Contents

Computer science is one of the most important, yet one of the most misunderstood, fields of study. Unlike other sciences, until recently it was not taught at the K-12 level, and even now, the majority of such classes focus on computer applications or programming, with little attention to computer science itself. Engineering disciplines, while also not taught at the K-12 level, are for the most part understood better than computer science by the public, who know (correctly or not) that civil engineers build bridges, mechanical engineers design cars and other machines, and so forth.

So what do people think computer science is, and what do they think computer scientists do? Probably, if you ask your acquaintances, they will have a vague idea that we "do things with computers", or "design computers", or "program". And while all of these are true, they are not really what computer science is.

Computer science is first and foremost about problem solving, in particular, problem solving using computers and computation. We study computation itself, that is, the process of solving problems by algorithmic and other means. A good definition, from our professional society, the ACM (Association for Computing Machinery), is:

> Computer science (CS) spans the range from theory through programming to cutting-edge development of computing solutions. Computer science offers a foundation that permits graduates to adapt to new technologies and new ideas. The work of computer scientists falls into three categories: a) designing and building software; b) developing effective ways to solve computing problems, such as storing information in databases, sending data over networks or providing new approaches to security problems; and c) devising new and better ways of using computers and addressing particular challenges in areas such as robotics, computer vision, or digital forensics (although these specializations are not available in all computer science programs).
>
> – From Computing Degrees and Careers, ACM

As you can see, the focus is not so much on the computer as what one can *do* with computers and what one can make computing devices do. One of the pioneers of computer science, Edsger Dijkstra, is supposed to have commented, "Computer science is no more about computers than astronomy is about telescopes."

But this does not mean that computer scientists are just theoreticians, unconcerned with actually doing things with computers? Not at all, as the quote from the ACM makes clear. There is, to be sure, an area of computer science theory, and most of us have at least some theoretical aspects to our work. But the focus of computer science on problem solving and computation naturally leads to designing better computing architectures and devices, better ways to handle large amounts of data, better ways to tell a computer what to do—i.e., programming—and better ways to use computing technology to enrich people's lives, help society, support science, and foster communication and understanding between people.

Science or not? A question that comes up a lot is whether computer science is a science, or if it is an engineering field, or something else entirely. In the early days, computer science was often found in math or electrical engineering departments; this was due to its origins in those fields, prior to becoming a field in its own right. Today, in many universities, CS is in lumped in with electrical engineering, say in a Department of Electrical and Computer Engineering. In others, it is its own department, but still within an engineering college. In many more, it is in a liberal arts college (as it is in the authors' university). And in a small but increasing number of universities, it stands alone as a college-level unit, separate from either engineering, the sciences, or liberal arts.

The reason for range of where CS lives is part historical, part based on the outlook of the department and faculty, and part pragmatic. While CS is its own field, there are legitimate arguments for its affinity either with engineering (creating artifacts) or liberal arts or science (e.g., studying an existing thing, computation and problem solving, by means that happen to be a computer). There are also arguments to be made that it is like none of the above, and so should be separate. And, given the incredible and increasing importance of computing to all facets of society, science, and daily life, highlighting that by putting CS in its own college-level entity also makes sense.

Part of computer science. Computer science is a very broad field containing many subfields. Some of the fundamental subfields include:

Digital logic and computer architecture: The study of how a computer works, and how computers can be designed to increase their efficiency, power, and other desirable properties.

Programming languages: The study of the design and implementation of programming languages.

Operating systems: The study of the design and implementation of the software of the computer that is the interface between the user programs and the hardware, and that manages the computer's resources.

Networking: The study of the design and implementation of mechanisms and protocols for communication between computers.

Database systems: The study of how to store and retrieve large amounts of structured or unstructured data using the computer.

Software engineering: The study of how to design and implement large computer software systems.

Human-computer interaction (sometimes called "human factors"): The study of how to best design and structure the interface between the computer and humans, including both software and hardware considerations.

Computer graphics: The study of how to create realistic representations of scenes using computers, as well as the best way to visually present information to users (e.g., scientific visualization).

Computer modeling: The study of how to design, implement, and analyze computer-based models and computer simulation systems.

Cybersecurity: The study of risks to computer security and how to prevent or ameliorate those risks.

Artificial intelligence: The study of ways to (1) make computers able to handle things that may be easy for humans and animals, but difficult for computers (e.g., vision); (2) make computers able to be "smarter" than they currently are (e.g., medical diagnostic systems); and/or (3) design and implement truly intelligent artifacts.

Relation to other fields. Computer science overlaps with many other fields. Within the broader area of computing fields, it overlaps with information systems (including such things a geographic information systems, GIS), information technology, management information science, and so forth.

Outside computing per se, computer science has a rich history of interaction with other sciences and engineering fields and, increasingly, liberal arts areas. Often these areas have names of the form "computational X". For example, computational biology has to do with modeling and investigating biological systems using computer techniques. Sub-areas of this include

bioinformatics and computational ecology. Computational chemistry has to do with modeling and investigating chemical systems using the computer, and so forth for other sciences. And, of course, computers are heavily used in the sciences for data storage, data processing and reduction, solving mathematical equations, scientific visualization, and, increasingly, data mining. One science–psychology–stands out in that it uses computers not only in these ways, but also for a central metaphor (in cognitive psychology) for some of the ways a brain works.

Computer engineering overlaps with computer science, of course, but other engineering disciplines have come to make heavy use of computers and computer science, for example to model physical, chemical, or biological systems, to visualize data, and to solve mathematical problems and reduce data.

Computers are being increasingly used in the liberal arts, both to solve problems (e.g., to analyze texts) as well as to enable new ways of doing things (e.g., computer art, computer music, etc.).

Why study computer science? The ACM also offers 10 reasons to major in computing that we repeat here; we doubt we could come up with better:

1. Computing is part of everything we do
2. Expertise in computing enables you to solve complex, challenging problems
3. Computing enables you to make a positive difference in the world
4. Computing offers many types of lucrative careers
5. Computing jobs are here to stay, regardless of where you are located
6. Expertise in computing helps you even if your primary career choice is something else
7. Computing offers great opportunities for true creativity and innovativeness
8. Computing has space for both collaborative work and individual effort
9. Computing is an essential part of well-rounded academic preparation
10. Future opportunities in computing are without boundaries

A brief history

No introductory textbook would be complete without some history of the field it introduces. Here, we give a very brief and very informal history of the field. There are better sources for more complete histories; we will provide links to them in the Further Reading section of this chapter.

In one sense, since computing at its root is problem solving, the history of computing is very old—in some sense, as old the time when the first person told another the first informal algorithm for accomplishing some task, perhaps tanning hide or lighting a fire.

Aside from that, however, we can trace the true beginnings of the field to mid-1800s, when Charles Babbage and Ada Lovelace were working on the difference and analytical engines. In a very real sense, Ada Lovelace was the first computer programmer; she was interested in how one would specify algorithms for the analytical engine to solve problems. In fact, she was even interested in how chess might be played. She was so influential that a programming language, Ada, is named after her.

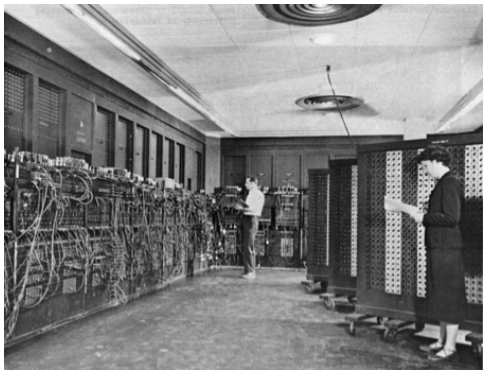

Figure 1: Programming ENIAC [US Army photograph]

The field's formal and practical foundations began to be laid in earnest in the 1940s with seminal work by Claude Shannon, Alan Turing, John von Neumann, and others. The basics of the kinds of computers that have reigned since (digital stored program computers, sometimes called von Neumann machines) were developed during this time, and fundamental properties of information (Shannon) and computing (Turing) were explored. Alan Turing proposed a formal, abstract machine, now referred to as a Turing machine, that is equivalent to a computer and that can, it is thought, compute anything that is computable. (Not everything is, it turns out.)

Beginning in the 1940s, but especially in the 1950s and 1960s, practical computing machines were developed; the term "computer" came to mean these machines, whereas in the past it referred to a person who did computations. In 1946, the first general purpose computer was built, called the Electronic Numerical Integrator and Computer (ENIAC), shown in Figure 1. It was designed by John Mauchly and J. Presper Eckert. Unlike the com-

puters that most of us have in our pockets—smartphones—ENIAC was 8 by 3 by 100 feet in size and weighed about 27 tons.

As can be seen in the photograph, programming ENIAC was not like programming computers today. It was not a stored program computer. Programs would be entered via a combination of cables to hook functional units of the computer together and "portable function tables" into which tables of numbers could be entered. Most of the programming was done by six women, who were later inducted into the Women in Technology International Hall of Fame.

Like some of its successors, ENIAC was one of a kind. Even though it was general-purpose, it was still built for a particular application (military calculations), and it was very expensive. It was followed shortly by true stored-program computers, but it wasn't until the 1950s that computers became more like products for sale, albeit still at a steep price.

One of the most successful computer companies was IBM (International Business Machines). Their early computer, the IBM 701, was for scientific use in 1952, and their (slightly) later computer, the IBM 702, was for business purposes. There were two, since it was thought that the needs of science and business were sufficiently different to warrant it. Scientific computing needed numerical operations, typically floating point (approximations to real numbers), while business needed to process text and to deal with decimal numbers (dollars and cents, e.g.; and decimal, since roundoff and other errors from floating point calculations could not be tolerated).

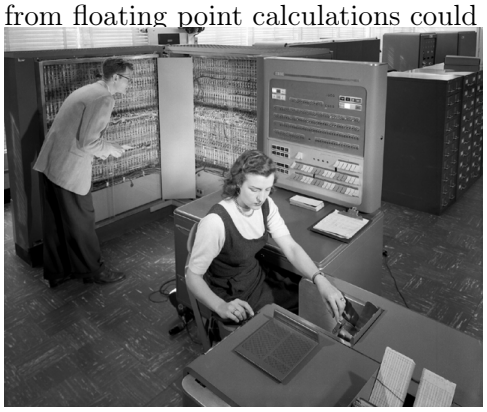

Figure 2: IBM 704 electronic data processing machine [NASA photograph].

Up until this time, computers had been programmed mostly either by rewiring them (e.g., ENIAC) or, more commonly by this time, by entering binary machine language programs into the computer, either directly, with switches, or in some other manner. A more human-friendly way was also common, called assembly language. This encoded the binary machine language as mnemonics that were easier for human programmers to deal with, for example, "ADD" rather than "1010010".

In the late 1950s, however, researchers began to develop high-level languages—languages much closer to humans', in which each line might

represent tens of assembly language lines. The first of these that saw any sort of widespread use was a scientific language, FORTRAN—which stood for FORmula TRANslation. As the name indicates, the focus was on quickly performing calculations. This language—which is still in use, by the way!—revolutionized the way people interacted with computers.

A bit later, a high-level language was invented by Grace Hopper and others for the business-oriented computers. This was COBOL, the COmmon Business-Oriented Language. COBOL was more concerned with moving and manipulating data than in numeric calculations, although it could do that too, of course, just as FORTRAN could manipulate text. It was an extremely verbose and English-like language, meant, in part, for non-specialists to be able to use. COBOL, too, is still around and in used; by some estimates, it is the language with the most lines of code written in it of all the programming languages in the world.

In between these two was a third language, developed not for numerical or text calculations, but for artificial intelligence: LISP (LISt Processing language). LISP (or Lisp, as we will write it in this book) was radically different from these other two "imperative" languages; instead, it was based on applying functions to data and other functions. Its data, too, reflected its original purpose: symbols and lists, as well as other types. Lisp, too, is still in widespread use.

A very influential language, ALGOL, was also being worked on about this time. This language, which influenced virtually all imperative and a good many object-oriented languages following it, was itself not used much. However, from it came many innovations.

About the start of the 1960s, IBM created a family of computers with the aim of unifying the two types that had come before, scientific and business-oriented. This was the IBM 360 family of machines, likely the most successful series of general-purpose computers of that era. The instruction set—the set of operations the computer can carry out—for the 360 family included both scientific and business-oriented operations. In addition, an innovation was that a customer could buy a computer that fit his or her needs, since there was a family of machines of varying abilities. A program developed for one computer in the family would run on any of the others, and so a customer could upgrade his or her computer without having to rewrite code.

A wealth of programming languages were available by this point. ALGOL, FORTRAN, COBOL, PL/1 (Programming Language/1), and others were available for use; a programmer could choose a language that best suited his or her needs.