as in most writing systems based on alphabets, it is normal to separate the words that we write, by means of spaces—not to mention all the other things that may come between words, such as punctuation marks. This comes so naturally to us that not to do so seems perverse in the extreme. But it was not always the case. Much early writing did not separate the words at all, and even in the Roman period, although some writing would mark the word boundaries in some way, this was not always or consistently applied.

It was not until the sixth and seventh centuries CE that monks in Irish monasteries began to make systematic use of spacing. This was the period when the written culture of the West was largely kept alive in monasteries, in whose libraries books were copied out longhand. A somewhat flip explanation of the introduction of the inter-word space is that the Irish monks were not very good at Latin (the language in which all books were written).

But Paul Saenger, in his book *Space Between Words: the Origins of Silent Reading*, links the practice of word spacing to both the spread of literacy and the practice of reading silently, in one's head. Again, this comes so obviously to us in the universal literacy of the twenty-first century that it is hard to imagine the absence of this practice. But if we go back to classical Greece for a moment, writing was seen in quite a different light. A written text was something a little akin to our present notion of a written musical score: a script for an expert reader to interpret and read out loud to an audience. In such a context, the notion of making it easy for readers simply does not figure.

The spread of the practice of word spacing owes a lot to an English monk. In the late eighth century, Alcuin, a well-known teacher, was invited by King Charlemagne of the Franks to come to his court in Aachen, in order to educate Charlemagne's sons. Among very many contributions to the culture of the court and more widely, Alcuin contributed to the development of a highly legible script (what we might now call a typeface), Carolingian minuscule, and wrote a manual of writing style. It covered many of the things we now take for granted, including punctuation, paragraphs, initial capitals for sentences—as well as spaces between words.

Charlemagne encouraged and presided over a new period of high culture, and eventually became the most powerful man in Europe, reigning as Emperor over a large area. Thus the culture and practices of his court were spread far and wide, and the use of spaces spread to other scripts and languages. We will see below how inter-word spacing comes into the era of telecommunications.

Coding the letters

Although all systems of telegraphy depend on having a small alphabet, the man who saw the connection most clearly was Samuel Morse, together with his collaborator Alfred Vail. His great leap forward was to see that we can take the process one stage further, and work with an 'alphabet' of just two elements: a short and a long electrical pulse, generally referred to as dot and dash. The step is simple, requiring only a small codebook, and makes it very much easier to think about electrical processing of text.

Morse and Vail were very conscious of movable type printing as inspired

by some of the same considerations. The Morse-coding scheme involved different length codes for different letters, and they had the inspiration, which is actually the basis for some modern data compression schemes, that it would be most efficient if the commonest letters had short codes. So Vail visited a newspaper printer's workshop to count the stocks of each letter that they kept—because printers know full well exactly what stocks to keep to satisfy most printing requirements.

The idea of encoding the letters in this kind of way has gone through a number of versions since his time. Morse's short and long pulses were designed to allow humans to do the encoding and decoding easily, but we can think of them as any pair of distinguishable states (for example up/down, off/on, black/white). We then need a number of these, spread out in time or space, in groups. A code book lists each distinct group with the object it represents—for example, the Morse code book says that the letter A is represented by the group *dash-dot*, and D by *dash-dot-dot*. In this case, the things we want to represent are the letters of our usual alphabet.

If we think of this pair of elements as itself an 'alphabet'—in the abstract, rather than in a specific physical form—what we have is the modern concept of a 'bit', a binary digit. Usually nowadays we think of the two states as 0 and 1. So the letter A in Morse is 01, and D is 100. (Morse code uses different numbers of bits for different letters, but most schemes allocate codes in fixed-size groups.)

Actually, Morse was not the first to use such a binary coding scheme for letters. The familiar Braille system of embossed dots on paper (invented by Louis Braille), designed to allow the blind to read with their fingers, predates Morse code by a decade or so. The dots are in groups of six—that is, the group is a rectangular array with six positions, in each of which the dot is either present or absent. The codebook specifies which dots are actually present for each letter.

The later Baudot code (invented by Émile Baudot) used for telex is a fixed-length, 5-bit code. A small calculation will show you that this gives 32 different combinations—enough for the 26 letters of the Latin alphabet, though not for upper and lower case. Actually this is not quite enough, even if we do not care about case—it does not allow for any punctuation marks or the digits (Morse has codes for the ten digits and one or two punctuation marks). For this reason the Baudot scheme includes a shift code—a little like

the shift key on a keyboard, or more precisely a caps-lock key—which doubles up the meanings of the remaining codes. Braille uses a similar method for extending the range of characters represented.

Coding for the modern era

As we entered the computer age in the 1960s, new coding systems were defined. In fact there were two main rival schemes, EBCDIC (pronounced *ebsidik*, for IBM machines) and ASCII (pronounced *askey*, for all other computer manufacturers). I will leave EBCDIC aside, but ASCII is worth some discussion. The American Standard Code for Information Interchange is a more ambitious system than Baudot, and was used for multiple purposes in the transmission and storage of data in the early computer age, and in fact is still in use. It is a seven-bit code, allowing a total of 132 different combinations. These include the 26 letters, in both upper and lower case (making 52), the ten digits (62), a significant number of punctuation marks and special symbols (96), and 32 codes reserved for control purposes. Minor variations on this system were defined for various European languages with features not seen in English, e.g. accented characters. More systematic variation is provided for by the scheme known as ANSI, which started from ASCII, but has different code pages for different languages. Each code page provides a complete coding of a set of characters for a language—but the computer must 'know' which code page is in use to interpret ANSI correctly.

The coding scheme known as Unicode, which is currently becoming the standard for many purposes, is a much larger set. It includes not only all the characters for other alphabets than the Latin one, e.g. Greek, Russian and Arabic, but also characters for non-alphabetic languages, e.g. Japanese, Chinese. This is a fascinating development: the *idea* of a coding of characters could only have developed in the context of a small alphabet; but given the idea, it now becomes possible to apply it to much larger character sets. Unicode in its original full form requires 16 or 32 bits per character, but there are alternative encodings for the same scheme, which allows the old ASCII character set to be represented as it traditionally was, in eight bits. (Yes, I know I said seven. ASCII is a seven-bit code, but since most computers operate with multiples of eight bits, ASCII is usually embedded in eight bits.)

Although there remain some languages and scripts in the world that have not yet been incorporated into the scheme, nevertheless we seem (in the early twenty-first century) to be approaching the state where any text character in any language can be represented by means of a standard binary code. This is a remarkable achievement.

The last alphabet

The bit—the binary digit, a character from a two-letter 'alphabet'—might be seen as the final stage of a process that began when we started inventing systems for writing, something like five-and-a-half millennia ago. At the start of the third millennium CE, we realise that we can represent *any* record by means of bits. Not just language, but also, as we shall see in later chapters, numbers, images, sounds, moving pictures, and so on. The universal alphabet consists of just two symbols, a zero and a one.

But it is not just a matter of representation. We have already seen how the alphabet has helped us towards new ways of *doing* things with information. Printing, and thus the publishing revolution that followed it; and Morse code, and the telegraphic revolution that followed that: each of these would have been inconceivable if we had not invented the alphabet in the first place. Now, in the new revolution, the biggest changes have come about in the ways of *processing* information: of systems and methods and mechanisms that operate on information rather in the way that a loom operates on the raw material of thread to produce something quite different, cloth. Sending messages was just one such operation; but the possibilities are almost limitless.

The strange story of the keyboard

I am writing this text by means of a device that has become so common that it passes almost without notice: a QWERTY keyboard. This is something else that the alphabet made possible. Although my keyboard has rather more than 26 keys, its existence depends on the small number of possible characters—a keyboard with a key for each of the tens of thousands of distinct Chinese characters is quite inconceivable.

We have had keyboards for musical instruments for centuries. But the

idea of associating keys with letters of the alphabet (or with numbers, come to that) has been around since the mid-nineteenth century at least. As we have already seen, Hughes used a piano-like keyboard for his telex-like machine. Various attempts were made to develop typewriters from earlier in that century.

But the development of an effective and useable typewriter had to wait until a little later. The primary inventor was Christopher Sholes; over a period from the 1860s until the 1890s, he and Remington, the company he worked with and eventually sold out to, pushed the typewriter from the status of one of those fascinating but impractical Victorian inventions to that of a common business accoutrement. To do this, Sholes had to solve a number of tricky mechanical problems. The design and layout of the keyboard he produced, as a result of confronting these mechanical problems, is with us to this day. If Sholes were to walk into your twenty-first-century office, one of the very few things he would recognise would be the QWERTY sequence on your computer keyboard. That this is so is even more extraordinary than you could possibly have imagined.

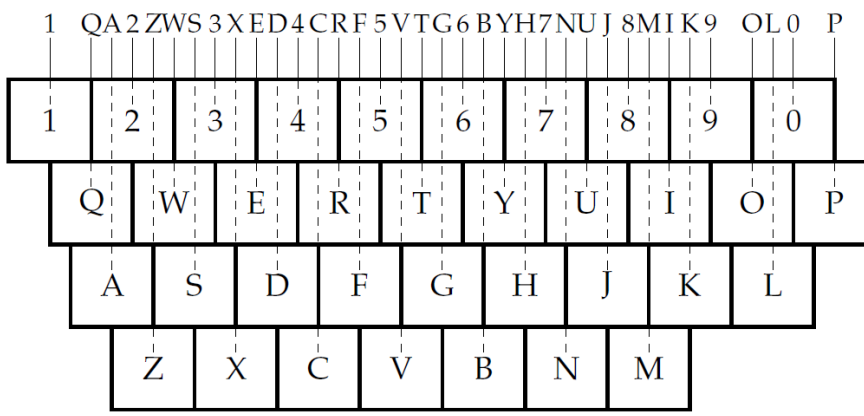


Figure 2: Basic QWERTY keyboard (with the interleaved bars of a traditional typewriter). Diagram: the author.

It's not just the sequence of letters we are talking about here. Look at Figure 2 (if you have a keyboard close to you, compare it to the diagram). In particular, look at the way that successive rows of keys are offset from

each other. Notice that the ZXCVB line is offset from the ASDFG line by half a key width; in other words, Z is half way between A and S. Also the QWERTY line has a half-key offset from the numeral line. But the offset between the QWERTY line and the ASDFG line is—what?—one-quarter of a key? But why on earth?

No, it is most certainly not any ergonomic or ease-of-use reason; in fact it is quite hard to learn to use. The reason is purely mechanical. Imagine that each key perches on the end of a metal bar, which comes out of the back of the machine (where the paper would have been). These bars have to be straight and parallel—or their movement would not be true when pressed—and cannot be allowed to interfere with each other. So they have to be carefully interleaved. The Q bar goes between the 1 bar and the 2 bar, and W between 2 and 3, that's easy. But now the A bar has to go between the Q bar and the 2 bar, S between W and 3; and Z between 2 and W, and so on. Now you see why it had to be so.

It is just possible that you have a keyboard that does not follow this offset convention. Some PDAs and other small machines, and tablets with on-screen keyboards, have the QWERTY sequence but use either no offset at all or a universal half-key offset. But real keyboards invariably use the Sholes offsets. Even if you are (say) French, and have one of those keyboards where the letter sequence is AZERTY or some other variation on Sholes, you will still have those offsets. Some keyboards are split into two parts, for ergonomic reasons associated with the way you place your hands; but they *still* use the Sholes offsets on each half.

But, you may argue, my keyboard no longer has those metal bars: indeed, you have to be of a certain age even to remember their existence. Nowadays, each key operates its own microswitch, and they could be arranged in any way we choose. So why do we persist in using these offsets? Well, this is part of the story.

Keyboard wars

In the 1880s and '90s, several rival typewriter companies were formed, and a number of different keyboard arrangements were in use. Another characteristic of the Sholes keyboard is that when they introduced lower as well as upper case, they did this by means of the familiar shift key—which has

now, of course, been joined by a few imitators, like the CTRL control and ALT alternate keys. But at least one of the rival companies just added more keys, so that upper and lower case letters were on separate keys.

As typewriting became more common, schools to train typists were set up, and various systems of fingering were devised for the different keyboards to help typists work faster—the earliest typists were almost certainly one- or two-finger typists. Claims and counter-claims were made about the relative speeds of these different combinations. And pretty soon, they became competitions.

An 8-finger method of typing was devised by Margaret Longley, who ran such a school, in the early 1880s. She applied this method to different makes of typewriter—but as applied to the Sholes keyboard, it is similar to the fingering taught today. Frank McGurrin, a court stenographer, used it with great skill on an early Remington. Another student and later principal of the Longley school, Louis Traub, used a similar fingering on Caligraph machine with a six-row keyboard.

The first competition, in 1888, pitched Traub against McGurrin. But McGurrin had a card up his sleeve. The trick was that he had discovered that he could memorise the keyboard layout, not looking at the keyboard while typing, but at the paper (he could also type blindfolded). He invented what we now know as touch-typing.

This turned out to be the ace. McGurrin thoroughly beat Taub, who shortly afterwards switched to a Remington. McGurrin went on to win many more competitions; and the keyboard never looked back. Gradually, the rival companies adopted the Sholes layout. Having a single system, a standard keyboard layout and method of typing, was a big advantage from the labour point of view. There is a delightful account of this event in Stephen Jay Gould's essay *The Panda's Thumb of Technology*, published in the collection *Bully for Brontosaurus*.

In the twentieth century, it was common to denigrate the Sholes keyboard, and to claim that it is very inefficient and unergonomic for the typist (even that it was *designed* to slow the typist down, which is not actually the case). A rival system was designed on ergonomic grounds, the Dvorak keyboard, which has the same basic structure as Sholes but a very different arrangement of the letters. In a series of experiments, it was demonstrated that Dvorak was easier to learn and faster to type on than Sholes. However,

the Sholes keyboard was so well established by then that it proved impossible to dislodge. In fact, the experiments (rather like the 1888 competition) were somewhat suspect as scientific evidence; probably the differences are not very large. Besides, Dvorak did nothing at all about the offsets, which are certainly one of the sources of ergonomic problems with the Sholes keyboard.

In the second half of the twentieth century, we saw the development of (successively, inter alia) the IBM golf-ball typewriter, the word processor, the PC, the laptop. With each of these developments, we could, in principle, have abandoned Sholes and devised something that might have been better. But this is not the way things work: technologies have to co-exist; people have to switch between them; people have to maximise the benefit they get from the investment they have put into learning something. If you are an experienced typist, your fingers remember not only the locations of the letters, but also the offsets. Even moving ASDFG a quarter-key to the right, so that all the key offsets are half-keys, would confuse you.

I once pointed out the offsets to a man with touch-typing skills who had managed to transfer them to one of those tiny PDA keyboards, about ten centimetres wide. The designers of this keyboard had retained the QWERTY layout, but (obviously not expecting anyone actually to touch-type on it) had made all the offsets a half-key. His instant response was "I *knew* there was *something* wrong with it!".

All these things conspired to ensure the persistence of almost every aspect of the Sholes design, including the offsets. Designers of laptops, with their fairly severe space limitations, have contrived to follow the Sholes offsets but to make interesting use of them by changing the shapes of the keys on the side edges of the keyboard, so as to fit into a rectangle. The laptop on which I am typing just now has a normal size shift key on the left, next to \ next to Z, but above that it has a one-and-a-half size Caps Lock key next to A, and a one-and-a-quarter Tab key next to Q. On the right, there is a Return key which is an upside down L shape covering two rows, and a one-and-three-quarter Backspace key. At the top is a row of smaller-than-standard function keys, so that more can be fitted in the row, with a few more at the bottom right.

Other languages

There is of course some variation between countries and languages. Languages that use the Roman alphabet do not have to do much to make Sholes work for them—maybe add a few accents or special characters. Non-Roman alphabets obviously need more drastic change; but it's really just a question of making substitutions. But what about non-alphabetic languages? Chinese, for example, has very many more characters than could possibly be represented on a Sholes-like keyboard.

There was a form of typewriter developed for Chinese. It consisted of a tray of several thousand embossed metal characters, each one in mirror image, like those on a traditional western typewriter. But in this case the characters are all separate, not attached to any part of the machinery. In order to type a character, the typist has to locate a movable frame above the correct character, then press a lever, which causes that character to be lifted out of the tray and struck against the ribbon and paper. Although it was possible to achieve quite fast typing speeds (if measured in words per minute), it required the typist to train for a couple of years.

Nowadays, in China as in the West, most such work is done on computers, with western (i.e. Sholes) keyboards. There are a couple of different ways of typing Chinese on a western keyboard (involving multiple keystrokes per character and/or menus), which obviously have to be learnt. But essentially this is very much easier than trying to construct a direct representation of Chinese on a keyboard.

What would Sholes think?

Let us return, for a moment, to the fantasy of communicating across time with Christopher Sholes.

If you were to go back to 1877 and explain to Sholes that his keyboard design will still be in use at the start of the next millennium, despite the fact that *every one* of the mechanical constraints that determined the design in the first place will have disappeared, he might be flattered but would probably think you a little crazy. If you were to add that a keyboard based on his design will be attached to practically every typewriter-like device in the world, *including China*—he would surely have no doubt that you were

certifiably insane.

Yet, at the start of the third millennium CE, such is the case.

The concept of a character

Earlier, I referred to 'letters or other characters'. We have already seen the letters of the alphabet and the digits of the Arabic numbering systems as characters, and inventions such as the typewriter or the ASCII coding scheme give us extra characters such as punctuation marks and currency symbols. Every so often, an old symbol gets resurrected for an entirely new purpose. Consider for example the @ sign, which used to be used to indicate the unit price of some commodity ('2lbs margarine @ 4d per lb = 8d', lb being a pound weight and d being a penny in the old UK coinage). @ has now been taken over as the universal email address sign and for other uses.

One character for which ASCII has a code (though neither Morse nor Baudot did) is the famous inter-word space, which I discussed earlier in this chapter. In this respect ASCII, following Baudot before it and inspired by the typewriter, has somewhat extended the notion of a character. Following the space-bar on the typewriter (which is treated very much like an invisible letter), ASCII defines space as a 'printable' character, distinguished from 'control' characters like *newline* or *tab*. We have now become completely familiar with the idea that the space is just another character. Further, the ASCII distinction between printable and control characters now seems rather strange, at least to computer programmers. Even Tab or Newline is just another character, with its own key on the keyboard and its own code in the coding system.

As an aside, an immense source of confusion and problems with machines has resulted from the fact that despite ASCII, there has been no agreement on which character should be used to represent the end of a line. ASCII has two, defined as 'carriage return' (CR) and 'line feed' (LF), both terms again being relics of traditional typewriters—a CR moves back to the beginning of the line on the typed page, and an LF advances by one line down the page. Files on the Windows operating system have lines ending with CRLF, on the Unix system the convention is LF, and on the Apple Mac it used to be CR. And this is not an exhaustive list of the conventions that have been used!

ASCII also distinguishes clearly (as Morse did before it) between letters and numbers. The Sholes typewriter on the other hand had digit keys for 2-9 but not for zero or one; the typewriting convention was to use lower-case ell for one and upper-case oh for zero.

I have assumed, and the typewriter and the ASCII scheme both encourage me to assume, that there are well-defined, separable things called characters, not only in alphabetic systems but also in syllabaries and other writing schemes. This is something of an over-simplification; we can see oddities even within ASCII, and more so when we consider other languages than English. Actually, much the cleanest character system is the Chinese: each character is self-contained and occupies a square block on the page—beautifully simple, if you forget for a moment about the number of different characters.

One oddity in ASCII is that it has *two* codes for every English letter: lower and upper case. The ASCII code for 'A' is different from that for 'a'. There is some reason for this—although there are certain rules about when to use capitals and when to use lower case, these rules are not clear or unambiguous enough for us to leave the decision to a machine. So when we are typing, we use the shift key to indicate a capital letter, and the coding is done accordingly.

We could easily have decided to make the shift key a character in its own right, a control code saying to the machine 'Now go into upper case', either to apply only to the following character, or 'locking' and requiring a corresponding 'down-again' code (this latter method is used in the Baudot system to represent numbers). But we didn't. On the other hand we typically use exactly this method to represent font or typeface variations; I have to use some such convention in order to produce *italics* or **boldface** in this book. We don't have separate codes for Roman A, italic *A* and/or boldface **A**.

This decision has many ramifications. Think for example of how names are typically organised in a directory, or words in a dictionary or an index. Traditionally, we do not distinguish between upper and lower case when arranging things in dictionary order. Similarly we expect modern search engines not to distinguish. But these expectations require our machines to be told that (for some purposes at least) 'A' and 'a' are the same.

In Arabic, each letter has not two but four different forms. But here the

rules are pretty clear: the four shapes occur when the letter is (1) at the beginning of a word, (2) at the end of a word, (3) elsewhere in a word, and (4) on its own. Although making a traditional typewriter do that would be hard, this decision can now safely be left to a machine, so only one code for the letter is necessary.

In English printing, we have some letters that are usually, in many typefaces, joined together—called ligatures. The most common examples are f l, represented as fl, and f i, represented as fi. The typesetting system I am using for this book will do these ligatures automatically for me (except that I have to tell it *not* to do so when I want to show the letters separately). In older books, you sometimes see other ligatures (for example s and t are sometimes joined), though most other ligatures have now died out. Also it is traditional to form a single character from an a followed by an e in some circumstances, for example 'archæology'. But this example is more tricky, for two reasons. First, it only applies to some words of Latin origin, it is not a general rule for when these letters occur together. Second, if it is ever encountered in modern English it is regarded as a ligature of the two letters a and e—dictionary order treats it as two separate letters. But in the Scandinavian languages and in Old English, this character is regarded as a letter in its own right, with a position in the alphabetical order distinct from the two component vowels.

As we explore other languages, we find many complex examples. In German there is a double-s symbol ß (nevertheless treated as two esses in alphabetical order). In Spanish we have a letter that is printed as ll (two ells), but is regarded as a single letter in its own right, with its own alphabetical position. Decorations on characters, such as accents or umlauts or cedillas, introduce their own complications. Sanskrit is written in an alphabetical system (the Devanagari script), but all the letters that make up a syllable are joined by ligature into a single syllable-symbol; there are hundreds of different ligatures. Sanskrit and Arabic also share the property that vowels are typically regarded as decorations on the consonants, rather than letters in their own right.

These complexities are hard to deal with in a coding system; eventually, instead of representing self-contained characters, some codes have to be used to represent instructions to the machine as to how to interpret the characters. or how to render them in readable form.

So the idea of a 'character' is a little complex. We English speakers and writers are lucky to be spared some of these complexities. And, just possibly, the development of computing in the English-speaking world benefited from the relative simplicity of our script.