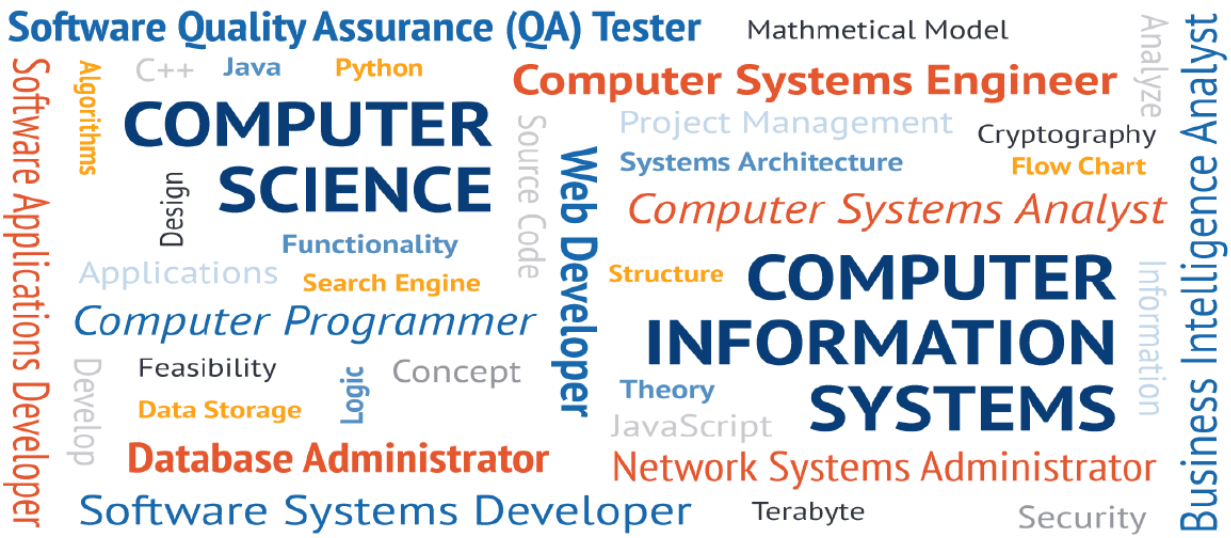
Taverham Sixth Form
One vision. Our passion. Your future.

Computer Science



Why am I taking computer science? What doors does this open up for me?

We are living in a digital age which has a growing need for computer scientists because computer programmes have all but infiltrated every aspect of our lives from the way we do our shopping to how we perform our operations. As a computer scientist you will need to plan / design and apply the software and hardware for the programmes we use in our daily lives.

This specification has been designed for students who wish to go on to higher education courses or employment where knowledge of computing would be beneficial. One can study computer science and go on to a career in medicine, law, business, engineering or any type of science. This course feeds into computer science/studies university courses or for those who wish to progress into writing their own programs/games/apps/software. It is also suitable for those candidates wishing to have a greater insight into how PCs actually work, which would suit those whose career will take them heavily into ICT as every industry uses computers in one way or another so the potential of future employment is high.

Examination Board: AQA **Course Title:** Computer Science

Subject Content

The following table shows the theory topics that you will be covering over the two years. By clicking on the topic you will be taken to the relevant section on the AQA website.

10	Fundamentals of programming	17	Consequences of uses of computing
11	Fundamentals of data structures	18	Fundamentals of communication and networking
12	Fundamentals of algorithms	19	Fundamentals of databases
13	Theory of computation	20	Big Data
14	Fundamentals of data representation	21	Fundamentals of functional programming
15	Fundamentals of computer systems	22	Systematic approach to problem solving
16	Fundamentals of computer organisation and architecture	23	Non-exam assessment: the computing practical project

Assessment of Subject

	Paper 1	Paper 2	NEA
What's assessed?	This paper tests your ability to program, as well as your theoretical knowledge of Computer Science from subject sections 10-13 above and the skills from section 22	This paper tests your ability to answer questions from subject sections 14-21 above	The non-exam assessment assesses your ability to use the knowledge and skills gained through the course to solve or investigate a practical problem using skills from section 22
How assessed?	On screen exam: 2 hours 30 minutes 40% of A-level	Written exam: 2 hours 30 minutes 40% of A-level	Assessed: 75 marks 20% of A-level
Style of questions	A series of short questions and write/adapt/extend programs in an electronic answer document	Compulsory short-answer and extended-answer question	

Useful Websites

Computing at School
A community of computing teachers, industry professionals and other interested parties, with an active discussion forum and resource sharing website.

Stack Overflow
A language-independent collaboratively edited question and answer site for programmers

CS4FN (Computer Science for Fun)
Website run by Queen Mary College, University of London, produces an online and in print magazine for students and teachers covering a range of computer science topics

Python Programming on YouTube
Python tutor videos with schematic animations

Codeacademy
Are your Python skills not up to scratch? Use this online tutorial to help build your confidence up – a necessity on this course.

Online Interactive Modules for Teaching Computer Science
Animations covering Computer Science concepts, from Virginia Tech

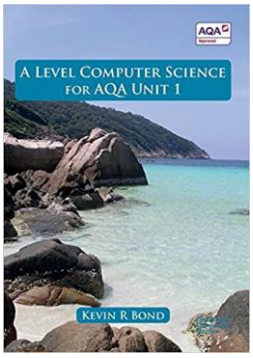
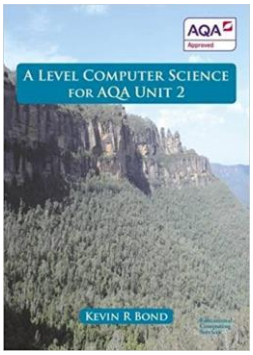
Past Papers

Since this is a new qualification there are none specifically for this. Previous specification exams are available on the exam board website with the mark schemes and the examiners' reports, all of which are essential background reading for the student who wants to succeed:

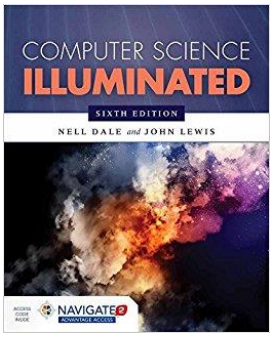
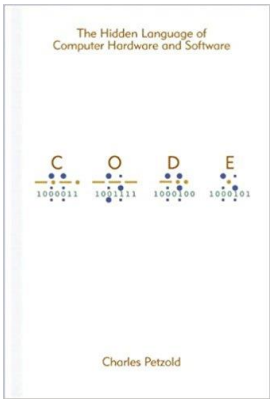
Assessment resources

Resources

The following books are essential to study computer science at A-level

Title		Author	Publisher	ISBN-13
A-Level Computer Science For AQA Unit 1		Kevin Roy Bond	Educational Computing Services Ltd (9 Feb 2017)	978-0992753610
A-Level Computer Science For AQA Unit 2		Kevin Roy Bond	Educational Computing Services Ltd (9 Feb 2017)	978-0992753627

The following books are not essential to study computer science at A-level but they would certainly help and guide you through the subject knowledge.

Title		Author	Publisher	ISBN-13
Computer Science Illuminated Sixth Edition Includes Navigate 2 Advantage Access		Nell Dale John Lewis)	Jones and Bartlett Publishers 6th edition (25 Feb 2015)	978-1284055917
Code: The Hidden Language of Computer Hardware and Software		Charles Petzold	Microsoft Press 1st edition (21 Oct 2000)	978-0735611313

Summer Task

Below is an introduction to the beginnings of computer science, and a historical view on all of the elements that combined to develop computers as we know them today. Your task this summer is to read the text below and to answer the questions that follow. This will give you a strong foundation for what is then taught in the A-level and an understanding for how the separate parts of a computer system are intimately linked.

The History of Computing

The historical foundation of computing goes a long way toward explaining why computing systems today are designed as they are. We examine the history of computing hardware and software separately because each has its own impact on how computing systems evolved into the layered model we use to understand computers. This history is written as a narrative, with no intent to formally define the concepts discussed.

A Brief History of Computing Hardware

The devices that assist humans in various forms of computation have their T0015 in the ancient past and have continued to evolve until the present deity. Let's take a brief tour through the history of computing hardware.

Early History

Many people believe that Stonehenge, the famous collection of rock monoliths in Great Britain, is an early form of a calendar or astrological calculator. The abacus, which appeared in the sixteenth century BC, was developed as an instrument to record numeric values and on which a human can perform basic arithmetic.

Stonehenge Is Still a Mystical Place

© vencavolrab/iStock/Thinkstock

Stonehenge, a Neolithic stone structure that rises majestically out of the Salisbury Plain in England, has fascinated humans for centuries. It is believed that Stonehenge was erected over several centuries beginning in about 2180 BC. Its purpose is still a mystery, although theories abound. At the summer solstice, the rising sun appears behind one of the main stones, giving the illusion that the sun is balancing on the stone. This has led to the early theory that Stonehenge was a temple. Another theory, first suggested in the middle of the twentieth century, is that Stonehenge could have been used as an astronomical calendar, marking lunar and solar alignments. Yet a third theory is that Stonehenge was used to predict eclipses. The latest research now shows that Stonehenge was intended for and used as a cemetery.[3] Human remains, from about 3000 BC until 2500 BC when the first large stones were raised, have been found. Regardless of why it was built, there is a mystical quality about the place that defies explanation.

In the middle of the seventeenth century, Blaise Pascal, a French mathematician, built and sold gear-driven mechanical machines, which performed whole-number addition and subtraction. Later in the seventeenth century, a German mathematician, Gottfried Wilhelm von Leibniz, built the first mechanical device designed to do all four whole-number operations: addition, subtraction, multiplication, and division. Unfortunately, the state of mechanical gears and levers at that time was such that the Leibniz machine was not very reliable.

In the late eighteenth century, Joseph Jacquard developed what became known as Jacquard's loom, used for weaving cloth. The loom used a series of cards with holes punched in them to specify the use of specific coloured thread and therefore dictate the design that was woven into the cloth. Although not a computing device, Jacquard's loom was the first to make use of an important form of input: the punched card.

It wasn't until the nineteenth century that the next major step was taken, this time by a British mathematician. Charles Babbage designed what he called his analytical engine. His design was too complex for him to build with the technology of his day, so it was never implemented. His vision, however, included many of the important components of today's computers. Babbage's design was the first to include a memory so that intermediate values did not have to be re-entered. His design also included the input of both numbers and mechanical steps, making use of punched cards similar to those used in Jacquard's loom.

Ada Augusta, Countess of Lovelace, was a very romantic figure in the history of computing. Ada, the daughter of Lord Byron (the English poet), was a skilled mathematician. She became interested in Babbage's work on the analytical engine and extended his ideas (as well as correcting some of his errors). Ada is credited with being the first programmer. The concept of the loop—a series of instructions that repeat – is attributed to her. The programming language Ada, used largely by the U.S. Department of Defense, is named for her.

During the later part of the nineteenth century and the beginning of the twentieth century, computing advances were made rapidly. William Burroughs produced and sold a mechanical adding machine. Dr Herman Hollerith developed the first electro-mechanical tabulator, which read information from a punched card. His device revolutionized the census taken every ten years in the United States. Hollerith later formed a company known today as IBM.

In 1936, a theoretical development took place that had nothing to do with hardware per se but profoundly influenced the field of computer science. Alan Turing, another British mathematician, invented an abstract mathematical model called a Turing machine, laying the foundation for a major area of computing theory. The most prestigious award given in computer science (equivalent to the Fielding Medal in mathematics or a Nobel Prize in other sciences) is the Turing Award, named for Alan Turing. A recent Broadway play deals with his life. Analysis of the capabilities of Turing machines is a part of the theoretical studies of all computer science students.

In the mid to late 1930s, work on building a computing machine continued around the world. In 1937, George Stibitz constructed a 1-bit binary adder using relays. Later that year, Claude E. Shannon published a paper about implementing symbolic logic using relays. In 1938, Konrad Zuse of Berlin built the first mechanical binary programmable computer.

By the outbreak of World War II, several general-purpose computers were under design and construction. In London in 1943, Thomas Flowers built the Colossus, considered by many to be the first all-programmable electronic digital computer (figure 1.4). In 1944, the IBM Automatic Sequence Controlled Calculator was given to Harvard; it was subsequently known as the Harvard Mark I. The ENIAC, pictured in figure 1.5, was unveiled in 1946.

John von Neumann, who had served as a consultant on the ENIAC project, started work on another machine known as EDVAC, which was completed in 1950. In 1951, the first commercial computer, UNIVAC I, was delivered to the U.S. Bureau of the Census. The UNIVAC I was the first computer used to predict the outcome of a presidential election.

The early history that began with the abacus ended with the delivery of the UNIVAC I. With the building of that machine, the dream of a device that could rapidly manipulate numbers was realized; the search was ended. Or was it? Some experts predicted at that time that a small number of computers would be able to handle the computational needs of mankind. What they didn't realize was that the ability to perform fast calculations on large amounts of data would radically change the very nature of fields such as mathematics, physics, engineering, and economics. That is, computers made those experts' assessments of what needed to be calculated entirely invalid.

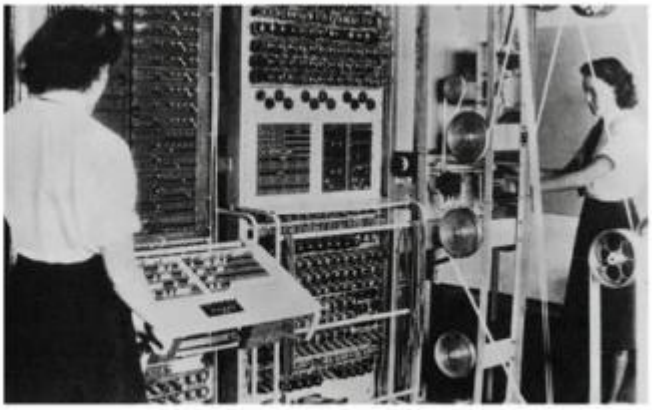


FIGURE 1.4 The Colossus, the first all-programmable digital computer
© Pictorial Press Ltd/Alamy Images

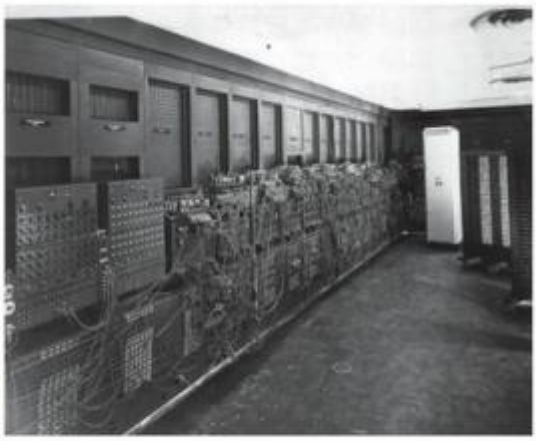


FIGURE 1.5 The ENIAC, a World War II–era computer
Courtesy of U.S. Army.

After 1951, the story becomes one of the ever-expanding use of computers to solve problems in all areas. From that point, the search has focused not only on building faster, bigger devices, but also on developing tools that allow us to use these devices more productively. The history of computing hardware from this point on is categorized into several "generations" based on the technology they employed.

First Generation (1951—1959)

Commercial computers in the first generation (from approximately 1951 to 1959) were built using vacuum tubes to store information. A vacuum tube, shown in figure 1.6, generated a great deal of heat and was not very reliable. The machines that used them required heavy-duty air conditioning and frequent maintenance. They also required very large, specially built rooms.

FIGURE 1.6 A vacuum tube
© SPbPhoto/Shutterstock, Inc.

The primary memory device of this first generation of computers was a magnetic drum that rotated under a read/write head. When the memory cell that was being accessed rotated under the read/write head, the data was written to or read from that place.

The input device was a card reader that read the holes punched in an IBM card (a descendant of the Hollerith card). The output device was either a punched card or a line printer. By the end of this generation, magnetic tape drives had been developed that were much faster than card readers. Magnetic tapes are sequential storage devices, meaning that the data on the tape must be accessed one after another in a linear fashion.

Storage devices external to the computer memory are called auxiliary storage devices. The magnetic tape was the first of these devices. Collectively, input devices, output devices, and auxiliary storage devices became known as peripheral devices.

Second Generation (1959–1965)

The advent of the transistor (for which John Bardeen, Walter H. Brattain, and William B. Shockley won a Nobel Prize) ushered in the second generation of commercial computers. The transistor replaced the vacuum tube as the main component in the hardware. The transistor, as shown in figure 1.7, was smaller, more reliable, faster, more durable, and cheaper.

The second generation also witnessed the advent of immediate-access memory. When accessing information from a drum, the CPU had to wait for the proper place to rotate under the read/write head. The second generation used memory made from magnetic cores, tiny doughnut-shaped devices, each capable of storing one bit of information. These cores were strung together with wires to form cells, and cells were combined into a memory unit. Because the device was motionless and was accessed electronically, information was available instantly.

FIGURE 1.7 A transistor, which replaced the vacuum tube
Courtesy of Dr. Andrew Wylie

The magnetic disk, a new auxiliary storage device, was also developed during the second computer hardware generation. The magnetic disk is faster than magnetic tape because each data item can be accessed directly by referring to its location on the disk. Unlike a tape, which cannot access a piece of data without accessing everything on the tape that comes before it, a disk is organized so that each piece of data has its own location identifier, called an address. The read/write heads of a magnetic disk can be sent directly to the specific location on the disk where the desired information is stored.

Third Generation (1965–1971)

In the second generation, transistors and other components for the computer were assembled by hand on printed circuit boards. The third generation was characterized by integrated circuits (ICs), solid pieces of silicon that contained the transistors, other components, and their connections. Integrated circuits were much smaller, cheaper, faster, and more reliable than printed circuit boards. Gordon Moore, one of the co-founders of Intel, noted that from the time of the invention of the IC, the number of circuits that could be placed on a single integrated circuit was doubling each year. This observation became known as Moore's law.

Transistors also were used for memory construction, where each transistor represented one bit of information. Integrated-circuit technology allowed memory boards to be built using transistors. Auxiliary storage devices were still needed because transistor memory was volatile; that is, the information went away when the power was turned off.

The terminal, an input/output device with a keyboard and screen, was introduced during this generation. The keyboard gave the user direct access to the computer, and the screen provided an immediate response.

Fourth Generation (1971–?)

Large-scale integration characterizes the fourth generation. From several thousand transistors on a silicon chip in the early 1970s, we had moved to a whole microcomputer on a chip by the middle of this decade. Main memory devices are still made almost exclusively out of chip technology. Over the previous 40 years, each generation of computer hardware had become more powerful in a smaller package at lower cost. Moore's law was modified to say that chip density was doubling every 18 months.

By the late 1970s, the phrase personal computer (PC) had entered the vocabulary. Microcomputers had become so cheap that almost anyone could have one, and a generation of kids grew up playing Pac-Man.

The fourth generation found some new names entering the commercial market. Apple, Tandy/Radio Shack, Atari, Commodore, and Sun joined the big companies of earlier generations – IBM, Remington Rand, NCR, DEC (Digital Equipment Corporation), Hewlett-Packard, Control Data, and Burroughs. The best-known success story of the personal computer revolution is that of Apple. Steve Wozniak, an engineer, and Steve Jobs, a high school student, created a personal computer kit and marketed it out of a garage. This was the beginning of Apple Computer, a multibillion-dollar company.

The IBM PC was introduced in 1981 and was soon followed by compatible machines manufactured by many other companies. For example, Dell and Compaq were successful in making PCs that were compatible with IBM PCs. Apple introduced its very popular Macintosh microcomputer line in 1984.

In the mid-1980s, larger, more powerful machines were created; they were referred to as workstations. Workstations were generally meant for business, not personal, use. The idea was for each employee to have his or her own workstation on the desktop. These workstations were connected by cables, or networked, so that they could interact with one another. Workstations were made more powerful by the introduction of the RISC (reduced-instruction-set computer) architecture. Each computer was designed to understand a set of instructions, called its machine language. Conventional machines such as the IBM 370/168 had an instruction set containing more than 200 instructions. Instructions were fast and memory access was slow, so specialized instructions made sense. As memory access got increasingly faster, using a reduced set of instructions became attractive. Sun Microsystems introduced a workstation with a RISC chip in 1987. Its enduring popularity proved the feasibility of the RISC chip. These workstations were often called UNIX workstations because they used the UNIX operating system.

Because computers are still being made using circuit boards, we cannot mark the end of this generation. However, several things have occurred that so dramatically affected how we use machines that they certainly have ushered in a new era. Moore's law was once again restated in the following form: "Computers will either double in power at the same price or halve in cost for the same power every 18 months."

Parallel Computing

Although computers that use a single primary processing unit continue to flourish, radically new machine architectures began appearing in the late 1980s. Computers that use these parallel architectures rely on a set of interconnected central processing units.

One class of parallel machines is organized so that the processors all share the same memory unit. In another class of machines, each central processor has its own local memory and communicates with the others over a very fast internal network. Parallel architectures offer several ways to increase the speed of execution. For example, a given step in a program can be separated into multiple pieces, and those pieces can be executed simultaneously on several individual processors. These machines are called SIMD (single-instruction, multiple-data-stream) computers. A second class of machines can work on different parts of a program simultaneously. These machines are called MIMD (multiple-instruction, multiple-data-stream) computers. The potential of hundreds or even thousands of processors combined in one machine is enormous, and the challenge of programming for such machines is equally daunting. Software designed for parallel machines is different from software designed for sequential machines. Programmers have to rethink the ways in which they approach problem solving and programming to exploit parallelism.

Networking

In the 1980s, the concept of a large machine with many users gave way to a network of smaller machines connected so that they can share resources such as printers, software, and data. ethernet, invented by Robert Metcalfe and David Boggs in 1973, used a cheap coaxial cable to connect the machines and a set of protocols to allow the machines to communicate with one another. By 1979, DEC, Intel, and Xerox joined to establish ethernet as a standard.

Workstations were designed for networking, but networking personal computers didn't become practical until a more advanced Intel chip was introduced in 1985. By 1989, Novell's Netware connected PCs together with a file server, a PC with generous mass storage and good input/output capability. Placing data and office automation software on the server rather than each PC having its own copy allowed for a measure of central control while giving each machine a measure of autonomy. Workstations or personal computers networked together became known as LANs (local area networks).

The internet as we know it today is descended from the ARPANET, a government-sponsored network begun in the late 1960s, which originally consisted of 11 nodes concentrated mainly in the Los Angeles and Boston areas. Like ARPANET and LANs, the internet uses packet switching, a way for messages to share lines. The internet, however, is made up of many different networks across the world that communicate by using a common protocol, TCP/IP (Transmission Control Protocol/Internet Protocol). Paul E. Ceruzzi, in A History of Modern Computing, comments on the relationship between ethernet and the internet:

> *If the internet of the 1990s became the Information Superhighway, then ethernet became the equally important network of local roads to feed it. As a descendent of ARPA research, the global networks we now call the internet came into existence before the local ethernet was invented at Xerox. But ethernet transformed the nature of office and personal computing before the internet had a significant effect.*

Questions (Section 1)

Write your answers in full sentences. These will serve as notes that we will sometimes refer back to throughout the course.

For exercises 1 – 10, choose from the following list of people:

- Leibniz
- Pascal
- Babbage
- Lovelace
- Hollerith
- Byron
- Turing
- Jacquard

1. What French mathematician built and sold the first gear-driven mechanical machine that did addition and subtraction?
2. Who built the first mechanical machine that did addition,~ subtraction, multiplication, and division?
3. Who designed the first mechanical , machine that included memory?
4. Who was considered the first programmer?
5. Who proposed that a punched card be used for counting the census?
6. Who edited Babbage's work?
7. Who was Ada Lovelace's father?
8. Who would have been mentioned in the book the Code Breakers?
9. Who developed the concept of punched holes used in weaving cloth?
10. Who is associated with IBM?

For exercises 11 – 23, match the hardware listed to the appropriate generation:

A. First
B. Second
C. Third
D. Fourth
E. Fifth

11. Circuit boards
12. Transistor
13. Magnetic core memory
14. Card input/ output
15. Parallel computing
16. Magnetic drum
17. Magnetic tape drives
18. Integrated circuits
19. Personal computer
20. Vacuum tube
21. Large-scale integration
22. Magnetic disk
23. Networking

A Brief History of Computing Software

The hardware of a computer can be turned on, but it does nothing until it is directed to do so by the programs that make up the computer's software. The manner in which software evolved is crucial to understanding how software works in a modern computing system.

First Generation Software (1951–1959)

The first programs were written using machine language, the instructions built into the electrical circuitry of a particular computer. Even the small task of adding two numbers together used three instructions written in binary (1s and 0s), and the programmer had to remember which combination of binary digits meant what. Programmers using machine language had to be very good with numbers and very detail oriented. It's not surprising that the first programmers were mathematicians and engineers. Nevertheless, programming in machine language is both time-consuming and prone to errors.

Because writing in machine code is so tedious, some programmers took the time to develop tools to help with the programming process. Thus the first artificial programming languages were developed. These languages, called assembly languages, used mnemonic codes to represent each machine-language instruction.

Because every program that is executed on a computer eventually must be in the form of the computer's machine language, the developers of assembly language also created software translators to translate programs written in assembly language into machine code. A program called an assembler reads each of the program's instructions in mnemonic form and translates it into the machine-language equivalent. These mnemonics are abbreviated and sometimes difficult to read, but they are much easier to use than long strings of binary digits.

The programmers who wrote these tools to make programming easier for others were the first systems programmers. So, even in first-generation software, there was the division between those programmers who wrote tools and those programmers who used the tools. The assembly language acted as a buffer between the programmer and the machine hardware. See figure 1.8. Sometimes, when efficient code is essential, programs today may be written in assembly language.

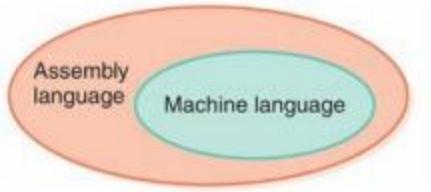


FIGURE 1.8 Layers of languages at the end of the first generation

Second Generation Software (1959–1965)

As hardware became more powerful, more powerful tools were needed to use it effectively. Assembly languages certainly presented a step in the right direction, but the programmer still was forced to think in terms of individual machine instructions. The second generation saw more powerful languages developed. These high-level languages allowed the programmer to write instructions using more English-like statements.

Two of the languages developed during the second generation are still used today: FORTRAN (a language designed for numerical applications) and COBOL (a language designed for business

applications). FORTRAN and COBOL developed quite differently. FORTRAN started out as a simple language and grew as additional features were added to it over the years. In contrast, COBOL was designed first and then implemented. It has changed little over time.

Another language that was designed during this period that remains in use today is Lisp. Lisp differs markedly from FORTRAN and COBOL and was not widely accepted. It was used mainly in artificial intelligence applications and research. Indeed, dialects of Lisp are among the languages of choice today in artificial intelligence. Scheme, a dialect of Lisp, is used at some schools as an introductory programming language.

The introduction of high-level languages provided a vehicle for running the same program on more than one computer. Each high-level language has a translating program that goes with it, a program that takes statements written in the high-level language and converts them to the equivalent machine-code instructions. In the earliest days, the high-level language statements were often translated into an assembly language, and then the assembly-language statements were translated into machine code. A program written in FORTRAN or COBOL can be translated and run on any machine that has a translating program called a compiler.

At the end of the second generation, the role of the systems programmer was becoming more well-defined. Systems programmers wrote tools like assemblers and compilers; those people who used the tools to write programs were called applications programmers. The applications programmer was becoming even more insulated from the computer hardware as the software surrounding the hardware became more sophisticated.

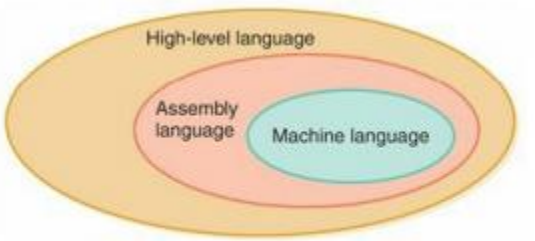


FIGURE 1.9 Layers of language at the end of the second generation

Third Generation Software (1965–1971)

During the third generation of commercial computers, it became apparent that the human was slowing down the computing process. Computers were sitting idle while waiting for the computer operator to prepare the next job. The solution was to put the computer resources under the control of the computer – that is, to write a program that would determine which programs were run when. This kind of program is called an operating system.

During the first two computer software generations, utility programs had been written to handle often-needed tasks. Loaders loaded programs into memory and linkers linked pieces of large programs together. In the third generation, these utility programs were refined and put under the direction of the operating system. This group of utility programs, the operating system, and the language translators (assemblers and compilers) became known as systems software.

The introduction of computer terminals as input/output devices gave users ready access to computers, and advances in systems software gave machines the ability to work much faster. However, inputting and outputting data from keyboards and screens was a slow process, much slower than carrying out instructions in memory. The problem was how to make better use of the machine's greater capabilities and speed. The solution was time sharing – many different users, each at a terminal, communicating (inputting and outputting) with a single computer all at the same time. Controlling this process was an operating system that organized and scheduled the different jobs.

For the user, time sharing is much like having his or her own machine. Each user is assigned a small slice of central processing time and then is put on hold while another user is serviced. Users generally aren't even aware that there are other users. However, if too many people try to use the system at the same time, there can be a noticeable wait for a job to be completed.

As part of the third generation, general-purpose application programs were being written. One example was the Statistical Package for the Social Sciences (SPSS), which was written in FORTRAN. SPSS had a special language, and users wrote instructions in that language as input to the program. This language allowed the user, who was often not a programmer, to describe some data and the statistics to be computed on that data.

At the beginning of the computer era, the computer user and the programmer were the same person. By the end of the first generation, programmers had emerged who wrote tools for other programmers to use, giving rise to the distinction between systems programmers and applications programmers. However, the programmer was still the user. In the third generation, systems programmers were writing programs – software tools – for others to use. Suddenly, there were computer users who were not programmers in the traditional sense.

The separation between the user and the hardware was growing wider. The hardware had become an even smaller part of the picture. A computer system – a combination of hardware, software, and the data managed by them – had emerged. See figure 1.10. Although the layers of languages kept getting deeper, programmers continued (and still continue) to use some of the very inner layers. If a small segment of code must run as quickly as possible and take up as few memory locations as possible, it may still be programmed in an assembly language or even machine code.

Fourth Generation (1971–1989)

The 1970s saw the introduction of better programming techniques called structured programming, a logical, disciplined approach to programming. The languages Pascal and Modula-2 were built on the principles of structured programming. BASIC, a language introduced for third-generation machines, was refined and upgraded to more structured versions. C, a language that allows the user to intersperse assembly-language statements in a high-level program, was also introduced. C++, a structured language that allows the user access to low-level statements as well, became the language of choice in the industry.

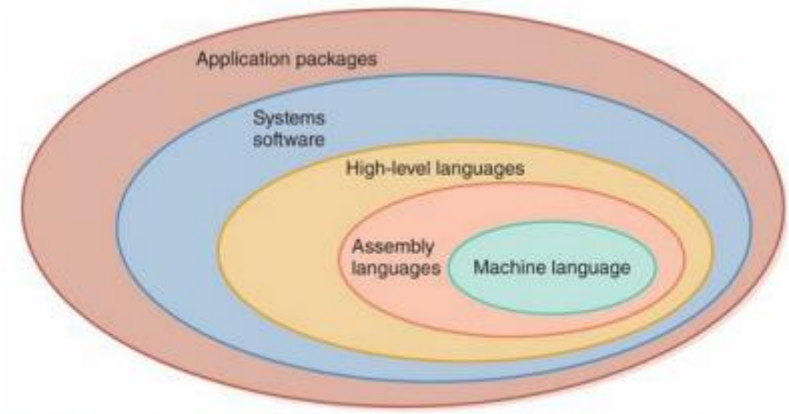


FIGURE 1.10 The layers of software surrounding the hardware continue to grow

Better and more powerful operating systems were being developed, too. UNIX, developed at AT&T™ as a research tool, has become standard in many university settings. PC-DOS, developed for the IBM PC, and MS-DOS, developed for PC compatibles, became standards for personal computers. Apple capitalized on research done at Xerox PARC by incorporating a mouse and point-and-click graphical interface into the operating system for the Macintosh, which ushered in an important change to computer–user interaction on personal computers.

High-quality, reasonably priced applications software packages became available at neighbourhood stores. These programs allow the user with no computer experience to perform a specific task. Three typical kinds of application packages are spreadsheets, word processors, and database management systems. Lotus 1-2-3 was the first commercially successful spreadsheet that allowed a novice user to enter and analyse all kinds of data. WordPerfect was one of the first word processors, and dBase IV was a system that let the user store, organize, and retrieve data.

Fifth Generation (1990–Present)

The fifth generation is notable for three major events: the rise of Microsoft® as a dominant player in computer software, object-oriented design and programming, and the World Wide Web.

Microsoft's Windows operating system emerged as a major force in the PC market during this period. Although WordPerfect continued to improve, Microsoft Word became the most used word processing program. In the mid1990s, word processors, spreadsheet programs, database programs, and other application programs were bundled together into super packages called office suites.

Object-oriented design became the design of choice for large programming projects. Whereas structured design is based on a hierarchy of tasks, object-oriented design is based on a hierarchy of data objects. Java™, a language designed by Sun Microsystems for object-oriented programming, began to rival C++.

In 1990, Tim Berners-Lee, a British researcher at the CERN physics lab in Geneva, Switzerland, created a set of technical rules for what he hoped would be a universal internet document centre called the World Wide Web. Along with these rules, he created HTML, a language for formatting documents, and a rudimentary, text-only browser, a program that allows a user to access information from websites worldwide. In 1993, Marc Andreesen and Eric Bina released Mosaic, the first graphics-capable browser. To quote Newsweek: “Mosaic just may have been the most important computer application ever.”

There were now two giants in the browser market: Netscape Navigator (derived from Mosaic) and Microsoft’s Internet Explorer (IE). Microsoft bundled IE with its Windows operating system, which made IE the winner in the browser wars. This bundling led to a monopoly lawsuit filed by the U.S. government, the 2001 settlement of which required Microsoft to be more open with its competitors. Netscape’s future became uncertain after America Online purchased it in 1998. AOL stopped supporting Netscape products ten years later. Mozilla Firefox, a web browser that retained some of the flavour of Mosaic, was released in November 2004. As of 2014, Firefox had captured 25% of the browser market.

Although the internet had been around for decades, the World Wide Web made it easy to use the internet to share information around the world (see figure 1.11). Around 2002, the Web began changing. Social networking sites such as Facebook and Twitter have become wildly popular. Online blogging has turned anyone and everyone into an author or social critic. User generated and edited content characterizes these new websites. For example, Wikipedia is an online encyclopaedia for which anyone can enter or edit content. The term Web 2.0 has been used by some to describe these emerging sites and uses.

The fifth generation must be characterized most of all by the changing profile of the user. The first user was the programmer who wrote programs to solve specific problems – his or her own or someone else’s. Then the systems programmer emerged, who wrote more and more complex tools for other programmers. By the early 1970s, applications programmers were using these complex tools to write applications programs for non-programmers to use. With the advent of the personal computer, computer games, educational programs, and user-friendly software packages, many people became computer users. With the birth and expansion of the World Wide Web, web surfing has become the recreation of choice, so even more people have become computer users. The user is a first-grade child learning to read, a teenager downloading music, a college student writing a paper, a homemaker planning a budget, and a banker looking up a customer’s loan record. The user is all of us.

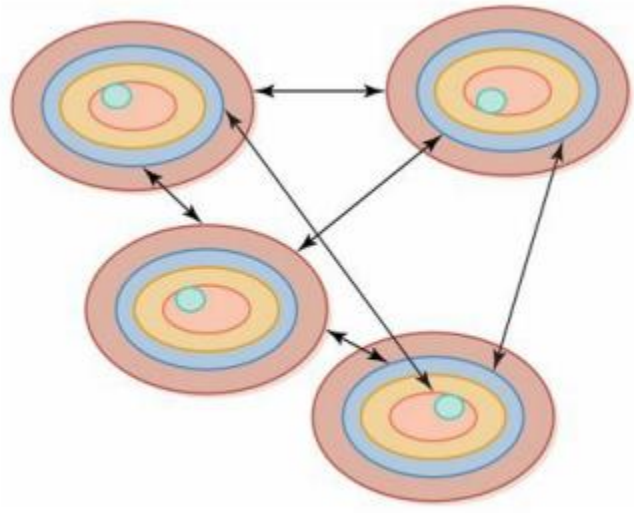


FIGURE 1.11 Sharing information on the World Wide Web

In our brief history of hardware and software, we have focused our attention on traditional computers and computing systems. Paralleling this history is the growing use of integrated circuits, or chips, to run or regulate everything from toasters to cars to intensive care monitors to satellites. Such computing technology is called an embedded system. Although these chips are not actually computers in the sense that we are going to study in this book, they are certainly a product of the technology revolution of the last 55 years.

Predictions

We end this brief history of computing with a few predictions about computers that didn't come true:

"I think there is a world market for maybe five computers." Thomas Watson, chair of IBM, 1943.

"Where … the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and weigh only 1.5 tons" Popular Mechanics, 1949

"I have travelled the length and breadth of this country and talked with the best people, and I can assure you that data processing is a fad that won't last out the year." The editor in charge of business books for Prentice Hall, 1957

"But what … is it good for?" Engineer at the Advanced Computing Systems division of IBM, commenting on the microchip, 1968

"There is no reason anyone would want a computer in their home." Ken Olsen, president, chairman, and founder of Digital Equipment Corporation, 1977

"$100 million is way too much to pay for Microsoft." IBM, 1982

"I predict the internet … will go spectacularly supernova and in 1996 catastrophically collapse." Bob Metcalfe, 3Com founder and inventor, 1995

"Folks, the Mac platform is through—totally." John C. Dvorak, PC Magazine, 1998

Computing as a Tool and a Discipline

In the previous section on the history of computer software, we pointed out the ever-changing role of the user. At the end of the first generation, users were split into two groups: systems programmers, who developed tools to make programming easier, and applications programmers, who used those tools. Later, applications programmers built large domain-specific programs such as statistical packages, word processors, spreadsheets, intelligent browsers, virtual environments, and medical diagnosis applications on top of the traditional language tools. These application programs were, in turn, used by practitioners with no computer background.

So who is using the computer as a tool? Everyone, except for those people who are creating the tools for others. For these toolmakers, either computing is a discipline (low-level tools) or the discipline of computing has made their tools possible (applications built upon applications).

A discipline is defined as a field of study. Peter Denning defines the discipline of computer science as "the body of knowledge and practices used by computing professionals in their work…. This discipline is also called computer science and engineering, computing, and informatics."20 He continues, "The body of knowledge of computing is frequently described as the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, What can be (efficiently) automated?"

Denning states that each practitioner must be skilled in four areas:

- algorithmic thinking, in which one is able to express problems in terms of step-by-step procedures to solve them
- representation, in which one is able to store data in a way that it can be processed efficiently
- programming, in which one is able to combine algorithmic thinking and representation into computer software
- design, in which the software serves a useful purpose

A debate has long raged about whether computing is a mathematical discipline, a scientific discipline, or an engineering discipline. Computing certainly has strong roots in mathematical logic. The theorems of Turing tell us that certain problems cannot be solved, Boolean algebra describes computer circuits, and numerical analysis plays an important role in scientific computing. Scientific disciplines attempt to understand how their systems work. The natural sciences exist to "fill in the instruction book that God forgot to leave us."21 Thus computing is a scientific discipline, as we use them to build and test models of natural phenomena. As we design and build larger and larger computing systems, we are using techniques from engineering.

In 1989, a task force of computer science educators proposed a curriculum model that covered the subareas of computing from the three perspectives represented in our history: theory (mathematics); experimentation, called abstraction by computer scientists (sciences); and design (engineering).22 Theory refers to the building of conceptual frameworks and notations for understanding relationships among objects in a domain. Experimentation (abstraction) refers to exploring models of systems and architectures within different application domains and determining whether the models predict new behaviours. Design refers to constructing computer systems that support work in different application domains.

Table 1.1 shows the topic areas outlined by the task force. Of the nine subject topic areas, six relate to understanding and building computing tools in general: Algorithms and Data Structures, Programming Languages, (Computer) Architecture, Operating Systems, Software Methodology and Engineering, and Human–Computer Communication. Not surprisingly, these are called systems areas. Three of the subareas relate to the computer's use as a tool: Database and Information Retrieval, Artificial Intelligence and Robotics, and Graphics. These areas are called applications areas.

Table 1.1 Topic Areas of the Computing Discipline, 1989

Algorithms and Data Structures
Programming Languages
Architecture
Operating Systems
Software Methodology and Engineering
Database and Information Retrieval
Artificial Intelligence and Robotics
Human-Computer Communication
Graphics

Revised curriculum documents, published in 2001, reorganized and expanded the topic areas to a total of 14. Algorithms and Data Structures has been expanded and put under the title "Programming Fundamentals." With the rise of the Web, networks get their own category: Net-Centric Computing. Artificial Intelligence and Robotics has been expanded to include all Intelligent Systems. Databases and Information Retrieval are now called Information Management.

The new topics include Discrete Structures, an area of mathematics that is important to computing, and Algorithms and Complexity, the formal study of algorithms rather than the study of how to write them. These would be systems areas. Computational Science includes the application of numerical techniques and simulation to fields such as molecular dynamics, celestial mechanics, economic forecasting, and bioinformatics. The last new topic is Social and Professional Issues, which relates to professionals in both systems and applications areas. This area is now called the Tenth Strand. We have more to say about this in the Ethical Issues section. Table 1.2 shows a listing of the topic areas as of 2001. The report "Computer Science Curriculum 2008: An Interim Revision of CS 2001," published in December 2008, leaves these 14 topic areas unchanged.

Table 1.2 Topic Areas of the Computing Discipline, 2001

Discrete Structures
Programming Fundamentals
Algorithms and Complexity
Architecture and Organization
Operating Systems
Net-Centric Computing
Programming Languages
Human-Computer Interaction
Graphics and Visual Computing
Intelligent Systems
Information Management
Social and Professional Issues
Software Engineering
Computational Science

Research is ongoing in both systems and applications. Systems research produces better general tools; applications research produces better tools for the domain-specific applications. There is no doubt that the relationships between the people who investigate computing topics as a discipline directly affect those who use computers as a tool. Computing research fuels the applications people use daily, and the turnaround for the technology is amazingly fast. This symbiotic relationship is more dynamic in computing than in any other discipline.

Questions (Section 2)

For exercises 24 – 38, match the software or software concepts listed to the appropriate generation:

A. First
B. Second
C. Third
D. Fourth
E. Fifth

24. Assemblers
25. FORTRAN
26. Operating systems
27. Structured programming
28. Time sharing
29. HTML (for the Web)
30. Loaders
31. Spreadsheets
32. Word processors
33. Lisp
34. PC-DOS
35. Loaders/linkers bundled into an operating system
36. Java
37. SPSS
38. C++

Exercises 39—55 are short-answer questions

Write your answers in full sentences. These will serve as notes that we will sometimes refer back to throughout the course.

39. What do we mean by the statement that "the 1980s and 1990s must be characterized by the changing profile of the user”?
40. Why was Mosaic important?
41. Discuss the browser wars
42. Describe how the web changed after 2002
43. Of the predictions listed in this chapter, which do you consider the biggest error in judgment? Explain.
44. Name the four areas in which the practitioner must be skilled
45. Distinguish between computing as a tool and computing as a discipline
46. Is computing a mathematical, scientific or engineering discipline? Explain.
47. Distinguish between systems areas and applications areas in computing as a discipline
48. Distinguish between machine language and assembly language.
49. Distinguish between assembly language and high-level languages.
50. FORTRAN and COBOL were two high-level languages defined during the second generation of computer software. Compare and contrast these languages in terms of their history and their purpose.
51. Distinguish between an assembler and a compiler

52. Distinguish between a systems programmer and an applications programmer
53. What was the rationale behind the development of operating systems?
54. What constitutes systems software?
55. What do the following pieces of software do?
    a. loader
    b. linker
    c. editor
56. How was the program SPSS different from the programs that came before it?