Computer Science: The Computer Introduction Computers are machines that process information. Most modern computers are digital, meaning that they manipulate symbols according to logical rules. The basic symbols used in most digital computers are 0 and 1, which are grouped to designate numbers, words, colors, and the like. Early computer designs, dating from the nineteenth century, were purely mechanical devices. In the 1950s electronic computers came into wide use, although only by large organizations. Computers gradually became a consumer commodity, however, and by the early 2000s hundreds of millions were in use around the world; many millions more were embedded in products such as automobiles, ovens, telephones, personal digital assistants, music players, and toys. Computers, now essential to science and engineering, have effected deep transformations in almost every aspect of society. topian predictions of a world freed from drudgery, poverty, political oppression, and other ills by the computer, however, have not been fulfilled.

Historical Background and Scientific Foundations For most of human history, computation was performed with some kind of aid: fingers, piles of pebbles, marks on clay or paper, abaci, and the like. The physical parts of these aids always symbolize or stand for quantities: a finger on the left hand, for instance, can stand for a 1, a finger on the right hand for a 5; beads on one wire of an abacus can stand for 1s, the beads on the next wire for 10s. Mechanical aids eased the burden on human memory and freed people to transform the information they stored with addition, multiplication, and other numerical systems.

A more complex class of devices began to be built in ancient times—mechanical analog computers. Around the end of the second century BC for example, the Greeks produced a sophisticated geared device known as the Antikythera mechanism, which was used to calculate lunar and solar eclipses as well as other astronomical movements. In the seventh or eighth century AD, Islamic scientists produced the first astrolabes, circular metal calculators used to predict the positions of the sun, moon, and stars that were used for navigation and for astrology.

Another early calculator was invented by Scottish scientist John Napier (1550–1617) in the late sixteenth century. "Napier's bones" were a set of marked ivory bars that sped up calculations by substituting addition and subtraction for multiplication and addition. After Napier invented logarithms in the early 1600s, the first slide rules were developed, using marked sliding sticks to record logarithmic relationships between numbers. The modern slide rule, invented in the 1850s, was standard in science and engineering work until the invention of the handheld electronic calculator in the 1960s.

Gottfried Wilhelm von Leibniz (1646–1716) might be called the world's first computer scientist. He invented binary (base 2) arithmetic, which uses only the numerals 0 and 1, unlike decimal or base 10 arithmetic, which uses the numerals 0 through 9. In 1673 he built a new type of calculator that could do multiplication; mechanical calculators that evolved from this design were used until the 1970s.

Another important step was taken by British mathematician George Boole (1815–1864) in 1854, when he described a system of logical rules, today called

Boolean algebra, by which true-or-false statements could be handled with mathematical rigor. The 1s and 0s of Leibniz's binary arithmetic can be identified with the true and false statements of Boolean algebra. Since any two-state device—an on-off switch, for instance—can store the value of a Boolean variable, Boole's new algebra paved the way for mechanized general logic, not just arithmetic.

In the mid-1800s English mathematician Charles Babbage (1791–1871) conceived a revolutionary design that he called the Analytical Engine. If completed, it would have possessed all the basic features of a modern computer: Babbage had designed a memory, or "store," of a thousand 50-digit numbers, the equivalent of about 20 kilobytes—more memory than some electronic digital computers had in the 1970s. These numbers were to be processed in a central processing unit called the "mill." Finally, the machine would have been programmed with instructions on punch cards, using technology borrowed from the Jacquard loom.

Due a complex series of misfortunes, this brilliant invention was never built, and Babbage was largely forgotten after his death. It was not until the 1930s that the

concept of the programmable computer was approached again. By this time, mechanical on-off switches (relays) driven by electricity had been perfected for use in the telephone system; vacuum tubes, which can be also used as on-off switches, had been developed for radio. Both devices were adapted for use in digital computers.

In 1930 American engineer Vannevar Bush (1890–1974) built an analog computer called the Differential Analyzer. Although it could not perform logical operations on symbols as digital computers do, it allowed mechanical or electrical quantities (cam motions, voltages, etc.) to behave like particular mathematical functions. By measuring how these quantities behave, it could solve various mathematical problems, including differential equations. One advanced differential analyzer built by Bush in the late 1930s weighed 100 tons and contained some 2,000 vacuum tubes and 200 miles (322 km) of wiring. These analyzers were used to compute firing tables for artillery gunners during World War II (1939–1945).

But the future of computing did not lie with analog computers. In the late 1930s American engineer George R. Stibitz (1904–1995) and colleagues built a small digital computer based on telephone relays and Boolean algebra. Harvard mathematician Howard Aiken (1900–1973), collaborating with International Business Machines Corp. (IBM), began building the IBM Automatic Sequence Controlled Calculator in 1939, a relay-based computer that could be programmed with instructions coded on a reel of paper tape with small holes punched in it. In the same year mathematician John Atanasoff (1903–1995) and Clifford Berry (1918–1963) designed (but did not finish) a fully electronic vacuum tube computer that stored information on a rotating drum covered with small charge-storage devices called capacitors, a forerunner of the modern hard drive.

Increasingly sophisticated "second-generation" electronic computers built during the 1940s and early 1950s gradually eliminated all mechanical parts. Many of these advances were driven by military needs; England's Colossus, for example, was devoted to cracking the secret German military code Enigma. Individual computers were bult for specific projects, each with a unique name: Colossus,

ENIAC, EDSAC.

The shift to standardized commercial units began with the UNIVAC I (Universal Automatic Computer I). The first one, bought by the U.S. Census Bureau in 1951, filled a large room. Nineteen were sold to government and industry from 1951 to 1954. Other large standardized computers were soon on the market, dominated for many years by IBM.

Until they were replaced by transistors, computers ran on vacuum tubes—sealed glass cylinders containing

electrical components but almost no air. Each tube acted as an amplifier (in analog devices) or as an on-off switch (in digital devices). Vacuum tubes use a lot of energy and break down frequently, factors that limit computer size. Transistors, invented in the late 1940s, are small power-sipping devices made of solid crystal. The first all-transistor computer was the TRADIC (transistor digital computer), a special-purpose computer built in 1955 and installed in B-52 Stratofortress bombers. In 1957 IBM announced that it would replace vacuum tubes with transistors in all its computers.

In these "third-generation" computers, the transistors were small wired cylinders. These were made largely obsolete by the integrated circuit, a small solid tile or chip of semiconducting material (e.g., silicon) in which multiple transistors, resistors, capacitors, and wires have been created on layers and regions of the chip by depositing elements called dopants. The first commercial chip was produced in 1961; it contained four transistors and stored one bit ("binary digit," a 0 or 1). By 2007 some chips contained over a billion transistors. For example, the Dual-Core Intel Itanium 2 processor released in 2006 contained 1.72 billion transistors.

In 1970 an entire computer was put onto a single chip, and the microprocessor was born. This led to the development of handheld calculators the following year and the debut of personal computers in 1974. Since then, the history of computing has largely been the evolution of increasingly powerful microprocessor chips at lower prices.

Starting in the 1980s, networked computers—arranged to exchange data over phone lines or other electronic channels—led to the development of the Internet and World Wide Web. In the 2000s, computing power, storage capacity, and connectivity continued to skyrocket, expanding the ways in which computers could be used, from scientific calculation to personal entertainment.

The Modern Digital Computer

Modern digital computers are based on binary arithmetic, which symbolizes all numbers as combinations of 0 and 1, such as 0110 for the number 8. Computers represent each 1 or 0 with an on-or-off electrical signal. Feeding these signals into devices produces other electrical signals according to the rules of Boolean algebra. All the millions of tasks carried out by modern computers, including graphics, sound, game playing, and the like, are built on Boolean bit-level operations.

The bit is the logical basis of computing; the transistor is its on-off switch. A small group of transistors can be linked together into a device called a flip-flop, which stores the value of a single bit. Other combinations of transistors take

bits and perform Boolean operations on them. Millions of transistors are hidden inside most modern computers, many of them changing state (flipping from 0 to 1 or back again) billions of times per second. Market pressures have forced engineers to figure out how to make transistors smaller, faster, and more energy efficient. As of 2007 prototype transistors could change state almost a trillion times a second; others were only 3 nanometers wide.

Computers interface with humans through screens, printers, speakers, pads, keyboards, and mice. They communicate with other computers on high-speed communications channels, receive data from special sensors, and operate machinery using a wide array of actuators. As with computing hardware, there is intense market pressure to produce cheaper, lighter, more flexible, more impressive interface devices.

Computer Science

Computer science is the field of knowledge devoted to designing more efficient computer architectures (arrangements of hardware and software) and more useful methods, called algorithms, for tasks such as searching memory, identifying patterns, encrypting or decrypting data, and solving mathematical problems. Programs that perform specific tasks when executed by a computer are as essential to computer function as the hardware itself, making software production a major industry.

One rule of computer science is described by Moore's Law, which was first posited by Intel Corporation cofounder Gordon Moore (1929–) in 1965. In its modern form, the law says that the number of transistors on a single chip doubles every two years. Graphs of cost versus memory, processing power, or pixels (for digital cameras) show that the law has held approximately true for over 40 years. Driven by human ingenuity, motivated by the desire for profit, and working with the limits imposed by the laws of physics, engineers have found clever ways to make transistors smaller and stuff more of them into every square millimeter of chip surface.

Obviously the trend described by Moore's Law cannot continue forever. Some experts believe that device miniaturization will be limited by the size of the atom in as little as 10 or 20 years After that, the methods used to make computers will have to change fundamentally if more computing power is to be packed into the same tiny space.

Several options beckon, all in the research stage today. One of the likeliest candidates is quantum computing, which exploits the properties of matter and energy at the atomic scale. Quantum computing enthusiasts hope to store multiple bits in single atoms and to teach bits stored in groups of atoms to perform simultaneous calculations as interlinked wholes, breaking out of the one-step-at-a-time limitations of Boolean logic. The underlying physics is real, but whether computers based on quantum bits or "cubits" can be produced remains to be seen.

IN CONTEXT: THE UNIVAC MYTH In 1952, the presidential candidates were Republican Dwight D. Eisenhower (1890–1969) and Democrat Adlai Stevenson (1900–1965). To spice up its election-night TV coverage of the close race, the CBS network decided to have the world's most powerful computer, the UNIVAC

I, predict the election outcome on the air.

That night, tense engineers teletyped messages between CBS studios in New York and the UNIVAC in Philadelphia. The UNIVAC's programmers fed it information about early returns, voting trends dating back to 1928, and other factors.

By 9:00 p.m., UNIVAC was predicting a huge victory for Eisenhower. Its program was altered to produce more reasonable results, but as more returns were fed into the computer, it still insisted on a huge margin of victory for Eisenhower. A UNIVAC team spokesman apologized on TV, attributing the freak figures to human error, but the computer was right: Eisenhower won by a landslide. In fact, its prediction of the electoral vote was off by only 1

The event entered computer mythology as proof that a "thinking machine" could outwit human experts. But in U.S. congressional elections two years later UNIVAC predicted large Democrat majorities in both the House and Senate that did not materialize. "Probably the outstanding TV casualty of the night was Univac," Time magazine commented wryly the following week.

In reality, computers have limited ability to model complex human activities such as politics. Even with today's extremely powerful computers and algorithms based on a further half-century of research, reliable computer political forecasting is not practical. UNIVAC's stunning success of 1952 was beginner's luck, a fluke.

Modern Cultural Connections Scientists and engineers were among the first to use computers and they rely on them more than ever today. Many scientific tasks can only be accomplished by solving equations, often by numerical methods—that is, the painstaking manipulation of many numbers. Given the extreme tedium, time, and expense of carrying out such calculations by hand, mechanized computing is an obvious need for science. Modern computers not only store data and communicate scientific results, but make many scientific projects possible that would not exist otherwise. Space probes, for example, could not return data from distant moons and planets without onboard computers.

IN CONTEXT: ASSEMBLING THE TREE OF LIFE In 1859 English biologist Charles Darwin (1809–1882) proposed a theory that all living things are related through an unbroken web of descent. He identified a natural process to explain how living things adapt to their environments—natural selection. Scientists have now turned to computers to better understand the patterns of evolution.

By the early 2000s large amounts of genetic code from thousands of different organisms had been deciphered and made available in computer databases such as GenBank, which is maintained by the U.S. National Center for Biotechnology Information. By 2005 GenBank contained at least partial DNA information for about 6

In 2002 the National Science Foundation launched a project called Assembling the Tree of Life to fund computer research to trace life's family tree. Three years later, scientists proved that by knowing only fragments of genomes (a species' complete DNA complement), it is possible to reconstruct subtrees—clusters of twigs, as it were—if not the whole tree, from the bottom up. This is good news because most genomes are not yet known in full.

Genomic evolutionary analysis is growing rapidly. "The algorithms are just

not keeping pace with the data that are available," biologist Keith Crandall said in 2005, "so anyone with a better mousetrap is going to have a huge impact."

Impact on Society

The earliest practical computers were gigantic and extremely expensive, affordable only to government agencies and a few large private corporations. Starting in 1971, however, handheld microprocessor-based electronic calculators made the slide rule obsolete. But this still did not put general-purpose computing power on the desks of ordinary citizens. That only began to happen in the mid-1970s, when personal computers—computers intended for home use by individuals—first appeared. The most famous was the Altair 8800, a build-it-yourself computer originally shipped with no screen or keyboard. Although it sold only a few thousand units, it is now credited with jump-starting the market. In 1977 the Apple II became the first mass-marketed personal computer, selling millions of units. After 1981 Apple computers were eclipsed by the IBM and compatible machines made by other manufacturers. In 1983 Time magazine declared the personal computer its Person of the Year for 1982.

In the 1980s personal computers became commonplace in American and European homes. They also appeared in schools, but early claims that computers would replace human teachers turned out to be false. In the early 1990s a network of university and government computers called ARPANET (after its sponsoring organization, the Advanced Research Projects Agency of the U.S. Department of Defense) evolved rapidly into the Internet, which now connects virtually all computer users to each other in a global web offering literally millions of cultural and scientific connections.

Computers now permeate all aspects of science, engineering, and manufacturing. They analyze genetic data, communicate with and control probes in deep space, analyze data returned by those probes for clues to the fundamental structure of the universe, and much more. Almost all scientific and engineering work that requires numerical calculation now involves computers.

Computers have become pervasive in industrialized countries. In 2003, for example, about 63% of American adults could access the Internet either at home or work, and this was not even the highest access rate in the world. At that time, 148 million Americans used the Internet and 80% of American Internet users were connected via high-speed broadband connections. By 2005 the average U.S. Internet user spent 3 hours online daily, compared to 1.5 hours watching television. The social effects of all this Internet use have been hailed by enthusiasts, assailed by critics, and studied by sociologists with somewhat uncertain results.

Some experts argue that communicating through the Web encourages the formation of communities, but the analogy with face-to-face interaction is questionable, especially since some studies show that people often find time for the Internet by spending less time with family and friends.

However, scientific results as of 2007 were contradictory. Some studies showed that Internet usage to be socially isolating, others did not. Indeed, both positive and negative claims about the impact of the Internet on society may have been exaggerated. Many important aspects of life, including people's emotional lives,

close relationships, and choices about how to spend large blocks of time, turn out to be fairly stable and to resist sudden change even in the face of radical technological innovations like the Internet. For example, despite predictions by Microsoft CEO Bill Gates and others in the late 1990s that computerized books (e-books) were about to replace the printed variety, sales of e-books remain slight compared to paper ($20 million versus 25$ billion).

There is less controversy, however, about the economic benefits of the Internet, which has produced a new class of professionals able to work from home all or part of the time, exchanging documents with employers rather than taking themselves physically to workplaces. Internet access, including broadband, is associated with higher personal income; poorer people are less likely to have Internet access and so are less likely to experience its economic benefits, a phenomenon known as the "digital divide."

Usually, computational systems are seen as composed of two ontologically distinct entities: software and hardware. Algorithms, source codes, and programs fall in the first category of abstract entities; microprocessors, hard drives, and computing machines are concrete, physical entities.

Moore (1978) argues that such a duality is one of the three myths of computer science, in that the dichotomy software/hardware has a pragmatic, but not an ontological, significance. Computer programs, as the set of instructions a computer may execute, can be examined both at the symbolic level, as encoded instructions, and at the physical level, as the set of instructions stored in a physical medium. Moore stresses that no program exists as a pure abstract entity, that is, without a physical realization (a flash drive, a hard disk on a server, or even a piece of paper). Early programs were even hardwired directly and, at the beginning of the computer era, programs consisted only in patterns of physical levers. By the software/hardware opposition, one usually identifies software with the symbolic level of programs, and hardware with the corresponding physical level. The distinction, however, can be only pragmatically justified in that it delimits the different tasks of developers. For them, software may be given by algorithms and the source code implementing them, while hardware is given by machine code and the microprocessors able to execute it. By contrast, engineers realizing circuits implementing hardwired programs may be inclined to call software many physical parts of a computing machine. In other words, what counts as software for one professional may count as hardware for another one.

Suber (1988) goes even further, maintaining that hardware is a kind of software. Software is defined as any pattern that is amenable to being read and executed: once one realizes that all physical objects display patterns, one is forced to accept the conclusion that hardware, as a physical object, is also software. Suber defines a pattern as "any definite structure, not in the narrow sense that requires some recurrence, regularity, or symmetry" (1988, 90) and argues that any such structure can indeed be read and executed: for any definite pattern to which no meaning is associated, it is always possible to conceive a syntax and a semantics giving a meaning, thereby making the pattern an executable program.

Colburn (1999, 2000), while keeping software and hardware apart, stresses that the former has a dual nature, it is a "concrete abstraction" as being both abstract and concrete. To define software, one needs to make reference to both a "medium of description", i.e., the language used to express an algorithm, and a "medium of execution", namely the circuits composing the hardware. While software is always concrete in that there is no software without a concretization in some physical medium, it is nonetheless abstract, because programmers do not consider the implementing machines in their activities: they would rather develop a program executable by any machine. This aspect is called by Colburn (1999) "enlargement of content" and it defines abstraction in computer science as an "abstraction of content": content is enlarged rather than deleted, as it happens with mathematical abstraction.

Irmak (2012) criticizes the dual nature of software proposed by Colburn (1999, 2000). He understands an abstract entity as one lacking spatio-temporal properties, while being concrete means having those properties. Defining software as a concrete abstraction would therefore imply for software to have contradictory properties. Software does have temporal properties: as an object of human creation, it starts to exist at some time once conceived and implemented; and it can cease to exist at a certain subsequent time. Software ceases to exist when all copies are destroyed, their authors die and nobody else remembers the respective algorithms. As an object of human creation, software is an artifact. However, software lacks spatial properties in that it cannot be identified with any concrete realization of it. Destroying all the physical copies of a given software would not imply that a particular software ceases to exist, as stated above, nor, for the very same reason, would deleting all texts implementing the software algorithms in some high-level language. Software is thus an abstract entity endowed with temporal properties. For these reasons, Irmak (2010) definies software as an abstract artifact.

Duncan (2011) points out that distinguishing software from hardware requires a finer ontology than the one involving the simple abstract/concrete dichotomy. Duncan (2017) aims at providing such an ontology by focusing on Turner's (2011) notion of specification as an expression that gives correctness conditions for a program (see §2). Duncan (2017) stresses that a program acts also as a specification for the implementing machine, meaning that a program specifies all correct behaviors that the machine is required to perform. If the machine does not act consistently with the program, the machine is said to malfunction, in the same way a program which is not correct with respect to its specification is said to be flawed or containing a bug. Another ontological category necessary to define the distinction software/hardware is that of artifact, which Duncan (2017) defines as a physical, spatio-temporal entity, which has been constructed so as to fulfill some functions and such that there is a community recognizing the artifact as serving that purpose. That said, software is defined as a set of instructions encoded in some programming language which act as specifications for an artifact able to read those instructions; hardware is defined as an artifact whose function is to carry out the specified computation.

1.2 The Method of Levels of Abstractions As shown above, the distinction

between software and hardware is not a sharp one. A different ontological approach to computational systems relies on the role of abstraction. Abstraction is a crucial element in computer science, and it takes many different forms. Goguen Burstall (1985) describe some of this variety, of which the following examples are instances. Code can be repeated during programming, by naming text and a parameter, a practice known as procedural abstraction. This operation has its formal basis in the abstraction operation of the lambda calculus (see the entry on the lambda calculus) and it allows a formal mechanism known as polymorphism (Hankin 2004). Another example is typing, typical of functional programming, which provides an expressive system of representation for the syntactic constructors of the language. Or else, in object-oriented design, patterns (Gamma et al. 1994) are abstracted from the common structures that are found in software systems and used as interfaces between the implementation of an object and its specification.

All these examples share an underlying methodology in the Levels of Abstraction (henceforth LoA), used also in mathematics (Mitchelmore and White 2004) and philosophy (Floridi 2008). Abstractions in mathematics are piled upon each other in a never-ending search for more and more abstract concepts. On this account, abstraction is self-contained: an abstract mathematical object takes its meaning only from the system within which it is defined and the only constraint is that new objects be related to each other in a consistent system that can be operated on without reference to previous or external meanings. Some argue that, in this respect at least, abstraction in computer science is fundamentally different from abstraction in mathematics: computational abstraction must leave behind an implementation trace and this means that information is hidden but not destroyed (Colburn Shute 2007). Any details that are ignored at one LoA must not be ignored by one of the lower LoAs: for example, programmers need not worry about the precise location in memory associated with a particular variable, but the virtual machine is required to handle all memory allocations. This reliance of abstraction on different levels is reflected in the property of computational systems to depend upon the existence of an implementation: for example, even though classes hide details of their methods, they must have implementations. Hence, computational abstractions preserve both an abstract guise and an implementation.

A full formulation of LoAs for the ontology of digital computational systems has been devised in Primiero (2016), including:

Intention Specification Algorithm High-level programming language instructions Assembly/machine code operations Execution Intention is the cognitive act that defines a computational problem to be solved: it formulates the request to create a computational process to perform a certain task. Requests of this sort are usually provided by customers, users, and other stakeholders involved in a given software development project. Specification is the formulation of the set of requirements necessary for solving the computational problem at hand: it concerns the possibly formal determination of the operations the software must perform, through the process known as requirements elicitation. Algorithm expresses the procedure providing a solution to the proposed computational

problem, one which must meet the requirements of the specification. High-level programming language (such as C, Java, or Python) instructions constitute the linguistic implementation of the proposed algorithm, often called the source code, and they can be understood by trained programmers but cannot be directly executed by a machine. The instructions coded in high-level language are compiled, i.e., translated, by a compiler into assembly code and then assembled in machine code operations, executable by a processor. Finally, the execution LoA is the physical level of the running software, i.e., of the computer architecture executing the instructions.

According to this view, no LoA taken in isolation is able to define what a computational system is, nor to determine how to distinguish software from hardware. Computational systems are rather defined by the whole abstraction hierarchy; each LoA in itself expresses a semantic level associated with a realization, either linguistic or physical.