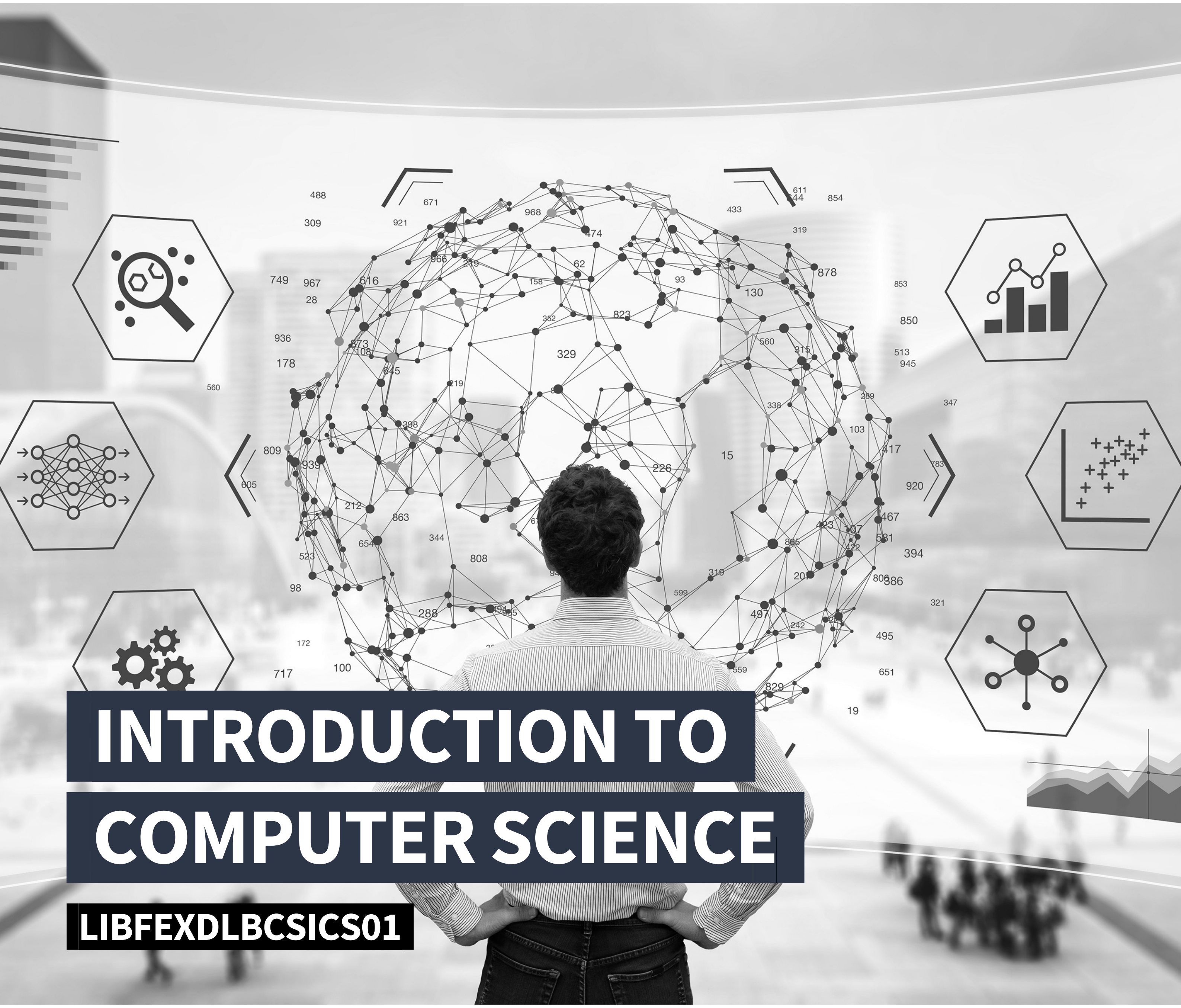


INTRODUCTION TO COMPUTER SCIENCE

LIBFEXDLBCSICS01

The London Institute of Banking & Finance

LEARNING OBJECTIVES

Computers were initially created to assist humanity in the calculation of data. The idea was simple: computers will perform these tasks with greater certainty and speed than a human being. A computer does not have a human being's propensity to become tired or bored. It does not make simple mathematical errors, no matter how many times it executes the same equation—that is, as long as the computer is programmed and designed correctly.

You will begin your **Introduction to Computer Science** by looking at the most basic concept of computing: data. Interpreting numerical data as an abstract human concept is the goal of man-to-machine cooperation. Data are processed by hardware, firmware, and software working together.

To create a computing machine, it is not only essential to assemble the parts, but also to program them. This programming involves a complex language that was created to be understood by humans and machines alike—you will be introduced to the concepts of programming languages, data types, syntax, and semantics.

From hardware to advanced software, computer logic and structure permeates the essence of computer science. You will encounter algorithms, data structure, circuit design, propositional logic, and basic computer architecture. The invention of networks revolutionized computing, and we will be examining these concepts along with TCP/IP, fault tolerance, and network topologies.

This overview of data, hardware, software, programming, logic, and networking will equip you, the student, with a foundational knowledge of the vocabulary and concepts necessary for studying the discipline of computer science.

UNIT 1

BASIC CONCEPTS OF DATA PROCESSING

STUDY GOALS

On completion of this unit, you will have learned …

- about the concepts of data, information, and computer messaging.
- the difference between hardware, firmware, and software.
- the basics of binary and data interpretation.
- what syntax and semantics refer to in programming languages.
- the history of computers and data processing.

1. BASIC CONCEPTS OF DATA PROCESSING

Introduction

In 1977, a high school student named Richard Garriott took a summer course in computer programming at the University of Oklahoma. An avowed fan of Tolkien's *Lord of the Rings,* he became enamored of a pencil and paper game called *Dungeons & Dragons* (D&D), which is set in a similar fantasy world. Not long after, Garriott started taking the rules of the game and programming them into a computer.

The game of D&D is a nearly perfect match for the computer. It involves ability scores, numbers that are assigned to different human attributes such as strength, intelligence, and dexterity (data). These numbers are then used to calculate modifiers to determine chances of success (probability) to perform imaginary actions in the game, such as hitting an orc with a sword, deciphering ancient mystical ruins, or climbing up the side of a steep tower. As a player attempts to take certain actions, dice are rolled, with these rolls—modified by said attributes—determining success or failure. Simple computer equations and algorithms could easily simulate dice rolling (randomization) and data calculation.

Garriott created his own fantasy world based on the concepts of *D&D* and completed the game *Alakebeth: World of Doom* while still in high school. He then got a job working at a software store where the owner proposed to sell Richard's game, saying that it was actually better than the games in the store. Eventually, the game sold 30,000 copies. With Garriott's $5 per game share, he earned $150,000, which was more than his father, a NASA astronaut, earned in a year at that time (Bebergal, 2020).

Any type of programming involves taking human concepts and ideas and boiling them down to numbers, logic, rules, and, finally, output that can be interpreted by a human user. To begin our study of Computer Science, we are going to take a look at the core concept of data.

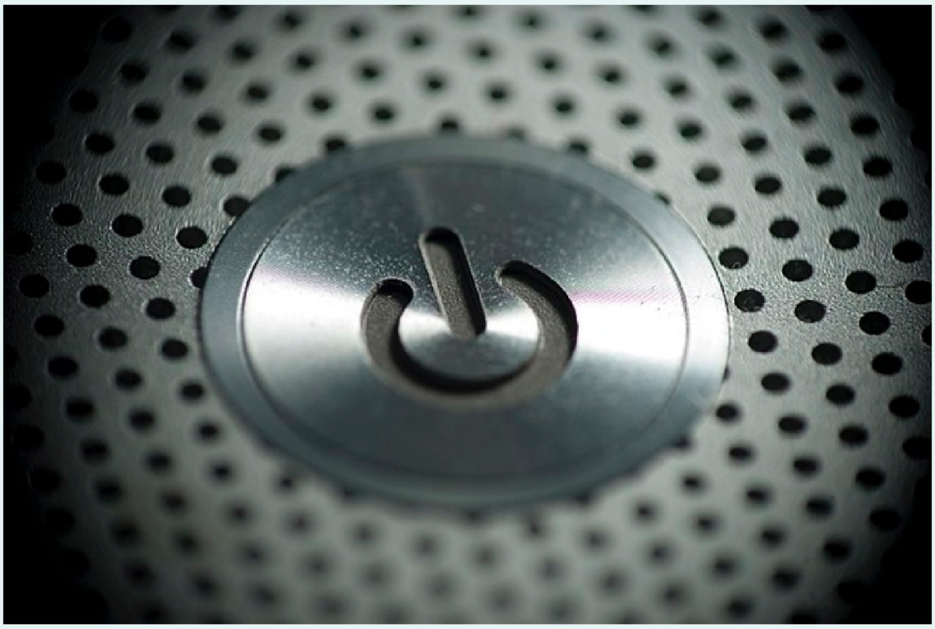
1.1 Data, Information, and Messages

Data consists of raw numbers and symbols that can be arranged into meaningful information, which is then used for specific research or analysis purposes. To a computer, data are numerical and the basic data storage system is the **binary** numbering system. While human beings are used to using many symbols to represent information, including letters of the alphabet and decimal numbers, computers are restricted to representing every single kind of information imaginable as ones and zeroes. If you asked a computer to remember how you liked your coffee, it would store this information as something like this: 01011100. To you, this would be meaningless, but this combination of zeroes and ones could easily represent sugar, cream, size, and temperature of the first drink you have in the morning. Your favorite song, to a computer, would be a longer string of ones and zeroes, as would a picture of your grandma, your résumé, and that cat video you love so much.

Binary
This is a base 2 numbering system consisting of only ones and zeroes.

The reason for this is simple: when computers were first designed nearly a century ago, the only way to store different states that represent information was to turn switches ON or OFF, which corresponded to the numerical values 1 and 0. In fact, this is still represented today in most digital devices: the power button is a zero intersected by a one.

Figure 1: Modern Power Switch



Source: Vincentg, 2007.

If we put a series of switches together, we can represent more combinations and therefore more information. For instance, placing eight binary digits together gives us a way to represent 256 different combinations; these 8 digits together are called a byte. A byte could then be used to represent a letter of the alphabet, a specific color, or different application preferences. The type of information represented to a computer depends on the context.

Context is Everything

Without context, even the letters you are reading right now are meaningless. However, once the rules of the English language are applied, these letters form meaning. We can use the same alphabet to represent other information in other languages or codes. Without context, data are completely meaningless, especially binary data, which are merely a long progression of zeroes and ones.

As mentioned earlier, binary can represent letters of the alphabet. Using a byte for each letter, the English alphabet can be encoded into a data file. Fortunately, we don’t have to create our own encoding rules for this—someone already did that years ago when they formed ASCII, the American Standard Code for Information Interchange. For example, the letter “A” in ASCII is represented as:

01000001

Since a capital “A” is distinct from a lowercase “a”, there is a separate code for it: 01100001. This way, a computer can represent all of the letters of the English alphabet along with various punctuation symbols. Using this context, we can now communicate with letters. You may have realized at this point that some data translation must take place. Humans do not input data in binary (at least, they probably would not find it very efficient) so computers must translate to binary to store the data—and then translate them back to show them to humans again! This is a good introductory concept of how computers work. They are constantly translating back and forth between human context and binary.

Figure 2: Full ASCII Table

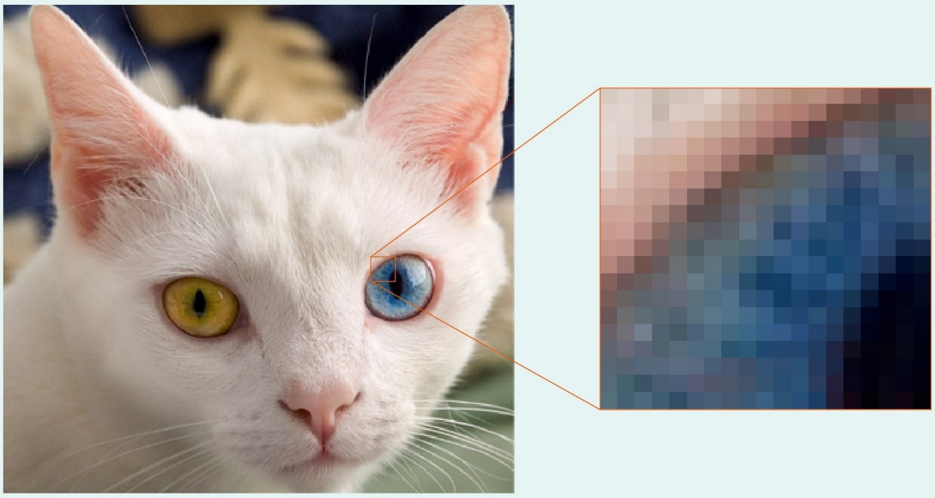
	Hexa-					Hexa-			
Decimal	decimal	Binary	Octal	Char	Decimal	decimal	Binary	Octal	Char
0	0	0	0	[NULL]	64	40	1000000	100	@
1	1	1	1	[START OF HEADING]	65	41	1000001	101	A
2	2	10	2	[START OF TEXT]	66	42	1000010	102	B
3	3	11	3	[END OF TEXT]	67	43	1000011	103	C
4	4	100	4	[END OF TRANSMISSION]	68	44	1000100	104	D
5	5	101	5	[ENQUIRY]	69	45	1000101	105	E
6	6	110	6	[ACKNOWLEDGE]	70	46	1000110	106	F
7	7	111	7	[BELL]	71	47	1000111	107	G
8	8	1000	10	[BACKSPACE]	72	48	1001000	110	H
9	9	1001	11	[HORIZONTAL TAB]	73	49	1001001	111	I
10	A	1010	12	[LINE FEED]	74	4A	1001010	112	J
11	B	1011	13	[VERTICAL TAB]	75	4B	1001011	113	K
12	C	1100	14	[FORM FEED]	76	4C	1001100	114	L
13	D	1101	15	[CARRIAGE RETURN]	77	4D	1001101	115	M
14	E	1110	16	[SHIFT OUT]	78	4E	1001110	116	N
15	F	1111	17	[SHIFT IN]	79	4F	1001111	117	O
16	10	10000	20	[DATA LINK ESCAPE]	80	50	1010000	120	P
17	11	10001	21	[DEVICE CONTROL 1]	81	51	1010001	121	Q
18	12	10010	22	[DEVICE CONTROL 2]	82	52	1010010	122	R
19	13	10011	23	[DEVICE CONTROL 3]	83	53	1010011	123	S
20	14	10100	24	[DEVICE CONTROL 4]	84	54	1010100	124	T
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	85	55	1010101	125	U
22	16	10110	26	[SYNCHRONOUS IDLE]	86	56	1010110	126	V
23	17	10111	27	[END OF TRANS. BLOCK]	87	57	1010111	127	W
24	18	11000	30	[CANCEL]	88	58	1011000	130	X
25	19	11001	31	[END OF MEDIUM]	89	59	1011001	131	Y
26	1A	11010	32	[SUBSTITUTE]	90	5A	1011010	132	Z
27	1B	11011	33	[ESCAPE]	91	5B	1011011	133	[
28	1C	11100	34	[FILE SEPARATOR]	92	5C	1011100	134	\
29	1D	11101	35	[GROUP SEPARATOR]	93	5D	1011101	135	]
30	1E	11110	36	[RECORD SEPARATOR]	94	5E	1011110	136	^
31	1F	11111	37	[UNIT SEPARATOR]	95	5F	1011111	137	_
32	20	100000	40	[SPACE]	96	60	1100000	140	`
33	21	100001	41	!	97	61	1100001	141	a
34	22	100010	42	"	98	62	1100010	142	b
35	23	100011	43	#	99	63	1100011	143	c
36	24	100100	44	$	100	64	1100100	144	d
37	25	100101	45	%	101	65	1100101	145	e
38	26	100110	46	&	102	66	1100110	146	f
39	27	100111	47	'	103	67	1100111	147	g
40	28	101000	50	(	104	68	1101000	150	h
41	29	101001	51	)	105	69	1101001	151	i
42	2A	101010	52	*	106	6A	1101010	152	j
43	2B	101011	53	+	107	6B	1101011	153	k
44	2C	101100	54	,	108	6C	1101100	154	l
45	2D	101101	55	-	109	6D	1101101	155	m
46	2E	101110	56	.	110	6E	1101110	156	n
47	2F	101111	57	/	111	6F	1101111	157	o
48	30	110000	60	0	112	70	1110000	160	p
49	31	110001	61	1	113	71	1110001	161	q
50	32	110010	62	2	114	72	1110010	162	r
51	33	110011	63	3	115	73	1110011	163	s
52	34	110100	64	4	116	74	1110100	164	t
53	35	110101	65	5	117	75	1110101	165	u
54	36	110110	66	6	118	76	1110110	166	v
55	37	110111	67	7	119	77	1110111	167	w
56	38	111000	70	8	120	78	1111000	170	x
57	39	111001	71	9	121	79	1111001	171	y
58	3A	111010	72	:	122	7A	1111010	172	z
59	3B	111011	73	;	123	7B	1111011	173	{
60	3C	111100	74	<	124	7C	1111100	174	|
61	3D	111101	75	=	125	7D	1111101	175	}
62	3E	111110	76	>	126	7E	1111110	176	~
63	3F	111111	77	?	127	7F	1111111	177	[Del]

Source: ZZT32, 2007.

Pixel
A pixel is a single square which is assigned a color and used to assemble a larger image.

Another form that data can take is a graphical image. In the 1950s, Russell Kirsch invented the **pixel** (Ehrenberg, 2010). He decided that the simplest way to divide up the data in a photo or image was to demarcate it into discrete squares—that is what we have used to this day. It was a simple solution and worked well enough that humans could recognize photographs on a computer screen.

Figure 3: Pixels Used in a Photo



Source: Kissel, 2012.

The concept behind pixel images is to break any image up into small squares, so that each square would then have one (and only one) color. If the squares are small enough, the human eye cannot distinguish them individually, and so our brains will see a photographic image. Pixels are used in all types of images including fonts, icons, and graphics, along with photos. How are these data stored? The basic idea is that each pixel has a sequence; the image could start at the top left for the first pixel, and then go across the first row in that order until completing it and then jumping down to the next row of pixels. Since the order is implicit, there is no need to number the pixels. Each pixel has a color as its basic data, so we must represent each separate color as a binary number. If we use eight bits (a byte), then there are 256 different colors we can represent uniquely. This is called 8-bit color.

?

In earlier computers, the VGA graphics system used 8-bit color schemes where the first three bits represented red, the next three green, and the last two were blue. This is where the term RGB comes from for color displays—these three colors of light, in combination, can create any color visible to the human eye. However, with only 8 bits, there are only 256 possible color combinations to represent out of the potentially infinite grades of colorization.

Figure 4: Digital Representation of Color

Bright Yellow in 8-bit color: 11111100

Source: Stephen Weese, 2020.

If we combine the most intense red and the most intense green with no blue, we will get a bright yellow color.

Music can also be stored digitally (which simply means "as digits") in binary. In the same way that we chop an image into little squares (pixels) we can cut a sound or song into tiny slices of time and then string them back together. If the slices are small enough, our ear cannot tell the difference in much the same way that the eye cannot see very tiny pixels.

A document in Microsoft Word, for instance, is also stored completely as ones and zeroes. The context for this data includes some binary codes that stand for formatting, such as bold and italic, and others that stand for page margins, font color, other settings, and, of course, binary code that stands for letters, numbers, and punctuation. MS Word, in a sense, has invented its own code for storing data in a .docx file. When you open a Word document, the binary gets translated back into what you view on the screen so you can begin to read and edit it.

As a programmer, your job will be to decide how to encode and store data and give this data meaning and context so that both a human user and the computer system can understand it.

Data Messaging

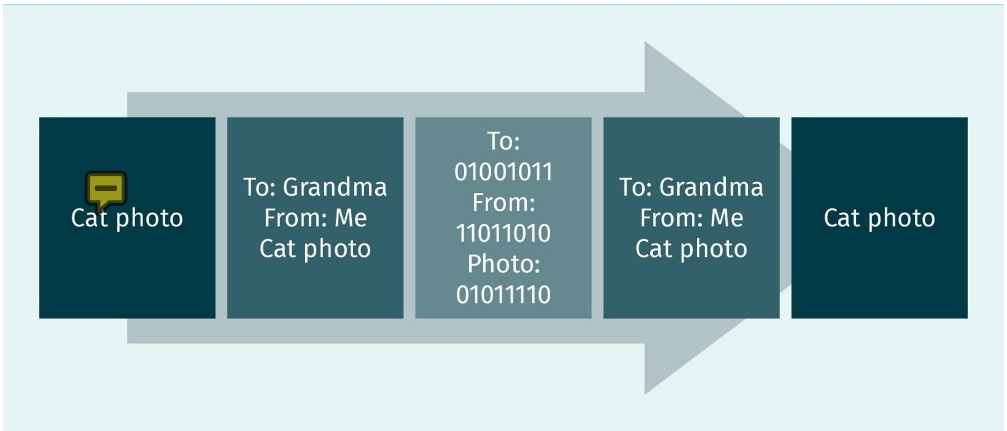
Computers use different hardware components to perform tasks and must communicate using binary messages. Computer components have a special list of commands that indicate information and instructions to each other; these are sent primarily through the motherboard or through data cabling. Computers also send messages to each other through networking.

Now that we have seen how to store letters, images, music, and documents as binary data, we will examine the process of sending these data over a network. The actual process of sending data is complex—there are many stages that include establishing and verifying a connection, agreeing on the mode of communication between computers, preparing the data to be sent, and then, finally, sending it and performing error checking to make sure it has arrived properly.

For our purposes let's say you're going to send a picture of a cat over the internet to your grandma. The cat picture, of course, is just zeroes and ones. But we have to make sure they get to grandma's computer. Here is a simplified demonstration of that process.

lol

Figure 5: Sending a Cat Photo


Cat photo
To: Grandma From: Me Cat photo
To: 01001011 From: 11011010 Photo: 01011110
To: Grandma From: Me Cat photo
Cat photo

Source: Stephen Weese, 2020.

You see the photo of the cat on your screen and save it to a file. Then you send it to your grandma, say, through email. But to send it to her, you have to have two addresses; both TO and FROM fields need to exist for delivery. Once you fill this in, everything is converted to binary: To, From, and all the data of the cat picture. It is sent out over the network to Grandma's computer, which then translates it back so she can see that she has received a cat photo from you. Once the photo data is received, the To and From data are stripped away, as they are no longer needed. Grandma opens up the file, and the computer translates the binary data into a viewable photo once again.

1.2 Software, Firmware, and Hardware

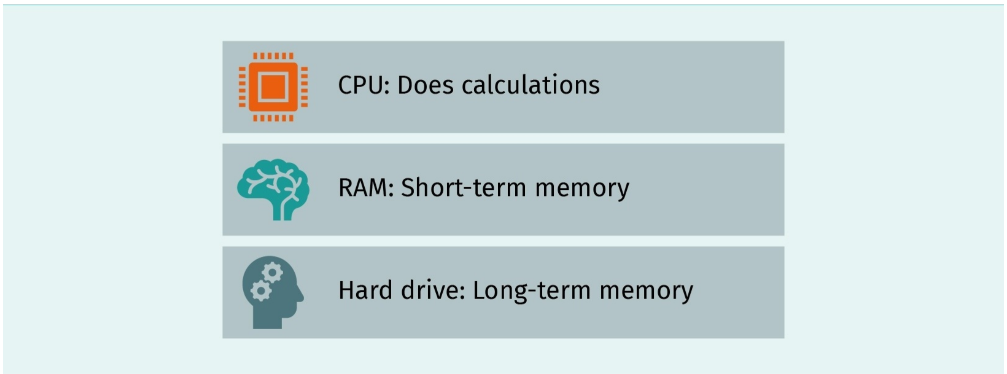
Three types of components exist that work together to enable a computer to function. The most visible component is the hardware; it consists of all the physical components of a device. For our purposes in this unit, we will examine the modern personal computer. Other types of computers also use a similar hardware architecture.

Hardware

The Central Processing Unit (CPU) is often referred to as the "brain" of the computer. It does all the processing, which includes doing calculations and sending instructions to other hardware. It is a square computer chip which is inserted into a motherboard—this is a circuit board designed for computer components to connect to and communicate through. ¶Computers need memory storage. A personal computer will need both long-term and short-term memory storage. awk

One analogy to use is actually human consciousness. We process with our brain, but we also store things in our memory. We have short-term and long-term memory.

Figure 6: Analogy: Computer to Human Thinking



Source: Stephen Weese, 2020.

We all have short-term memory; it contains the things you are thinking about right now. How many things can you think of at once? It may seem like a lot, but let's compare it to your long-term memory. Your long-term memory stores everything that you know. You can't possibly think of all of those things at once; they won't fit into your short-term memory. A computer works the same way. Everything that the computer knows (data) is stored on the hard drive. Whatever you are currently working on is moved into **RAM**, the short-term memory, while it is being used.

RAM
This stands for Random Access Memory and means that the memory can be read randomly and at any point in the data.

Let's say you are working on your résumé. At that moment, the computer is using an application such as MS Word to edit it. While you're working on the document the data in your resume as well as the Word application are taking up RAM (short-term memory). Once you **save** the document, a copy is taken from the RAM and placed in the hard drive which is long-term storage. When you're done working, you close Word, which means RAM is freed up to perform the next task. Since you've saved the file to long-term storage, you know it will be there when you want it again. Just like in real life, when you finish the résumé you stop thinking about it and start thinking about the next thing you are going to do. Your short-term memory clears out space, and you think about the current task. Most modern computer systems follow this model.

Saving
The process of converting data from short-term memory to a file stored in long-term memory is called saving.

You might wonder what the need for RAM is when the hard drive works perfectly well as a storage medium. The short answer is that RAM is a lot faster than the hard drive in terms of memory accessing speed and reading and writing data. It serves as a middle stage between the CPU and the hard drive to provide data quickly for calculations and other uses.

Another comparison is a library. The hard drive is the entire library, whereas the RAM is your desk. It has a smaller data size than the hard drive. You may want to look at one book, or a few at a time, but only so many will fit on your desk. Besides, you can only really look at a few books at a time due to the way your brain (CPU) is structured. When you're done with the current books, you put them back into the library, freeing up the desk (RAM space) so you can put some different books on it.

Firmware

Before we discuss firmware, we must briefly explain software. Software is data; it is represented in binary. The difference between normal data and software is that it consists of instructions to the computer—in other words, it is a specialized type of data. Some software is optional or can vary, but other software is required by the hardware for essential functionality. This type of software is known as firmware. When you first power on a computer, it uses special firmware chips to boot up.

Boot up process

Boot process
This is the startup sequence of a computer. The term comes from the phrase "pull yourself up by your bootstraps."

RAM is volatile—this means that it only preserves data when it has electrical power. This is why in years past (before laptops with batteries became more popular) if you lost power while working on a computer project, it was gone unless you had saved it to the hard drive. When a computer first starts up **(the boot process)**, the RAM is empty and the computer is waiting for instructions on what to do. It finds those instructions (software) in a special chip called a BIOS, which stands for Basic Input Output System (Tarnoff, 2007). It is also referred to as ROM or Read Only Memory. This means it is static and cannot normally be written to. (There is a special process to update these chips.) This is how firmware is designed to be: stable and not easily changed. These instructions are what the computer follows when first it starts up. They are always the same, so they are programmed into a stable chip that retains the data even without electrical power (a non-volatile chip). A modern computer first runs the instructions on the BIOS chip for booting up, which includes identifying all of the hardware attached to the machine (hard drives, amount of RAM, graphics card, etc.). Once that is done, it finds the operating system (OS) on the hard drive or other long-term storage device and begins the process of copying it into RAM. That is what you are waiting for when a computer is "booting" up. You stare at the logo for your computer until this process is finished and you can interact with it.

Firmware describes computer chips that have instructional software on them that remain the same. They are either used for booting up a computer or electronic devices that always perform the same task, such as your smart refrigerator or a child's toy. A single-task device does not need RAM to run different applications. Advanced personal computers can perform many tasks and need the RAM and extra hard drive storage to do so.

Software

Software, as mentioned above, are data that specifically tells the computer to do something—in other words, instructions. (Remember, all computer data is binary.) There are two main types of software: operating systems and applications. Both types of software are updatable, but applications are optional; this means both are not stored on permanent ROM chips, but long-term storage such as hard drives that can be written to.

Operating systems

An operating system (OS) is the first software loaded into the RAM during the boot process. An OS is essential for a computer to function (assuming a standard personal computer). Examples include Windows, Linux, and MacOS. This software provides three fundamental functions for the computer:

1. Provide an **interface**
2. Control hardware
3. Run applications (apps)

Interface
The interface is how a human talks to the computer. Most modern interfaces include a mouse pointer and icons on a desktop.

An interface is required so a human being can interact with a computer. The hardware in a computer system would sit, dormant, without the OS to give it instructions. Applications need an operating system to run. They are a second layer of functionality. This is why when you get an application you must get the application for Windows or for MacOS; they are designed to work with a specific set of instructions that are part of the unique OS.

Applications

Applications (apps) are used to perform specific tasks. If you want to write a document, you load an app. If you want to play a game, listen to a song, edit photos, or create music—all these are tasks performed on a computer that rely on apps. You open the app (loading it into RAM) and use it for your task. When you're done, you close or quit the app. Apps provide specific instructions to the hardware to complete your task, which is why they are also considered software.

1.3 Languages, Syntax, and Semantics

Now that we understand that machines work on a binary level, the next step is to find a way for humans to communicate with a computer. We've learned that an OS and applications provide instructions so people can use the computer, but where does this software come from? The answer is: programmers. Programmers (or coders) have to use a special language (or languages) that both humans and computers can understand. To this end, programming languages were invented.

There are many languages, but they all perform a similar function: they are words and symbols that translate directly to binary commands for the computer. Fortunately, one simple word in English could stand for a long series of binary commands, making it much easier for the programmer to specify what they want to do (Eck, 2009). One such language is Java.

Syntax

Besides coming up with words and symbols which will translate to computer instructions (or **machine language**), the correct ordering of the symbols and words must also be established. Just like the words in an English sentence, programming instructions only

Machine language
A machine language is a set of binary instructions that can be executed directly by a CPU.

make sense in the correct order. Otherwise, it is just a jumble. If you took the words of this sentence and put them in a random order, it would be ignoring proper syntax and make little sense. Syntax also includes rules on when certain symbols are allowed to be used, just like rules for English as to where punctuation can go in a sentence.

```
foodItem = JOptionPane.showInputDialog("Your choices are:
\n1.Hamburger\n2.Cheeseburger\n3.Fish Burger\n4.Veggie Burger");
```

The above line of code in Java is an assignment statement—it takes a variable, `foodItem`, and stores in it the result of a dialog box that appears on the screen.

The text to be displayed is in quotes; however, there is a special code "`\n`", which means: insert an End of Line here. To humans, the "`\n`" probably looks odd, but in Java, the computer knows it means it should go to the next line. Syntax is crucial in computer programming. If just one character or symbol is out of place, the code will not work. Computers are terrible at guessing—if you don't say exactly what you want according to the rules, they will just stop working.

Semantics

After you're sure you've followed all the rules, you have to make sure that you've got the correct meaning—the semantics. If I very clearly describe how to make a peanut butter sandwich, but actually wanted a chicken salad, then I have a semantics problem. It's possible to create perfectly readable code for a computer, yet have it do something different than your intention. Experienced coders learn many skills to prevent that kind of thing from happening; it is best if you are aware of this from the beginning, since it is difficult to unlearn bad programming habits. One such skill is adding comments to code. These are notes to yourself (and other coders) that describe what the code is doing in a certain section. It's especially helpful if you haven't looked at that section of the program for a day or longer, so you know what the code was intended to do instead of trying to figure it out all over again. Advanced programs consist of thousands of lines of code; without comments, they are very difficult for any coder to work with.

Speaking computerese

In many ways, a computer language is similar to human language. There are words and symbols that have meaning, and these elements have a specific set of rules about the order that they can be used in and when they can be used at all. Finally, following these rules, you must assemble a message that makes sense and that explains the concept that you intend to convey. Just like with learning a new human language, you have to start with the basics, and it definitely takes time. The good thing is, once you have learned one computer language, the concepts and functionality are similar so that most others are also easily learned.

At first, the semantics of programming languages may seem pedantic and difficult to understand. However, if we "think like a computer" it will become easier to understand. In English, we have several ways of conveying the same concept: for humans, this is artistic expression, but for a computer this is ambiguous and unnecessary. The more precise and exact you are with a computer, the better results you can expect. Look at the following sentence:

“One day I ate a hot dog in my underwear.”

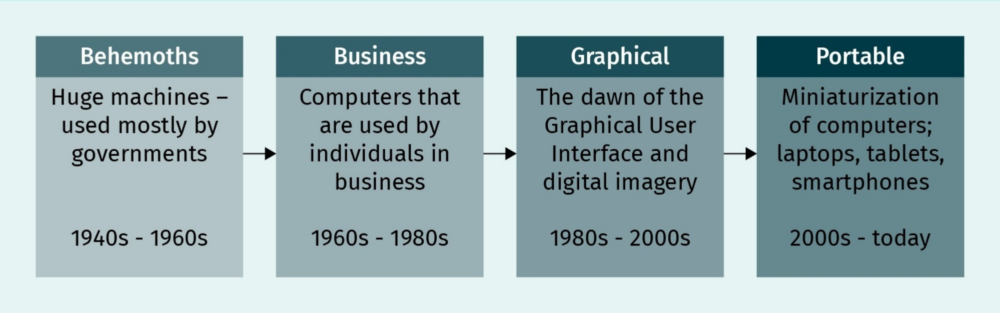
You’d probably imagine someone sitting in their boxers eating a hot dog. But you might also imagine that the hot dog is wearing the underwear. The sentence is grammatically correct, but semantically it could be clearer. Computer code should be used precisely to get the proper results.

1.4 Historical Overview of Computers

There’s no machine that we can point to and say “here is the first computer.” It was more of an evolution of advanced machines into something that eventually resembles the computers we have today. In 1801, Joseph Maria Jacquard invented a loom that used punch cards made of wood to create fabric designs automatically (Zimmermann, 2017). Technically, this was programming—a pre-made set of instructions changed into a “machine language” that tells a machine what to do. However, this machine wasn’t doing any computing, simply weaving cloth.

Later, during World War II, machines were used to encode secret messages. These machines used gears that would align with symbols to create a coded text from regular or “plaintext” messages. Breaking those codes, however, required actual computing power. The 2014 film *The Imitation Game* tells the story of Alan Turing; his Turing machine was able to take in encrypted messages, process them, and output an answer. However, his machine had no keyboard or monitor, let alone a mouse. To look at the history of modern computing machines, we will break it down into four eras.

Figure 7: Eras of Computing

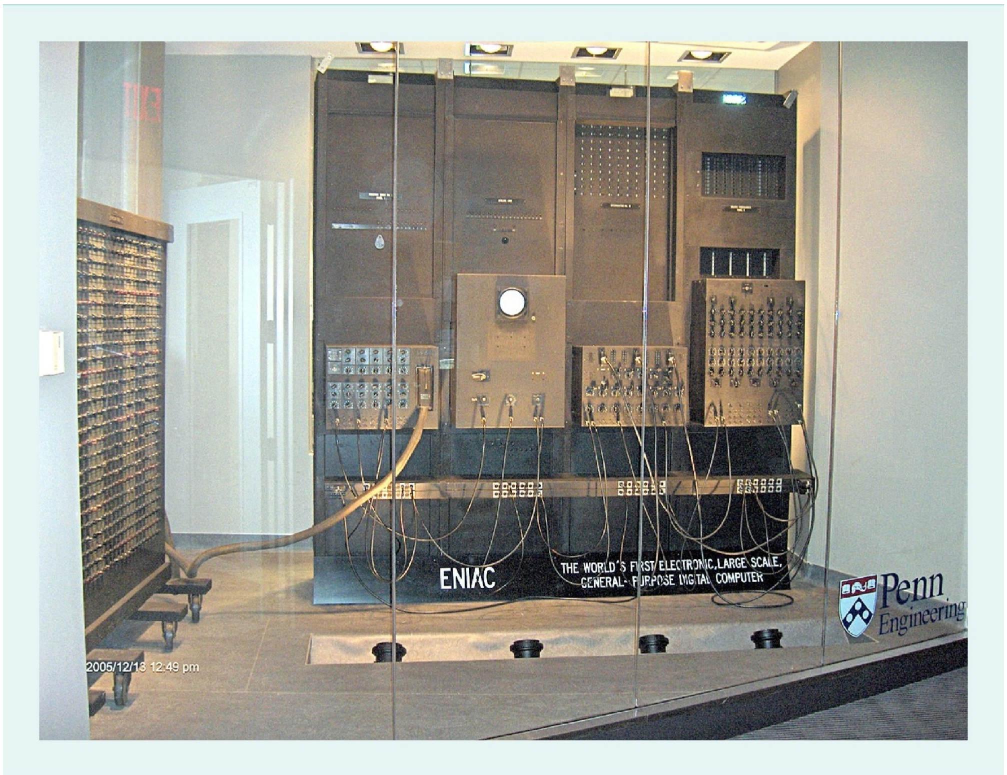

Behemoths
Huge machines – used mostly by governments
1940s - 1960s
Business
Computers that are used by individuals in business
1960s - 1980s
Graphical
The dawn of the Graphical User Interface and digital imagery
1980s - 2000s
Portable
Miniaturization of computers; laptops, tablets, smartphones
2000s - today

Source: Stephen Weese, 2020.

Behemoths

These enormous computers were built with vacuum tubes and wires and took up entire rooms. The input was usually entered by flipping switches and turning dials. The output was often given via lights or punched holes on paper. They were basically number-crunching machines that were used to perform large mathematical computations, or to work with large amounts of numerical data. The U.S. Census Bureau purchased one of these huge computers, the UNIVAC, in 1951 to help with counting the population of the United States (Fabry, 2016). The ENIAC (Electronic Numerical Integrator and Computer), the first general-purpose computer, which began operation in 1946 and was programmed by a team of six female operators, should also be mentioned in this context (Schwartz, 2019).

Figure 8: ENIAC Computer 1945

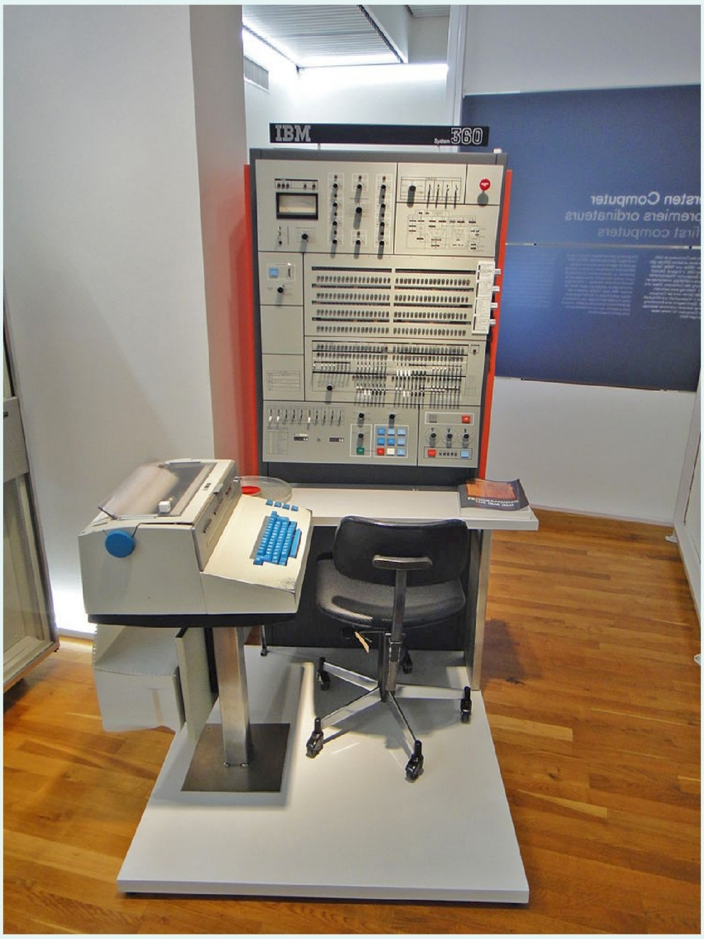

ENIAC
THE WORLD'S FIRST ELECTRONIC, LARGE SCALE, GENERAL-PURPOSE DIGITAL COMPUTER
Penn Engineering
2005/12/13 12:49 pm

Source: TexasDex, 2006.

Business

Eventually, these large behemoths got a little smaller and were able to be produced more quickly. They eventually became affordable for medium to large-sized businesses (instead of requiring an entire government's GDP to purchase).

Figure 9: IBM System/360 Mainframe 1964



Source: Sandstein, 2011.

The output of these machines was a teletype terminal, where text would come automatically out of a typewriter like device. Eventually, in the 1970's, "dumb terminals" were created with **monochrome** CRT screens. Unlike the personal computers of today, these screens with a keyboard did not have their own CPU or RAM, they were connected via an electrical signal to the mainframe which did all of the calculations. Several "dumb terminals" could be connected to a mainframe at the same time, sharing the central computer's processing power. Universities also became major purchasers of these **mainframes** at this time. Many of these mainframes ran the Unix operating system, the predecessor of Linux. It wasn't until the 1980s that the idea of taking a computer home for personal use became widespread. Early home computers were text only—there were no graphics yet and no need for a mouse, so only technical people and hobbyists owned them.

Monochrome
This means using only a single color. Early monitors were monochrome; green, orange, or white on a black background.

Mainframe
A large, powerful, centralized computer, a mainframe is what terminals connect to for processing requests.

Graphical User Interfaces

The invention of Apple's Macintosh computer in the 1980s started a personal computer revolution. Though there were previously computers with graphics capabilities (e.g., the Commodore 64 in 1982), the Macintosh included a mouse and a Graphical User Interface (GUI). Instead of having to type in precise and obscure text commands to use a computer, users could now point-and-click. This made them much more accessible and led to a new, booming market for home computers. They were not only used for business applications, home budgeting, and the like—they were also used for gaming. Richard Garriott's *Ultima II* was released for the Macintosh in 1985. At first, the Mac's display was only black and white, but as computers became more powerful, more colors and pixels were available. The main competitor to Macintosh was the PC, a computer originally made by IBM using an Intel CPU. Even though the Mac was technically also a personal computer, "PC" became synonymous with the Intel-based IBM system (and later other similar "clones"). In the 1990s, Microsoft released Windows, a GUI operating system for the PC. This was the beginning of the legendary "Mac versus PC" divide that still exists today. Modern Macs and PCs now both use Intel CPUs and have very similar hardware designs.

Portable

The next generation of personal computing was driven by display technology—the invention and widespread use of flat screens. If you were born in the mid-1990s, you may never have known a world filled with clunky **CRT** monitors giving off waves of heat. Once flat-screen technology was refined, it enabled truly portable computers to be mass-produced. In the past, some enterprising computer manufacturers had attempted to create "portable" computers such as the Osborne, which weighed over 24 pounds and had a five-inch monochrome CRT built-in. Unsurprisingly, these did not become popular. The flat screens of the 1990s were used to make easy-to-carry laptops and also replaced the bulky CRT monitors used for desktops. As the technology developed, more pixels and colors were made to fit on ever-shrinking screens for tablets and smartphones. Today's smartphone is still a computer—it has a CPU, as well as long-term and short-term storage.

CRT
A cathode ray tube (CRT) was a type of monitor or television that used a large tube to display images.

SUMMARY

Data, in reference to computing, is stored in the binary numbering system. This base 2 system only allows for 0 and 1. Large strings of binary numbers, or bits, can be used to represent all different types of human data.

Text, documents, photographs, music, and video data can all be translated to binary; however, this means that there must be context for the digital information. The bits have to have meaning beyond just the values of zero and one.

Once we have translated and stored ideas from the human world into a computer with binary, they must be translated back again so humans can understand them. That is the job of the computer.

A basic computer consists of a brain, the CPU, and memory storage. Some storage is short-term, like RAM, and other types are long-term, like the hard drive.

Software is binary instructions to the computer. It takes the form of either an operating system or an application. The OS provides basic functionality including an interface, while applications (apps) are available to perform specific tasks.

To program a computer, languages had to be invented that were somewhere between natural language and binary. These provide a way for humans to give instructions to a computer without having to write everything in binary. Programming languages have rules, including syntax, which dictate how to use the words and symbols of that language.

Computers were designed to automate complex tasks and process data. Some of the earliest machines were used by governments to decode encrypted messages during World War II. Eventually, the size and price of computers was reduced, and businesses became able to purchase them. The invention of CRT monitors allowed people to have “personal computers” in their homes. The invention of the GUI made computer use easier, causing its popularity to explode.

UNIT 2

INFORMATION REPRESENTATION

STUDY GOALS

On completion of this unit, you will have learned …

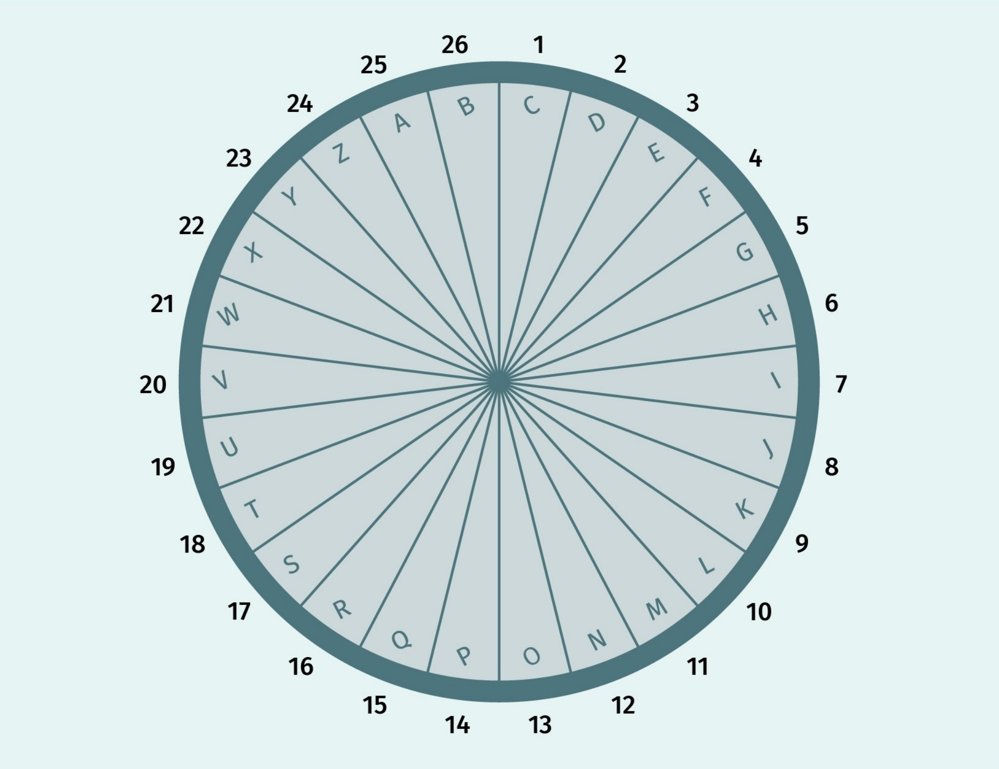
- about the binary numbering system and how to convert decimal to binary.
- about data storage sizes, such as kilobyte, megabyte, and terabyte.
- how graphics, documents, music, and other data are represented in memory.
- about the different data types used in programming such as char, string, int, and float.
- how computers perform error checking on stored and network data.

2. INFORMATION REPRESENTATION

Introduction

In 1936, kids from all over the United States would tune their radios in to the Little Orphan Annie radio show. They knew that sometime during the show, it would be time to break out their decoder pin—the one they ordered after sending in proof of purchase of the chocolate drink Ovaltine, a sponsor of the show. This pin had two concentric rings: the inside was a ring of letters and the outside a ring of numbers. The kids would listen for what letter to turn to the top of the inner ring, and then be given a sequence of numbers which were the secret code for that week. The kids eagerly wrote down these numbers and began the work of decoding the secret message, which was usually a preview of what would happen on the show next week. It was brilliant marketing as it not only encouraged the children to ask their parents to buy more Ovaltine, but also made kids look forward to tuning in the next week, keeping up the audience numbers.

Figure 10: Ring Cipher



Source: Stephen Weese, 2020.

The children probably didn't realize that they were using something called a ring cipher: a simple device used to encode and decode messages.

The Orphan Annie membership pin they used is an apt representation of how computers deal with non-digital data. The 26 letters of the English alphabet were converted into decimal numbers. You may notice that this conversion is not symmetric—there are only ten different symbols for numbers (0—9), not nearly enough to have one for each letter. Therefore, some of the letters would require two-digit numbers to symbolize them. Since all data in computerized machines are stored digitally (as numbers), conversion has to occur from all types of non-numeric data, not unlike the decoder pin. In the case of computers, the data are not necessarily a “secret” message, but they certainly are in code, a code that computers are fluent in: the binary numbering system.

2.1 Number Representation: Formats

Human beings have ten fingers. It is a common reason given for why we use a base 10 numbering system: decimal. Decimal uses ten symbols, 0 through 9, to represent all possible numbers. Computers, however, don’t have fingers. What early computers did have were switches. These could be turned on or off, giving two possibilities and a base 2 numbering system: binary. There are two symbols, 0 and 1, that are used in binary. Since humans and machines use different numbering systems, translation must occur.

Figure 11: Decimal Table

100 thous.	Ten thous.	Thousands	Hundreds	Tens	Ones
10^5	10^4	10^3	10^2	10^1	10^0
100,000	10,000	1,000	100	10	1
8	**0**	**6**	**4**	**1**	**2**

Source: Stephen Weese, 2020.

In decimal, the number 806,412 is easily decipherable to a human. We recognize that each successive number to the left of the first one represents the next power of ten. In other words, 800,000 + 6,000 + 400 + 10 + 2 = 806,412. Most of us learned this very early in school. Each time a digit is added, there are ten new combinations of possibilities, which is why we multiply by a power of ten. In binary, there are only two possibilities for each digit, which means adding a digit creates a power of 2 more possibilities.

Figure 12: Binary Table

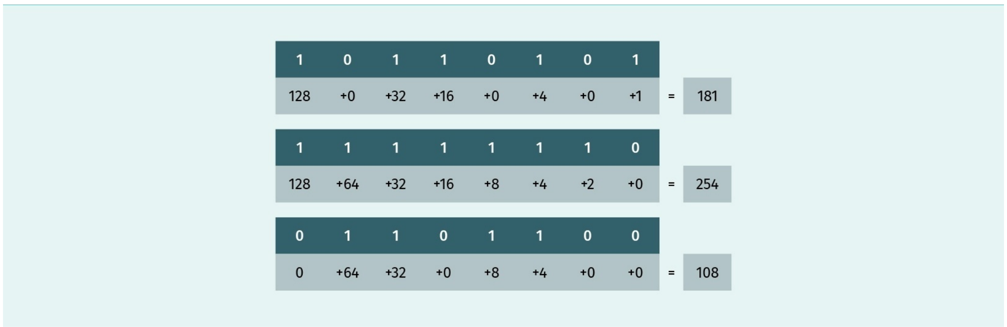
One twenty eights	Sixty fours	Thirty twos	Sixteens	Eights	Fours	Twos	Ones
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	1

Source: Stephen Weese, 2020.

This eight-digit binary number represents a numeric value to a computer. A human would have to translate 00101101 this way: 32 + 8 + 4 + 1 = 45. There is a 1 in the 32's place, the 8's place, the 4's place, and the 1's place. There are no 16s or 2s. This demonstrates how to translate binary into decimal. Since a **byte** is a common unit of data in computing, it is useful to know that 2^8 is 256, meaning that 8 digits in decimal can represent 256 combinations, or 0 to 255. Just as we can keep adding digits in decimal to get larger numbers, we can add more digits in binary to increase the value.

Byte
This refers to eight bits (digits) together in binary, often used to measure data sizes.

Figure 13: Binary to Decimal Examples



1	0	1	1	0	1	0	1		
128	+0	+32	+16	+0	+4	+0	+1	=	181
1	1	1	1	1	1	1	0		
128	+64	+32	+16	+8	+4	+2	+0	=	254
0	1	1	0	1	1	0	0		
0	+64	+32	+0	+8	+4	+0	+0	=	108

Source: Stephen Weese, 2020.

It is simple to convert from decimal to binary—the process is to keep dividing the decimal number by 2 while recording the remainder. This will reveal the binary number (in reverse order of digits). For instance, let’s look at the number 97 in decimal and change it to binary.

Figure 14: Decimal to Binary Examples

Number	Operation	Result	Remainder
97	÷ 2	48	1
48	÷ 2	24	0
24	÷ 2	12	0
12	÷ 2	6	0
6	÷ 2	3	0
3	÷ 2	1	1
1	÷ 2	0	1 Start Here

Source: Stephen Weese, 2020.

Going in reverse order of the remainder, the binary result is 1100001 for a decimal value of 97. We can quickly check our math by going back to decimal:

$$\begin{array}{ccccccccc} \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \\ 0 & +64 & +32 & +0 & +0 & +0 & +0 & +1 & =97 \end{array}$$

Bytes and Data Measurement

Since a single bit only represents two possibilities, a larger grouping of bytes is often used for measuring data. A byte can represent simple data like a letter from a certain alphabet, a color, or a musical note. Different types of data can take up varying amounts of space, which are measured in bytes or multiples of bytes.

Figure 15: Data Storage Quantification

Name	Abbreviation	Size	Bytes	Data size
Kilobyte	KB	1,000 bytes	1 thousand	Document
Megabyte	MB	1,000 KB	1 million	Photograph
Gigabyte	GB	1,000 MB	1 billion	Movie
Terabyte	TB	1,000 GB	1 trillion	Hard disk
Petabyte	PB	1,000 TB	1 quadrillion	All photos on Facebook

Source: Stephen Weese, 2020.

For instance, a Word document might take up around 28KB of storage, a JPG photograph about 3MB, and a DVD movie around 4GB. Most modern hard drives are capable of storing around 1TB of data. Video, applications (especially video games), and operating systems take up the largest amounts of data storage. Notice that each naming convention above is 1,000 times the previous.

Octal

You may have observed that it takes quite a large number of digits to represent numbers in binary in comparison to decimal. A good method to abbreviate binary is to use the octal number system, which, as you may have surmised from the name, is a base 8 system. This gives us combinations of digits, with 0 through 7 as possible values for each digit. Each digit's place value is a multiple of 8, which is a power of 2, so it maps nicely—one digit of octal can represent exactly three digits in binary.

Table 1: Octal Numbers

Binary	Octal	Decimal
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	10	8
1001	11	9
1010	12	10

Source: Stephen Weese, 2020.

As you can see, though we are using some of the same symbols in all three systems, they mean different things. When we write "10" in decimal, to us humans it means 10. In octal, "10" means 8 to us, and "10" means 2 in binary. In fact, there is an old joke which illustrates this concept.

Figure 16: Binary Humor

**There are 10 types of people in the world:
those who understand binary,
and those who don't.**

Source: Stephen Weese, 2020.

Hexadecimal

Though octal may be useful in some situations, Hexadecimal is the preferred intermediate between decimal and binary. One digit of hexadecimal (or “hex”) can represent four digits of binary. Hexadecimal is base 16—that means there are sixteen possible symbols per digit. Obviously, we will run out of decimal numbers for symbols, so hex recruits letters from the alphabet.

Table 2: Hexadecimal Numbers

Binary	**Hexadecimal**	**Decimal**
1000	08	8
1001	09	9
1010	0A	10
1011	0B	11
1100	0C	12
1101	0D	13
1110	0E	14
1111	0F	15
10000	10	16
10001	11	17

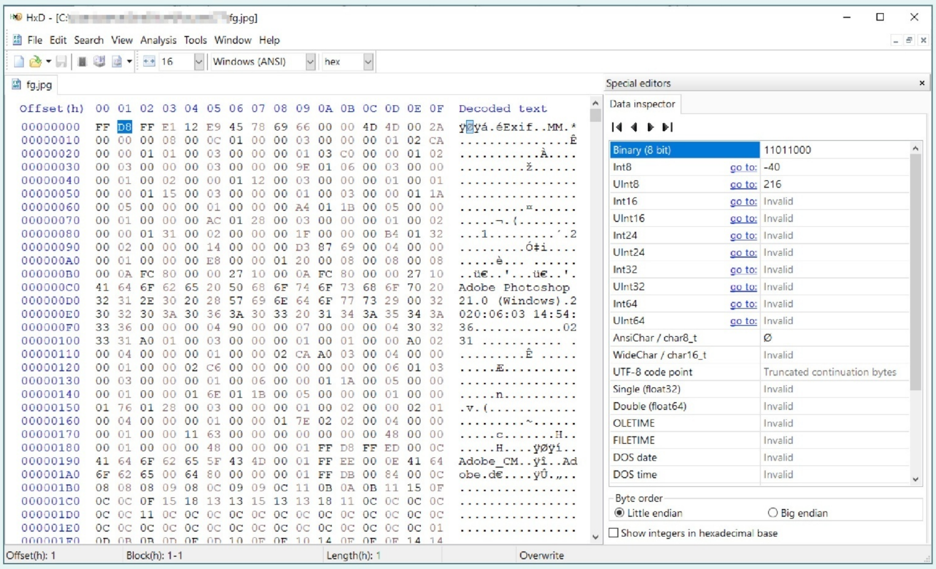
Source: Stephen Weese, 2020.

In computer applications, hexadecimal numbers are usually written in groups of two, since two digits represent eight digits in binary (a byte). You will often see a leading zero before a hex number to represent this, as in the table above. Hex is often used to view raw data in a file; it is a sort of shorthand for binary. It represents 16 different possibilities with one digit, 0—F. “A””represents decimal 10, “B” represents 11, and so on.

Often, computer scientists may want to analyze the contents of a file. Viewing it in binary is very difficult and nearly meaningless for a human. “Hex editors” are often used that show a view of the hex translated content, as are **ASCII** decoders.

ASCII
This is a binary code scheme where letters, numbers, and symbols are represented by exactly one byte of data.

Figure 17: Hex Editor



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  FF D8 FF E1 12 E9 45 78 69 66 00 00 4D 4D 00 2A  ÿØÿá.éExif..MM.*
00000010  00 00 00 08 00 0C 01 00 00 03 00 00 00 01 02 CA  ...............Ê
00000020  00 00 01 01 00 03 00 00 00 01 03 C0 00 00 01 02  ...........À....
00000030  00 03 00 00 00 03 00 00 00 9E 01 06 00 03 00 00  .........ž......
00000040  00 01 00 02 00 00 01 12 00 03 00 00 00 01 00 01  ................
00000050  00 00 01 15 00 03 00 00 00 01 00 03 00 00 01 1A  ................
00000060  00 05 00 00 00 01 00 00 00 A4 01 1B 00 05 00 00  .........¤......
00000070  00 01 00 00 00 AC 01 28 00 03 00 00 00 01 00 02  .....¬.(........
00000080  00 00 01 31 00 02 00 00 00 1F 00 00 00 B4 01 32  ...1.........´.2
00000090  00 02 00 00 00 14 00 00 00 D3 87 69 00 04 00 00  .........Ó‡i....
000000A0  00 01 00 00 00 E8 00 00 01 20 00 08 00 08 00 08  .....è... ......
000000B0  00 0A FC 80 00 00 27 10 00 0A FC 80 00 00 27 10  ..ü€..'...ü€..'.
000000C0  41 64 6F 62 65 20 50 68 6F 74 6F 73 68 6F 70 20  Adobe Photoshop 
000000D0  32 31 2E 30 20 28 57 69 6E 64 6F 77 73 29 00 32  21.0 (Windows).2
000000E0  30 32 30 3A 30 36 3A 30 33 20 31 34 3A 35 34 3A  020:06:03 14:54:
000000F0  33 36 00 00 00 04 90 00 00 07 00 00 00 04 30 32  36............02
00000100  33 31 A0 01 00 03 00 00 00 01 00 01 00 00 A0 02  31 ........... .
00000110  00 04 00 00 00 01 00 00 02 CA A0 03 00 04 00 00  .........Ê .....
00000120  00 01 00 00 02 C6 00 00 00 00 00 00 00 06 01 03  .....Æ..........
00000130  00 03 00 00 00 01 00 06 00 00 01 1A 00 05 00 00  ................
00000140  00 01 00 00 01 6E 01 1B 00 05 00 00 00 01 00 00  .....n..........
00000150  01 76 01 28 00 03 00 00 00 01 00 02 00 00 02 01  .v.(............
00000160  00 04 00 00 00 01 00 00 01 7E 02 02 00 04 00 00  .........~......
00000170  00 01 00 00 11 63 00 00 00 00 00 00 00 48 00 00  .....c.......H..
00000180  00 01 00 00 00 48 00 00 00 01 FF D8 FF ED 00 0C  .....H....ÿØÿí..
00000190  41 64 6F 62 65 5F 43 4D 00 01 FF EE 00 0E 41 64  Adobe_CM..ÿî..Ad
000001A0  6F 62 65 00 64 80 00 00 00 01 FF DB 00 84 00 0C  obe.d€....ÿÛ.„..
000001B0  08 08 08 09 08 0C 09 09 0C 11 0B 0A 0B 11 15 0F  ................
000001C0  0C 0C 0F 15 18 13 13 15 13 13 18 11 0C 0C 0C 0C  ................
000001D0  0C 0C 11 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C  ................
000001E0  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 01  ................
000001F0  0D 0B 0B 0D 0E 0D 10 0E 0E 10 14 0E 0E 0E 14 14
```

Binary (8 bit)		11011000
Int8	go to:	-40
UInt8	go to:	216
Int16	go to:	Invalid
UInt16	go to:	Invalid
Int24	go to:	Invalid
UInt24	go to:	Invalid
Int32	go to:	Invalid
UInt32	go to:	Invalid
Int64	go to:	Invalid
UInt64	go to:	Invalid
AnsiChar / char8_t		Ø
WideChar / char16_t		Invalid
UTF-8 code point		Truncated continuation bytes
Single (float32)		Invalid
Double (float64)		Invalid
OLETIME		Invalid
FILETIME		Invalid
DOS date		Invalid
DOS time		Invalid

Byte order
● Little endian ○ Big endian

□ Show integers in hexadecimal base

Offset(h): 1 Block(h): 1-1 Length(h): 1 Overwrite

Source: Hörz, M, 2020.

This is the raw hexadecimal data from a photo taken by the author. Note the hex number D8 is highlighted, showing a binary value of 11011000. On the “Decoded text” pane the ASCII interpretation is displayed.

Computing and Numbering

Since computers use binary, people must use different ways to represent data that are more compatible with human thinking and concepts. These ways include different codes, such as ASCII, and different numbering systems such as octal and hexadecimal. Hexadecimal is often used as a go-between for humans since: a) it has an exact 1:4 ratio of digits with binary, and b) it has more symbols, so it is easier for humans to read. Displaying data in hex takes four times less space than in binary.

2.2 Representation of Non-Numerical Information

We have demonstrated how decimal is converted to binary and stored in a computer. This will work for any data set that consists only of numbers. However, human data are also stored in words, images, and even sound. This means that to store this information on a computer we must also have a system to convert all of this to ones and zeroes.

Everything is Binary

Think of your favorite song right now. On a computer, it is stored in binary. The entirety of all the subtle notes and sounds are captured as zeroes and ones. What about a beautiful painting? This, too, is broken down into zeroes and ones and then displayed to you as light on a monitor screen. From Handel's greatest works to The Rolling Stones, from DaVinci to Dali, a computer sees everything, stores everything, as binary numbers. The Bible, the writings of Goethe, the sayings of Confucius—all become digital data when stored on a computer. How is this done? It all depends on context.

Data Contextualization

One of the simpler ways to represent data is to convert alphabetical letters into numbers. Looking back to our decoder ring from the 1930s, we can see that it is a simple matter to assign a specific number to a letter. But what about upper and lowercase? Then we create separate numbers for that. Punctuation? Once again, they get unique numbers. Once we have decided how many letters and symbols we have to represent (including things like spaces) then we can create a 1-to-1 assignment of numbers. It goes without saying, of course, that these numbers will all be in binary. We have discussed ASCII before, but this concept was expanded for more modern computers and international languages with an encoding standard called Unicode.

Digital Documents

ASCII can only display English characters (with a few exceptions), while Unicode was designed to represent most of the alphabets across the entire globe. It includes world currency symbols and italics. Since it uses 16 bits, it can represent a total of 2^{16} or 65,536 different symbols. Unicode is often used for internet communication, from email to webpages. With Unicode we are able to represent raw, unformatted characters and symbols in many languages, but to create something like a Word document, another layer of complexity is needed.

In 2007, Microsoft released an update of their Office software suite which included a new way to store its data files. This new method was based on XML, or Extensible Markup Language. A document, Word or otherwise, must have other information besides letters and symbols: this includes margins, page size, fonts, sizes, and colors, for example. Put simply, Word documents and other documents use binary codes to "tag" different sections of text with different attributes such as bold or a certain indent. This, of course, is also all stored in binary in a very specific order. When a Word document is read by an application, the binary is interpreted according to the rules for the file type and it is displayed on your screen.

A Word document (or any other data file) is a meaningless string of zeroes and ones without context. When you use the Word application to open a Word document, it knows the specific context, and therefore exactly what all those numbers mean.

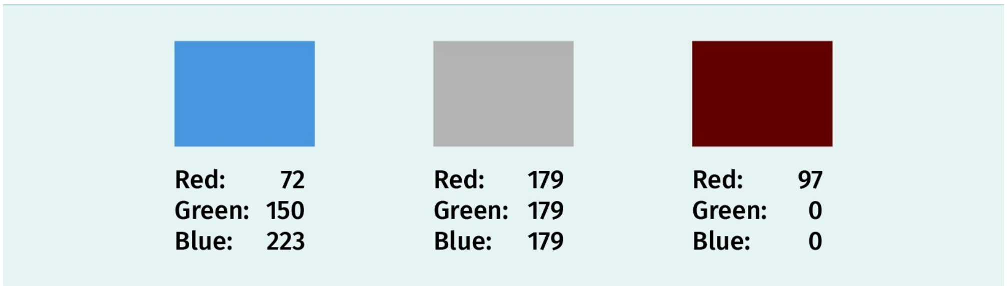
Images

Photos and other graphics that are displayed on a screen are broken down into pixels, or very small squares (sometimes rectangles). For instance, right now you are viewing a document that is stored as text and formatting; these must be converted into pixels on your screen. When a human uses a computer, several layers of conversion and translation are continually occurring, carried out by the operating system and the application. Once we developed large enough data storage, humans were able to design computers with very small pixels so that millions of them could fit on your display screen, giving you a high **resolution,** which makes photographic images look realistic and text very readable.

Resolution
The number of pixels able to be displayed is called the resolution. This is usually measured as width by height, e.g., 1920 x 1280.

Many displays now use 24-bit color. This means there are 8 bits each for red, blue, and green. These colors of light combined together can represent the full spectrum of visual light. To store any image as binary, it is broken up into pixels which are each given a binary value that shows how much red, blue, and green light intensity to display it with.

Figure 18: Digital Pixel Colorization



Source: Stephen Weese, 2020.

Since 8 bits (a byte) can store values from 0—255, each color can have that many degrees of intensity. Absence of a color would be zero, while the most intense is 255. Using 255 for all colors will display white, while using 0 for all colors would be black. This requires enough memory to store 24 bits multiplied by the number of pixels on the screen. This means that to store an HD image (1920 x 1080 pixels) with 24-bit (3 byte) color, you would need:

$$1920 \cdot 1080 \cdot 3 = 6{,}220{,}800 \mathrm{bytes}$$

Which, in other words, is about 6MB. Older computers from the 1990s, with only 1MB of memory, had no means of displaying anything with close to this many pixels and colors, which is why images on those screens looked very **pixelated.**

Pixelated
This term describes an image represented on a computer with large enough pixels that they are individually visible.

Audio and Music

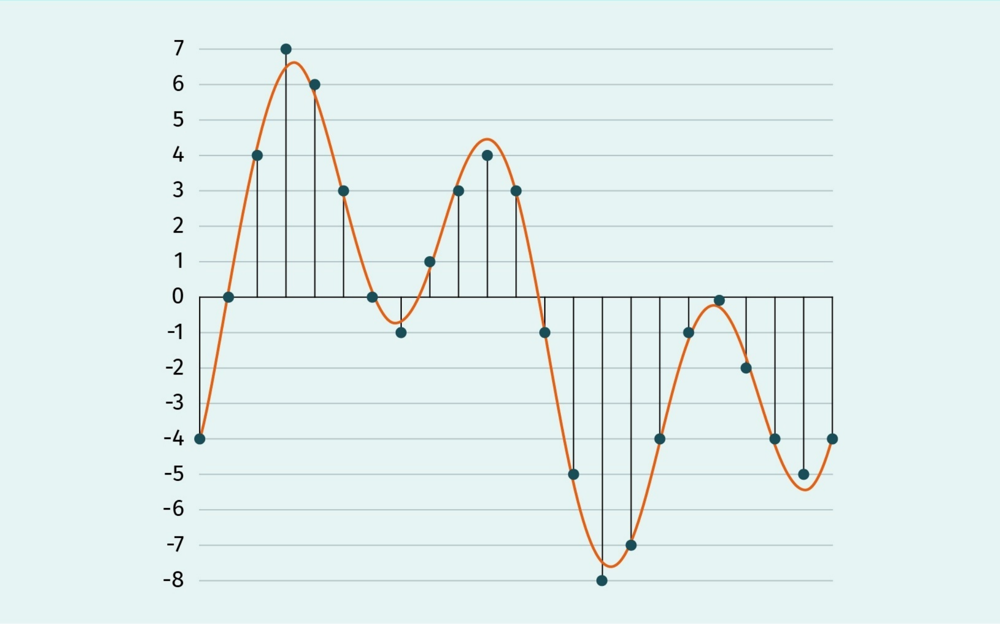
In the real world, light and sound exist in waves. Information represented in waves (e.g., an audio cassette tape) is referred to as analog. When we store it on a computer it must be converted into digital data. Does this mean that some of the information is lost? Definitely. Analog waves have an infinite continuum of values, whereas digital representations of waves must simulate them with **sampling.**

Sampling
This is the representation of analog data by taking samples of the wave values over small time intervals.

Analog data are sliced into very small time samples and a digital value is assigned for that time slice. The smaller the slices, the more accurate a representation. However, we cannot have infinitely small samples, just as we cannot have infinite pixels.

When we look at the world with our eyes, we are not seeing pixels, but continuous waves of light. However, when a computer makes the pixels on the screen small enough our eye simply cannot tell the difference. This is the same principle with audio sampling. If we cut up an audio wave into small enough slices and reassemble it, the human ear cannot tell the difference.

Figure 19: Digital Audio Sampling



Source: Aquegg, 2013.

Even before digital video, the same concept was used for film. A movie viewed in the theater is really a series of still images shown rapidly enough that to the human eye (and brain) it seems like real motion.

Whether the audio is music, singing, sound effects, or someone speaking, it is sampled and stored as binary data, where the context tells the computer to reproduce a sound through a speaker so humans can listen.

Context is Key

Computers need assistance in deciphering the message we are sending with binary. There are two primary ways that a file (a discrete unit of data) is labeled to contextualize the data. The first, simple way is through the file extension. The three or four letters appended to the filename tell the operating system what type of application is able to read the data in this file properly. Not only that, but files also have headers (or file signatures) at the beginning with specific binary sequences that indicate that the data are an image, a document, an executable program, or something else.

For any specialized type of data, a computer system must first be created to store all of the information in binary. That standard will then be used by any application that wants to read or write these types of data. This is where we get file types such as .jpg, .docx, .exe, .dmg, etc. New file types and data contexts are continually created every day.

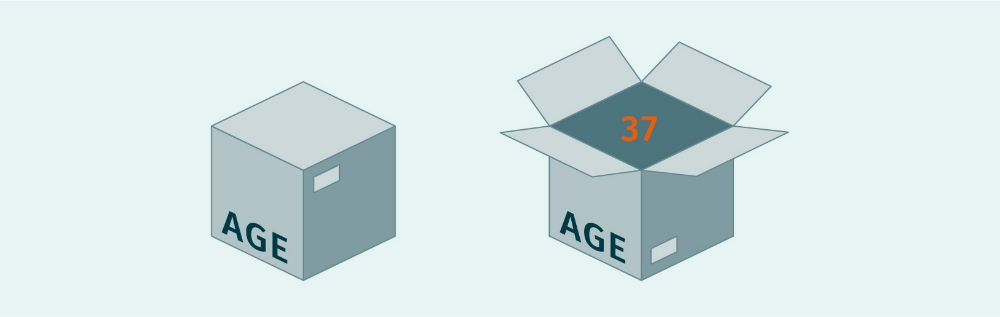
2.3 Data Types

In this unit, we have learned that files, or large blocks of data, need to have context. On a more granular level, individual units of data must also have context. When we work with a data point, it must be defined to the computer as a number, a character, a memory location, or something else. For instance, a Word document contains all kinds of different data: some numerical, some character, some formatting. Each separate character would need a data type to identify it correctly.

Variables

In programming, the coder creates variables to store individual units of data. A variable is a container for information. It has three important defining factors: a name, a type, and a memory location. The name is what it will be referred to in code, just as if you wrote a label on a box. The memory location is where the computer can find this box in RAM.

Figure 20: Variable as a Box



Source: Stephen Weese, 2020.

Let's say you create a variable called "AGE". The computer will label that box with the name and store it somewhere in memory. But there's nothing in the box yet. We need to store a value in the box; a variable is a container for a data value. The value of 37 is assigned to the variable "AGE". Now when the computer goes to find the box and opens it, the value 37 is discovered within. Just like in real life, a box can be reused; we can put other values in the box at different times. This is why it is called a variable—the value inside it can vary. It is different from a constant which is declared at the beginning of the program and cannot ever change. The data inside must also have a "type". Just as a paper bag is not made to contain liquids, but solids, variables are designed to contain specific types of data.

Integers

An integer is a unit number; this means it contains no decimal component. Integers include all the natural numbers (1,2,3, etc.), their negative versions, and zero. This definition is important to a computer since it does not have to assign any memory to decimals, only unit numbers. In the previous case of the variable "AGE", an integer would be a good data type to use, since it is a standard convention to list someone's age as a unit number without a fraction or decimal on many forms and records.

In Java, the integer data type is abbreviated "int." To declare (create) a variable for age, it would look like this:

```
int age = 0;
```

When declaring a variable, in some languages a starting value must be given; this is referred to as initializing. In this case, the variable is initialized to zero. It presumes that an actual age will be calculated or input by the user later.

Numerical data types

Other standard data types include longer integers and decimals. The following are some examples of numerical data types in the programming language Java.

Byte: integer from -128 to 127 (8 bits)

Int: integer from -2,147,483,648 to 2,147,483,647 (32 bits)

Float (smallest value): 1.4E-45 to 1.4E-45 to 3.4E38 (32 bits)

Float (largest value): 3.4028235E38 (32 bits)

Double (smallest value): 4.9E-324 to 49E-324 to 1.7E308 (64 bits)

Double (largest value): 1.7976931348623157E308 (64 bits)

As you can see, data types are specified in bits—this is important to programmers. You don't need to use a huge box to store a pair of sunglasses; it's a waste of space. Coders also shouldn't use large variables such as `int` when a byte will do. Note that a "byte" data type can store 256 different values; some of them are used for negative, some positive, and one is zero. To declare these variables in Java, you'd do something like this:

```
byte age = 37;

int jellybeans = 2243;

float fluidOunces = 28.73;

double solarDistance = 932842983.2340273;
```

The reason a decimal is called a "float" is because the decimal point can float around to different places in the number. This is an important distinction for a computer: where is the decimal? That must be stored somewhere in the variable itself. It is something a human would take for granted, but something that has to be specified for a computer. Many programming languages are also case-sensitive; this means that the variable "age" is not the same as "Age" or even "AGE". Good coders should follow a naming convention for all of their variables for consistency.

Non-numeric data types

Most languages, including Java, have a basic data type to store one "character." This represents one symbol in a language, so would include letters, numbers, spaces, and punctuation. You might think that we have already covered numbers, but remember computers are very specific. The numeric value of "2" is a separate concept than the character for "2". Humans automatically deal with the different contexts where we see the symbol for "2", but computers are more specific. Consider that displaying the number 2 on the screen as a specific group of pixels is completely different operation than multiplying a value by 2.

In English and many other languages, words and sentences are represented by groups of characters strung together—this is the reason that computers have a data type called a string. A string is a group of characters strung together. Therefore, it is actually a complex data type consisting of several simple (or "primitive") variables combined together (primitive data types). Here are some non-numeric data types from Java (Oracle, 2020):

- Char (a character): letter, number or symbol (16 bits)
- Boolean: represents "true or false" (1 bit)
- String: a sequence of character information (varies in size)
- Array: an ordered sequence of a primitive data type (varies in size)

A Boolean variable (named after mathematician George Boole) represents either a TRUE or FALSE value, and therefore only has two possible states, which technically can be stored as one bit; zero is used for false and one for true.

An array is an ordered arrangement of a simpler data type. You could, for instance, have an array of integers, doubles, or even Booleans. However, an array of characters is so common that the data type string usually exists in most languages for storing language text. Let's look at how these data types would be declared as Java variables:

```
char middleInitial = 'A';

String firstName = 'Joseph';

int[] lottoNumbers = {5, 22, 11, 7, 16};
```

The array contains **elements** that represent different lottery number picks. To reference the second number, you would type "`lottoNumbers[1]`" and it would return the value "`22`".

Element
This is a single value in an array, addressed by its ordinal number; these start at 0 for the first element

Arrays are often used when you have a collection of data of the same type that you want to store in order. You can search through them sequentially, pick out singular items that you want to work with, and even sort the list of elements in order. Uses of arrays include lists of options to choose from, arranging data groupings, or collecting user input, to name a few.

Using Variables Correctly

When creating variables, you should be considering the correct data type as well as the memory usage. Not only that, but descriptive names should be given that also follow the standard naming convention that is used for that language. Many new programmers begin by naming their variables "x" and "y" for simple programs, but these names are non-descriptive and will be problematic in code of any reasonable length. In code with hundreds of variables, non-descriptive names will leave the programming unreadable to anyone, including the original coder. Variables should not be a mystery; they should be named in a way that describes their use in code. Computers have very specific data types and expect them to be used in the correct context. You cannot "add" together two char data types and expect the same result as adding two integers. The computer will see this as two different things. If you have the characters (char data type) "4" and "2" and combine them, in many computer languages you will get "42" instead of 6. The characters are seen as symbols and simply placed together; if they were numeric variables (such as int) then adding them would produce a numeric answer rather than a character based answer. Computers do exactly what they are told. This is why data types and variables must be used with precision.

2.4 Redundancy and Error Tolerance

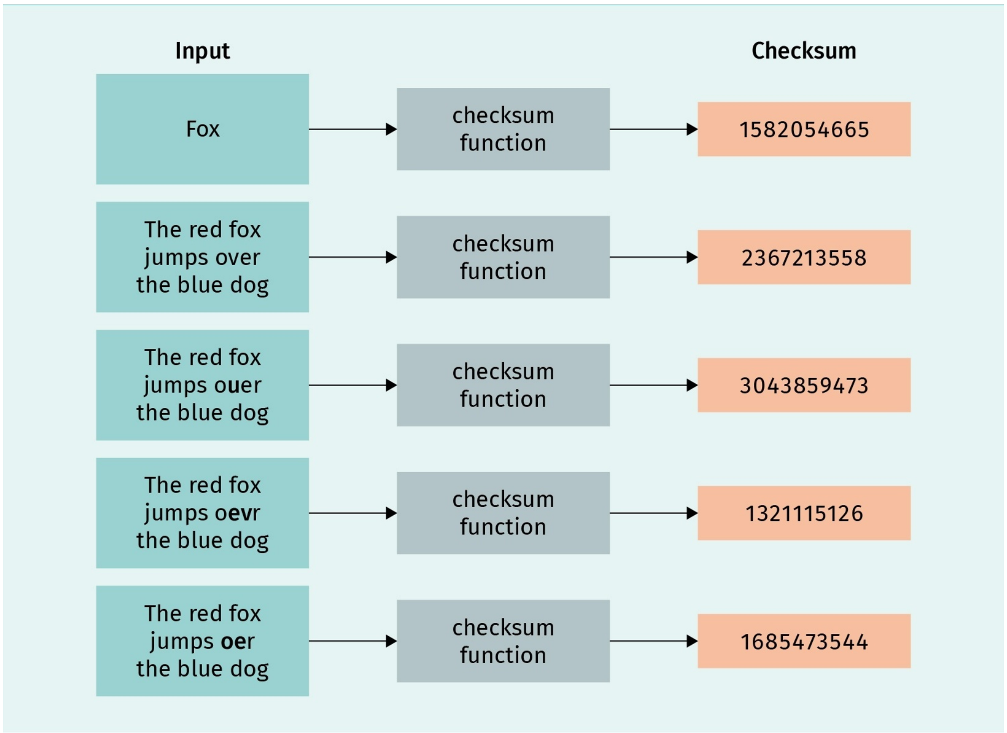
Imagine writing a 100,000-word novel. You carefully read the manuscript after your first draft, and check for mistakes—but did you really find all of them? There is still a chance that errors slipped through. You might do a second check or hire an editor. Are you sure you got all of them this time? It is very difficult to be sure of perfection in such a large volume of words. Computers process words and information at a much higher rate than human beings. A simple image on your screen of 800 x 600 pixels has 480,000 pixels. Are they all exactly the right color? Did some bits get confused in transmission? A computer, though very accurate in its calculation, can also suffer from errors in data transmission. Whenever data is moved from one location to another, there is a chance that the electrical signal may be garbled, interfered with, or interrupted. Data can move from RAM to the hard drive, from the hard drive to the internet, then from the internet to your friend's house in Argentina. So just sending a photo of a cat to Mari Rubi in Buenos Aires has several points where errors can occur. Therefore, computers must have built in precautions against error corruption.

Storage Error Checking

Some errors are not caused by transmission, but media corruption. Hard drives typically last about three to five years. Some drives experience a gradual corruption, where parts of the physical surface become unstable and will no longer store data. An early method to protect files in storage from corruption is the checksum.

Recall that any file on a computer is a series of binary numbers—in fact, it could be considered one very long binary number. One way to check if it has been changed (corrupted) would be to store an exact copy somewhere and then compare it. But because some files are megabytes, or even gigabytes, in size, this would be very inefficient. Instead, since the file is one large binary number, it is placed into an algorithm. This algorithm is like a mathematical function—it has a numeric solution. The checksum is the solution to this problem; it is calculated and appended to the file when it is saved. When the file is opened, the calculation is run again to see if it matches the number stored in the file. If it does, then the chances of it being uncorrupted are incredibly high. (There is a small chance that changes to a file might give the same checksum, but it is remote enough that the error checking is considered reliable.) Usually corruption in files is very minor and easy to detect in a checksum.

Figure 21: Checksum Examples



Source: Helix84, 2008.

Notice in this example the text is only changed slightly in the later texts, but the checksum is very different. Since the number is 10 digits, there are 10,000,000,000 different possibilities, making the random chance that two texts have the same checksum 1 in 10 billion.

Transmission Data Fault Tolerance

The problem of corrupted transmission of data has existed since before computer networks were invented. Radio transmission of secret codes by different militaries risked this problem, as did communication by telegram, and even a letter written by hand could be damaged by water or other environmental hazards.

Parity

One of the earliest methods of error-checking for binary data is called parity. Communication with telephone-based **modems** was serial, or one bit at a time. For every seven bits, an eighth parity bit was added for error checking. These basically worked like a very simple checksum—in this case we can even do the math in our heads. Communications via modem were set to either "even" or "odd" parity, which meant that for every 8 bits there must be an even or odd number of ones in the data. If it did not match, then it was assumed there was an error, and a request was made by the receiving computer for retransmission.

Modem
A portmanteau of "modulator-demodulator", this is a device used to translate one type of signal to another.

Table 3: Odd Parity

01101011	**1**1101001	**1**0011000	**0**1101011	**0**0000100

Source: Stephen Weese, 2020.

The parity bit is sent first in a group of eight bits, and its value must make the total number of ones in the byte odd. In the example above, you can see that the values are 5, 5, 3, 5, and 1 when the ones are tallied. This first bit is not counted as the primary data, it is used for error-checking and discarded, just like the checksum on the end of a data file. If a computer set to odd parity received 8 bits with an even number of ones, it would assume it was an error. Of course, if two bits happened to flip during transmission it could still pass parity. The assumption here is that the most common error is losing one bit in an 8-bit sequence. This also does not account for the possibility that the parity bit itself is corrupted, but this was used as a basic method of error-checking for many years in telecommunications.

Redundancy Check

A more advanced type of error-detection is the Cyclic Redundancy Check, or CRC. It works with a similar principle to the checksum. A block of data is processed through a mathematical algorithm and a result is appended to the data. After the data is sent, a calculation is repeated to check the integrity. The CRC can be applied to any length of binary data and will always return a code of the exact same length. This makes CRC a “hashing” algorithm, one that returns a value of consistent length in digits.

TCP/IP Error Detection and Correction

TCP/IP
This stands for Transmission Control Protocol/Internet Protocol, a specification for internet communications.

The main protocol, or set of rules, for internet communication is **TCP/IP.** This network standard has several layers of protection for data corruption. The first one is one we’re already familiar with: the checksum.

TCP/IP divides data into pieces to be sent over a network into segments or datagrams. These segments have a “header” at the beginning which defines the sending computer, the receiving computer, transmission settings, and also a 16-bit checksum to verify the data. The maximum size of this segment is 65,536 bytes (Stevens, 2004.) When each segment arrives at the destination computer it is checked against the checksum value; if the values don’t match, a retransmit request is sent to the source computer.

Along with a checksum, TCP/IP also checks for transmission errors on other levels. The receiving computer must also send an acknowledgement (ACK) for each datagram received. This is accomplished by giving each datagram a sequence number in the header. The recipient machine sends an ACK for each sequence number—if one of the numbers is not acknowledged after a certain period of time, the source computer will retransmit.

At another level, TCP/IP will also detect broken routes over the network and re-route to new ones. This is one reason why TCP/IP has been used for decades on the internet: it is fault-tolerant. Part of the network can stop working without taking the entire network offline. Routers and devices that stop forwarding network traffic can be worked around by checking for alternate routes to the destination.

SUMMARY

All data on computers is stored as binary. There must be a method of translating these numbers into useful human concepts. One of these is to give this data context. This means assigning the binary numbers a significance beyond their numeric value.

For images, that means pixels and colors. Documents are stored as symbols, punctuation, formatting, margins, and other settings. Audio is sampled digitally and saved as binary numbers to be reproduced as sound waves. A pre-formed code or set of rules is determined for each context to give the binary data an extra layer of meaning. This is done just as different letter combinations in English can be used to create words and paragraphs with more meaning that just their original symbology as individual letters.

In computer programming, all information must be given a specific data “type” which defines how the data will be used and their limitations. A good coder knows which variable types to use and how to name them properly for maximum code readability.

There are many standard data types. Some are numeric and used to represent integers and decimals. Others can represent letters, symbols, and strings of characters. We can group simple data types together into arrays to arrange and search them.

The volume of data that computers deal with necessitates some type of error checking. It is possible that during data storage or transmission some of the information can be corrupted. To detect this, various error checking systems have been created. Checksums and CRC take the binary data through a mathematical function that returns a specific value for that data stream. If the sending value matches the receiving value, then the data is assumed to be unchanged.

TCP/IP has various error-prevention technologies built in, including a checksum in the header of its data segments. It also has a system of acknowledgement, where the recipient computer confirms the receipt of data sent. If some data is not confirmed as received, or found to be corrupted, a retransmission request is sent to the source computer.

The more we understand how digital data is used by computers, the better we can utilize computing and code to create software applications.

UNIT 3

ALGORITHMS AND DATA STRUCTURES

STUDY GOALS

On completion of this unit, you will have learned …

- how the term algorithm is used in applications of computer science.
- how to create flowcharts.
- the details of data structures, such as arrays and stacks.
- how sorting algorithms work, including quick and bubble sort.
- how the quality of different algorithms is evaluated.

3. ALGORITHMS AND DATA STRUCTURES

Introduction

For the last 20 years, Google has been the main search engine used on the internet, accounting for over 90 percent of the market share internationally. (StatCounter.com, 2020.) But how does it work? A Google search works by a set of rules called an algorithm.

An algorithm is a set of rules or instructions for completing a task. This applies to human beings as well as computers. The vast amount of data that is available on the internet poses a challenge to anyone who would want to search through it. Google has established a method to search through this data with excellent results.

According to Google, there are actually several algorithms working together to achieve the results for an internet search. One of these is used to determine meaning: Google can substitute synonyms for words in its search such as “move” and “relocate” to find similar matches. Another algorithm is used to analyze relevance. The keywords typed in the search are matched to the page—the better the match, the higher the search results. The search algorithms for Google also incorporate a quality assessment of the website itself in terms of expertise, authoritativeness, and trustworthiness on a given topic (Google, n.d.).

This seems like a good start, but the algorithms go even further. The page will also be ranked for “usability”, which means how well the page displays in different browsers and how quickly the pages load. Finally, the algorithm includes personal information about the user that is searching. If that user lives in Barcelona and searches for “fútbol”, Google will prioritize local teams and FC Barcelona would be a higher result compared to results for someone in Berlin.

Of course, the exact details of Google’s algorithms are secret, like the secret spice recipe for Kentucky Fried Chicken. It is, after all, the most popular search engine in the world, and Google would not want anyone stealing their market share by implementing the same algorithms.

Is the algorithm perfect? No. New sites take a while to start to appear in search results. There is also a chance that a good site match for a particular search does not come up in the first page of searches. However, it is a good enough algorithm that it is the top choice of most web searchers around the world.

Often in data processing, there will be a data space that is extensively large, like the internet, that poses a challenge. Certain mathematical problems also have a vast number of solutions and can be difficult to manage. Often, it is impossible even for a computer to search through every possibility (at least, if quick results are desired) and an algorithm will be designed to give a “good” result quickly instead of a “perfect” result that takes much longer.

3.1 Algorithms and Flowcharts

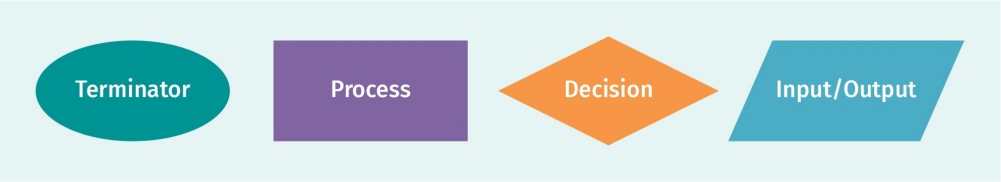
An algorithm at its most simple definition is a set of instructions. This works for computing applications as well as human beings. Let's say that you want to tell someone how to fix a broken lamp. You'd probably ask the usual questions like, "Is it plugged in?" and "Did you replace the light bulb?" However, to put these into a set of instructions, you'd have to organize things a bit more. You'd do something like this:

1. Check to see if the lamp is plugged in. If it isn't, plug it in. If it works, you're done. If not, go to step 2.
2. Check the lightbulb. Is it burned out? If so, replace it. If it works, you're done. If not, go to step 3.
3. The lamp is broken. Buy a new lamp.

Flowcharts (Flow Diagrams)

To translate human instructions into computer instructions, coders have used flowcharts for many years. These charts make it easy to observe the structure of an algorithm and create it with a programming language. They use standard symbols to indicate different types of instructions, as shown below.

Figure 22: Flowchart Symbols

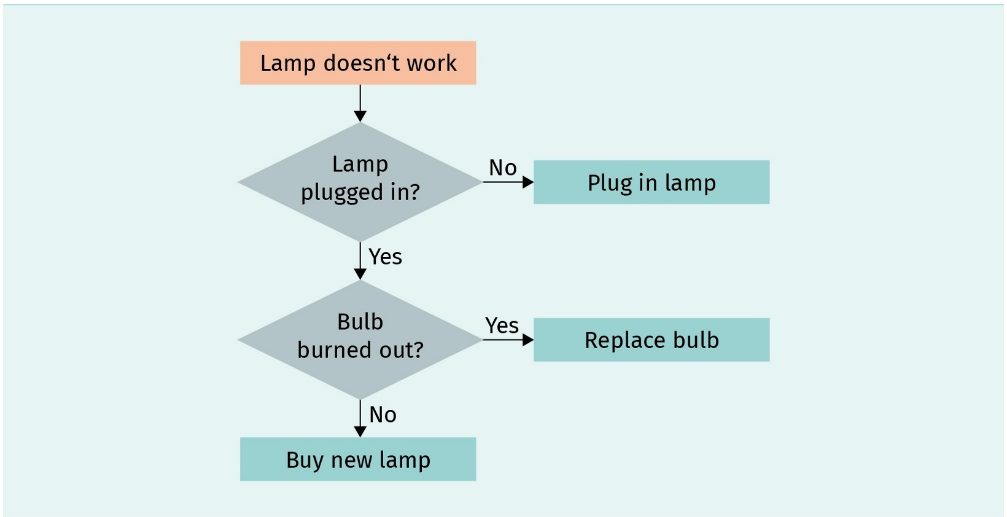


Source: Stephen Weese, 2020.

A terminator symbol (oval or rectangle) is used at the beginning of a flowchart and at all the endings. They represent the external nodes of flowcharts. A rectangle represents a process, for instance, some type of data calculation or filtering. The diamond is a decision —a decision has multiple pathways that can be taken. A parallelogram in a flow chart represents input or output, where data are sent into or out of the process. Finally, the arrows represent the flow from beginning to end of the sequence.

There are many other symbols that can be used in a flowchart, but these will suffice for now. If we return to our example about fixing a lamp, we can use a flowchart to illustrate that process.

Figure 23: Lamp Repair Flowchart

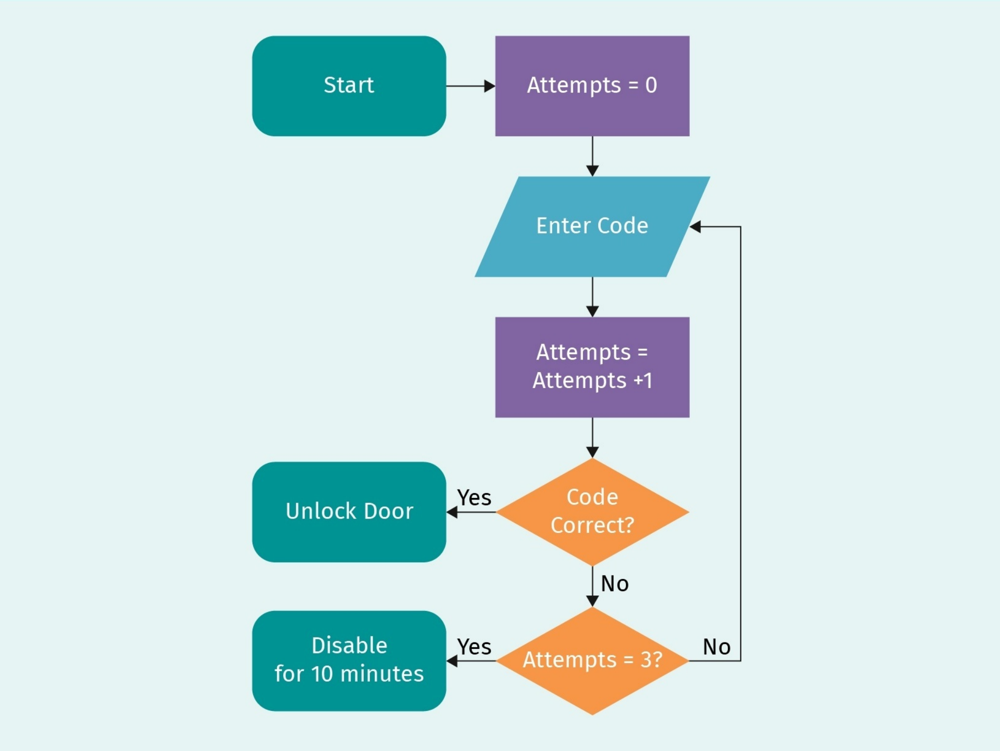


Source: Ilkant, 2006.

The flow starts at the beginning; from there we just follow the arrows along the pathway chosen. You can see that the beginning is a terminator with our starting point: “Lamp doesn’t work.” From there we go to a decision asking if the lamp is plugged in. There are “yes” or “no” options here. These represent binary thinking in computers where a “yes” can be symbolized by a one and a “no” by a zero.

Let’s take a look at a computer-based example of a flowchart. For this example, we will use a simple key code door lock. The lock works like this: the user has three attempts to enter the correct four-digit code to unlock the door; if they fail three times, the door will be locked for ten minutes and will not take any more input.

Figure 24: Door Lock Flowchart



Source: Stephen Weese, 2020.

This illustrates the algorithm for our door lock. At the beginning, when the lock is activated we move from start to the processing step. This is where the counter variable, attempts, is set to zero. Then the input step shows that the four-digit code is entered by the user. Then a decision is made: is the four-digit code correct? If so, the door is unlocked. So far, so good. If it's not correct, what happens next? It flows to the next decision. To a human, we would ask the question, "Is this the third failed attempt?" However, a computer is different. We have to use the variable "attempts" to count them. We check the value of attempts to see if it is "3." If it is, then the user is out of attempts and the door will lock for ten minutes. If they haven't reached three, the flow will loop back to where they enter the code again. Notice that every time they enter the code, the counter is incremented by one. This flowchart, which represents an algorithm, can be used as a guide to create programming instructions for a computer.

3.2 Simple Data Structures

Data are often entered into a computer sequentially, meaning in a specific order. Think of this process as similar to what you are doing now; you are reading letters and words in sequence. Simple data types such as "integer" and "char" will store one unit of information, but to store them in a sequence they must be arranged in a series. There are different ways to do this in computer memory and each has its advantages and drawbacks.

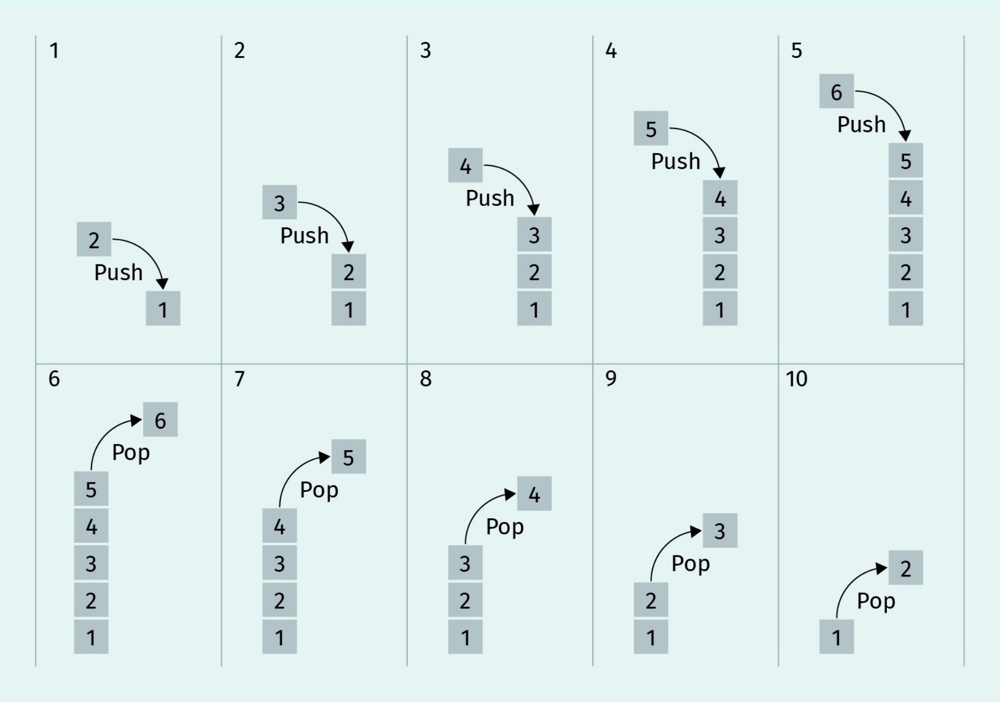
Stacks

A stack is a data structure that works under a specific philosophy: Last In, First Out. This is often abbreviated as LIFO. Imagine you are inputting data into a computer program—in this case, we will say it is a card game simulator. It will use a standard deck of 52 cards, and they will be randomized and placed into the stack.

Just like in real life, when you are playing cards, you can only draw the card from the top of the deck. That is the last card that was placed on top, so it is the last in, but the first out. Once the game is started, the order cannot be changed, the next card in play is always the one on top of the deck—or in our simulation, the stack.

A stack in computer programming allows for two basic operations: push and pop. Pop will take the top piece of data from the stack, which removes it. Push will add a new item to the top (and only the top) of the stack.

Figure 25: Using a Stack Data Structure



Source: Maxtremus, 2015.

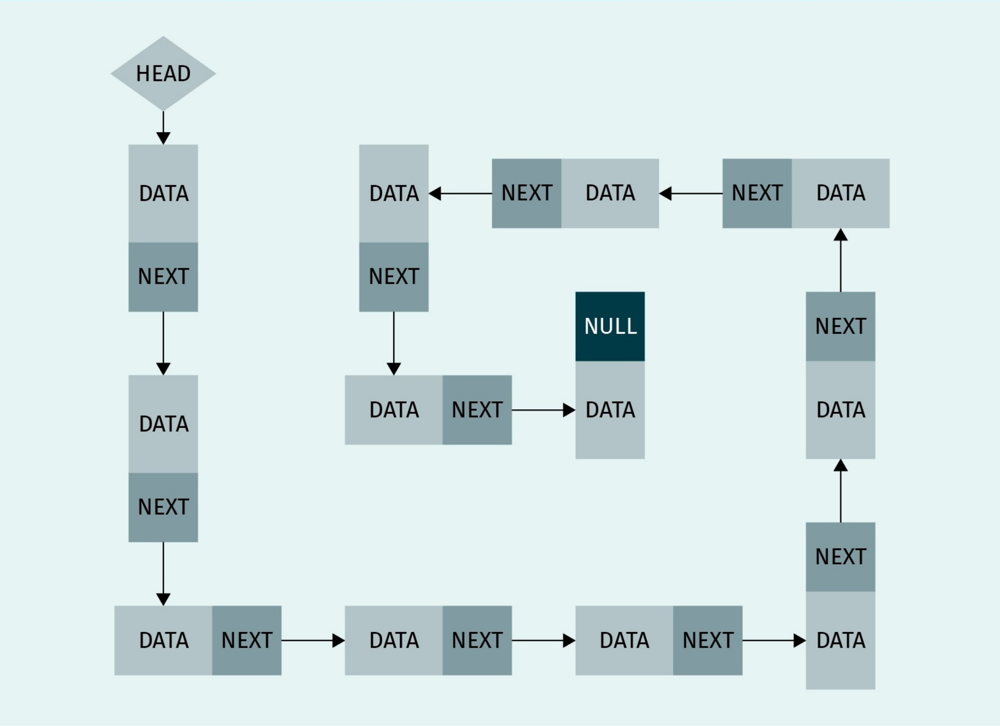
To populate the stack with data, we use the push operator, one element at a time. Once all of the data has been entered, it is read by using the pop operator. Note that the pop operation returns the value of the element and removes it from the stack. This is a very simple and efficient way to store data. Pushes and pops can be performed at any time to add or remove elements from the top of the stack. This structure has the advantage of taking up less space than more complicated data structures—however, it cannot be searched randomly.

RAM is called “Random Access Memory” because any part of it can be accessed at any time. This might seem obvious, but initially computer storage was only accessible in a sequence. This means that any time you wanted to read some data, you had to start at the very beginning until you got to the part you wanted. This is similar to reading a book. Last night you were on page 77 in the book. The next day you wanted to read it you’d have to start at page 1 and read your way all the way back to 77. Obviously, this isn’t very efficient. Random access means you can jump right to whatever page you want, in this case page 77. You could also skip ahead even to parts you haven’t read yet, if you wanted. Stack data structures can only be accessed sequentially, not randomly. Not only that, but the sequence is LIFO, giving you the last element first (imagine reading a book backwards!). There are times that this order might be useful (such as our deck of cards example) and times when it is not. Stacks are used because they take up less memory that other data structures. Basically this means that when you can use a stack for your purposes, you should.

Linked Lists

A linked list is also a sequential data storage structure. Instead of being stored in a simple stack of data, each element is linked to the next. In other words, an element in the data structure linked list contains two pieces of information, the data and the link.

Figure 26: Linked List Structure



Source: Vhcomptech, 2009.

Null
This is a special value used in computer code. It represents zero, nothing, or the end of a sequence, depending on the context.

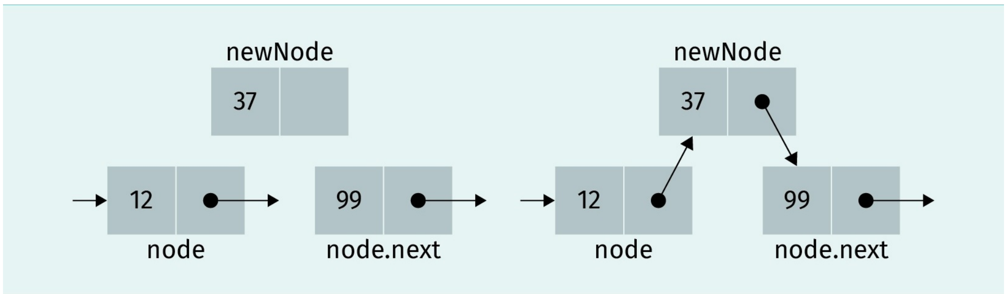
The last element in the linked list structure does not have a pointer to the next value, instead it points to **null**, which indicates that there are no more elements. Null is often used to end a series of data. The link in each list is a pointer, which is a memory location. It identifies the part of memory that contains the next element.

In many computer implementations, a linked list is unbounded. This means that there is not a specific number of elements that it is restricted to. (Of course, the size would have an upper limit based on the computer memory or the programming language structure.) This is useful when you are working with a series of data that you do not know the length of, including one that has a continually changing length.

One thing that linked lists can do that stacks cannot is insert data into the middle of the sequence. If you imagine the linked list as a chain, the chain could be temporarily broken and have a new link inserted, making it whole again. To do this with the data, we can simply change the pointers to include the new element.

Let’s say you have a linked list of integers.

Figure 27: Linked List Example



Source: Pluke, 2011.

The list is comprised of the elements 83, 23, 12, 99, 52, and 94. That means that the element containing the 12 also points to the next one containing 99. If we wanted to insert a new node with 37 (a link in the chain) into the list, we would simply have 12 point to 37, and then 37 point to 99 and now we once again have an unbroken chain.

Linked lists work well if you need speedy access to sequential data with the ability to insert new data into any part of the list.

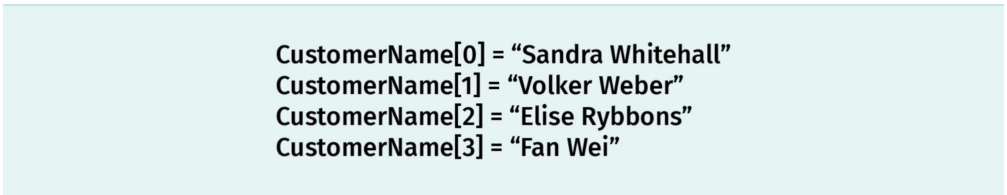
Arrays

One of the most useful and common data structures is the array. Though it takes up more memory than a stack, for instance, it provides random access for reading and writing data. Usually, when arrays are declared in a programming language, they are “bounded,” meaning that they have a set size. Linked lists offer similar functionality with an unbounded size. However, linked lists do not allow random access to elements. This means that, while arrays are more costly in size, they offer the most functionality of the sequential data structures.

The data in an array is arranged in elements, just like our other sequential types. However, each element in the array is given a specific index—a number that identifies it. The index is ordinal and starts at zero (in most programming languages.) This means that the first element is element zero. It may take a while to remember that computers start counting at zero, but there's a good reason for it. When you are working with small amounts of data and fail to use zero as a counter, you are basically wasting that piece of memory. So, to be more efficient, zero is used as the first number when counting.

Let's say you were going to store a list of customer names in an array. It would look something like this:

Figure 28: Customer Name Array



```
CustomerName[0] = "Sandra Whitehall"
CustomerName[1] = "Volker Weber"
CustomerName[2] = "Elise Rybbons"
CustomerName[3] = "Fan Wei"
```

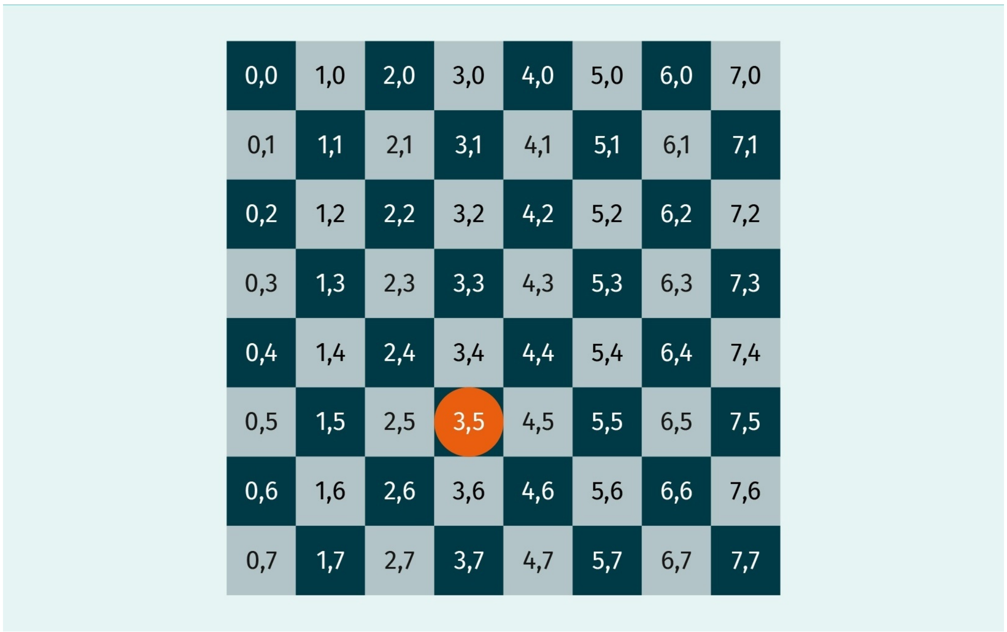
Source: Stephen Weese, 2020.

The array name is CustomerName, which will hold all of the elements. The first element is element zero [0] in the list. The number after the array name is called the index. The value stored in element zero is "Sandra Whitehall." The advantage of this array is that you can access any element immediately. If you want to read or write to element 3 (the fourth element), you can go directly to it without having to read 0, 1, and 2 first.

Multi-Dimensional Arrays

Another advantage of arrays is that they can be multi-dimensional. If you think of a list of items, for instance, you think of a linear progression from start to end—a line, or one dimension. Adding a second dimension means that the list can be read in two directions. One way to imagine this is to think of a checkerboard. A standard checkerboard is 8 by 8 squares, giving us a total of 64. If you are playing checkers, the squares can have three possibilities for their contents: a red checker, a black checker, or no checker. Let's store these as RED, BLACK, and NULL.

Figure 29: 2D Checkerboard Array



0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

Source: Stephen Weese, 2020.

The top left square is the first one at 0,0—remember computers start counting at zero. This means there is a row zero and a column zero on the board. This array references the squares similar to coordinates. The square at 3,5 for example means column 4, row 6 (remember to think like a computer!). Let's say we simply called this array "`checkerBoard`". If a red checker was in square 3,5, we could say:

```
checkerBoard[3,5] = "RED"
```

We could store information for the entire checkerboard using this array, where every square would have `RED`, `BLACK`, or `NULL` stored in it as data. Then we could implement algorithms to move the checker on the board.

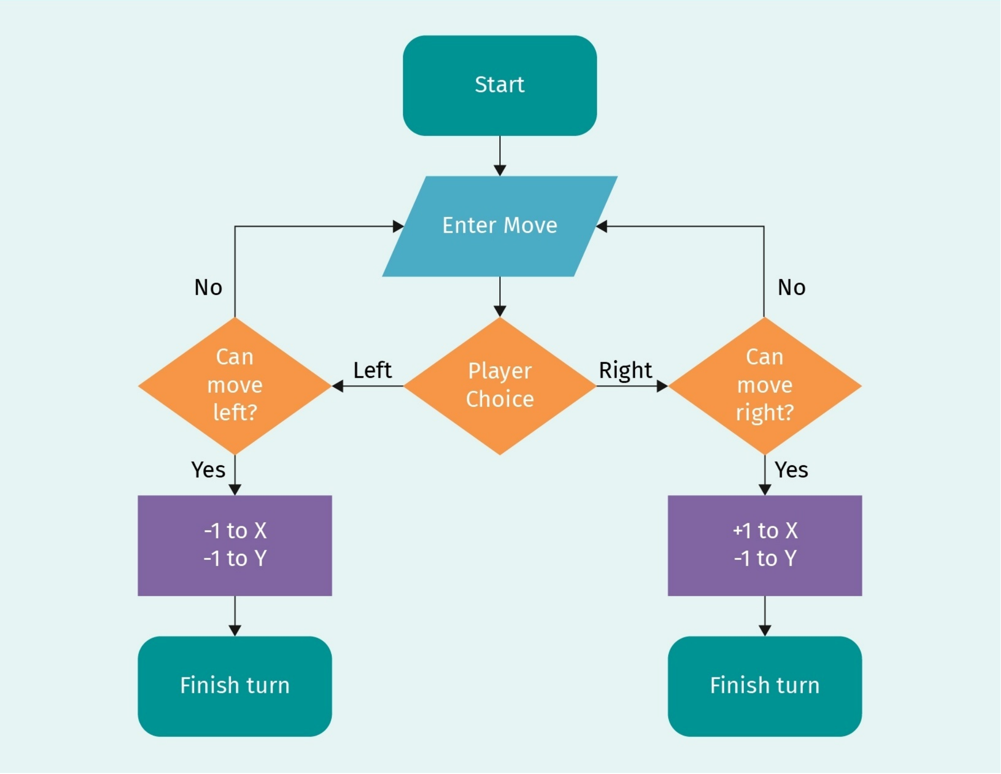
There are two types of moves, a regular move and a "capture" where you take one piece (or more) of your opponent's. The checkers must only be on the black squares and therefore must move diagonally. If we have a red checker on 3,5, and we are facing the opponent (we are on the bottom of the board), then the valid regular moves are to go to 4,4 or 2,4. The algorithm would look something like this:

Regular move

1. Player inputs move choice.
2. Can you move right?
3. Move right is +1 X and -1 Y (Where X and Y are array indices.)
4. Can you move left?
5. Move left is -1 X and -1 Y

You would have to test if you could move by checking to see if you are at the edge of the board, and also check whether there was an opposing piece in the way. In other words, if the destination number for X or Y is larger than 7 or less than 0 you are not on the board. If the space you want to move into contains a NULL, it is fine, but if it contains a RED or BLACK piece, you cannot move there.

Figure 30: Checkerboard Flowchart: Regular Move



Source: Stephen Weese, 2020.

Of course, this would just be an algorithm for using a regular move on one piece. There would be a larger algorithm for what happens during a player's turn, for example, which would include the rules for both a regular move and a capture move. Arrays make it easy to map out coordinates in two or three-dimensional space, like with checkers on a board.

More dimensions in arrays

There is no need to stop after only two dimensions, but keep in mind that for every dimension you add, you are raising the complexity of the data by an exponent. For instance, our checkerboard takes up 8 x 8 elements of space, for 8^2 or 64. If we made it a three-dimensional cube, 8 x 8 x 8, it would be 8^3 or 512 elements. As a programmer, you must be careful never to waste precious memory and only use what you need.

3.3 Searching and Sorting

Computers are very good at performing certain repetitive tasks—if they are programmed correctly. That can be a big “if.” It is important to realize that any mistake that is made in code can be amplified exponentially by the power of a computer. If your algorithm has a tiny flaw that makes it less efficient that flaw will be magnified as the task is repeated millions and billions of times by a computer.

The tasks of searching and sorting are both comparable to everyday tasks completed by humans. Humans search and sort objects, papers and things, whereas computers sort data. It is easy to compare these type of tasks to the human perspective to explain them. For instance, we search for things all the time. “Where did my keys go?” you may have asked for the hundredth time in your life. We try to find our favorite shirt, our wallet, our cup of coffee: “I swear it was just here a minute ago!” Usually there is a method to our searching for things. Sometimes we want to find something that is kept with other things of the same type. This is why we have things like a silverware drawer. We know that in this drawer, we keep our silverware. And what helps us find the fork? The silverware is sorted by type. So we also sort things and as it turns out, sorting things makes it easier for us to search for things. This is true for computers as well.

We began this unit with a discussion of Google search—the search of all searches in the biggest database ever made, the internet. Usually programmers don’t have such a daunting search space, they have a simple data set, something like a linked list or an array.

Gathering and Sorting Data

When data is input into a computer, the process depends on the type of data collected. It’s similar to real life—sometimes things come to you randomly, but sometimes they are already in order. Or, as things often go, they are in a kind of order, but not perfectly arranged. It may be that once data is input into a computer that it is happily already sorted, but most times this is not the case. We must do some additional processing of the data: a sort.

Let’s say you have a box full of kids' toys and you want to sort them in some kind of order. First you have to choose what the sort criteria will be. What will you use to sort these toys? You could sort them alphabetically by name, but what if the stuffed “bear” is also called “Peter”? Do you use “b” or “p”? It can be ambiguous. You could sort by size of the toy—but would you be using length or total volume? Sorting criteria must be exact. Even when sorting alphabetically decisions must be made—is there a different between capital and lowercase? Which is first?

Computers cannot work with ambiguity—everything must be exact. Fortunately, all data in a computer is represented in binary, and we can sort by the value of these binary numbers. The most common sorts, then, would be by numeric value or by alpha-numeric codes using one of the common computer encoding systems such as **Unicode.** Since each letter and symbol has a binary value attached to it, that value can be used to sort.

Unicode
An international encoding standard, Unicode is used for numbers, letters, and symbols.

Of course, the sort must be either ascending or descending; in other words, smallest to largest or vice versa. Once you have figured out your sorting method, it's time to sort.

Looking back at our box of toys, imagine what you would do first if we sorted by largest toy. Your eye would quickly scan all the toys for the one taking up the most space. Then you'd take it and put it first (or last) in the order. You are making order out of chaos. Each time when you pick the next toy, you will scan them all—and each time you do this, the number of toys left is smaller so it should take slightly less time. The last toy will take almost no time since it is the only one left and no visual scanning is required. Some steps will be easier since the next toy choice might be obvious, but some steps might take longer because two toys might be very close in size (or identical, which also requires a rule).

As you can see, sorting is a bit of a complex task that takes time and also must have a very specific set of criteria or rules. In fact, we could call our set of rules for searching an algorithm.

Searching for Data

Let's imagine a different kind of scenario. Today, we live in a very digital world, but not long ago everyone in the United States and many other countries had their own phone book. This enormous tome listed the phone numbers of everyone (with permission) in the local area, along with the numbers of businesses.

Figure 31: Old Phonebooks

Source: Gentle (2010).

Today, phonebooks are still to be found littering the landscape, an echo of the age of paper. However, phonebooks serve as a perfect example of sorting and searching. The list of all of the phone numbers was first sorted. Of course, the publishers could have chosen to sort by number, but most people were looking for the number and already knew the name. The sorting algorithm for a phone book was decided as the last name followed by the first name. Once all the names were sorted, the phone numbers and addresses were printed next to them in the book.

Once the books were delivered, a person searching for the name could open the book and look. Let's say the person in our example is Ken, and he is looking up Josh Rodriguez. How would it be best for him to search? You might say that he should open the book near the end, since "R" is farther down through the alphabet. However, there may be many names that start with "S" and so the bulk of the last part of the book would be those names. However, Ken decides to choose his starting point, he will then have to decide whether to go forward or backward in the book. If he opens about three quarters of the way through and sees the letter "S," he will go backwards. However, if he opens to "O," he knows he must go forward in the book. He also can search through one page at a time, looking at the letters, or simply skip a number of pages to see what letter he finds next.

Obviously, leafing through every page will take more time than jumping to a new page, but once you are close to the answer, leafing might become faster. It really depends on the data. We can pick a search algorithm for Ken and make it very simple. We tell him, "Ken, this is what you're going to do. You open the book halfway. Then, you decide to go forward or backward. Once you decide that, you will split the remaining pages in half and open to that page. You will then decide again and keep splitting the pages in half until you have arrived at Mr. Rodriguez." This is actually what we call a binary search, since it continually splits the data size into two.

There are many ways we could tell Ken, or a computer, to search a data set.

Search Algorithms

Let's take a look at some common searching rules. Some are more efficient than others. Many of them depend on the data itself, and data could be random, sorted, partially sorted, or in some other unusual order.

Linear search

This kind of search is pretty simple. It starts at the beginning and looks at each element in order. This would be like telling Ken to start at page one of the phonebook and read every single entry until he found Josh Rodriguez. If he was lucky, Josh would be near the beginning, but with a last name of "R" it seems unlikely.

Jump search

For this one, we need a little more information. Let's say the book is 2,500 pages long. We calculate the square root of n, the number of pages, which gives us 50, which we will use for the jump size, or m. Ken will first jump to page 50 and see if Josh is higher or lower, then keep jumping until he is past Josh. Then, he will go back to the previous jump and do a linear search (page by page) on the next 50 pages to find him. As you can see, this is definitely an improvement over a straight linear search.

Binary search

This is the one we have already described in our first example. Ken would start at page 1,250 of our 2,500-page book, and if Josh was higher in the list then jump half the remaining space, 625 pages, and keep splitting it until he arrived on the proper page.

Depending on the data itself, any of these techniques could be faster or slower than the others. The last two are also dependent on the data being sorted first. Since a linear search simply reads every entry until it finds the correct one, it will work on sorted or unsorted data lists.

Sorting Algorithms

We have seen that there is an advantage to searching data if it is first sorted. But how is the data sorted? There are also several different methods we can instruct a computer to use to organize our data.

If we go back to our toy box example, we can imagine a computer doing something similar. However, our brains work a bit differently. We can look at a collection of objects and pick out the largest one without pointing our eye at each one separately—it's one of the amazing things our brain does for us. However, a computer has to look at each element individually—it can't look at an entire array or linked list from "above" like we do to pick. For a computer it is more like looking through a phone book where the pages are hidden: it can't see them until it opens each one.

Bubble sort

One of the simplest types of sorting algorithms is the bubble sort. You can imagine how it works by thinking of the largest values "bubbling" to the top, like air bubbles in your drink. The rules for bubble sort are this: starting at the beginning, compare the first two values. If the first one is larger, move it to the second. If it is not, they are already in order. Then compare the next two values and do the same until you reach the end. Once this is done, do it again for every element except the last one, since it will be in order now. Each pass you complete will guarantee one more element is in order. Others may happen to be in order, but the bubble sort will check them all regardless. Let's use a simple array for our example.

(Kayli, Katie, John, Nathan, Anna)

As you can see it has five elements. The first step is to compare Kayli and Katie. Since "y" is after "t" they are out of order, so bubble search will switch them.

(Katie, Kayli, John, Nathan, Anna)

Then Kayli and John are compared. Kayli comes after John, so they are switched.

(Katie, John, Kayli, Nathan, Anna)

Kayli and Nathan are compared and found to be in the correct order, so the list is unchanged for this step.

(Katie, John, Kayli, Nathan, Anna)

Finally, Nathan and Anna are compared, and Nathan is switched with Anna.

(Katie, John, Kayli, Anna, Nathan)

With this method we now know that the last element, "Nathan" is in its proper place. Now we must compare the remaining four elements. After that, then the remaining three, then the last two. A regular bubble sort will always take the same number of steps based on the list size. However, a modified bubble sort could be instructed to stop if the list is sorted just by detecting if there were no swaps made. If no swaps are needed, the list is already in order. You can see that the number of comparisons in each pass is n-1, where n is the array size. In this case there were four comparisons on five elements.

Binary insertion sort

The binary insertion sort is based on the same method we used for a binary search—by splitting the data in half repeatedly. To sort the data, it builds a new output list and takes items from the input list. The input list are items yet to be sorted, and the output are the ones already sorted. Imagine you're sorting your toys in the toy box. First you dump them all on the floor—that's your input list. Then you place them back in the box one at a time in order, that's the output list. This example will make more sense with a larger list. Let's use numerical data this time and assume that the first five elements have already been sorted. So far, we have:

Input (66, 3, 23, 76, 27, 1, 11): yet to be sorted

Output (13, 35, 37, 44, 56): sorted

The original list, then, had 12 elements. We now reach down for the next toy or element in the list, 66. Remember a computer can't see a top down view like we do of all the items, it needs to check them individually. However, we can speed this up by splitting the ordered list in half, just like Ken did with the phonebook. The computer looks at the middle element out of the five, which is the third element, 37. 66 is higher, so now it jumps to the top half of the data. It then splits the difference again and looks at the fourth element, which is 44. 66 is larger, so it splits the two remaining into a space of one—now we are at the last comparison. 66 is larger than 56 so it is added to the end of the list. We now have:

Input (3, 23, 76, 27, 1, 11)

Output (13, 35, 37, 44, 56, 66)

This took three comparisons for a five-element list. A bubble sort would take four comparisons for each pass, so generally speaking, this algorithm is an improvement.

The next step would continue with the input of 3. Since there are an even number of elements (six) we have to split the middle with either the third or fourth. Let's say the algorithm chooses the third. We compare 3 to 37. It is lower, so we split the remaining space and compare it to 35. 3 is lower again and we compare it to 13. It is lowest again, so 3 is placed at the beginning of the list. The advantage of the binary insertion sort is that it can at the very least eliminate comparing to half of the data elements in the list in the first step. Now we have:

Input (23, 76, 27, 1, 11)

Output (3, 13, 35, 37, 44, 56, 66)

This process would continue as each element is added to the output list in order, so when the last element is added, the list will be sorted.

Quicksort

The performance of search algorithms depends on the data set, but Quicksort is regarded as one of the most efficient algorithms overall. Quicksort is more complex than the previous algorithms, but the explanation of how it works is simple. Imagine a teacher with a stack of graded papers. The teacher wants to sort all of the papers by grade. The pile is split into all the grades higher than 55 (this number is the pivot and can be chosen by different methods) and those less than 55. Then the two piles are split around another pivot bringing the total number of piles to four. The teacher then sorts each pile separately and when they are done, can simply put all the piles back together in order from lowest pile to highest. It can be thought of as a "divide and conquer" type of method. In this example, the stack was split into four piles but, in reality, the data determines how many stacks it is split into for sorting.

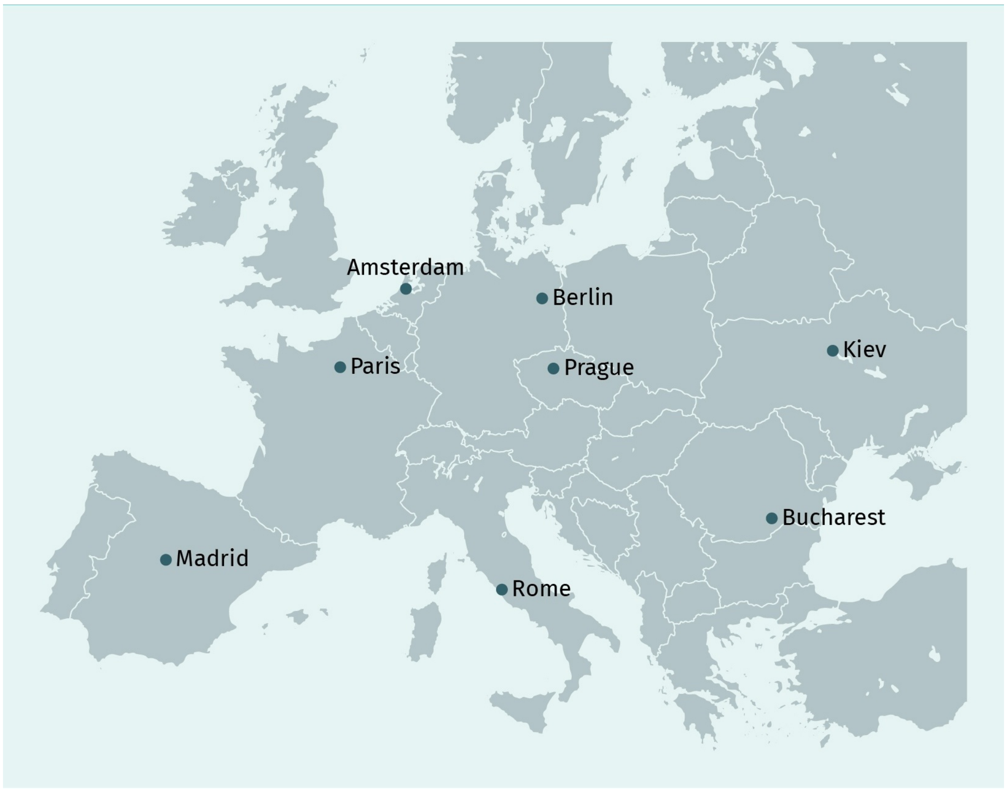
3.4 Quality of Algorithms

There is much to be said about algorithms—enough to fill an entire course. This unit discussion serves as an overview that will facilitate further study in the future. We have mostly examined searching and sorting algorithms here, however there are an infinite number of possible algorithms for an infinite number of potential tasks. Even within searching and sorting, there are an infinite amount of different kinds of data sets. Some data sets are smaller and static, others are multi-dimensional and dynamic. One illustration of a more complex data set used often is the Traveling Salesman Problem (also known as the Traveling Salesperson Problem or TSP). In fact, there is an entire book published exploring the many different challenges and permutations of this problem, titled *The Traveling Salesman Problem: A Computational Study* (Applegate et al., 2007).

The Traveling Salesman Problem

Imagine a salesperson, let's say, of computer parts, who must travel to a series of cities all over Europe to make sales presentations. Each city is a different distance from the other. What is the route that is the shortest? Obviously, the salesperson does not want to fly more miles than necessary and waste money, fuel, and time.

Figure 32: Traveling Salesman Map



Source: Stephen Weese, 2020 based on San Jose, 2006.

Our salesperson, in this case, is Alexa. She has to travel to the eight European capitals shown in this map. She could look at the map and do some calculations, of course, but this is where computers are excellent problem solvers. The route Alexa takes will have to comprise all eight cities. If she starts in Madrid, she then has seven choices after that. If she chooses Rome second, there are then six cities left. Thus, the number of routes is:

$$8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 8!$$

Eight factorial (!) is the number of possible routes, which is calculated as 40,320 routes. This would be a lot of work for Alexa to do herself if she wanted to find the best route. This problem involves more than just a simple search for a solution: each route choice must be added together to get the distance between all the cities. This does involve a significant amount of processor time, at least, when the data size grows. 40,320 x 7 distance calculations (between each city) probably doesn't intimidate a modern computer much, but what happens if the problem grows to thirteen cities? This is 13 factorial (13!) and comes out to: 6,227,020,800 routes, multiplied by the 12 in-between distance calculations for each route. The data space is growing quickly. In fact, factorial growth passes exponential growth after a certain point, depending on the data set.

Huge data spaces

Some data spaces are so large that an **exhaustive search** of the data space will simply cost too much time. Instead, programmers create an algorithm that will create a "good" solution as opposed to an optimal solution. With thirteen cities our salesman problem is up to six billion routes—increase it to sixteen cities, and now we have twenty trillion solutions. At this point, creating an algorithm to find a good solution for this problem is a trade-off between time and quality.

Exhaustive search
Sometimes referred to as "brute force," this is a search algorithm that looks through the entire data set.

Algorithm accuracy

Some algorithms are required to give the optimal solution. For instance, if we are sorting data, the algorithm cannot stop until everything is sorted with 100 percent accuracy. If we are searching for a specific data point, it must be found. However, if we are trying to find a good solution to a complex mathematical problem, a standard of "good enough" is applied by the programmer to the situation.

Correct algorithms

Any algorithm that is used must be mathematically correct. This one of the many reasons that a study of computer science also requires an in-depth study of mathematics. Of course, many types of programming algorithms have been implemented previously, so doing diligent research into the formulas for these algorithms will also yield an appropriate answer. As mentioned before, if an algorithm is incorrect, the effect of the flaw could be magnified exponentially.

Algorithm termination

An algorithm must know when to stop its calculations. A good programmer will terminate the algorithm at the first moment the task is complete without wasting any processor time on the computer. As an example, let's look back at bubble sort. A standard bubble sort will go through the data set with n-1 passes, where n is the number of elements. However, there is a chance that the data might be sorted before all passes are completed. All a programmer would have to do is put in a simple check to see if there were no elements swapped during the last pass—if no swaps were needed, the data is already in order and no further passes are required.

Algorithm Complexity

When we are evaluating the usefulness of algorithms, one thing we ask is: which one requires more operations to complete its task? That algorithm is the most complex. Of course, we want to use the least complex algorithm that will take fewer calculations and therefore less time and energy. Some algorithms have an exact pattern and will always perform a predictable number of steps. Others, like quick sort, have a variable number of steps that is determined by the data itself.

“Big O” notation

In computer science, the “Big O” function demonstrates the mathematical complexity of an algorithm, which is based on the data size of “n” elements. For instance, bubble sort takes approximately n^2 steps to be completed, so we write it as:

$$\mathrm{O}(\mathrm{n}^2)$$

However, n^2 isn’t very efficient compared to other sorting algorithms. The less complex an algorithm is, the more efficient it tends to be.

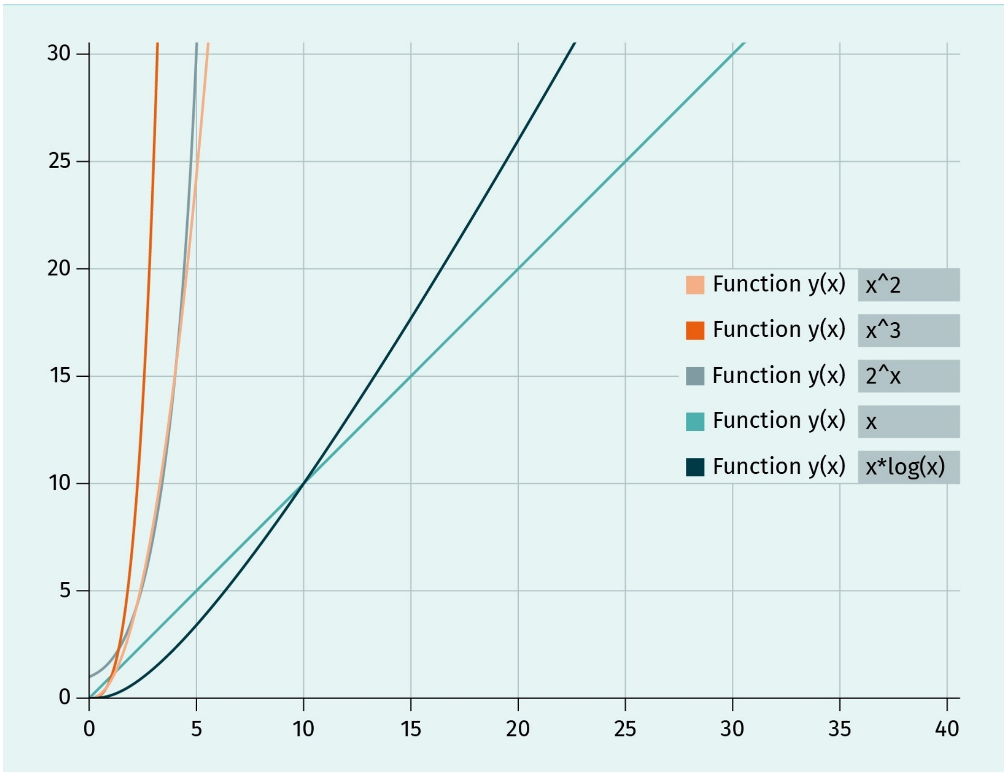
Table 4: Algorithm Complexity

Name	Complexity	Details
Constant	O(1)	This is an exact number of steps, no matter what the data size. The number in parentheses is the exact number of steps.
Linear	O(n)	This takes exactly or close to the number of steps as the size of the data set.
N*log(n)	O(n*log(n))	This is n times the logarithm of n. It is close to the same efficiency as linear for many data sets (see graph). Quick sort falls into this category.
Quadratic	O(n^2)	The number of steps is approximately the square of the data set size. Bubble sort and binary Insertion sort fall into this category.
Cubic	O(n^3)	The steps for algorithm are the data size cubed.
Exponential	O(n^k)	This includes any n to an exponent above 3, when a number is raised to the power of n, and factorials.

Source: Stephen Weese, 2020.

Exponential algorithms are very inefficient and only useful for problems with small data sets and become quickly unusable as data sizes grow—this means a better solution should be found. Cubic is similar.

Figure 33: Complexity Graph



Source: Venkatraman, 2020.

Any algorithm with an exponent in the complexity graph will grow in computation time very quickly as the data set grows.

SUMMARY

Humans and computers share the idea of an algorithm, which is a series of steps used to perform a task. Computers are ideally suited to perform complex and repetitive tasks and often can perform them at incredible speeds compared to a person.

One way to illustrate the process of an algorithm is to create a flowchart (or flow diagram). This chart identifies the path of decision-making that occurs when completing a specific task. It could be as simple as changing a light bulb or as complex as finding the shortest route between thirteen cities.

To organize collections of data, we have various data structures. A stack is a simple type of structure that follows the LIFO structure: last in, first out. A more complex structure is a linked list, which has the advantage of being expandable and allowing simple data insertion. Arrays are complex and can be multi-dimensional; they refer to elements in the structure by an index number.

Data that is stored in one of these collections may need to be sorted so it can be searched. We explored different search algorithms such as bubble sort, binary insertion sort, and quick sort. These sorting systems are rules to tell a computer how to process the sorting step-by-step. Once data is sorted, it is much more easily searched. An example would be an alphabetical phonebook where the general area of a listing can be predicted.

Algorithms must be mathematically and semantically accurate to produce the correct results. They must also not perform extra unnecessary operations which waste time. Some algorithms are optimal and find the best solution, but with large data spaces algorithms are often required to find a “good” solution without searching every possibility in an astronomically sized data set.

Measuring algorithm complexity is done by using the “Big O” notation. If an algorithm takes approximately n * n steps (where n is the number of data elements) then it is written as $O(n^2)$ complexity. Complexity is very important as algorithms can quickly become inefficient as data size grows.

UNIT 4

PROPOSITIONAL LOGIC, BOOLEAN ALGEBRA AND CIRCUIT DESIGN

STUDY GOALS

On completion of this unit, you will have learned …

- the basic language and concepts of propositional logic.
- how to create truth tables.
- how to use the conjunctive and disjunctive normal form.
- about the basic concepts of digital circuits and logic gates.

4. PROPOSITIONAL LOGIC, BOOLEAN ALGEBRA AND CIRCUIT DESIGN

Introduction

In the 1986 film *Labyrinth*, starring David Bowie, Sarah, the protagonist, is trying to get to the center of a labyrinth to save her younger brother, taken captive by the Goblin King (Bowie). At one point she is given the opportunity to take one of a set of two doors. In front of each door is a guard. The guards tell her this: One of these doors leads to the castle at the center of the maze, and the other one leads to certain death. They also say that one of the guards always answers questions with the truth, and the other one always lies.

This doesn't seem too difficult, eventually through asking many questions one could probably figure out which one lies, and then ask the truthful one about the door. However, there is a catch: you can only ask the guards one question. Whatever this question is, it must answer definitively which door is which.

Ultimately it comes down to logic—what is true or false? In this situation there is little room for subjectivity, it is a binary choice: one door or the other. And each guard always lies or tells the truth. Since these rules are established it puts neat little boxes around the choices; they are discrete with no fuzziness. That means we can apply rules of logic to it. Sarah (who, apparently, was paying attention in her logic classes) comes up with a solution to the problem. She asks the first guard, "Would he (the second guard) tell me that this door (the first door) leads to the castle?" The guard thinks for a while and finally answers "yes". Sarah then says that the other door leads to the castle. The guards are amazed and ask how it is true. There are only two possibilities, either the first guard is the liar or he is the honest one.

If the first guard is a liar, then:

He would lie about what the truthful guard would say. So the answer would be the opposite of the truth.

If the first guard tells the truth, then:

He would truthfully tell the false answer that the lying guard would say. So the answer would also be the opposite of the truth.

No matter which door is correct, by phrasing the question this way, Sarah is guaranteeing that she will get the opposite of the correct answer, which is good enough for her purposes. She is able to take the second door and continue toward the castle.

In this unit, we will learn how to apply this kind of logic to statements in common language and also computer language. Computers can be instructed to think and make logical decisions, but it all depends on the correct input from the human creators. If Sarah was wrong about her logic, her choice would have resulted in certain death. Fortunately, she created her logical statements properly and solved the riddle.

4.1 Propositions and Logical Conclusions

To understand how logic works, we must first establish definitions of terms and what the applicable rules are. First, we will start with a statement. A statement is a phrase that says that something "is" or something "is not." That means it can be true or false. It is different from the idea of a sentence. A sentence may or may not contain a statement. "Come over here now!" is a command that is not stating anything about the universe. "I only have five dollars" is a statement about the universe, declaring that the speaker only has five dollars. However, in reality, it could be true or false. The person speaking could be intentionally lying or could just be unaware of the actual amount of money they possess. Logical statements must be objective to be evaluated for truth. If Austin says "It is very hot outside," that is subjective and ambiguous. What is hot? At what temperature does it fall within the range of "very hot?" However, if Austin claims that "It is more than 27 degrees Celsius outside," then we have a statement that can be evaluated for truth (Rautenberg, 2010).

A proposition is the idea behind a statement. For instance, saying "It is more than 27 degrees Celsius" is a statement, but the proposition behind it is that the conditions of the environment include that the temperature is more than 27 degrees Celsius at the current moment outdoors. There are many statements that can be used for the same proposition. "Right now, it is hotter than 27 degrees Celsius" would be a different statement that is saying the same thing. Technically, when we evaluate logic, we are evaluating truthfulness of propositions. The real power of proposition logic is seen when different propositions are combined and evaluated together.

And, Or, and Not

When we speak to others, we can use the simple word "not" to indicate negation. "It is NOT more than 27 degrees outside" is the opposite of the statement Austin made before. In other words, it is the negative form of the proposition. In a true/false evaluation, this means that using not will change true to false, and false to true.

We can also combine statements together using the connectives "and" and "or". For instance, Darby might say: "It is colder than yesterday and it is raining."

This is a combined proposition containing two ideas: one, that it is colder than yesterday, and two, that it is raining. The way it is phrased using "and" means that for the sentence to be true, both propositions must be true. If it is cold and not raining, it's not true. If it's raining but not cold, again, untrue. If neither is true you will very much think Darby has a problem with lying or is confused. However, if it truly is cold AND raining outside, then and only then is the full sentence true.

We can also use the “or” connective to evaluate two propositions together: “It is colder than yesterday or it is raining outside.”

In this case, to be correct, only one of the propositions needs to be true for the sentence to be true, since it was qualified by the “or” connective.

Conjunction, Disjunction, and Implication

In propositional logic, sentences are translated into symbols that show how propositions relate to each other. We call the “and” relation a conjunction, and the “or” relation a disjunction. The previous sentence, “It is cold out and it is raining,” is written thusly:

$$\mathrm{p} \wedge \mathrm{q}$$

where p is the proposition “it is colder than yesterday” and q is “it is raining.” The $\wedge$ symbol denotes that p is conjoined with q. (A way to remember conjunct is that the symbol resembles an “A” as in “and”.) If we say that it is “colder than yesterday or it is raining,” then we have a disjunction. This would look like

$$\mathrm{p} \vee \mathrm{q}$$

Finally, implication means that a proposition being true would also mean another proposition is true. For instance, “If it is raining outside (p), then I should bring my umbrella (q).” Implication is saying something only if the first part is true; if it is not raining outside, then the full sentence does not imply anything. This is written as follows:

$$\mathrm{p} \Rightarrow \mathrm{q}$$

A common mistake many people make is confusing the terms “imply” and “infer.” Imply is something the information does, it points toward a conclusion. Inferring is noting or observing a conclusion that has been implied. For instance, if Sofia says to Liam, “If someone doesn’t hug me in thirty seconds I will scream,” Sofia is implying that Liam should hug her. If Liam figures this out, then he has inferred that she wants him to hug her and if he doesn’t, he had better cover his ears.

In propositional logic (PL) notation, negation (or “not”) also has a symbol. The negative of proposition “p” is written like this:

$$\neg \mathrm{p}$$

Truth Tables

Now that we have basic symbols and terms for logic, we can start to make some evaluations. Truth tables are a structured way of displaying true and false values for propositions. Let’s make one based on our previous compound proposition: “It is colder than yesterday (p) and it is raining (q).”

Table 5: Conjunctive Truth Table

p	**q**	**p ∧ q**
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

Source: Stephen Weese, 2020.

The only way the conjunction of p and q is true is when both propositions are true. What if we have “It is colder than yesterday or it is raining outside” in a truth table? Now it is a disjunction.

Table 6: Disjunctive Truth Table

p	**q**	**p ∨ q**
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

Source: Stephen Weese, 2020.

It might seem strange that the sentence “It is colder than yesterday, or it is raining outside” is true if both are true, but that is the way propositional logic sees it. As long as ONE of the parts is true, then the disjunction is true. If both parts are true, then one is still true. Though the English language might have a slightly different use of “or”, this is how it is used in PL. (The **exclusive disjunction** is more like “or” in standard English.)

Exclusive disjunction
Also known as an XOR logic gate, this indicates that only one out of two propositions is true.

In the case of implication, we can look at another example. A father says to his daughter, “If you score a goal today in the match (p), I will buy you candy (q).” What are the scenarios that make this implication true? These are the ones where the father keeps his word to the daughter.

Table 7: Implication Truth Table

p	**q**	**p ⇒ q**
T	T	**T**
T	F	**F**
F	T	**T**

p	q	$p \Rightarrow q$
F	F	**T**

Source: Stephen Weese, 2020.

So, if the daughter scores a goal (T), she gets candy (T), so it’s true. If she scores a goal (T) and he does not give her candy (F), then he is not keeping his word. In the last two situations where she does not score a goal, the father is still true to his word whether he gives her candy or not, so they are both true.

Review

A statement is a sentence that makes a claim as to whether something is or is not. The idea behind the statement is the proposition. A proposition can be true or false. Propositions can be combined together to form more complex propositions with connectives. The most common connectives include “and”, “or”, and implication. Truth tables are ways to represent the evaluation of a proposition depending on the truth of the component elements. True or false is a binary proposition; it can also be represented by 1 for true and 0 for false in a computer.

Table 8: Logic Symbology and Terminology

Name	Symbol	Description
Conjunction (AND)	$\wedge$	This operator connects two propositions; both of them must be true for the result to be true.
Disjunction (OR)	$\vee$	This operator connects two propositions; at least one of them must be true for the result to be true.
Implication	$\Rightarrow$	Implication means that if the first proposition is true, the second one should be true as well (this often uses an “If…then” structure).
Negation (NOT)	$\neg$	This negates the proposition. If it is true, the result is false. If it is false, the result is true.
Propositions	p, q, r, etc.	A proposition is a specific statement about reality. It is usually represented by letters starting at “p.”

Source: Stephen Weese, 2020.

4.2 Conjunctive and Disjunctive Normal Form

In computer science, propositional logic and truth tables are used to evaluate computer code. In digital electronics, the same logic is used to design circuits and computer chips. To make logical propositions easier to implement in computer code, they are often written in these normal forms: conjunctive and disjunctive.

Conjunctive Normal Form

We have learned that a logical conjunction is described as using a logical "AND" operator. To explain conjunctive normal form, a few more terms must be defined. Our propositions from before will be more generalized into literals. A literal is any element that can be evaluated as true or false. This includes negation. A system that functions using true and false values is referred to as a Boolean system, named after **George Boole**. Literals used in normal forms are either of the form p or ¬p, using any of the variables desired to represent Boolean values.

George Boole
A nineteenth-century mathematician and philosopher, Boole was the creator of Boolean logic.

The definition of conjunctive normal form is a logic formula that is a single conjunction of any number of disjunctions. For example

$$(\mathrm{p} \vee \neg \mathrm{q}) \wedge (\mathrm{r} \vee \mathrm{s})$$

There can be any number of disjunctions as long as they are joined by a conjunction. Technically, a proposition or negation on its own also qualifies. Meaning that p or ¬p is also in conjunctive normal form in the trivial case. To evaluate the results of the logic formula above, we can create a truth table. Since there are four variables (or literals) in this formula, it means there are 2^4 different possibilities for all of the input values of true and false; our truth table will have to be much larger than before. In the table we are using ¬q instead of q; this means that q has already been negated. To create the table, we first fill in all of the possible value combinations for p, ¬q, r and s.

It is essential to note the ordering rules in order to solve every logic formula. First, apply the negations; second, consider the expressions in the parentheses; third, resolve the conjunctions. Last, resolve the disjunctions. So, now you may try and answer the following question. What would the truth table look like if the parentheses were omitted from the given logic formula?

Figure 34: Conjunctive Normal Form Truth Table

p	¬q	r	s	p ∨ ¬q	r ∨ s	$(p \vee \neg q) \wedge (r \vee s)$
T	T	T	T	T	T	**T**
T	T	T	F	T	T	**T**
T	T	F	T	T	T	**T**
T	T	F	F	T	F	**F**
T	F	T	T	T	T	**T**
T	F	T	F	T	T	**T**
T	F	F	T	T	T	**T**
T	F	F	F	T	F	**F**
F	T	T	T	T	T	**T**
F	T	T	F	T	T	**T**
F	T	F	T	T	T	**T**
F	T	F	F	T	F	**F**
F	F	T	T	F	T	**F**
F	F	T	F	F	T	**F**
F	F	F	T	F	T	**F**
F	F	F	F	F	F	**F**

Source: Stephen Weese, 2020.

Evaluating p V ¬q gives a value of true when at least one of these literals is true. Another way to say it is as long as both values are not false, then the disjunction is true. If we look at the column for p V ¬q, we see that twelve of the sixteen evaluations are true. Evaluating r V s is similar—it is true as long as one or both of the literals are true. Again, there are twelve true values. Finally, the results of the two disjunctions are conjoined (think of our AND example), and the full statement is true only if both p V ¬q and r V s are true.

Boolean Decision Making

If we think of this as computer code, all four of the literals are inputs. The computer inputs these values into a logical algorithm, and the result is returned as true or false. This is very important in **branching code**. The computer code will decide which code to run next based on the true or false result of the formula. For instance, these types of procedures might happen:

Branching code
This is computer code that causes a computer to choose to execute a specific branch of instructions, while passing over others.

PROPOSITION: There is more than X money in the bank account (p), and the user is properly authenticated (q).

TRUE: Withdraw X money.

FALSE: Send insufficient funds message.

This would be represented as p ∧ q, which is in simple conjunctive normal form. Depending on the Boolean evaluation, the computer will take different actions. If both conditions are met, the money can be withdrawn; otherwise, the user will get an error message.

Disjunctive Normal Form

The other normal form, disjunctive, is one where any number of conjunctions are connected by a disjunction. In other words, something like this:

$$(\mathrm{p} \land \mathrm{q}) \lor (\neg \mathrm{r} \land \neg \mathrm{s})$$

This could be read as “p and q or not r and not s.” This is two conjunctions connected by a disjunction, and the parentheses tell us which to evaluate first. This can also be evaluated by a truth table with sixteen entries. First each conjunction is evaluated, then the disjunction. Following the ordering rules required to solve every logic formula, are the parenthesis required in the above logic formula?

Figure 35: Disjunctive Normal Form Truth Table

p	q	¬r	¬s	p ∧ q	¬r ∧ ¬s	$(p \land \neg q) \lor (\neg r \land \neg s)$
T	T	T	T	T	T	**T**
T	T	T	F	T	F	**T**
T	T	F	T	T	F	**T**
T	T	F	F	T	F	**T**
T	F	T	T	F	T	**T**
T	F	T	F	F	F	**F**
T	F	F	T	F	F	**F**
T	F	F	F	F	F	**F**
F	T	T	T	F	T	**T**
F	T	T	F	F	F	**F**
F	T	F	T	F	F	**F**
F	T	F	F	F	F	**F**
F	F	T	T	F	T	**T**
F	F	T	F	F	F	**F**
F	F	F	T	F	F	**F**
F	F	F	F	F	F	**F**

Source: Stephen Weese, 2020.

First, p ∧ q is evaluated true only if both p and q are true. Interestingly, ¬r ∧ ¬s is true only if r and s are false, since they are both negated. In the table, they are written as ¬r and ¬s, since they are “not r” and “not s.” Once those propositions are evaluated then they are disjoined—if one of them is true, then it will result in a true for the disjunction.

Satisfiability

When writing a logic formula in normal form, it is only useful to a computer program if there is at least one outcome that evaluates to true. This means that given the different inputs to the literal values at least one of the combinations must end up with a true result. To test a logical formula, it must be evaluated with each combination—once a “true” result is found, it is proven to be satisfiable. You may have noticed that the more literals a formula has, the larger the possibilities. In fact, it is calculated by 2^n, meaning that a formula with six literals would have 64 possibilities, and one with ten would have 1,024 possibilities. Writing out that truth table would be problematic. However, computer programs can be written to evaluate the formula to see if it is satisfiable. That means a computer would be telling you whether you should put this formula in your computer code. Fortunately, there are online tools that do exactly that. Stanford has an online truth table generator that can generate quite a large table of entries. (There are others easily found through a Google search.) You can test them with this formula:

$$(\mathrm{p} \wedge \mathrm{q}) \vee (\neg \mathrm{r} \wedge \mathrm{s}) \vee \mathrm{z} \vee (\mathrm{t} \wedge \neg \mathrm{q})$$

As you can see, there are six unique literals (not q does not need to be listed in the table since it is the implied opposite of q.) This means the table will have 64 entries. Feel free to come up with your own formulas to put into the generator. If a logic formula has at least one “true” in the answer, then it is satisfiable. This formula has several “true” evaluations.

A simple unsatisfiable formula is $p \wedge \neg p$. You can evaluate this one yourself by making a truth table or by pasting it into the generator. Since a conjunction requires both values to be true, this formula will never be satisfiable. One or the other values of p will always be false.

4.3 Digital Circuit Design

The subject of digital circuit design would normally encompass an entire class or even a major. As a programmer, it is extremely helpful to understand how processors and circuits work since these are the specific entities that you will be controlling. It also gives an excellent introduction into how computers “think.” When a coder is able to think more like a computer, their code becomes more streamlined. This section will serve as an overview of the terminology and concepts that are involved with this discipline.

All of the previous discussion of logic formulas directly relates to digital circuit design. It takes only a little bit of translation to change “true” to a binary 1 and “false” to binary 0. All of the operators learned in propositional logic (PL) have an equivalent in digital circuits. It is one of the main reasons to learn PL: if you understand logic, you understand basic circuit design.

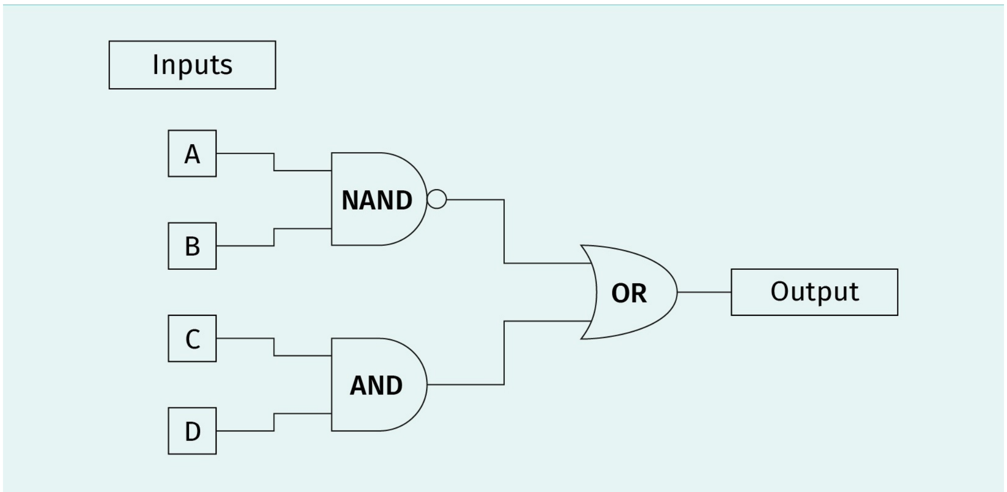
Logic Gates

Digital circuits send an electrical signal along a pathway indicating a 0 or 1 state. A basic logic circuit has this description: it accepts several different inputs (of 1 or 0) and returns a value at the end (also a 1 or 0.) This seems rather elementary, but once thousands of logic circuits are combined together it allows for complex operations. The good news is that with today’s level of technology we can actually use computers to design other computers. We can simply tell a design program what the inputs are and what the desired outputs are, and it will design the most efficient logic for the situation. However, to understand what is going on, a programmer or designer should understand the basics of how this works. Even though a computer programmer would not directly deal with circuits, the code that they write behaves in the same way. Simple logic gates take two binary inputs and give a single result for output. They form the building blocks of all computer logic.

AND, OR, and NOT

AND and OR perform true and false evaluations on input (remember that we are converting it all to 1 and 0 now). NOT provides a way to negate the value, turning a 1 to a 0 or a 0 to a 1.

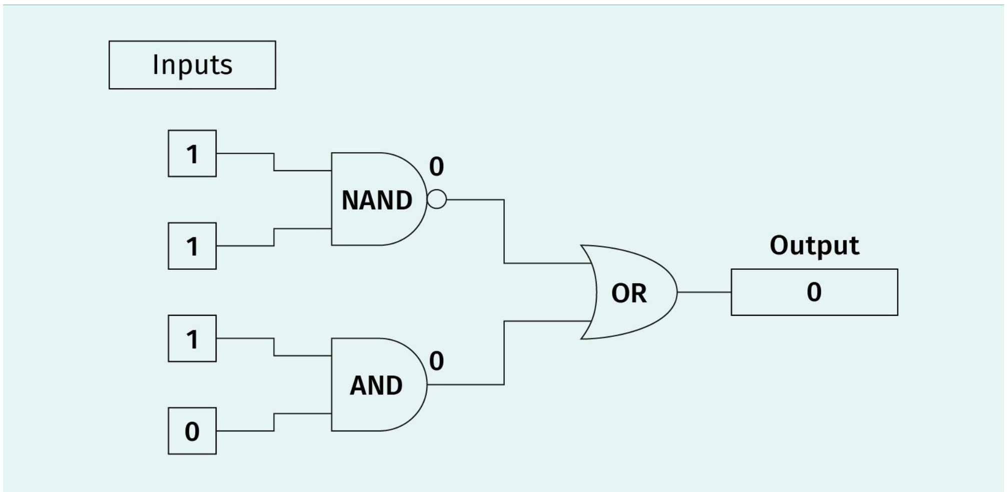
Figure 36: Simple Circuit



Source: Stephen Weese, 2020.

The logic formula that matches the above circuit is ¬(A ∧ B) ∨ (C ∧ D). Inputs for the circuit are A, B, C, and D; these can be either 0 or 1. The symbol after A and B is for a NAND gate, meaning “not AND.” The small circle after the gate indicates negation. The gate for C and D is a standard AND gate. After evaluating the first two gates, they are then sent to an OR gate, and the final result is output. Let’s input data into this circuit and see the results.

Figure 37: Simple Circuit with Binary Inputs



Source: Stephen Weese, 2020.

If we input 1 and 1 to a NAND gate, what do we get? NAND is just the negation of an AND. An AND of 1 with 1 gives a result of 1, but the negation of 1 gives us 0. The AND at the bottom, with 1 and 0 input also gives a 0 result. Both results then go into the OR gate—an OR with two "false" inputs (0) gives us 0. So, the result of this circuit is an output of 0. We can make a truth table for this circuit. Since this is a circuit, our truth table will be in binary form.

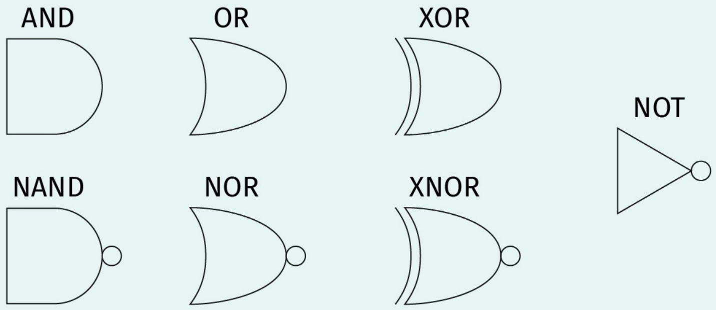
Figure 38: Truth Table for Circuit Example

1	1	1	1	0	1	**1**
1	1	1	0	0	0	**0**
1	1	0	1	0	0	**0**
1	1	0	0	0	0	**0**
1	0	1	1	1	1	**1**
1	0	1	0	1	0	**1**
1	0	0	1	1	0	**1**
1	0	0	0	1	0	**1**
0	1	1	1	1	1	**1**
0	1	1	0	1	0	**1**
0	1	0	1	1	0	**1**
0	1	0	0	1	0	**1**
0	0	1	1	1	1	**1**
0	0	1	0	1	0	**1**
0	0	0	1	1	0	**1**
0	0	0	0	1	0	**1**

Source: Stephen Weese, 2020.

The inputs of 1,1,1,0 are only one combination of three possible input combinations that will yield a zero result for this circuit. You can see the others are 1,1,0,1 and 1,1,0,0. Now that we've seen an example circuit, let's look at the common gates that make up a circuit.

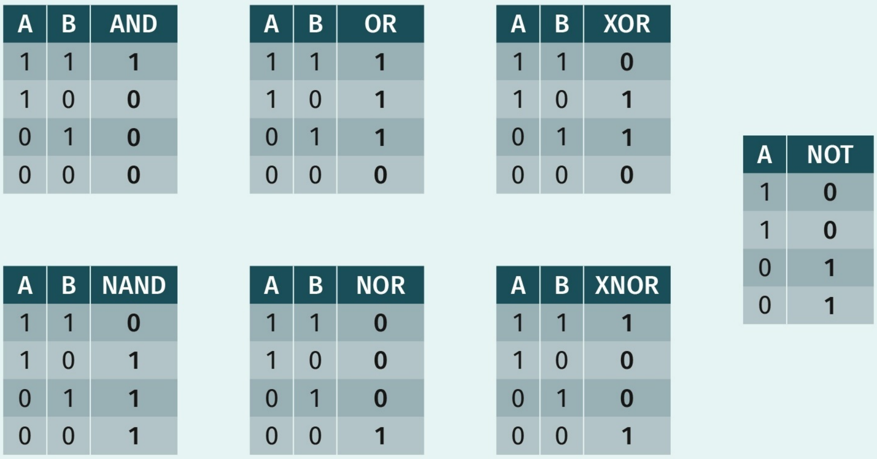
Figure 39: Common Logic Gates



Source: Stephen Weese, 2020.

We know how the AND and OR gates work since they work the same way as a conjunction and a disjunction in logic formulas. The NOT is the same as before. The NAND gate is simply a negation of an AND gate, and a NOR is the negation of an OR gate. The XOR (exclusive or) is a new gate—before, we mentioned that in English when we say "or" we usually mean one or the other but not both. This XOR expresses that exactly; it also could be thought of as indicating when both inputs are different. Of course, the XNOR is the negation of the XOR; it is true when both inputs are the same. Here are the truth tables for these gates.

Figure 40: Logic Gate Binary Truth Tables



A	B	AND
1	1	**1**
1	0	**0**
0	1	**0**
0	0	**0**

A	B	OR
1	1	**1**
1	0	**1**
0	1	**1**
0	0	**0**

A	B	XOR
1	1	**0**
1	0	**1**
0	1	**1**
0	0	**0**

A	NOT
1	**0**
1	**0**
0	**1**
0	**1**

A	B	NAND
1	1	**0**
1	0	**1**
0	1	**1**
0	0	**1**

A	B	NOR
1	1	**0**
1	0	**0**
0	1	**0**
0	0	**1**

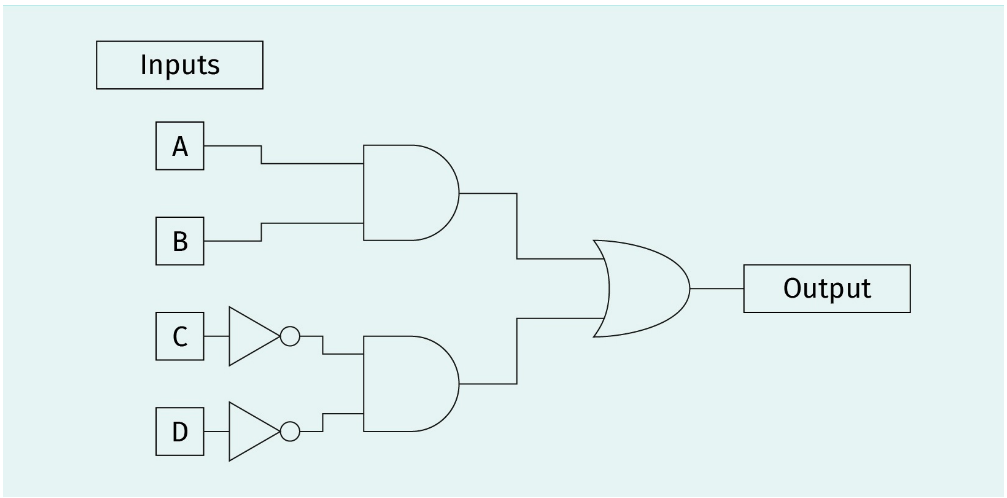
A	B	XNOR
1	1	**1**
1	0	**0**
0	1	**0**
0	0	**1**

Source: Stephen Weese, 2020.

Logic gates follow rules comparable to those given in propositional logic. When multiple gates are combined, more complex formulas can be represented. Let's adapt the formula $(p \wedge q) \vee (\neg r \wedge \neg s)$ from the previous section into a circuit design. We will change the variables to the more common A, B, C, and D and the operators to gates. It now looks like:

(A AND B) OR (NOT C AND NOT D)

Figure 41: Digital Circuit Based on Formula



Source: Stephen Weese, 2020.

To represent NOT C and NOT D, a NOT gate is placed just after each one is input, which will negate both. Note that (NOT C AND NOT D) is different than (C NAND D.) The reader is encouraged to make a truth table for each to see how they are different.

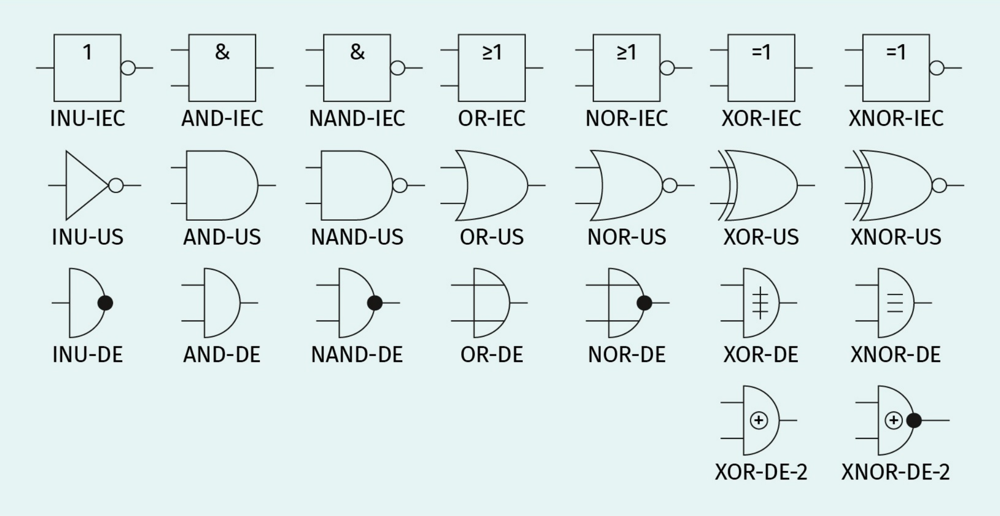
Logic, Formulas, and Design

Propositional logic (PL) gives us a good working basis for circuit design. The concepts from PL translate into similar constructs in a digital world. True becomes 1, and False becomes 0, for instance. Complex circuits can be designed by combining many gates to generate the desired output based on all possible input combinations. There are many tools online that allow you to design and test digital circuits. One is available from Academo.org, or you can search Google for others.

International usage

Symbols for logic gates can vary by country. The ones shown in this chapter are standard U.S. symbols. International and German symbols are shown in this diagram.

Figure 42: International, German, and US Logic Gates



Source: Stephen Weese, 2020 based on Stefan506, 2005.

SUMMARY

Propositional logic takes the concept of human statements and represents them as a formal equation. These logic equations can be used to evaluate the truth or falsity of claims made, or propositions. Propositions can be combined together with the connectives conjunct and disjunct. They can also be negated. Combining several propositions together with these gives us a logic formula. Since computers deal with true and false decisions often, this translates well for programmers to use.

Logic formulas can be written in these normal forms: conjunctive and disjunctive. Conjunctive normal form is several disjunctions connected by a conjunction. A formula in disjunctive normal form is a collection of conjunctions joined by a disjunction. These normal forms help to simplify logical formulas for use in computer programming and circuit design.

Digital circuit design goes down to the individual, bit level with inputs of 1 and 0 that are the equivalent of “true” and “false” in logic. There are several different logic gates that provide the functionality of logical operators such as AND, OR, and XOR. Given binary inputs, these gates are programmed to give consistent outputs. Many gates combined together can be used to represent complicated mathematical functions.

Truth tables are used to illustrate the results of a logical formula or a digital circuit. They demonstrate the exact output of these given specific inputs for the variables. Digital circuit design is a field of its own with many employed in the profession of integrated chip design.

UNIT 5

HARDWARE AND COMPUTER ARCHITECTURES

STUDY GOALS

On completion of this unit, you will have learned …

- the basic elements of computer architecture.
- how processors and memory work.
- how a computer processes input and output.
- how operating systems and hardware communicate.
- an overview of high-performance computing.

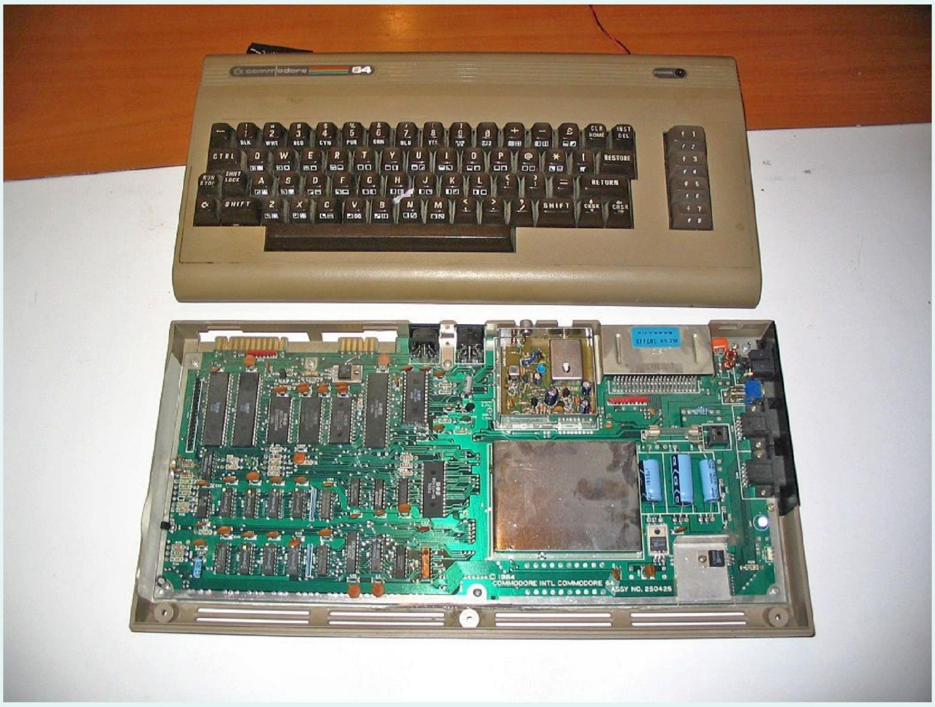
5. HARDWARE AND COMPUTER ARCHITECTURES

Introduction

In 1981, Jack Tramiel was the owner of Commodore, a computer and calculator company. Commodore also owned the chip making company MOS Technologies. It was at this time that the company embarked on an ambitious project: offering a high-powered personal computer for about $600.

MOS Technologies had created an iconic chip, the MOS 6510, which was the basis for the new computer. Since Commodore owned a chip-making company themselves, they did not need to negotiate with another independent company for deals on the chip—they were able to get them at cost. These chips were rated at about 1MHz clock speed, with a data width of 8-bits and an address width of 16-bits. Compared to 21st century chips, they are very limited, but for 1981 it was more than adequate. 16-bit addressing allowed this new computer, the Commodore 64 (C64), to address 64KB of RAM—which is where the name came from. At the time, this amount of RAM was unheard of, especially at the price of $600 (Commodore 64, n.d.).

Figure 43: Commodore 64



Source: Dake, 2005.

The Commodore 64 also had a sound chip that could play multiple voices at once—other computers at the time had far inferior sound quality. Eventually, as production and demand increased, the price of computer parts decreased, and the Commodore 64 was selling for $199. Nothing on the market for personal computers could match its performance and price; software companies created over 10,000 titles for it. It eventually went on to become the best selling personal computer ever made, selling between an estimated 17 million and 30 million units. It also had a joystick port and a modem port, enabling users to play thousands of games and connect with other computers. (There was no internet yet in the 1980s).

The Commodore 64 was built around the BASIC programming language; using the Commodore meant you had to learn some programming. In the 1980s it became the gateway for many young programmers to full-time jobs in the industry. Today many in the computer field have not heard of the Commodore 64 and its record-setting history, yet it is considered by some as the best computer ever built (McFadden, 2019). The C64 used Von Neumann architecture, a computer design that is still used today.

5.1 Computer Types and their Architecture

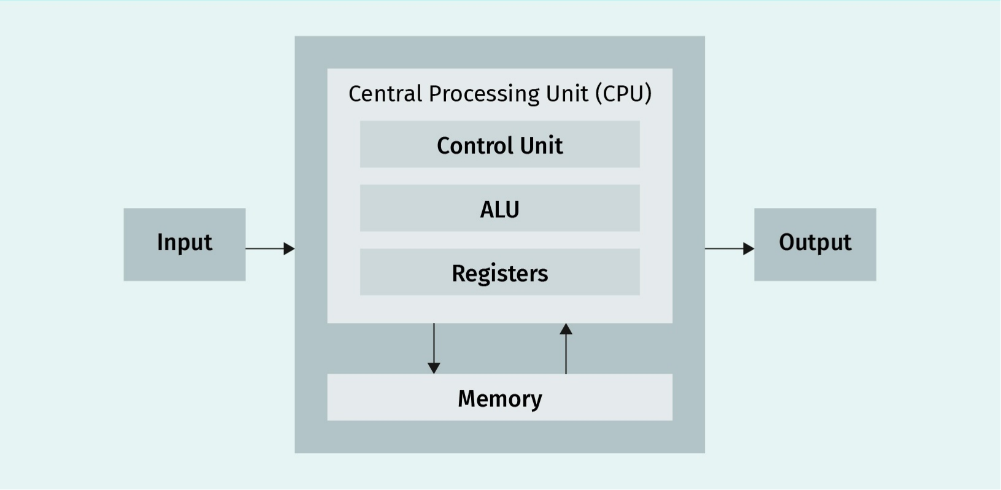
A computer, at its most basic level, consists of a processor and memory storage. The processor is able to perform various calculations and commands using the data stored in memory. It takes that input from memory and creates unique output, the results of the processing.

The bits (0s and 1s) stored in the memory fall into two categories: data and instructions. They are both binary numbers; the context tells the processor whether each group of bits is a command (telling it to do something) or data (information to be processed). This is the concept behind Von Neumann Architecture, a basic design for computer systems that is still in use today.

Besides the processor and memory, there also should be some type of input and output system—the basic architecture doesn't require a specific definition of this, but it is assumed since a human being needs to communicate with the device to use it. It is sufficient to say that there is some system for entering data (in binary, of course) and a system for outputting the data.

A CPU-based system works like this: it checks for the memory address of the next instruction and then reads that instruction. If the instruction requires data access, it gets the address of the data and reads it. Then it performs processing on the data and sends it to the output memory address. Repeat this millions or billions of times per second and you have an advanced computer.

Figure 44: Von Neumann Architecture



Source: Stephen Weese, 2020.

Von Neumann broke down the CPU into subcomponents. The Arithmetic and Logic Unit, or ALU, is responsible for just what its name says: it performs basic arithmetic calculations on data, such as adding and subtracting, but also performs logic operations such as AND and OR. The Control Unit also performs the tasks for which it is named: it controls the operation of the ALU and communication with the input and output devices. It interprets processor instructions and carries them out. A CPU has its own internal memory storage built right on to the chip as registers. These are very small in data size, measured in bits, 8 or 32 bits, for example. Newer processors have 64-bit registers. This simply means that in each register on the CPU it can store 64 bits (or eight bytes). These registers are used for different purposes.

Von Neumann Registers

There are five registers specific to the Von Neumann design. These are as follows:

1. Program Counter (PC) contains the memory address of the next instruction to read from memory.
2. Memory Address Register (MAR) contains the address of the current instruction in memory, or the next data transfer address.
3. Memory Data Register (MDR) contains the contents of the memory address that the MAR is pointing to and contains data to be transferred.
4. Accumulator (AC) this contains data that have been processed or are about to be processed, including arithmetic or logic results.
5. Current Instruction register (CIR) contains the current binary instruction that is being executed.

Data Buses

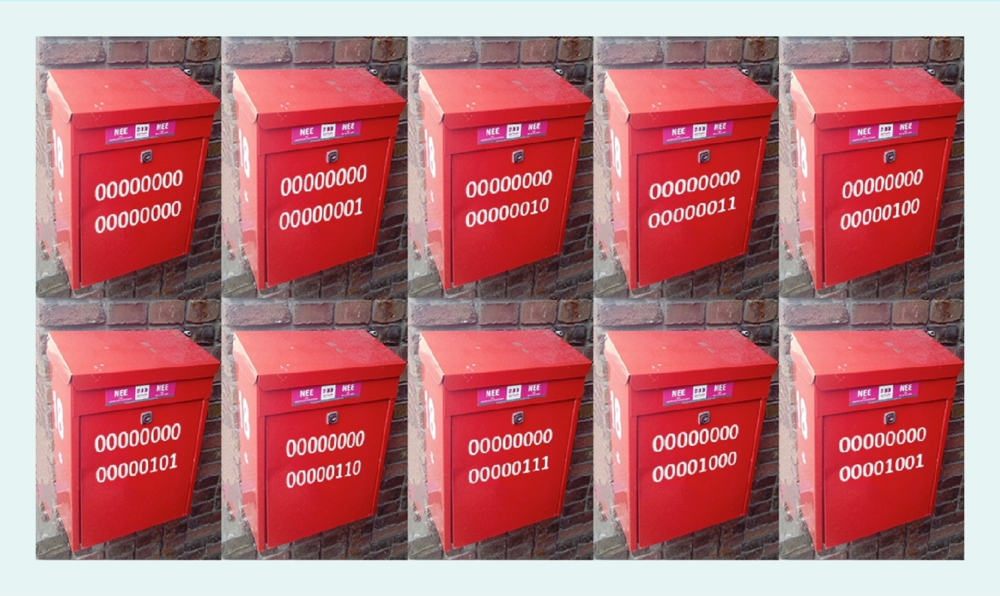
A bus is simply a connection that is used for data transfer. A CPU will need to have a way for input and output—therefore it will use buses. The standard Von Neumann design incorporates an address bus, which is used to send memory addresses, a control bus, which carries instructions and special signals from devices (such as their status), and a data bus, which simply sends and receives data.

Since the CPU is not only performing calculations but controlling devices, it needs a way to know what the other devices are doing, and whether they are ready to perform a new task. The control bus is used to send and receive these status updates from devices as well as to send instructions to them.

Memory Addressing

In this architecture, each byte of memory has a unique binary address. You could think of them each as little mailboxes, storing a different binary number and also having a binary address. That means that the number of bits used for addressing will identify the number of bytes available. For instance, the Commodore 64 had 64K of addressable memory. This was accomplished by using 16-bit addressing. The total number of bytes that can be addressed is 2^{16} which is 65,536 bytes. This was considered 64K. (At the time, a kilobyte was referred to as both 1,024 (2^{10}) and 1,000 bytes. In 1998, the standard was set to 1,000 bytes.)

Figure 45: 16-bit Addressing



Source: Stephen Weese, 2020. based on Trung Quoc Don, 2019.

Each 16-bit address points to a mailbox (memory location) that contains exactly one byte (inside the box). Some registers store memory addresses and use them to point to the data inside. It is used as a way to keep track of the last CPU-accessed memory byte so it can then read the next byte in sequence (if you are reading sequentially).

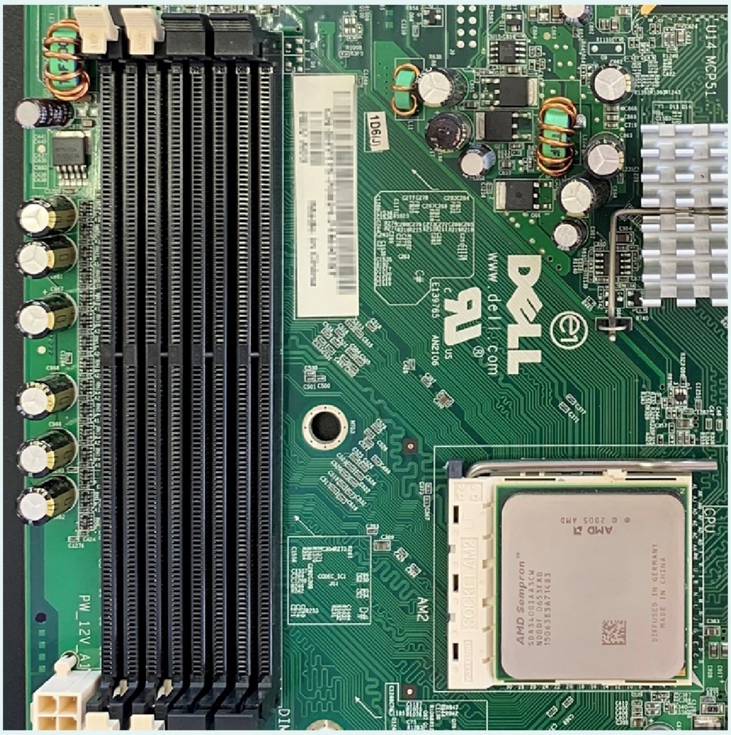
Memory Usage

In the Von Neumann architecture, memory can be used for both data and instructions. This lends versatility to the design as the available memory can be split between them any way the programmer wishes; the computer program itself can also be written to utilize the memory any way it wishes as it runs. There is a slight disadvantage with this design, as data and instructions are on the same memory bus. If the CPU is reading a large volume of data, it may have to wait many cycles to load the next instruction.

5.2 Processors and Memory

To solve the problem of delay in accessing the RAM directly from a CPU, a local cache of memory was added directly to the CPU itself. Since the processor (CPU) is always accessing memory (RAM), personal computer designers place the memory very close to it on the motherboard. The motherboard has tiny lines etched into its surface that act like little wires in order to transmit binary data signals back and forth.

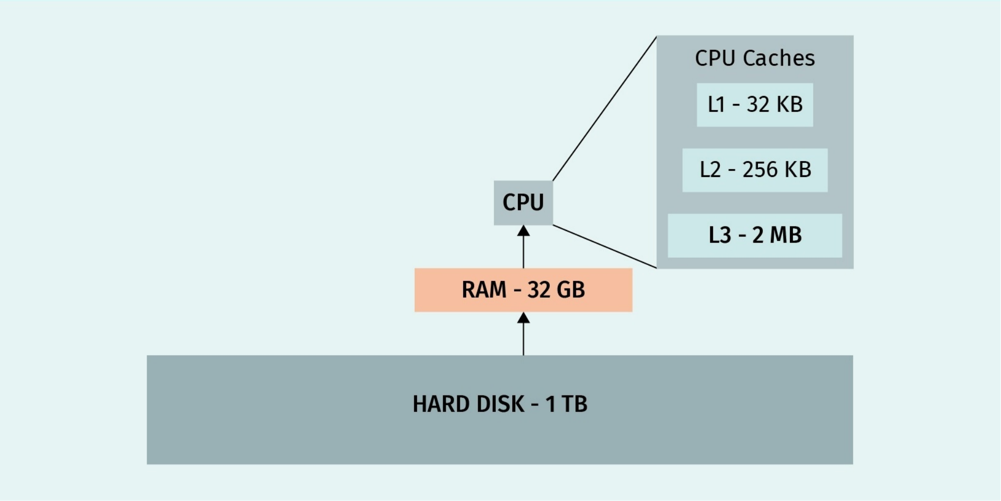
Figure 46: CPU and RAM Slots on Motherboard



Source: Project Kei, 2020.

Though the CPU is very close to where the RAM sits on a PC motherboard, when you are dealing with billions of cycles per second, this distance can still cause a slowdown. This is where cache memory comes in; it functions as a small storehouse of memory directly on the chip which improves memory access time. This establishes a “supply chain” of data from the largest and slowest storage medium (the hard disk) through the RAM and, finally, to the cache.

Figure 47: Computer Memory System



Source: Stephen Weese, 2020.

Let's say you have a PC with a 1TB hard drive, 32GB of RAM, and an Intel i7 CPU. On the CPU there are three levels of cache with 2MB, 256KB, and 32KB of storage. The closer you get to the actual CPU, the smaller the memory size needed. As you can see from the diagram, there are five steps of memory used by the processor. The hard drive is the slowest, but also the largest. It stores your entire library of applications and data. However, you don't need to access all of that at once—it depends on what task the user is performing at the moment. When you open up an application, it copies itself into RAM. RAM is smaller in memory size, but much faster than a hard drive (though solid-state drives (SSD) are faster than a hard disk drive (HDD), they still are both slower than RAM). RAM is meant to be smaller since it only stores what you are currently working on, instead of everything. The CPU accesses data from the RAM directly and copies a smaller segment of the current instructions or data to its L3 cache. As you can see, the i7 has two more stages of cache, L2 and L1, each smaller, and closer to the CPU itself. This is done because of the weakness of the Von Neumann design, where memory is accessed only through one bus, whether it be data or instructions. Using algorithms, the CPU decides which data should be placed in its own local caches, and it can get to them very quickly. Notice that the L1 cache in our example is only 32KB of memory, whereas the hard disk is 1TB, which is approximately 30,000 times larger. This system sifts the data down to the most significant at any given moment and provides it to the processor in the L1 cache. If needed data is not in the L1, then it checks L2, L3, and then RAM. The closer the needed data is, the faster it can be accessed.

Personal Computers and RAM

Currently, Windows 10 is the most popular desktop operating system in the world, hovering at about 75% of the market (StatCounter.com, 2022). When a desktop is started up, it first loads (copies) the operating system into RAM. Windows 10 requires at least 2GB. That means even if you have 32GB as in the last example, only 30GB would be actually available to you for data and applications. Remember that RAM is a holding space for current programs and data that you are working on and it can change. When you close an app, it frees up memory for a new app to use. RAM is **volatile** memory, unlike the hard disk which is more permanent. If you were opening up a picture of your cat in a photo editing software, both the software and the data from the cat photo would be stored in RAM while you were working on it. When you close the app (and, hopefully, save your changes), the memory is released.

Volatile
With respect to memory, volatile means that it needs power to store data. Turning off the device erases the contents.

5.3 Input and Output

Data is stored in memory from the hard drive up to the CPU cache. Of course, when a human uses a computer, they will interface with it and create new data for input. In response to this input, the computer will then respond with output. Input begins with the user and ends with an output device.

Table 9: Input Devices

Input device	Description
Keyboard	A user enters keystrokes which are converted into electrical signals, and then into binary codes.
Mouse	A user enters input by moving the mouse, creating electrical signals which are converted into binary coordinates.
External Devices	USB devices such as cameras and microphones also input electrical signals that are then turned into pixels and frames for video, or a digital sound wave for audio.
External Data Sources	Through a network card, a computer can receive new data. It can also receive data through a portable USB drive; these data are already in binary form.

Source: Stephen Weese, 2020.

Table 10: Output Devices

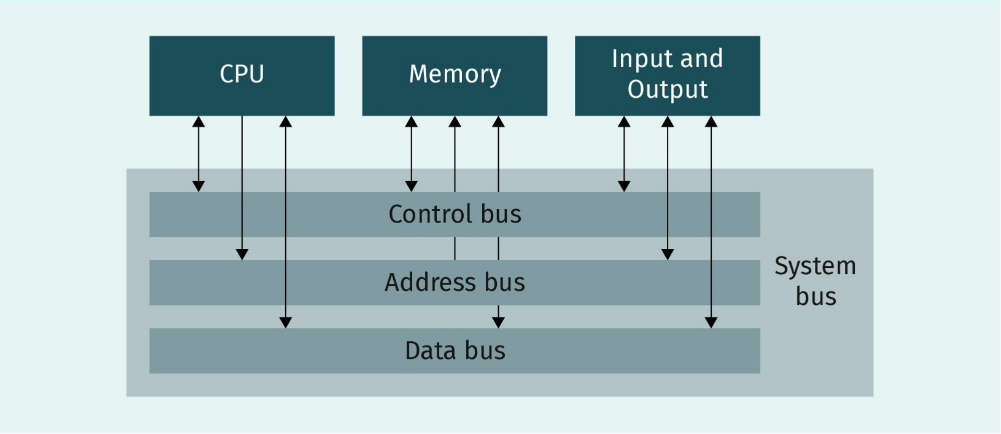
Output device	Description
Monitor	High-resolution monitors can display many picture elements (pixels), and they may use the Digital Visual Interface (DVI) to transmit moving images from a computer's graphics card.
Printer	A document is broken down into primary ink colors and pixels and then printed to paper.
Speakers	The computer converts sound to analog from digital and plays it through the speakers.
Network card	A computer can send (and receive) binary information over a network.

Source: Stephen Weese, 2020.

This is the input and output at each end of a full computer system. Internally, however, there is also an input and output system for the processor. Data from devices such as the keyboard or a microphone require some processing before being sent to the computer; this is often referred to as IOP, or input-output processing. Once the data are properly arranged into bytes matching the proper **word** size, they can then be sent through the data bus to the processor.

Word
This is the standard data size used by a processor. A 64-bit system uses 64-bit words; data is passed to registers in this size.

Figure 48: Input Output System



Source: Stephen Weese, 2020.

The CPU uses the control, address, and data buses to communicate with RAM and external devices. In some designs, these three are merged together into a system bus.

Input and output devices on a computer are typically assigned a binary “channel code” for the CPU; that means when a CPU receives this code it will be followed by a status message from that device. The CPU can also use this code to send information back to the device. This information flows along the control bus. Control codes for a device can include things like ready, busy, error, and request for input or output. Aside from the control bus, devices can also use the data bus to send data and the address bus to point to a specific memory address that may be read from or written to.

So far, we have discussed communication between hardware elements such as the CPU, RAM, monitor, mouse, etc. The operating system adds an extra level of complexity to a computer system, but a necessary one. Without an operating system, a user would not be able to communicate with the computer hardware.

5.4 Interfaces and Drivers

An operating system performs three main functions for a computer:

1. Provides an interface
2. Controls the hardware
3. Runs applications

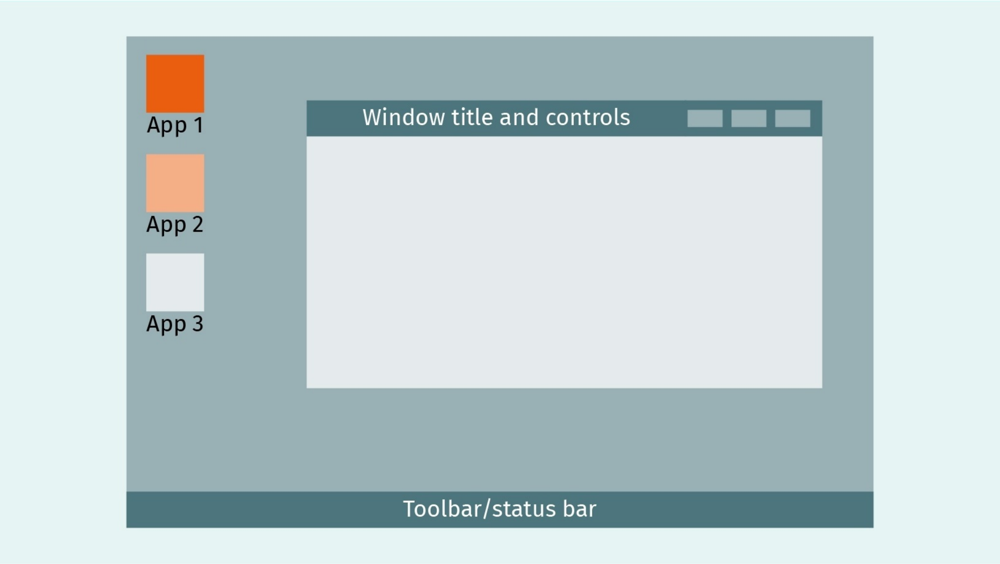
An interface is the combination of all of the methods designed for a user to interact with a computer, input and output. You might remember that the CPU controls the hardware after receiving commands from the operating system. A user gives a command, which is relayed through the operating system to the hardware, including the CPU. Then the CPU can direct the hardware to perform the individual tasks at a granular level.

The Computer Interface

The earliest computers used switches and punch cards to communicate with their users. Eventually, video screens were attached—these were the large and heavy CRT monitors. There were no graphics when these first came out, so all computer input and output was text on a screen. This was known as a **command line interface** operating system. Many mainframes used this type of interface; universities had large UNIX servers for their computer science departments so students could compile their code. As computers became personal, the graphical user interface, or GUI, grew in popularity. The Windows interface is the most popular for personal computers, followed by Mac OS (which is based on Linux, a type of UNIX.) They both provide a very similar experience for the user.

Command line interface
This was a visual computer interface that used a series of exact text commands entered through a keyboard.

Figure 49: Modern Desktop Interface



Source: Stephen Weese, 2020.

Whether you are using a Mac or a Windows machine, the same metaphors are being used. Applications can be run by clicking on “icons” on the desktop. Double clicking, single clicking, and right clicking all have specific meanings that are similar in both operating systems. Windows and MacOS also use “windows” for applications, where an app normally takes up a rectangular area on the screen that can be resized or temporarily “minimized” off the screen. As a computer user at this point in life, you have been trained for these computer interfaces. It is expected that you have a mouse, a monitor, and a keyboard. You know what icons mean, what a status bar is for, and how to move around and use windowed applications on your screen. But this didn't just come to you naturally, you had to learn it. If another computer came along with a very different interface, it would be difficult to use at first. Operating system programmers consider the user interface when each new upgrade is released and rarely make major changes.

The operating system takes input from the user primarily through keyboard and mouse, and shows output firstly through the monitor, and secondarily through the speakers (for example, those annoying little sounds that let you know you can't do something).

Consider this little part of the interface: the throbber (also known as the loading icon). When you click an app, it is copied into RAM (loading). Without the loading icon, you wouldn't know that the computer is doing anything and most likely would repeatedly click the icon, thinking that it is not working. This seemingly insignificant part of the interface provides important feedback to the user that something is happening.

Interface devices such as the keyboard and the mouse are still ubiquitous today, but there are also new trends emerging. Who still remembers the spinning wheel on Apple IPods - the first touch surface integrated into a device? This technology was subsequently refined further and further and found its climax in the smartphone and tablet. The operating concepts behind these devices have had a strong influence on further aspects of our lives. For example, think of a toddler trying out swipes or touch gestures in a conventional children's book. Or imagine an adult standing in front of a fully automatic coffee machine, attempting to select an option by touching the screen, despite the fact that there are buttons on the left and right for selection.

Touch and gestures, like pointing or swiping, play a significant role, but what about language? A lot has changed here in recent years. In the past, voice input was only used for special applications, for example, in medicine or automobiles for control purposes. Today, voice control is integral to the functionality of smartphones, tablets, smart speakers, and their operating systems. Whereas in the past, it was individual commands that had to be spoken and recognized by computers, today, the processing and understanding of natural language play a unique role in information technology. This is not all - think of technologies such as facial recognition or IRIS recognition, which can recognize faces, identify people beyond a doubt, and evaluate and interpret their facial expressions. Ultimately, we have reached a point at which all human senses are used to create a complete interface between a computer and a human, and in which reality and computer-generated reality intertwine.

Drivers: Talking to Devices

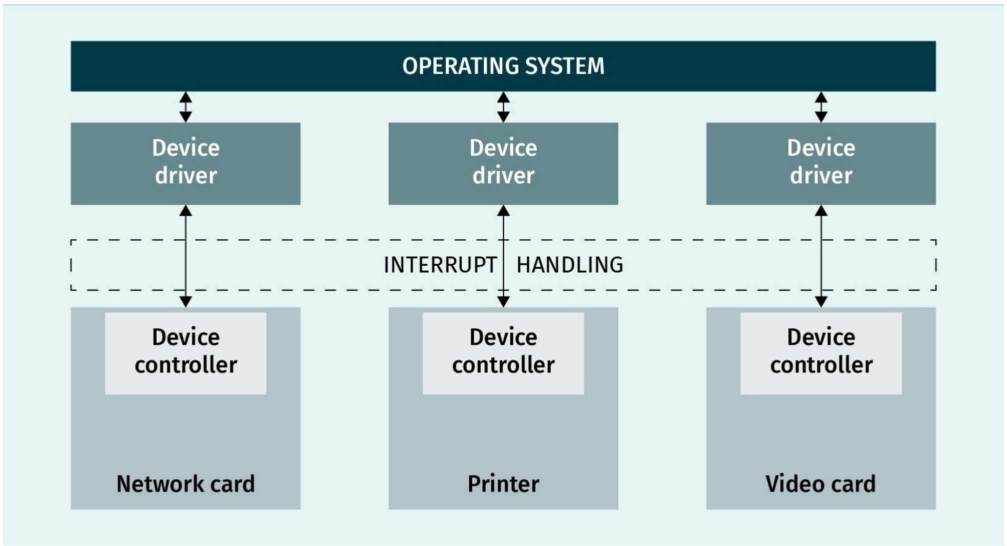
We mentioned that one of the jobs of the operating system is to control the hardware. This includes any device connected to the computer. Standard internal devices such as the hard disk and video card are included, as well as things like external webcams and printers. Each device is different and performs a different function, yet they all must talk to the same operating system. The method of communication involves a special piece of software called a driver.

In the early days of Windows, installing drivers for a piece of software was a complex task. The user often had to specify the **IRQ** and the I/O port address—it was important to make sure that no other device was using the same settings. Standard PC design allowed for 16 IRQs for devices to send an interrupt request to the CPU, basically demanding its attention. Modern Windows and Mac machines configure these automatically (known as plug and play), so the user has no need to worry about them (and most users do not know they even exist).

IRQ
This is an interrupt request line, labeled from 00 to 15 and assigned to a device to send status messages to the CPU.

Besides sending IRQ signals, the driver translates data between itself and the operating system. As you can imagine, a network card, a printer, and a video card would all be using very different contexts for their data and command signals. Since PC parts are open to almost any manufacturer, each company is responsible for the development and operation of their own drivers for their hardware.

Figure 50: Device Drivers and Operating System



Source: Stephen Weese, 2020.

Any device connected to a computer has a device controller, which is responsible for the binary input and output to the device. The device controller can also signal the CPU with an interrupt. A device driver has to be designed specifically to talk with both the operating system that is in use and the hardware device's controller. Sometimes, **bugs** are discovered as new driver updates are released, so it is a good idea to check from time to time that a computer system has the latest drivers. The operating system relays commands from the user to the appropriate device via the device driver, which talks to the device controller, which then communicates with the device.

Bugs
These are software flaws that range from a minor inconveniences to serious problems.

5.5 High-Performance Computing

Sometimes the demand for a specific computing task is more than just a single desktop or server machine can handle. This is where we enter the world of high-performance computing. On one hand, there is a simple way to perform these kinds of tasks: using a cluster. A cluster is a group of identical **servers** that share the workload for a certain task. On the other, sometimes the demand is even greater than a cluster can handle—in these cases we need a supercomputer.

Server
This is a dedicated computer set up to handle requests from users over a network. It is often kept in a separate server storage area.

Parallel Processing

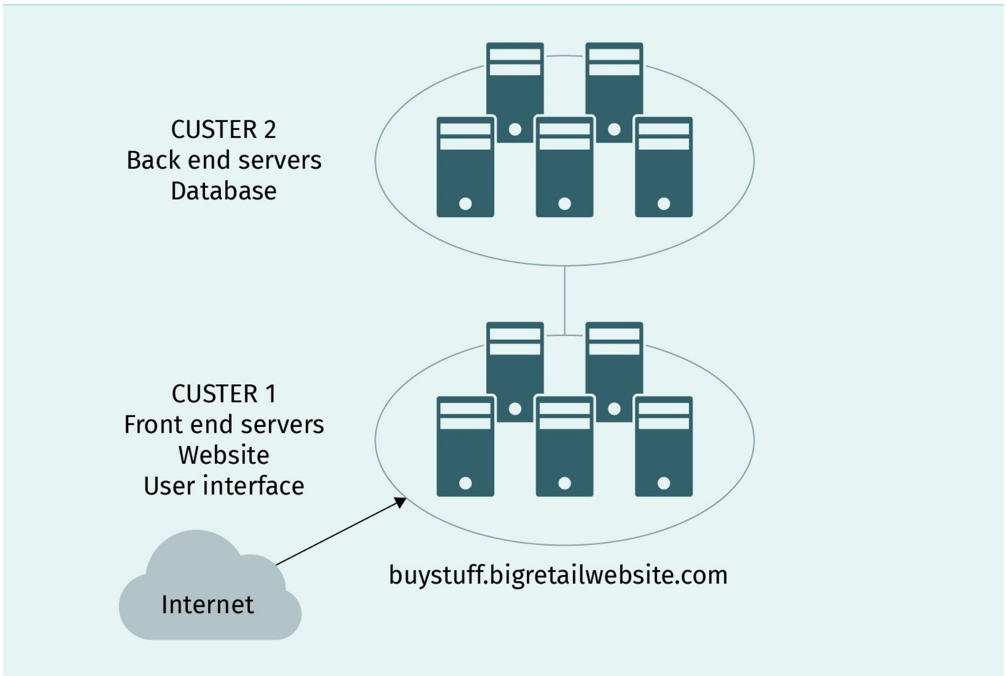
You might imagine taking a regular computer and just making it giant-sized to create a supercomputer. However, the technologies involved don't scale that way. You can think of it like an automobile—you can make bigger and bigger cars with bigger engines, but the larger engines will weigh more, eventually cancelling out any gains in horsepower and speed. A single desktop computer with an enormous RAM space would have to spend more time managing all of that memory and eventually would be useless. Instead of building a huge computer with one CPU, we use the concept of parallel processing. This is when we take a task and break it down into sub-tasks and assign a separate CPU to complete it. Several CPUs are linked together in parallel to accomplish this.

Server Clusters

Imagine a popular website such as Wikipedia or Amazon. There is no way a single computer could handle all of the processing power of all those Wiki articles or all of those Amazon orders. Yet, when a user accesses these sites, they expect to get the same experience every time. One method to achieve this is by making a server cluster. A cluster is a group of identical servers; think of them as "clones" that can respond to web requests. Not only do these clones exist to respond to multiple requests, but they also are spread out geographically all over the world. A user in South America, for instance, could connect to one close to their location. When a server is updated, there is a "master" server that is then copied or cloned to all of the servers in the cluster through a **version control** software that makes sure that all servers eventually get every update.

Version control
This is a change management system that tracks and updates the latest version of documents, databases, and websites.

Figure 51: Server Clusters



Source: Stephen Weese, 2020.

A large retail website could make use of multiple server clusters, splitting the load between the front end (the website users see) and the back end (the database with all the images, descriptions, and prices). When the user types in the URL for the site, it connects to one of the clones in the cluster (usually selected by a **load balancing** system). When the user performs a search for a “large black umbrella” the front end server connects to one of the back end cluster servers and requests the umbrella items list that match the query. A cluster provides the advantage of not only evenly splitting up a task (load balancing) but providing fault tolerance: if one server goes down, the others are still functioning.

Load balancing
This is a management process that distributes the processing load as evenly as possible among servers.

Supercomputers

When you need more power than just a cluster, then a supercomputer is the answer. These machines are rare and cost hundreds of millions of dollars. They are similar to clusters in at least one way: they use parallel processing, where tasks are split up between processors. One such supercomputer is in Germany, the SuperMUC-NG.

Computing power for desktops and supercomputers can be measured in something called FLOPS, which stands for floating point operations per second. Basically, this is an operation on numbers with a floating (movable) decimal point. Modern desktops are no slouch at this, clocking in at approximately 100 gigaFLOPS. A gigaFLOP is one billion calculations (per second).

The SuperMUC-NG is rated at approximately 26 petaFLOPS. A petaFLOP is 10^{15} floating point operations (per second). This means that this supercomputer is 260,000 times more powerful than an average desktop. This is accomplished by putting 300,000 very high-end, water-cooled CPUs together in the same room and connecting them together to work in parallel. This computer also has 700 TB of RAM (SuperMUC-NG, n.d.).

Figure 52: Supercomputer

Source: Argonne National Library, 2007.

Supercomputer Challenges

Supercomputers are often used in scientific fields to analyze vast amounts of data. For instance, predicting the weather involves an enormous number of data points; a supercomputer is a good match for these kinds of problems. Scientists can request time on supercomputers to do research, bringing them the massive power of Earth's most amazing computers.

SUMMARY

Many of the computers in use today use a common design plan: the Von Neumann architecture. This simple design involves the interaction of a CPU with a memory store. Input and output arrive to the CPU through registers, which contain a fixed number of bits. These registers can contain data or instructions that have been read from memory.

One way to speed up memory access times is to put some memory directly on the CPU chip; this is called a cache. There are different levels of cache: some processors have three levels labelled L3, L2, and L1, each smaller and with a faster access time. This provides a way to avoid memory bottlenecks as the CPU does not always have to make a request of system RAM; a small amount of currently used data sits in the cache.

The Von Neumann architecture also includes buses; these are connections used to carry data. They can carry input and output related to memory addressing, command instructions, and pure informational data. The results of processing, such as an arithmetic operation, are passed along the data bus.

One of the primary functions of an operating system is to provide an interface. Personal computers in the twenty-first century have graphical user interfaces (GUI) that enables users to use a mouse, point and click, open multiple applications, and resize them. This has become standard. The operating system also provides a way to control and communicate with hardware devices that are added to the computer such as a network card, a video card, or an external web camera. The device driver is a piece of software written to communicated between the OS and the device.

When a personal computer or single mainframe is not enough computing power for a job, supercomputers can provide a solution. Created by combining thousands of CPUs together in parallel processing, they can perform calculations in the petaFLOPS range.

UNIT 6

NETWORKS AND THE INTERNET

STUDY GOALS

On completion of this unit, you will have learned …

- about standard network topologies.
- about the hardware devices used in a network.
- how TCP/IP and the internet work.
- about wireless and wired network technologies.
- about the Internet of Things (IoT).

6. NETWORKS AND THE INTERNET

Introduction

In the 1960s, the U.S. government was researching computer networks. This fell to ARPA, the Advanced Research Projects Agency. Part of the design requirements for this network was one that would be able to withstand disruptions in the physical infrastructure of the network.

The idea was that the military, using a communications network, could possibly have part of it destroyed during action. Early network designs were not “fault-tolerant”: a disruption in part of the network caused the entire network to crash, or stop working. Eventually, ARPA started using TCP/IP for its network, the ARPANET. This technology was fault-tolerant: TCP/IP could direct data around damaged parts of a network to reach the undamaged parts. This network eventually was opened up to universities to do research. Large educational institutions like Stanford and USC became part of the network. Not long after that, commercial entities saw the value of a large interconnected network (internet) and the general public was able to access this network through a university account or through internet service providers (ISPs) such as America Online (AOL).

The World Wide Web is often confused for the internet, yet the web is a set of services that are accessed through the internet. It requires a web browser to access these services. There are other data services on the internet that do not require a browser: online games such as World of Warcraft are good examples.

6.1 Wired and Wireless Networks and Topologies

All computer data are stored in binary, i.e., zeroes and ones. The data sent over networks is no exception. Electrical signals over a wire are created to symbolize either a one or a zero. By themselves, the numbers don’t mean very much, but in the proper context, the data is interpreted by the recipient computer.

Before there were wireless networks, all network data was sent over wires. There originally were different ways to design the layout of the network or the physical topology. Over the years, these have been simplified, and only one or two topologies are primarily used over local networks and the internet.

A LAN, or local area network, consists of all the computers within a specific physical location, such as an office building or floor or a home. Once you connect a LAN to another LAN over a large distance, it is called a WAN, or wide area network. You can imagine two offices from the same company, one in Madrid and one in London, connected together; this is an example of a WAN. The internet is the largest WAN, connecting millions of LANs together through wired and wireless connections.

Network Devices

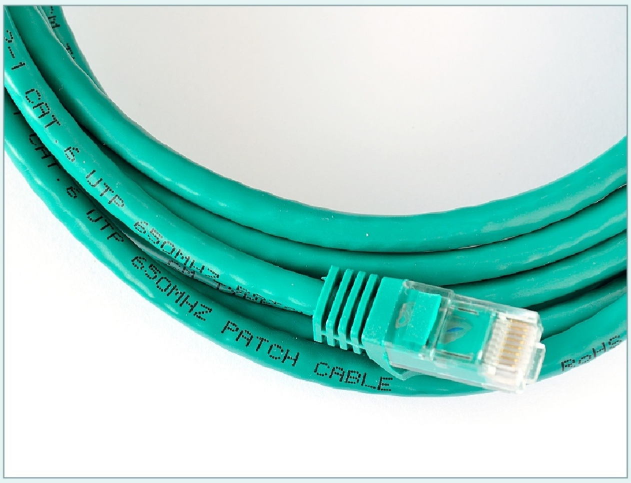
Using a wired network naturally requires cabling. Standard wiring for a LAN (inside a home or an office) consists of **UTP** cable of various categories. Data speed for networks is usually measured in bits per second (bps.) Occasionally, you may see bytes per second (Bps); keep in mind this is 8 times more data throughput than bits per second. The categories of UTP cable are as follows:

UTP
This stands for "unshielded twisted pair cable." The individual wires are twisted together in pairs without extra insulation.

Category	Data throughput
Category 5	Up to 100Mbps (Megabits per second)
Category 5e	Up to 1000Mbps (1Gbps)
Category 6	Up to 10Gbps
Category 7	Up to 40Gbps
Category 8	25Gbps or 40Gbps

These UTP cables appear physically identical. You must usually look for a label or printing on the wire to see the category if you are not sure.

Figure 53: Category 6 UTP Ethernet Cable



Source: Raysonho, 2015.

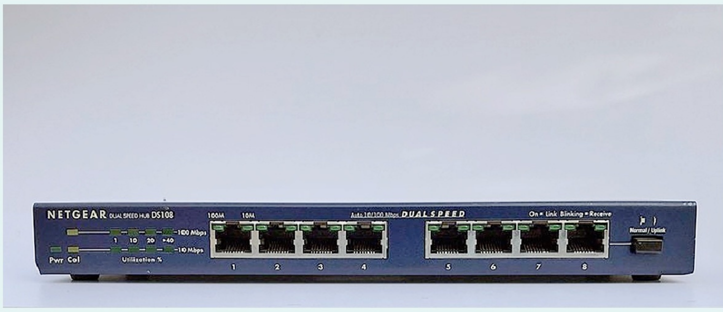
The type of connector used for UTP cabling is called an RJ-45 connector (shown above). Though it is similar to the standard U.S. telephone connector, it is actually larger and incompatible with phone wire, which is Category 1 and unusable for networks.

Hub

The simplest network connection device is a hub. The hub takes a signal from one cable and splits it out over several cables. If it was splitting out four signals, it would be called a “four port” hub. A hub does not do any signal processing—everything that comes in is sent out through all of the ports. Communication is two-way: the ports can also send communication through the **uplink**connection.

Uplink
This is a connection from a smaller network to a larger network. Usually this link sends data toward the internet connection.

Figure 54: Eight Port Hub


NETGEAR DUAL SPEED HUB DS108
100M
10M
Auto 10/100 Mbps DUALSPEED
On = Link Blinking = Receive
Normal / Uplink
100 Mbps
10 Mbps
1 10 20 >40
Pwr Col
Utilization %
1
2
3
4
5
6
7
8

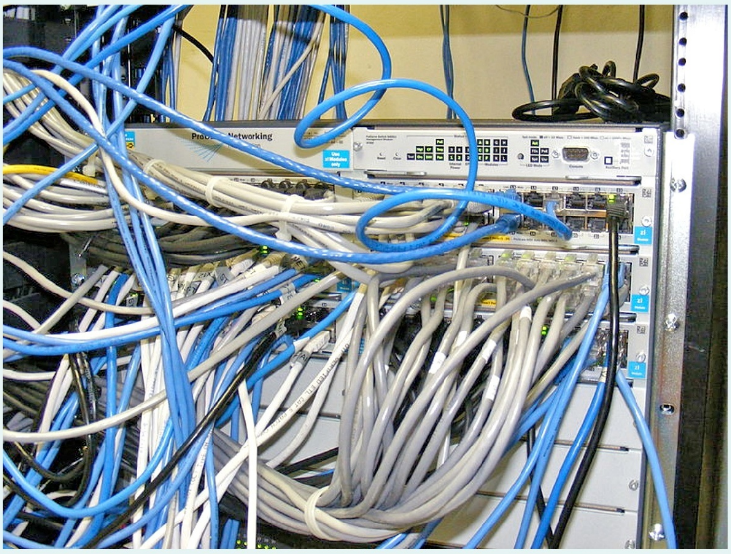
Source: Heimnetzwerke.net, 2018.

Switch

A switch is very similar to a hub. It has a connection for an uplink and multiple ports for communication. However, it does do basic filtering on data. Computers send data over a network in data **frames** with hardware addresses. Some data is sent out with the “broadcast” address; this address means it is for all computers who can receive it. A switch filters these out so they do not get propagated across the entire network (which can cause a flood of data known as a “broadcast storm”).

Frames
A unit of data transmitted over a network is called a frame. A frame has a “to” and “from” hardware (MAC) address.

Figure 55: Switch with Cabling Connected



Source: J.smith, 2007.

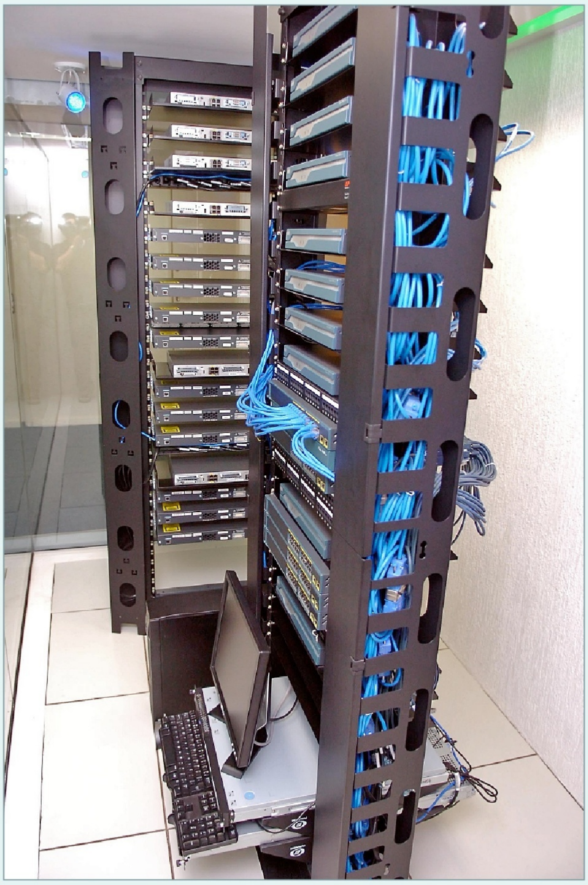
Switches are often placed together in a central location in an office building to split the signal out to different rooms and floors.

Router

The router is the most intelligent of the three connecting devices. It examines the IP address of each **packet** of data sent over a network and only sends the signal through the connection that will reach the address. Since it filters data, it can also block data from entering a network that does not belong there; this function is called a firewall.

Packet
Similar to a frame, this is a unit of network data that is identified by an IP Address (instead of a hardware address).

Figure 56: Routers in a Large Network

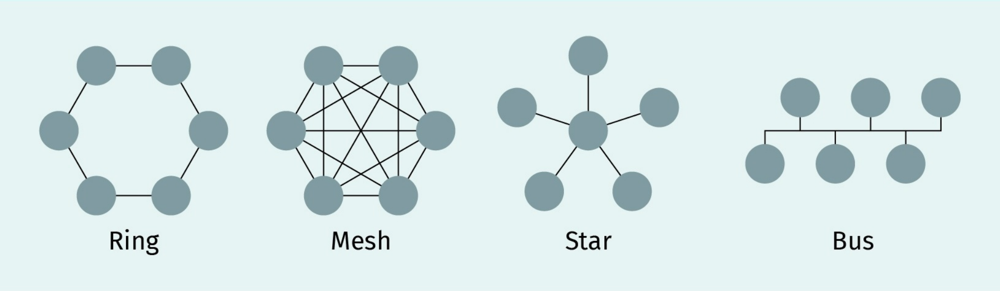


Source: de Lima, 2009.

Wired Network Topologies

A network topology is the physical layout of the connecting devices used. In early days of networks, several different topologies were used. Today, a large portion of networks use the same design: the star topology. Some network topologies are shown and described below.

Figure 57: Topologies for Wired Networks

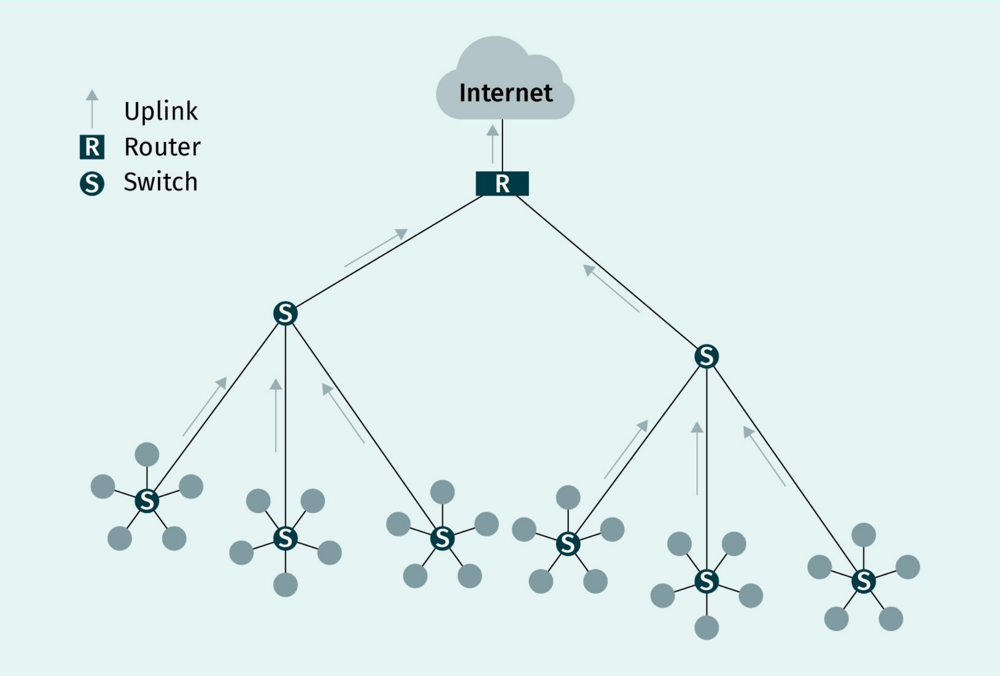


Source: Maksim, 2005.

Ring	A ring network consists of computers connected together in a ring, where the signal would pass around it. In this design, each computer was required to look at any data passed to it to see if it was required to pass it along. This design was not fault-tolerant, and would stop working if the ring was disconnected; this is why it has fallen out of use today.
Mesh	These types of networks connected every node (computer or device) to every other node. For each device that was connected, more cabling was required, which was extremely inefficient. It did have the advantage of being fault-tolerant since there are multiple paths to each node. Mesh networks are becoming interesting in wireless networks, as there is no need for cabling and the benefit of fault tolerance is significant.
Star	Star networks are by far the most commonly-used type of network today (which use multiple stars connected together). The central node in a star is responsible for sending traffic to each outer node. This network is fault-tolerant to a point, but if the central hub or switch fails the network will not function.
Bus	Bus networks require all computers to be connected to the bus. If the connection is broken anywhere the entire network fails, which means it is not fault-tolerant. This type of network is not commonly used today, except in some network **backbones**.
Hybrid	Any topology that connects multiple topologies is referred to as a hybrid. An example would be multiple star networks connecting through a bus network backbone.

Backbone
The core of a network that connects the main parts together is called the backbone; it can also connect multiple networks.

Figure 58: Example Office Star Network



Source: Stephen Weese, 2020 based on Maksim, 2005.

In this example, multiple star networks are connected through a larger star network. Switches for the center for each small network at the bottom of the figure. They connect five computers (or other devices) to the uplink, which goes up to another switch. Each switch at the second level has four connections, so it also forms its own star. The router sits at the edge of the private network, filtering data to and from the internet. The router is also the center of a star with two connections going to local switches and one going out to the internet. This is a common design within office buildings. Each second-level switch would handle the data from one floor of the building, where each lower-level switch is place in separate rooms with workers.

Wired Meets Wireless

Wireless networks were made available to the public in the 1990s. These networks help connect devices to a larger network. Inevitably, that network eventually connects to wires. Though a device you are using may be “wireless”, to reach devices at other geographical locations the signal will eventually have to be translated and sent over a wired network. Wireless networks have advantages and disadvantages compared to wired.

Table 11: Wired versus Wireless

Wired network	Wireless network
Devices physically limited by cable	Devices can move more freely within range
Easier to troubleshoot	Can be difficult to troubleshoot

Wired network	Wireless network
Fastest data speeds	Medium data speeds
Much more secure	Easy to intercept signal
Must purchase cabling	Only wireless access point must be purchased

Source: Stephen Weese, 2020.

Obviously it is easier to carry your device around if it is untethered from a cable. However, if the signal is weak in wireless, it is often hard to find the cause and where the signal is strong in a room—it is impossible to tell just by looking. Throughout the history of wired and wireless networks, wired networks have always had the advantage in data speeds, though wireless is catching up. A wired-only network can only be accessed by plugging into a cable, which allows for greater security, while anyone within range of the signal can connect (or attempt to connect) to Wi-Fi. Wireless is often easier to set up without the need to run cables.

Wireless Technology

The IEEE, or Institute of Electrical and Electronics Engineers, is a professional association of engineers. They have provided technical standardization for many aspects of electronics for decades. One of these standards is for wireless networks.

The original standard, set in 1997, is IEEE 802.11. New, updated standards were then introduced by adding additional letters after the name. There have been over a dozen standards in use, but there are three that are still in prominent use today.

Table 12: 802.11 Modern Wireless Standards

Wireless standard	Speed	Frequency
802.11g	54Mbps	2.4GHz
802.11n	600Mbps	2.4GHz and 5GHz
802.11ac	3Gbps	2.4GHz and 5GHz
802.11ad	Up to 7Gbps	6GHz
802.11be	Up to 46Gbps	1GHz - 6GHz

Source: Stephen Weese, 2020 based on Maksim, 2005.

The 802.11g protocol is the oldest widely used protocol; you can see that its theoretical speeds are over ten times slower than the next protocol available. However, just as it is stated, these speeds are theoretical—in practice, most networks do not achieve these maximum possible speeds. The 2.4GHz channel (frequency) for data can travel farther and better through physical obstacles than the 5GHz channels, but 5GHz frequencies can carry more data. 802.11n is able to switch between both frequencies. The problem with further development is the backwards compatibility of the network cards and routers, as these must also support all previous frequency bands. The support increases the price of new devices until the backward compatibility is discontinued.

Keep in mind that for a network to use a particular protocol, all wireless devices must be able to connect. In other words, if you buy an 802.11ac Wi-Fi access point, all of your network devices must have network cards that support it. If not, you will have to buy new network cards to take advantage of the higher speeds.

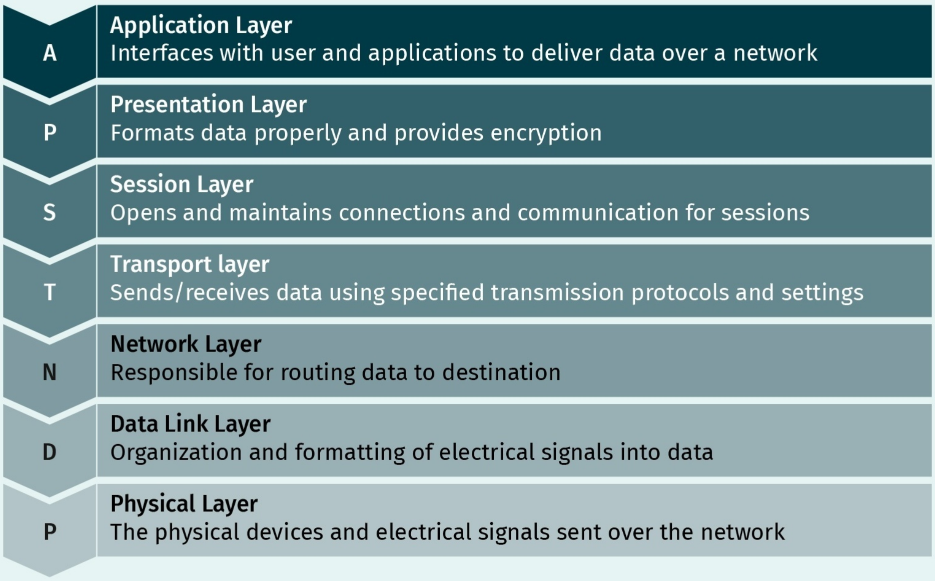
Figure 59: Wireless Access Point

Source: Pjpearce, 2011.

6.2 The OSI Model and TCP/IP

To explain the process of network communication, OSI, the Open Systems Interconnection standard, was created. It is used to create specific networking technologies and provide a map of how those technologies work. The main description of this is the OSI Model. It is divided into seven layers.

Figure 60: The OSI Model



Application Layer
Interfaces with user and applications to deliver data over a network

Presentation Layer
Formats data properly and provides encryption

Session Layer
Opens and maintains connections and communication for sessions

Transport layer
Sends/receives data using specified transmission protocols and settings

Network Layer
Responsible for routing data to destination

Data Link Layer
Organization and formatting of electrical signals into data

Physical Layer
The physical devices and electrical signals sent over the network

Source: Stephen Weese, 2020.

Here's a basic overview of how it works: the application layer communicates with applications that want to use the network (usually requested directly by a user). It takes the data and passes them to the presentation layer. Then the data are formatted into the correct encoding (for instance, Unicode). This is also where encryption is applied to the data. After that, a session is established with the other computer. The session layer establishes communication settings, such as **ports**, and synchronization of data. Once the settings are established, the transport layer handles the error checking and actual transmission of data, monitoring communications from end-to-end. The network layer is more specific; it handles the physical route of the packet each step along the way of the network. This is where IP addressing is mostly used. The data link layer exists to provide specifications for devices—such as Ethernet—that determine the way signals are sent electronically over the network, as well as standards for devices; this is where the hardware (MAC) address is used. The physical layer consists of the physical devices and wiring of the network, as well as the signal itself.

Port
In networking, a port is a virtual identification number for a specific data stream or type of data.

Application Layer

This layer is the closest to the user, yet it interacts with applications that request network access. Technically, this layer is for applications, not made of applications. It is the interface layer where applications make requests of the operating system to access the network. Once the request is accepted and delivered, it goes to the presentation layer. Applications that request network access can include internet apps such as browsers and other local apps using the network, such as MS Word. If you print a document to the network printer in the office, Word will request access through the application layer.

Presentation Layer

This layer prepares the data for being transferred. You might say that it being ready to be “presented”. Encoding happens here. For instance, if you are sending an email in Unicode, it will translate the symbols at this layer. Encryption occurs at this level. In other words, a plaintext or other unencrypted message is brought through a mathematical algorithm that created encryption. Without knowing the encryption key, the code is nearly unbreakable (though no code is perfect). The sending and receiving network devices share a key at the beginning of communication that allows for the algorithm to be reversed and the message to be turned back into the original.

Session Layer

Imagine you want to have a meeting with someone: you will set the time and place and some other requirements (e.g., bring your notebook). This is similar to a session for network communications. The start of communication is agreed upon, along with wait times for responses (acknowledgements) and how the session will be ended (terminated). Once communication is begun, lower layers handle the details.

Transport Layer

Inside a session, the transport layer does the important work of managing the data transfer between hosts. It makes sure that the data flows at the proper rate and performs some error checking. It also breaks down the data into chunks, or segments, that are sent out separately.

Network Layer

If you wish to mail a package to a friend, there are several steps. You put it into a box (the previous layers do this) and give it an address. This address is what the network layer uses to transport the data (your box) to its destination. It does this by using routing. You may remember a device called a router—this is what performs this task. At each step along the way, the address is checked and the correct path is chosen to send the data further along toward its destination. In TCP/IP this is where the IP address is used. Data sent at this level is organized into packets.

Data Link Layer

Once the data makes it to its destination network, it still must find the exact device it is looking for. The data link layer works at a more specific level than network. It uses hardware addresses to identify network devices. This is the layer where Ethernet specifications dictate how signals are formed as well. It includes local network error checking, including collisions. Collisions occur when two network devices send out a signal simultaneously, and they collide over physical media. Most networks use collision detection. This means after a collision is detected both devices are instructed to wait a random amount of time before sending again.

Physical Layer

Network devices are included in this layer such as hubs, switches, and routers as well as cabling. The signal itself is included at this layer, where zeros and ones in data are formed into a specific signal and placed on the network.

Data travels back and forth on a network from physical to application and back again. The OSI is a general model of how network data transmission works, but it is not a required list; different protocols use different concepts from the model.

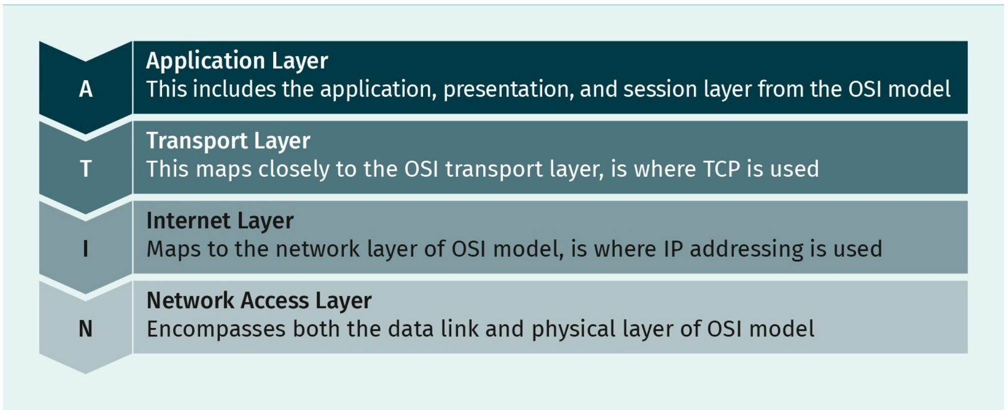
Ethernet

Ethernet is a set of standards (defined by IEEE 802.3) used for LANs and WANs. The word “Ethernet” is used commonly to refer to parts of a local wired network, such as Ethernet cabling (Cat 5, 5e, 6, 7 and 8), and Ethernet ports, where the cables are connected. The standards of Ethernet include the physical layer, which gives specifications for devices and signals, and the data link layer, where it defines how signals are organized.

TCP/IP and the OSI model

TCP/IP stands for Transmission Control Protocol/Internet Protocol. The transmission control protocol works at the transport layer, while the internet protocol works at the network layer of the OSI model. However, TCP has its own four-layer model of operating.

Figure 61: The TCP/IP model



A **Application Layer**
This includes the application, presentation, and session layer from the OSI model

T **Transport Layer**
This maps closely to the OSI transport layer, is where TCP is used

I **Internet Layer**
Maps to the network layer of OSI model, is where IP addressing is used

N **Network Access Layer**
Encompasses both the data link and physical layer of OSI model

Source: Stephen Weese, 2020.

TCP/IP is the **protocol** of the internet. That means that to access the internet, it must be installed on a device. Since it is also usable for LAN communication, most computers use TCP/IP as their primary and only network protocol. It consists of other sub-protocols that exist within it to facilitate communications. To use TCP/IP a network device must have three settings configured. These settings are 32-bit binary numbers (or 128-bit with the new IP version 6).

Protocol
This is a set of standards or rules to follow for communication over networks.

IP address

This is the unique address of the network device; there cannot be a duplicate on the same network. It is divided into two parts: a network ID (which identifies what network segment the device is on) and the host ID (which is the individual device number). A good analogy is that the network ID is like a street and the host ID is like a house number.

Subnet mask

This is a series of ones and zeros that identify which part of the address is the network ID and which is the host ID. IP addresses have the option of being split in different places depending on the need of the network.

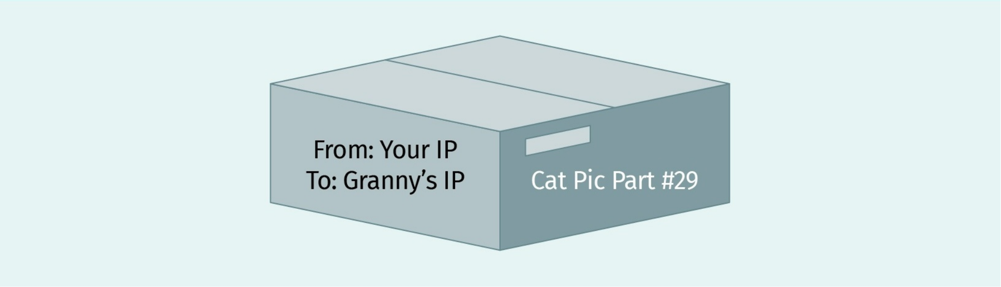
Default gateway

The gateway is the router or device that allows the computer to access other devices outside of its local network. If a computer is talking to a device on its own network it does not need a gateway, but once it wants to go outside of the local network it must use a gateway.

Packet delivery

Let’s return to our analogy of sending a box in the mail, with more details. Let’s say you are sending a picture of your cat over Skype to your grandmother in Canada. The data is formatted and presented for transport over the internet. Then a session is established. The picture of your cat data is then broken down into individual packets. Each packet is like a separate box of data. The box will have two addresses on it: the sender’s and the recipient’s.

Figure 62: TCP/IP Data Delivery



Source: Stephen Weese, 2020.

Granny’s IP address will be used to deliver the cat picture pieces to her. The packet goes to your router (the default gateway), which then sends it out through your internet service provider (ISP) to the internet. The internet is designed like a large road system—the routers keep deciding which road to take to get to grandma’s house. Eventually there are no turns left and the box is placed in her mailbox (i.e., her computer). The computer waits until all the packets have been received and then reassembles them back into the picture of a cat which is now on Granny’s screen.

Fault tolerance

TCP/IP is also fault-tolerant. During a session, it checks to make sure all packets are received and continually verifies the path. If, for some reason, a router along the path to the destination stops functioning, TCP/IP will automatically look for (and, most often, find) a different path to continue to deliver the data. This is why the internet never completely goes “down” as a whole; parts of it may stop working, but the rest of this vast network continues functioning without those parts.

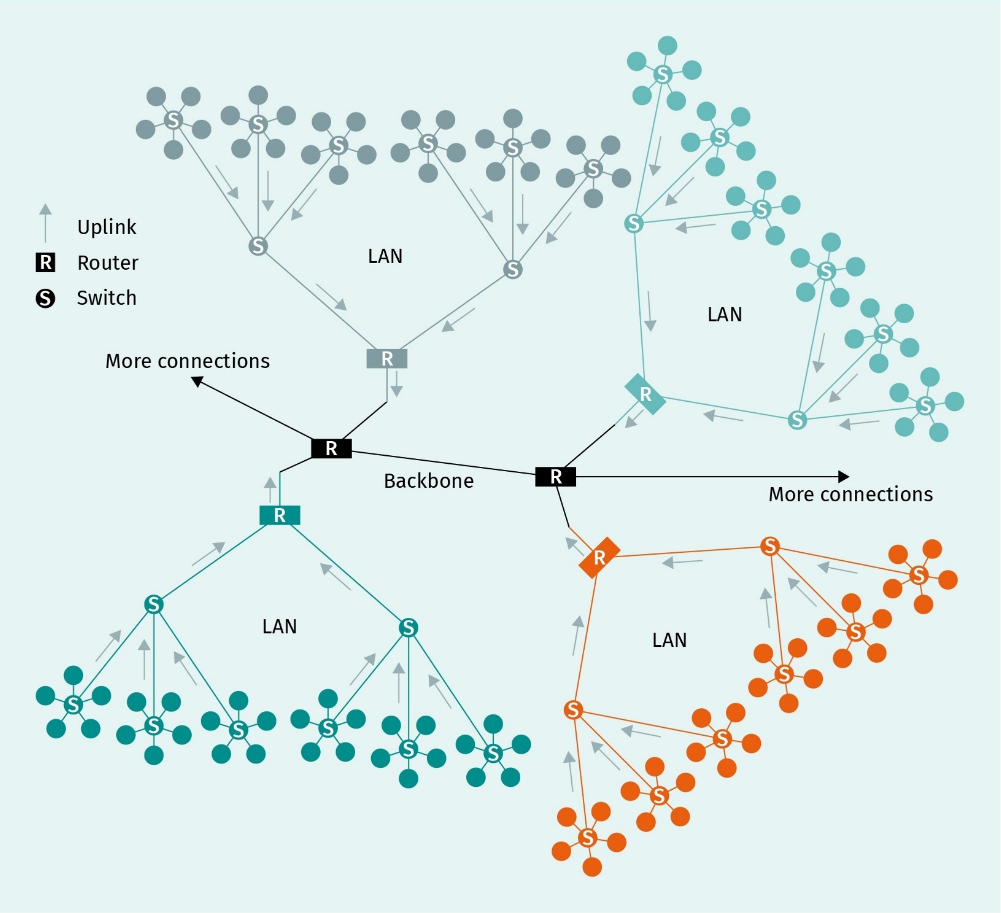
6.3 Internet Structure and Services

The internet is by far the largest computer network ever created. At its simplest level, it is a collection of LANs all joined together by routers and cabling all over the world. Using TCP/IP addressing, any computer on this huge network can talk to any other, thanks to the built-in data routing system.

The basic devices of the internet have already been discussed: the router, hub, and switch, as well as cabling. Once we leave the domain of the LAN, however, there is more complexity to how all of the devices are connected.

Individual residences and office buildings pay a telecom provider, or ISP, for a connection to the rest of the internet. If we imagine the internet as a series of buildings and roads, the cabling is the roads, the routers are the intersections, and computers and other devices are the buildings. If you have an individual neighborhood connected by roads, then the neighbors can all travel to each other. For the neighborhood to reach the rest of the road system, they must pay someone to build a road to connect all of their roads to it. The ISP (company) will make this connection and now the users are able to travel anywhere. Of course, since internet speeds are much faster than a road vehicle, this distance is covered in seconds (or even less).

Figure 63: LANs Connected to the Internet



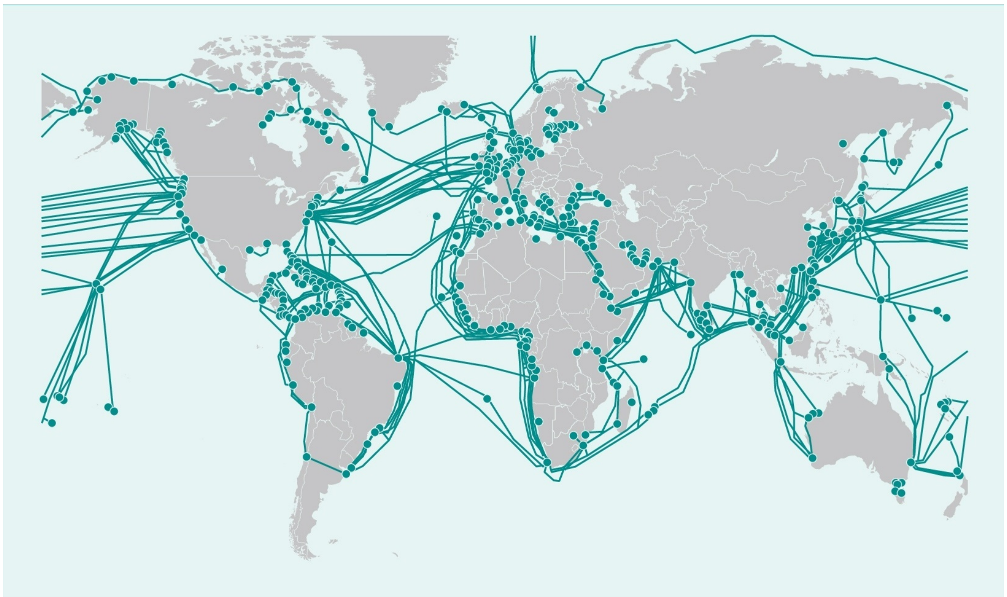
Source: Stephen Weese, 2020 based on Maksim, 2005.

As of 2022, there were over 46 billion devices connected to the internet, including PCs, smartphones, smart TVs, and vehicles (Steward, 2022). The cabling that connects the backbone together is a mix of copper cable and fiber optic. Fiber optic is much faster but it is a newer technology, so not all geographic locations will have the option of fiber.

Under the Sea

To connect all of the continents of the earth together in this network, kilometers of cable have been laid under the oceans using special equipment. Over 99 percent of intercontinental data is transferred by these cables, which include copper and fiber optic (Main, 2015). Fiber optic cabling uses light pulses to simulate the values for 0 and 1 over long distances. Copper cabling uses electrical signals that are slower and can degrade over distances; fiber optic signals travel much more reliably.

Figure 64: Transoceanic Communication Cables



Source: Mahlknecht & OpenStreetMap Contributors, 2015.

Above the Sea

In addition to communication via undersea cables, there is also communication above sea level. The decisive factors for communication technologies are their speed, worldwide availability and network coverage. Communication satellites following different orbits around the Earth are essential to worldwide availability and maximum network coverage above the sea and on land. One shortcoming, however, is so-called latency, i.e. the time it takes for a signal to travel from the Earth's surface to the satellite and back to Earth. Even if these signals travel at the speed of light, latency is noticeable when opening a website. To reduce this effect, satellites have to orbit at the lowest possible level, which also means that they must orbit in close proximity to the atmosphere. Even though the air here is thin, it slows the satellites down, so the satellites have to be able to maintain their low orbit. requires additional energy not to crash and burn up in the Earth's atmosphere.

The second issue is the higher number of satellites needed to cover the Earth completely. With 2453 satellites in Earth orbit (as of mid-2022), SpaceX is by far the largest satellite operator in the world(SpaceX, 2022). Starlink is a satellite network operated by the US space company SpaceX that is intended to provide worldwide internet access in the future, i.e. the provision of internet access with short packet turnaround time (latency) and provision in areas where previously internet connection was available or the connection was of poor quality (Starlink, 2022). Finally, suppose we restrict communication to the land masses and the shore area. In that case, our Global communication Systems for Mobile communication (GSM) are to be mentioned, and new developments in the Next Generation Mobile Networks (NGMN) project, including mobile communication standards such as Long Term Evolution (LTE), have to be recognized.

How the Internet Works

We have explored how internet devices are connected together and examined how data is divided into packets with IP addressing for delivery, yet there is much more to the operation of the internet.

Hardware

The hardware of the internet includes connecting devices such as routers, cabling, network cards, computers, smartphones, and any other physical device that connects to the internet, whether by wire or wireless.

Protocol

The internet uses the TCP/IP fault-tolerant protocol to deliver data to and from every node (device) on the network. Devices using TCP/IP must have an IP address, a default gateway, and a subnet mask (called a “prefix” in IP version 6).

Services

Computers and devices connected to the internet typically have a role of either a “client” or a “server.” A client is a device using services provided online, whereas a server is a dedicated machine that provides data services to clients. Servers are typically very powerful computers with a high-speed internet connection.

The World Wide Web

Now mostly called the “Web,” this is a special set of services that is delivered to a web browser. Basically, anything you can access in a web browser is part of the web. The web was created primarily to communicate formatted documents with pictures and hyperlinks using a simple markup language called HTML (Hyper Text Markup Language). Eventually, it evolved into what is used today including video services and websites with interactive JavaScript and connections to databases of information.

Other web services

Besides “surfing the Web,” internet users can access other data streams and services. Examples include music streaming services, online games, and direct video chat software. These internet applications do not require a browser and are therefore not part of the web.

DHCP

To access these services, an IP address is required. However, users are not able to make up any address they want and must acquire a proper address from somewhere that properly identifies their network. Fortunately, this is part of the package that an ISP provides. It will automatically provide you with an IP address using a service called DHCP. This service, Dynamic Host Configuration Protocol, provides not only an IP address, but a subnet mask and default gateway as well. It also provides you with addresses for DNS servers. By default, desktops and Wi-Fi devices search the local network for a DHCP server and request configuration. Once configuration is given, the communication can begin.

DNS

Another very important internet service is the Domain Name System. As we have mentioned, every internet device requires a 32-bit IP address to transmit and receive data. Imagine if human users had to memorize and type in 32 zeroes and ones just to access a website? This would prove quite cumbersome. Fortunately, DNS is a translation service that maps names to IP addresses. When you type in or click a link to a Web address, you are accessing something called a URL, or Uniform Resource Locator. It includes the full DNS name of the computer.

Figure 65: URL or Web Address

https://www.iu-fernstudium.de/studienplatz-sichern/index.htm

https://	Hyper Text Transfer Protocol (Securet
www.	the computer name
iu-fernstudium.de	the domain name
/studienplatz-sichern/	folder location
index.htm	file to open

Source: Stephen Weese, 2020.

A URL actually contains several pieces of information. The first part indicates the protocol used for communication. HTTP and HTTPS are both part of the TCP/IP protocol suite—they exist specifically to transfer data for a web page. The secure version uses encryption to protect private data. Many sites use encryption—you should never type in personal data on any site that is not using HTTPS (this is often indicated on a browser by a small padlock icon). Since the beginning of the Web, many people thought that you must simply type “www” to connect to a website. However, the web server could have any alphabetic name—“www” was simply the most common. The computer name identifies which computer you are communicating with on a network. The domain name, for instance, “iu-fernstudium.de”, identifies the network that the computer belongs to. When you put both of these together you get the hostname, such as “www.iu-fernstudium.de”. A hostname is very important; it is what the DNS service uses to look up the correct IP address.

You can imagine DNS like a large internet phone directory—you know the computer’s name, but you want to know its number. These special servers are placed all around the world to accept lookup requests. When you want to view a website, DNS automatically goes into action. If the request takes more than an instant, many web browsers will display in the bottom status window the message: “looking up host.” Once the IP address is retrieved, your computer will remember it for a specified amount of time in the DNS cache. This means it won’t have to keep looking up the same address for sites you visit frequently.

Once you have connected to the computer you are looking for, it needs to know exactly what file you wish to open. After the computer hostname in the URL is a file folder (or multiple folders) that indicates where the file is located, followed by the filename itself. That file is retrieved and sent out over the network to your web browser to be displayed.

A file or folder location is optional in a URL; if it is not provided, the web server will provide the default file it was configured to send.

The Growing Internet

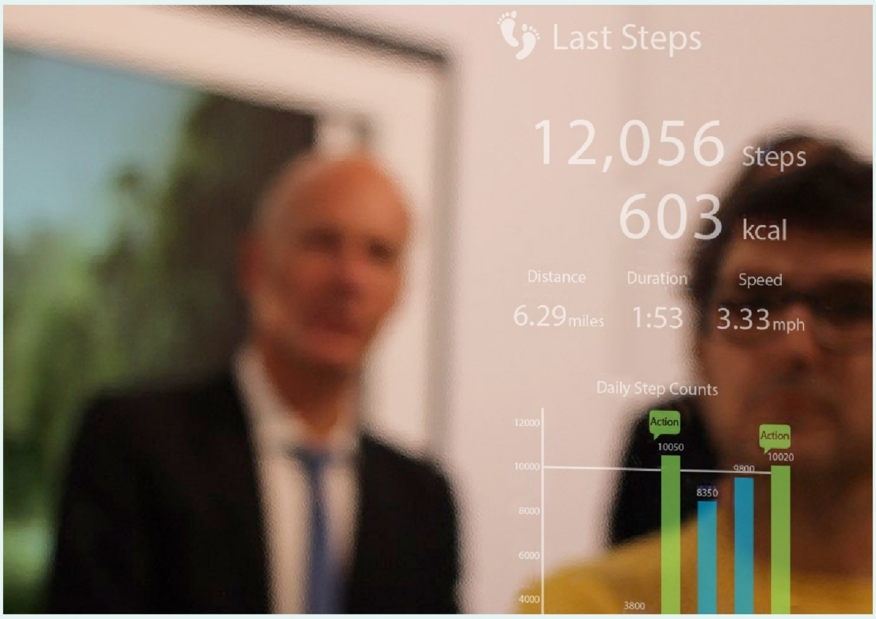
The internet is providing new services every day as they are invented. Companies now offer “cloud services” where users can store all of their documents and photos online where they can be accessed and synchronized on multiple devices. More smart devices are constantly being added to become part of the Internet of Things (IoT). New users in remote locations are being connected, enabling them to communicate all over the world. The internet has continued to grow by leaps and bounds since its creation, and it shows no signs of stopping.

6.4 The Internet of Things

At first, the devices that were connected to the internet were standard computers—they were divided into powerful servers and individual client machines. As the internet grew, the benefit was observed to connecting other types of devices to the internet: “non-standard” devices is a way to describe them. These have become the Internet of Things (IoT).

There are already over a billion of these devices connected to the internet. They include many “smart” devices. Light bulbs that can be controlled remotely are a type of smart device. Regular and self-driving vehicles are also connected to the internet. Some devices can provide feedback on their performance. A home appliance such as a smart refrigerator can tell you recipes, whether you are running low on eggs, how much power it is using, and an optimal temperature setting. Wearable technology such as smartwatches are part of the IoT as well as many medical monitoring devices. Security cameras all over the world are connected to the internet so users can access them from anywhere to see what is happening. Televisions are now smart and can access the internet for you, along with wireless speakers. People can work out in front of a smart mirror so they can watch themselves exercise while instructions and calorie counts are displayed on the screen. The mirror could also display the day’s weather or stock reports.

Figure 66: Smart Mirror



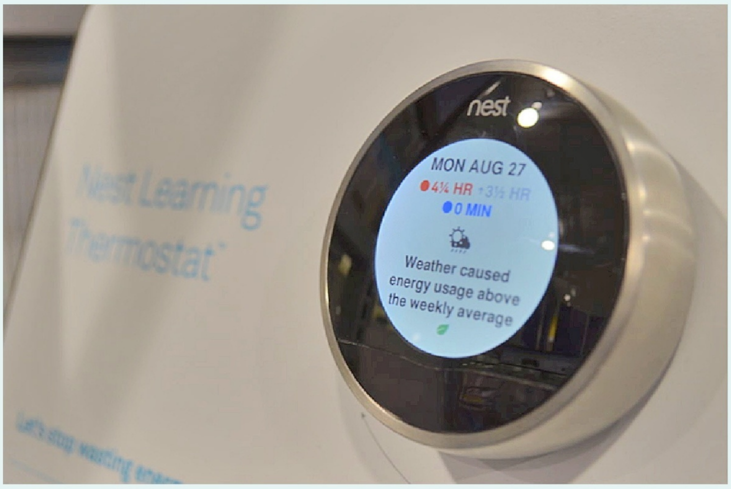
Source: Pesce, 2014.

Uses for IoT Devices

There are many different reasons to connect a device to the internet. One of the reasons is that these devices can be programmed to function in different ways, whereas a standard device would have to be continually manually adjusted.

A good example is having an IoT home thermostat system. It can be programmed to change the temperature settings for different days of the week at different times. A maximum and minimum temperature can be added to always keep a location between two temperatures, switching between heat and cooling without you having to worry about which is running. Not only that, but with a smartphone app you can change the temperature settings for your home without even being there. The Nest thermostat will track your energy usage for you and give you monthly reports over the web. It also can learn your preferences for temperature and create a weekly schedule automatically.

Figure 67: Nest Thermostat



Source: Raysonho, 2014.

IoT devices can store data and create web reports of statistics like your heat usage, but they can also retrieve data from the internet.

A smart refrigerator can keep track of what food you have and download recipes that are made of those exact ingredients. In fact, many smart refrigerators can use the expiration dates on food to recommend recipes to use up those items first. Using a home network, devices can even communicate with each other—the refrigerator could recommend a recipe that, if chosen, would then send a command to the oven to preheat for cooking.

Smart devices can also customize your home experience—including setting mood lighting and playing music. Not only that, personal AI assistants can talk to you, answer questions, and control household devices like televisions and speakers. This is possible due to average household internet speeds being fast enough to handle larger data streams. Without having to type or use a computer, you can simply talk to your home assistant and ask for light, music, information from the web, or to watch a specific sporting event on TV.

Security and IoT

Connecting billions of devices to the internet also presents a security issue. One issue is similar to a previous problem when Wi-Fi became popular. Home internet users purchased a Wi-Fi access point but never changed the default name or password; at the time, the default passwords were the same for each manufacturer. This enabled easy access to millions of home networks, basically “free internet” for anyone who took advantage of it. Eventually internet service providers started pre-programming Wi-Fi devices with complex passwords which mostly eliminated this problem.

Recently, people are buying more wireless security cameras and are doing the same thing: leaving the default passwords in place. There are millions of unsecured security cameras operating all over the world, so many that it has even created a new hobbyist community of users who randomly watch different cameras for entertainment.

Intercepting data from devices is a security risk, but more concerning is that these devices could be completely taken over without proper security measures. In the worst case it could mean that a self-driving vehicle is taken over, or a device like an oven is used to cause a fire or other property damage—including loss of life. Even simple, seemingly harmless devices can be used for nefarious purposes. Since IoT devices are given an IP address, they are able to make requests of other devices online. This means that millions of them could be compromised to make something called a botnet—an army of internet robots that can be controlled by a single user or program. Botnets are most often used in a DDoS attack—this is a distributed denial of service attack, where a server is flooded with so many simultaneous requests for data that it cannot answer them all and either stops working or becomes so slow that it is mostly unusable.

Other concerns include companies using these devices to gather data on users. These devices could learn your shopping, eating, sleeping, and movement habits. Parents are concerned that smart toys might be spying on children or, worse, trying to influence them somehow. There have been few standards set at this point for IoT devices to prevent such things.

IoT Standards

Companies like Microsoft and organizations like IEEE are working on creating standards that will help improve the function and development of IoT devices, while addressing security issues as well. One of these is the protocol 6LoWPAN. This protocol is designed for low-power IoT devices that may be very small and run on a simple battery—normal IP communication could eat up batteries quickly, so 6LoWPAN implements compression along with other power-saving settings (Leverage, 2020).

IoT and the Future

The addition of new devices to the IoT continues every day. New types of devices to connect are being created. This means that there will be more security issues and more problems, as well as new solutions for them. It is likely that, in the near future, human beings themselves will be connected to the internet through some type of brain interface—several organizations are working on this. The U.S. Army is creating brain chips to help with PTSD (Tucker, 2014), and Elon Musk's Neuralink is developing a brain interface to help with medical conditions that will also be able to interface with networks (Alexiou, 2020).

As more devices are added, computer programmers will find that more and more of their work is not writing applications for computers or even smartphones, but for IoT "smart" devices.

SUMMARY

Local area networks (LAN) use standard technologies such as Ethernet and TCP/IP, as well as standard devices such as routers, switches, and hubs. Most cable used is now Cat6 or Cat7 Ethernet cable, capable of speeds of 1Gbps or more. Local computers and devices (nodes) connect through a switch or router using a “star” topology. This allows multiple computers to network together and to connect with other networks using an uplink.

Wireless networking, or Wi-Fi, enables devices to connect to networks by using electromagnetic signals through the air. However, these signals can be interrupted, unreliable, and less secure. The advantages include mobility for devices and ease of connection.

Network communications are based on the OSI model which represents the different layers that network data passes through when sent over a network. The layers are Application, Presentation, Session, Transport, Network, Data Link, and Physical. The main protocol used for modern network connections, including on the internet, is TCP/IP. TCP/IP uses routing and IP addressing to deliver data that is broken up into packets to its destination where it is reassembled.

The internet is a massive group of interconnected networks that reaches all over the world. Communication is delivered through local Ethernet networks out through routers to ISPs who are connected to internet backbone connections—including massive undersea cabling between continents. Older types of copper cabling are being replaced with newer fiber optic cable. To make internet communication easier for users, DHCP is used to automatically configure computers and DNS is used to assign easy-to-remember host names to servers on the internet.

The Internet of Things (IoT) consists of non-standard devices connected to the internet such as smart appliances and self-driving vehicles. There are billions of these devices connected already. Protocols are still being developed for the IoT and there are security issues such as devices being hacked and controlled for various purposes. Since many new IoT devices are added every day, the demand for programmers to write code for these devices is increasing.

UNIT 7

SOFTWARE

STUDY GOALS

On completion of this unit, you will have learned …

- how BIOS works.
- about modern file systems.
- about application software for operating systems.
- about the concept of embedded systems.
- about the process of software development.
- what the functions of an operating system are.

7. SOFTWARE

Introduction

Microsoft was brought into existence by Bill Gates and Paul Allen in 1975. In the 1980s, IBM requested that they create an operating system for their personal computer. They bought a basic disk operating software, 86-DOS, from a smaller company and rewrote it to work on the IBM-PC (Curtis, 2014). This simple operating system was text-based: there were no icons, no desktop, no windows. It showed you a command line where you typed in instructions to the computer. The name DOS comes from Disk Operating System, meaning it would allow the user to access files on floppy and hard disks.

Gates decided to sell the OS to IBM but keep the copyright; his idea paid off when other computer manufacturers also wanted to use his MS-DOS (Microsoft DOS) product. Soon, Microsoft was a major player in the operating systems market. In 1989, they released Microsoft Office, a suite (collection) of applications that was used for office productivity. The names Word, Excel, and PowerPoint soon became recognized as the international standard for digital office files.

Microsoft released their first successful version of Windows in 1992: Windows 3.1 (Windows 3.0 was released in 1990 but was not nearly as well-adopted as 3.1.). This led to a great success where companies moved away from command line operating systems and toward a graphical system. Successful versions of Windows followed: 95, 98, NT, 2000, XP, 7, 8, and 10 (“9” was not used in order to disambiguate between 95 and 98). As recently as 2013 Windows operating systems were on over 90 percent of the worlds desktop computers (Liu, 2020). Microsoft Office and Windows are still going strong today, holding the majority of market share for both office applications and operating systems.

7.1 BIOS and Operating Systems

A computer system is comprised of both hardware and software: the physical devices that make up the computer and the instructions to use it. When a computer is activated, there must be a way for it to find and execute software instructions. If the computer always performed the same task every time it would be very easy—it would just be built into the hardware to always execute the same instructions. One of the advantages of having a computer is that it can perform many tasks. Every time it is turned on, a computer can be used for something different.

When a computer is turned on (or booted up) it knows nothing from its previous usage; it is a blank slate. It needs to know where to start. The BIOS provides a starting line for a computer every time it is turned on.

Figure 68: BIOS: The Starting Line

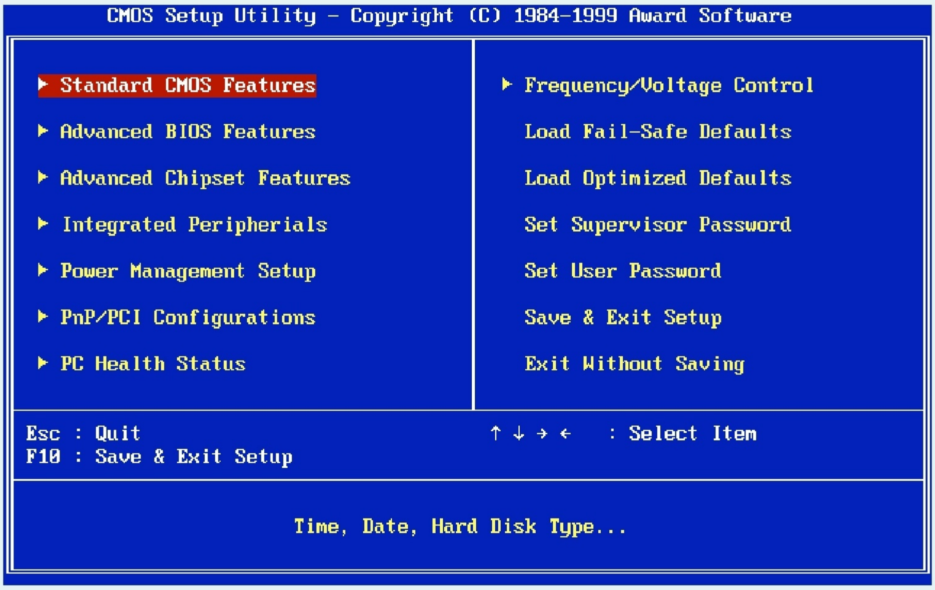
Source: Gladden, 2017.

Once you turn on a computer it is hard-wired to look for instructions on the BIOS chipset. At this starting line, it executes the instructions on the BIOS chip. The acronym BIOS stands for Basic Input Output System. Before the computer can do any input and output, it has to start up. Computers are shipped from the manufacturer with instructions already placed on the BIOS chip. These can be updated, or “flashed”, from time to time as new updates are released.

One of the very first things a BIOS chip will instruct the computer to do is detect and test hardware that is connected to it. Older systems from the 1980s and 1990s had to be configured ahead of time for the BIOS to find devices; current BIOS systems auto-detect most if not all hardware. BIOS performs a POST, or power on self-test to check that the required hardware is there and that it is functioning properly. Required hardware includes things like RAM, CPU, and storage. It quickly sends some queries to the hardware parts and if the responses are correct, they pass the test.

The BIOS then directs the hardware to the location of the operating system. The BIOS is “Basic” and can provide the user with simple testing and configuration functions, but it cannot run applications or display advanced graphics. Only an operating system can do these things. The BIOS will either already have been configured with the location of the OS, or it will start searching in the most likely locations (like hard disks) for one. Once an OS is detected, the rest of the boot process finishes—a copy of the operating system (like Windows or MacOS) is loaded into memory. If no OS is detected, the boot process will stop and return an error. This may require you to go into the BIOS configuration screen, usually accessed by hitting a special key during the boot process.

Figure 69: BIOS Configuration Screen



```
            CMOS Setup Utility - Copyright (C) 1984-1999 Award Software

 ▶ Standard CMOS Features              ▶ Frequency/Voltage Control
 ▶ Advanced BIOS Features                Load Fail-Safe Defaults
 ▶ Advanced Chipset Features             Load Optimized Defaults
 ▶ Integrated Peripherials               Set Supervisor Password
 ▶ Power Management Setup                Set User Password
 ▶ PnP/PCI Configurations                Save & Exit Setup
 ▶ PC Health Status                      Exit Without Saving

 Esc : Quit                           ↑ ↓ → ←    : Select Item
 F10 : Save & Exit Setup

                        Time, Date, Hard Disk Type...
```

Source: Kephir, 2019.

BIOS configuration screens have not changed much in the last three decades. They provide a basic text interface for the user to identify devices like hard drives, provide other basic hardware configuration settings, and often have password protection.

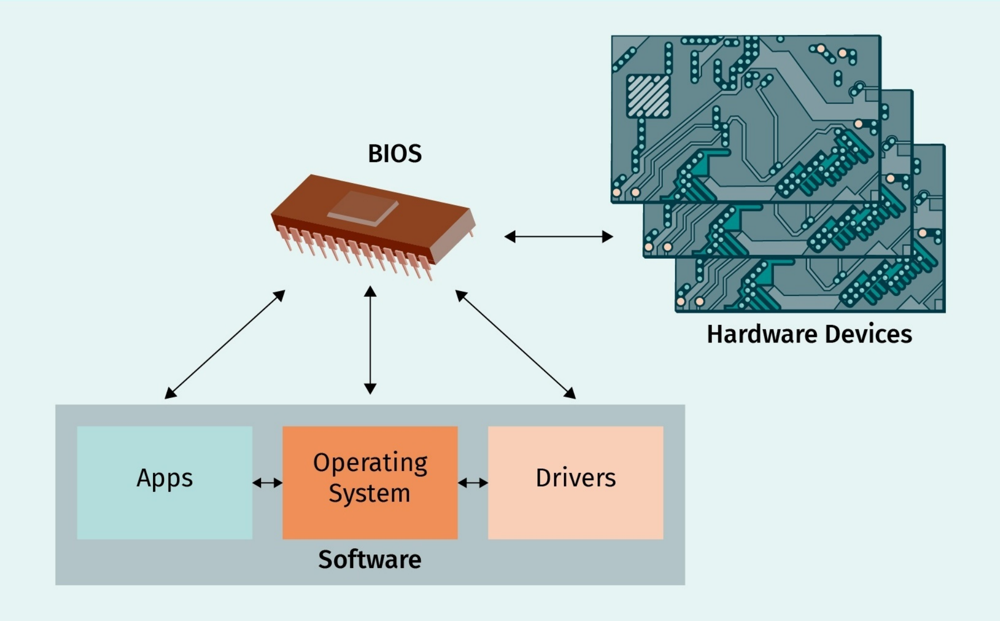
CMOS
This stands for Complementary Metal Oxide Semiconductor: a chip that stores data using battery power.

The **CMOS** is another chip used in computers. Originally, it contained all of the BIOS configuration information. A drawback of this chip is that it required a small battery connected to the motherboard and when the battery died, all the settings were lost. Today's computers use flash memory which does not need continual power to preserve the settings. The CMOS still exists in personal computers today, but its main function now is to preserve the system clock (current time) for the computer. If the CMOS battery dies in your modern computer, you will have to keep resetting the clock every time the PC is started until you replace it.

Input and Output

The BIOS, once it identifies the hardware on a computer, provides access for software (operating systems and applications) to access that hardware. BIOS chips provide channels to software to allow direct access to devices without having to know hardware addresses. This is helpful because sometimes hardware addresses change, as well as the actual PC components. It ensures seamless transition where the BIOS knows where the hardware is and software can always rely upon it to access devices. Since the BIOS chip has instructions on it, it may seem similar to software. However, it is in its own separate class: firmware. Firmware is hard-coded instructions built into a device that usually cannot be changed (though it is updated periodically). This is good because the BIOS needs to be the same every time to start up the computer.

Figure 70: BIOS Input and Output



Source: Stephen Weese, 2020.

The BIOS handles hardware requests from the operating system, applications, and drivers: these are all types of software. Applications make hardware requests to play sounds and display things on the screen and must actually pass through the operating system first. Drivers are used by the operating system to translate information into formats that a device can use. Input and output travels between the BIOS.

Applications use an Application Program Interface (API) to make requests of the hardware. The API interacts with the operating system. The operating system uses drivers to speak to the BIOS which then relays the requests to the hardware. Application programmers will have to learn the proper API procedures to communicate with the operating system that they are working with, be it Windows, Android, iOS, MacOS, Linux, or something else.

Operating System Function

The BIOS gets things started and then gives control to the operating system or “OS”. However, the BIOS is in continual use by the OS to access hardware after the full boot up of a computer is completed. The DOS operating system provided by Microsoft allowed users basic access to a computer’s functionality. One of its primary functions was disk access—this is crucial to an operating system. Along with that, an operating system has three main functions:

1. Providing an interface
2. Controlling hardware
3. Running applications

Disk storage is included in hardware, but disks need much more than just a driver; they need a complete file system that manages and stores information. Operating systems implement file systems for this purpose.

File systems

The MS-DOS operating system used a simple system called FAT—file allocation tables. This was used by "formatting" a disk. Both floppy and hard disks were formatted with FAT (and sometimes still are today). Formatting a drive prepares it for use; it also erases all the current content on the drive. FAT divides a disk into clusters, and the contents of the clusters are stored in the file allocation table. For instance, one of the clusters will have the beginning of the OS and the BIOS will be pointed there to start the system. The original FAT, now referred to as FAT16, uses 16-bit addressing, so a maximum of 65,536 clusters could be created. Later, as hard drives grew in size, it was replaced by FAT32, with a 32-bit address for over 260 million clusters.

File tables identify files by file name and their starting location on the drive. FAT systems put files in the next available open space. Since earlier files could be deleted, this ended up putting parts of files all over the drive and led to drive **fragmentation**. File systems and operating systems go together; every operating system has a primary file system.

Fragmentation
This is when files spread out across different hard drive locations, causing slow access times.

Table 13: Primary OS File Systems

Operating System	Primary File System	Features
Windows 10	NTFS—New Technology File System	File encryption File security Large file sizes Large amount of files User quotas Can resize volumes
MacOS 10.13	APFS—Apple File System	File encryption File security Large file sizes Large amount of files Improved access speed On-demand volume size
Linux (various types)	ext4—Fourth extended file system	File encryption File security Efficient file organization Large file sizes Large amount of files Fragmentation protection

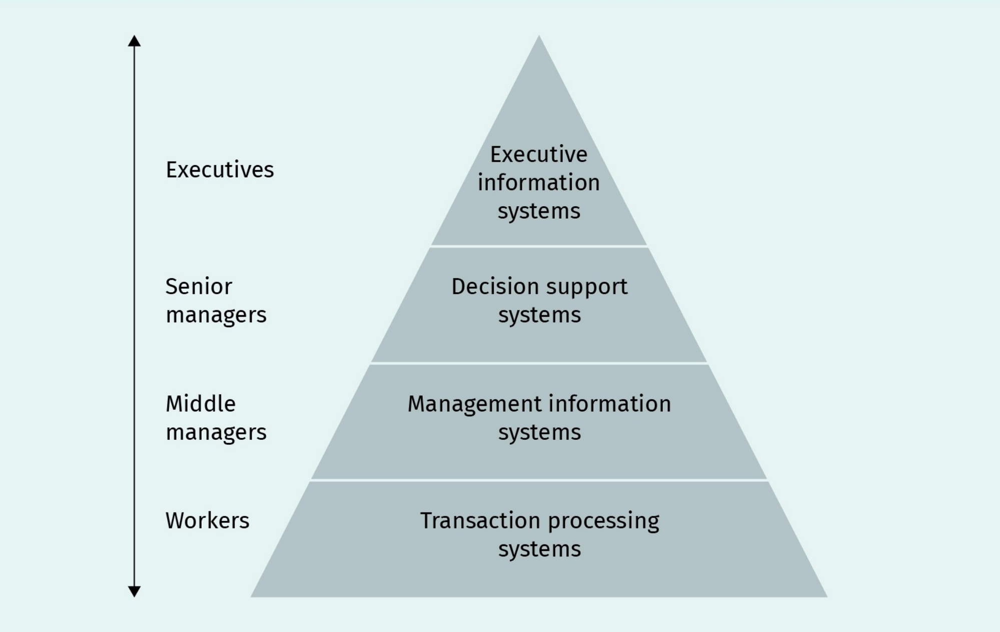
Source: Stephen Weese, 2020.

Computers with these preferred file systems have their primary hard disk partition or volume (a unit of space) formatted with that system. Windows must be installed on NTFS, for example. An issue can arise where portable external drives cannot be used between operating systems. For instance, Windows cannot read a hard drive formatted for a Mac.

7.2 Application Software and Information Systems

Software applications are used to perform specific tasks on a computer such as writing a document, listening to music, playing a game, or retouching photographs. In a business environment, many applications are used to manage information. These applications are part of an information system.

Figure 71: Information System Levels


Executives
Senior managers
Middle managers
Workers
Executive information systems
Decision support systems
Management information systems
Transaction processing systems

Source: Kimble, n.d.

In a traditional work environment, there are many levels of management and differing types of information needs. Executives deal with the highest level of decision-making of a company. It is rare that an executive would deal directly with a worker—usually they deal with employees in senior or middle management. Their needs are different than a senior manager who is entrusted to implement high level decisions made by the executive level. At the middle manager level, management of workers is key. They are assigned tasks and evaluated on those tasks. At the worker level, a task management system is essential to track exactly which employees are working on which job as well as deadlines and progress. Information system software falls into several different sub-categories depending on the needs of the organization.

Parts of Information Systems

Information is the considered result of processed data; it has been organized to provide insights into the subject of focus. An information system consists of the raw data itself that exists in the system. Software tools can provide analysis of the data and present it in different ways for user interpretation. This means that software is part of the information system, and so are the people using it. Of course, the software needs hardware to run on, so it also is part of the system. In such a system, it makes sense to assign tasks properly because a computer can perform many tasks more efficiently than a human, but not all tasks. Things like data searching are perfect jobs for computers but difficult for humans, especially with very large data spaces. This means that there is some kind of business organization that makes decisions about how to use the hardware, software, people, and data that are available. This organization is also part of information systems.

MIS

Management Information Systems (MIS) is a classification of information system that helps facilitate management within an organization. Highlights of MIS software include:

- managing internal files
- sorting company data
- creating action plans
- tracking inventory
- managing budgets
- personnel management

MIS software is used by lower and mid-level management (Kimble, n.d.).

TPS

Transaction Processing Systems, or TPS, handle operational transactions within a business. These are mostly used by the workers (bottom level of pyramid). Here are some examples of what these systems do:

- Manage payroll
- Process customer orders
- Track inventory/stock
- Data validation
- Funds transfers

Data from these systems is often used in higher-level systems (Kimble, n.d.).

DSS

A Decision Support System (DSS), is primarily designed to look to the future and help make informed decisions. A key of a good DSS is that the information is presented in an easily understandable way. These systems often have multiple ways to present data including in graphs and charts. DSS systems provide information such as:

- revenue predictions,
- hiring needs,
- inventory analysis, and
- future sales.

This system is most often used at the middle management level and above.

EIS

An Executive Information System (EIS) is similar to a DSS but it is specifically designed to be used by the executive level of the information systems pyramid. EIS data and analysis focuses on the whole company—the “big picture” level. Executives can use this data to determine future decisions for the company. EIS systems can

- provide overall performance analyses
- compare productivity between departments
- display market research data
- predict future performance

These systems are designed for executives who may not necessarily have technical skills; thus, a good EIS must be intuitive to use.

OLAP

Online Analytical Processing (OLAP) involves a high performance data analysis system. It is designed to query data across multiple tables and sources and provide a fast and detailed result. These systems are often used for

- sales figures
- product information
- product comparisons
- employee information

These systems are good for **data mining** and creating reports. They are frequently used by middle management.

Data mining
This is the process of producing useful information from large amounts of data, often from multiple sources.

7.3 Applications

There are two primary types of software: operating systems and applications, often referred to as “apps.” Operating systems allow use of the computer hardware, while apps are designed to perform specific tasks. You might think of app as a tool; you use the tool for the task you want and then you put it away until next time. Even though users can open up many apps at once, humans can only perform so many tasks at a time.

Apps for Every Purpose

There are many different types of applications. Apps are not only used on personal computers, but they are widely used on smart phones. They have many different characteristics and work in different ways.

App sizes

Applications can be small; sometimes they are just a few KB. Apps like a to-do list or notepad don’t use much memory (around 200KB). Some applications are medium-sized, like Zoom for video calling. These apps are in the MB range, around 50 to 100MB. Other apps, like Adobe Photoshop, perform many functions and take up much more space, often over 1GB. These apps will be divided up into several files that contain the program code and only the parts that are needed are loaded into memory. For instance, if you were editing a photo and wanted to use an artistic filter, Photoshop would load those filters into RAM. Other parts of the program not in use would remain in the files until needed. Some apps are very large, and many of these are video games. Games include 3D models, textures, pre-made video (cut scenes), and large worlds to explore. They can take up many gigabytes of space: the popular game World of Warcraft takes up more than 70GB of space on the hard drive. Just as with other large apps, it is split into separate files and the ones that are needed are loaded into memory. Some applications come in “suites,” or collections, like Microsoft Office or the Adobe Suite. These can take up very large amounts of space since they are several apps installed together.

App operating systems

Each operating system has a different API to communicate with the hardware. That means that once an app is written for a specific OS, it will only work on that OS until it is rewritten for a different operating system. This is the reason that some applications are available for both Mac and PC, and some are only available for one or the other. Final Cut Pro, the video editing app, is only available for the Mac OS, for instance, while Adobe Premiere Pro is available for both.

The same holds true for smartphones—apps must be made for either Android or iOS, but they can't be made for both. The version of Facebook made for Android will not work on an iPhone.

Since Windows and Android have the highest market share for PC and phones respectively, most software companies will make their app for these platforms first. They may decide to also make a version for other operating systems as well.

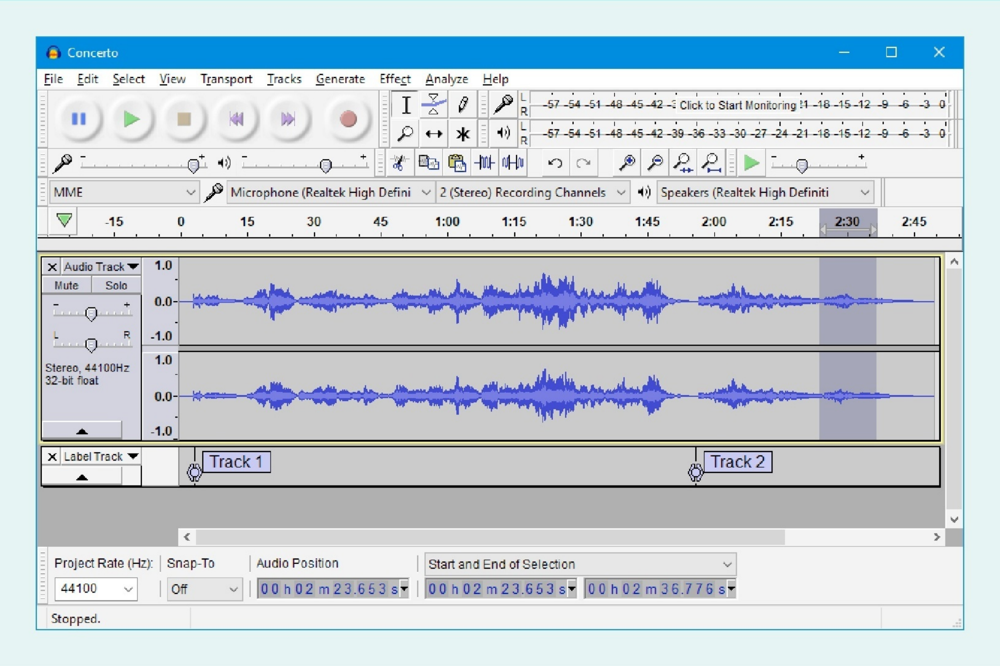
Free and paid apps

Anyone with basic knowledge of programming and a sufficient amount of time can create an app. However, that does not mean you should trust your valued hardware with them. Apps found through verified sites that are tested to be stable and virus-free are the best choice.

Freeware

Some apps are made to be given out without payment. Sometimes their developers will ask for donations, or they just want to make something to contribute to the software community. A good example of this is Audacity, a free audio editing software. Audacity is “open source," meaning that the program source code is also shared and people are free to modify it how they wish. The Audacity website also has a section where you can choose to donate if you like the program.

Figure 72: Audacity Open Source App


Concerto
File Edit Select View Transport Tracks Generate Effect Analyze Help
Click to Start Monitoring
MME
Microphone (Realtek High Defini
2 (Stereo) Recording Channels
Speakers (Realtek High Definiti
-15 0 15 30 45 1:00 1:15 1:30 1:45 2:00 2:15 2:30 2:45
Audio Track
Mute
Solo
Stereo, 44100Hz
32-bit float
1.0
0.0
-1.0
Label Track
Track 1
Track 2
Stopped.

Project Rate (Hz): 44100

Snap-To: Off

Audio Position: 00 h 02 m 23.653 s

Start and End of Selection: 00 h 02 m 23.653 s – 00 h 02 m 36.776 s

Source: Audacity Team, 2020.

Adware

Another type of app is adware. This software is sponsored by advertisements. Usually you can use all the features of the software but there will be occasional ads or a banner ad at the top or bottom. You might also be offered the opportunity to get rid of the ads if you buy the sofware.

Off-the-shelf

Before the internet was widely available, people went to computer stores and bought apps in boxes on CD-ROMs. Once you bought the software you were free to use it according to the license terms on the box. Usually, this meant you could install and use it personally on one or two devices. This model is still the same, except users can now purchase and download apps directly from the internet. This entitles the user to one specific version of the software. If a new version comes out, then the user must pay to buy it if desired. This model suffered greatly from piracy where people who would not or could not pay the cost of the software found ways to circumvent the copy protection. This problem eventually led to the subscription model.

Subscription apps

A newer model for apps is the subscription model. It might be considered similar to "renting" an application. You are able to install it on one or more machines, and use it as much as you wish for a monthly fee. There are three main benefits to the subscription model. First, it prevents piracy by continually verifying your subscription through an internet connection. Second, it makes software more affordable for users. Instead of paying $500 for a version of the software, a user can pay $30 a month for as long as they can afford it. This takes away some of the incentive for piracy as well. Third, when a new version comes out, subscriptions are designed to automatically upgrade. This means that a user is not tied to the purchase of an obsolete version but always has access to the newest and best software.

Malware

Some software also has some destructive or nefarious code. This makes it "malware," coming from the Latin word "mal," meaning "bad." Many websites offering "free" software are actually distributing malware. This includes viruses that can replicate to other computers and "Trojan horses" that masquerade as a useful program but actually steal your data, compromise your security, or even delete data from your device.

Coding for Applications

Creating complex applications is a multi-layered task. Very few apps are made by one person, and those tend to be small and simple. Larger applications are made by teams of programmers working together with UI designers under managers. When an application team has completed its design document, coders then begin to write the app. At this point the computer language (or languages) is decided on, as well as the operating system on which the app will run. The process begins with writing and debugging code until there is an early version called an alpha, which is then followed by the beta version. The beta version is tested extensively in a process referred to as "beta testing." Finally, when the app has been tested it will be released. Even then, users may find more bugs or other problems, and the development team will be called upon to change the code once again. This process takes months, or, in most cases, years.

Just as applications are subject to frequent changes, so are the processes involved in their development. The continuous maintenance and uninterrupted operation of software is becoming increasingly important. For example, it is now expected that software functions be made available to all users across all time zones (so-called 24-7 availability for all users worldwide). A second factor has also driven the development of cloud services – user expectation that the software functions be available on all end devices, from the smartphone to the tablet to the home PC or Mac.

For this reason, an application is broken down into several parts. The part of the application running on the user's device that depends on the device's operating system should only be a small element, which is then to be ported for all of the supported devices. The part with the actual functionality in the cloud, for example, the payment service, is executed in the secure backend financial system and not on the insecure user's device. While this improves the security of the application, it also means that personal data is often distributed across data centers worldwide, which has necessitated national regulations on data protection.

These demands on the development of applications have made it necessary to change the processes in software design, software development and operation. Some concepts you are most likely familiar with in this context are, agile development processes, like SCRUM, or SAFe (Scaled Agile Framework for Lean(R) Enterprises) and the interlocking of development and operation, the so-called DevOps. Due to increased security issues in cloud computing, this is somehow supplemented by the security part – DevSecOps. All of these terms are essential for the discipline of Software Engineering, and you will certainly encounter them again in your further studies.

7.4 Embedded Systems

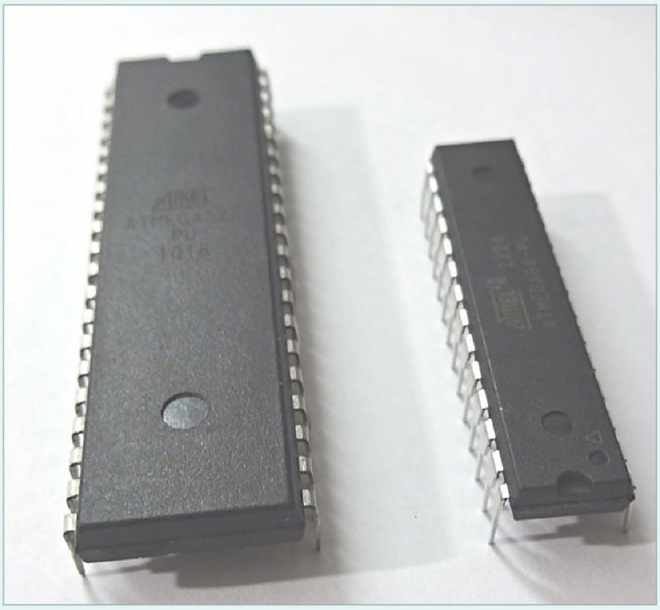
Think about the computers you have owned, including smartphones. All over the world, there are people like you who have had multiple computer devices. Now think about all of the different electronic gizmos and gadgets you’ve owned. Right now, count how many electronic devices you have in your house—things as simple as a digital clock. There are many times more electronic devices than actual computers. We are surrounded by them. Many of these devices are classified as “embedded systems.”

An embedded system is a system used in a special purpose device that usually performs one primary task over and over. Devices like medical equipment, fire alarms, voting machines, standalone GPS, washing machines, traffic lights, electronic toys—even switches and routers have embedded systems. The simple definition is a computer system that is designed for one specific task. It includes a CPU, memory, and an input/output system. Since the invention of microprocessors, these devices can be very small, such as a digital watch. Most of these systems use a microcontroller.

Microcontroller

A microcontroller is a computer chip with a processor, RAM, input, and output built in. The input and output travels through specific pins on the chip.

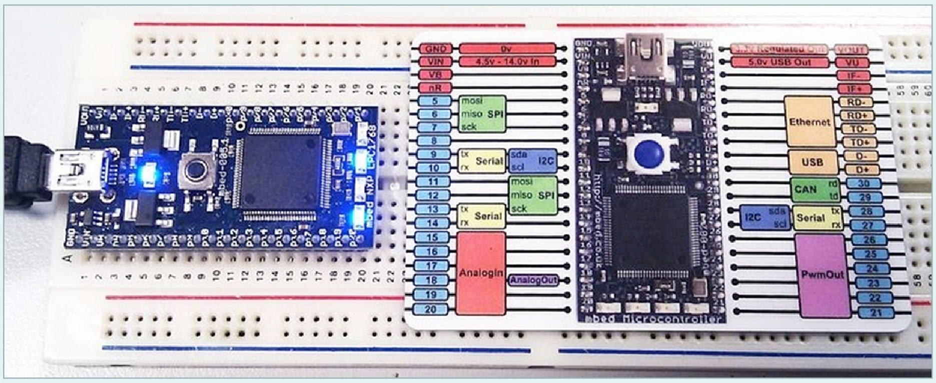
Figure 73: Microcontrollers



Source: Vahid alpha, 2013.

The chips can receive and send data to USB ports and even Ethernet through these I/O pins.

Figure 74: Microcontroller Pin Outs



Source: Nbauers, 2011.

Microcontrollers enable the creation of an enormous variety of different electronic devices; they are basically an entire computer on a chip.

Embedded System Design

Since an embedded system typically performs the same function over and over again, the software can be placed on a computer chip, making it firmware. This instruction set was created by programmers, but the requirements for an embedded system are different than an app for a computer.

As these devices are hard to upgrade (and upgrading is not part of the regular use of the device), they must have an extremely high degree of reliability. No one wants a device that crashes or has errors that cannot be corrected. This means that much more rigorous testing must be done of any software created to be placed on the device.

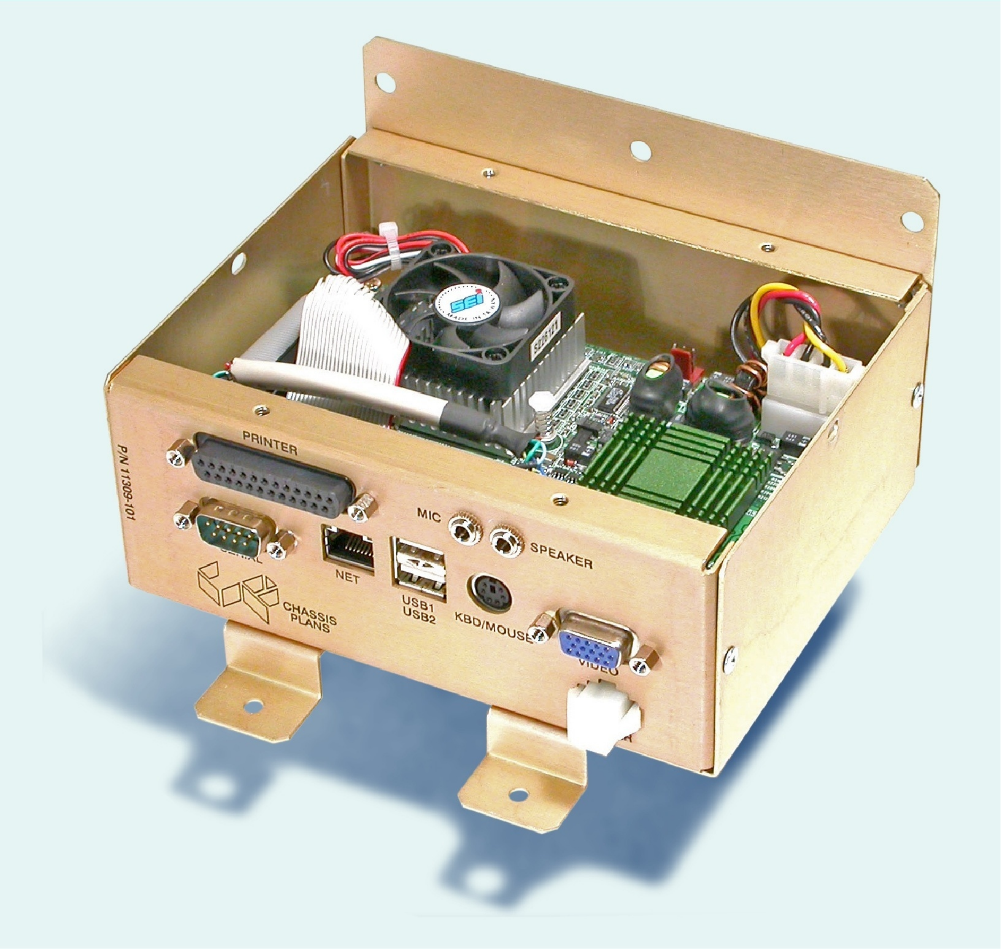
Electronic devices also have a very limited amount of RAM—programmers must be as efficient as possible to get the maximum amount of instructions into the smallest amount of memory. This is also different than programming for a computer as memory is a concern; PCs have a very large amount of RAM in comparison to embedded systems. In addition, PCs are made to run more than one app at a time, so they have even more memory.

Examining Embedded Systems

Imagine an ultrasound scanner in a medical office. This device has one purpose—to use ultrasound to create an image of the internals of a human being. The input would be selections from the user and various settings through an interface. Input would also include data from the scanner. These data would have to be converted and displayed as output—the final image of the patient. The device would be designed to connect to the scanner and also a display. It would have to be able to store an image and likely connect to a network.

A voting machine will have very specific requirements. It would have to report data externally without error. Counting a vote for an incorrect choice or counting a vote twice would be a catastrophic type of error for this machine. Crashing in the middle of voting would also be intolerable. The internal memory would store the progress of a voter’s choices and when it is done, there will be some kind of output to transfer voting data.

Figure 75: Voting Machine Internals



Source: Lippincott, 2003.

Embedded Systems and IoT

Originally, devices such as a fire alarm system or a washing machine functioned independently. With the proliferation of the IoT, more embedded systems are connecting to the internet. Some devices, such as fire alarms, can benefit from an internet connection. Besides sending an alarm signal to emergency responders, building managers could get other critical data such as where the alarm was tripped in the building and watch a real time update of what is happening through data. Other devices, such as your washing machine, might get some benefits from being connected to the internet (remote control), but there may not be enough advantages to justify the cost of additional network components yet. Since devices are often mass produced, and there is market competition, manufacturers are always striving to have the best device for the smallest cost. Additionally, size reduction is becoming a factor. People tend to want smaller and smaller devices that can perform the same functions. With new IoT protocols, even small devices with low power systems can be connected to a network.

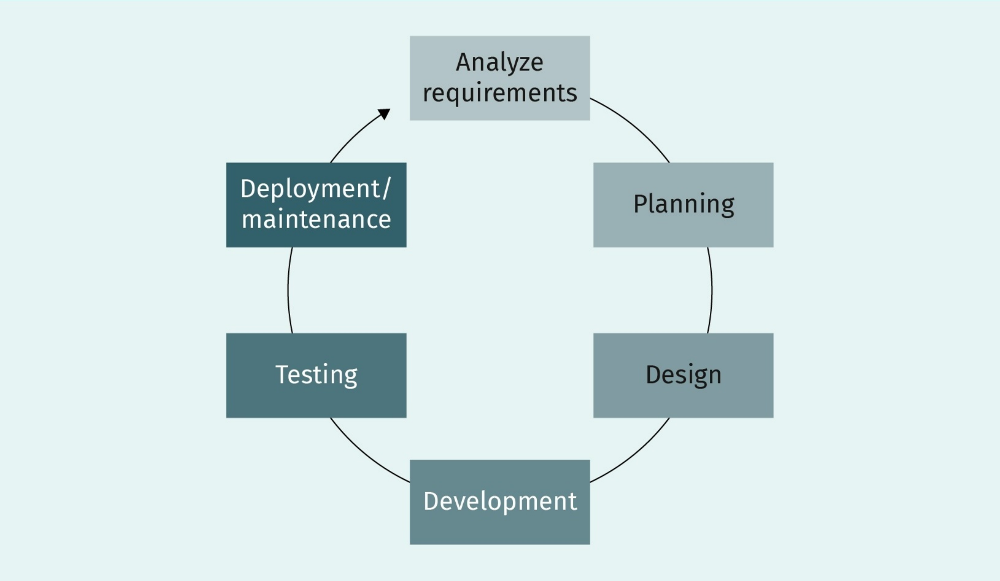
7.5 Software Development

In many ways, writing software is similar to writing a book. An author does not just sit down and start writing—many things have to be established first. Authors research the best title, write down their purpose and goals for the book, and then write an outline. Software developers do the same thing. Hours and hours of work must first be completed before the first line of code is even written.

Software Development Life Cycle

There are many methods of creating software; most follow the standard development life cycle. This is called a "life cycle" because it takes place over the life of a software application. After a software is released, maintenance and updates are required. These updates must go through a similar cycle to that of the initial development process.

Figure 76: Software Development Life Cycle Concept



Source: Stephen Weese, 2020.

The SDLC concept incorporates technology as well as management to put together planning and teams to meet a specific software need.

Analyze requirements

A development company is usually in one of two situations: they are either being hired to create specific software for a client, or they are creating a new software to release into the market. In either case, a common business question is asked: "What problem is this software going to solve?" In other words, what needs are unfulfilled, either in the market or for a specific business? The software should address those needs efficiently. This is when a company should be polling the users of the software as to what they want and need. Input from users, clients, and experts such as programmers and marketers is taken into consideration. Once these objectives are ready they are put together into a **requirements document**.

Requirements document
This document defines the purpose, requirements, and function of a software project.

Planning

This stage includes examining the feasibility of the project as well as looking at specific team assignments. Risks are examined and strategies to eliminate those risks are formed. Budgets are drawn up at this stage.

Design

Once a budget is approved, the relevant teams can begin working on the design—including creating a design document. This is a critical phase, since coding begins next. The design must be analyzed to make sure it will fit within the budget and that there are not any major design flaws. Otherwise, coders could begin working and waste hours developing something that must be redesigned.

Development

Coding finally begins. The documents created in earlier phases become the blueprints for the code. At this point, every team leader should clearly explain the specifications, standards, and phases of the development to the teams.

Testing

Once initial coding is complete, the software goes through extensive testing. This is where the QA (quality assurance) team goes to work. A good QA professional will test every aspect of a software application when looking for errors. During this stage, coders are notified of bugs and assigned to fix them.

Deployment and maintenance

Once internal testing is complete and QA checks have been passed, the software is released to the client or into the market. It is most likely that, even though extensive testing was done, other errors and problems will be found after deployment to teams. Part of software maintenance is updating the software to address these issues. If a new version is requested, then this new version of the software will start at the beginning of the life cycle and go through all the phases again.

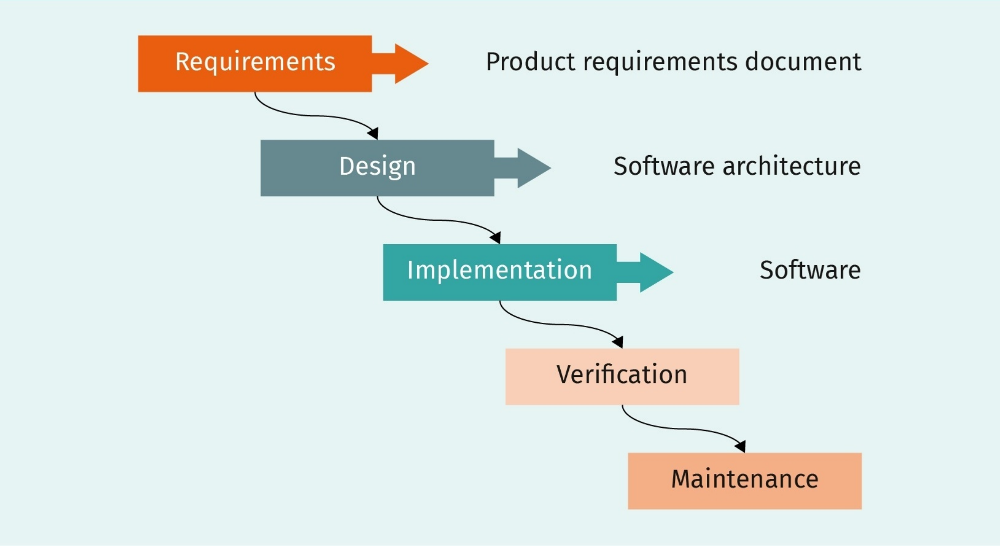
Implementations of the SDLC

The initial concept of the software life cycle has spawned many specific development models. We will take a look at three specific models: Waterfall, Spiral, and Big Bang.

The waterfall model

This model is a rigid model flowing downward through the different steps of the cycle. The idea is that after each stage is completed, it will not be returned to. It has similar steps to the SDLC concepts addressed earlier.

Figure 77: Waterfall Model for Software



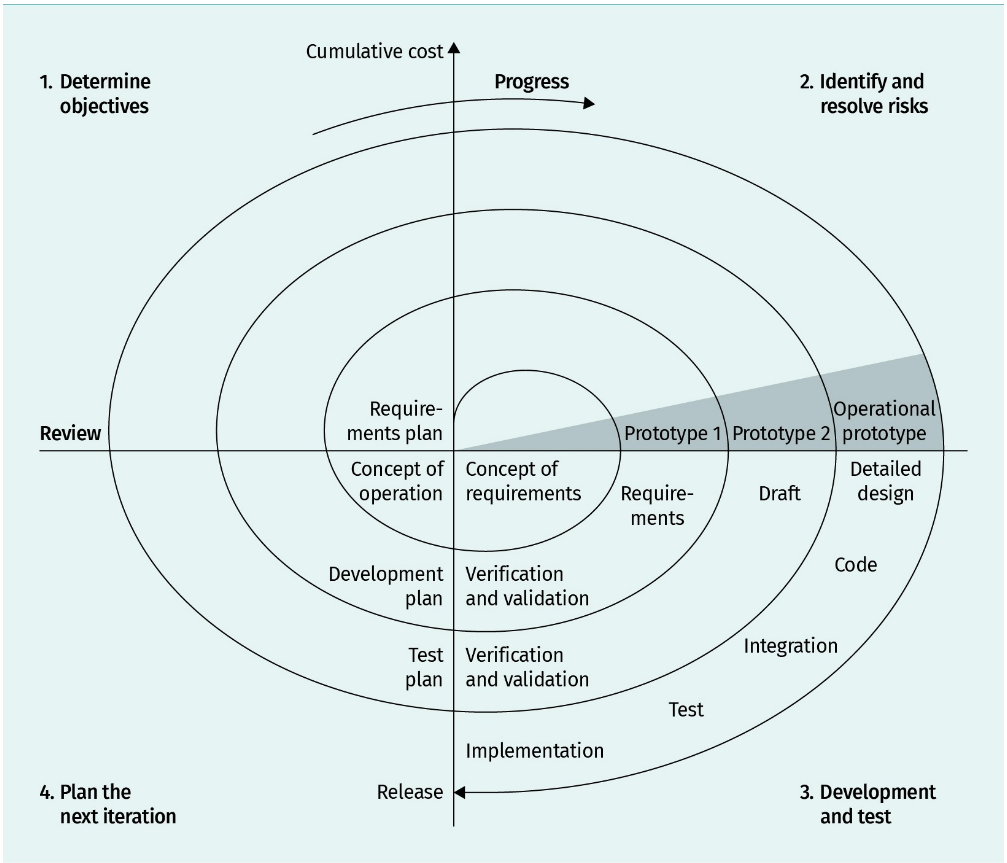
Source: Kemp & Smith, 2010.

The Requirements phase produces the requirements document, and Design the design document. Implementation is where the programming occurs. Verification is testing, and then the last step, Maintenance, includes the release or deployment.

The spiral model

A spiral approach to software development intentionally goes through several cycles of planning, designing, development, and testing, before final release. An advantage of this model is that various teams can get started working earlier in the process and some teams can work simultaneously.

Figure 78: Spiral Model for Software



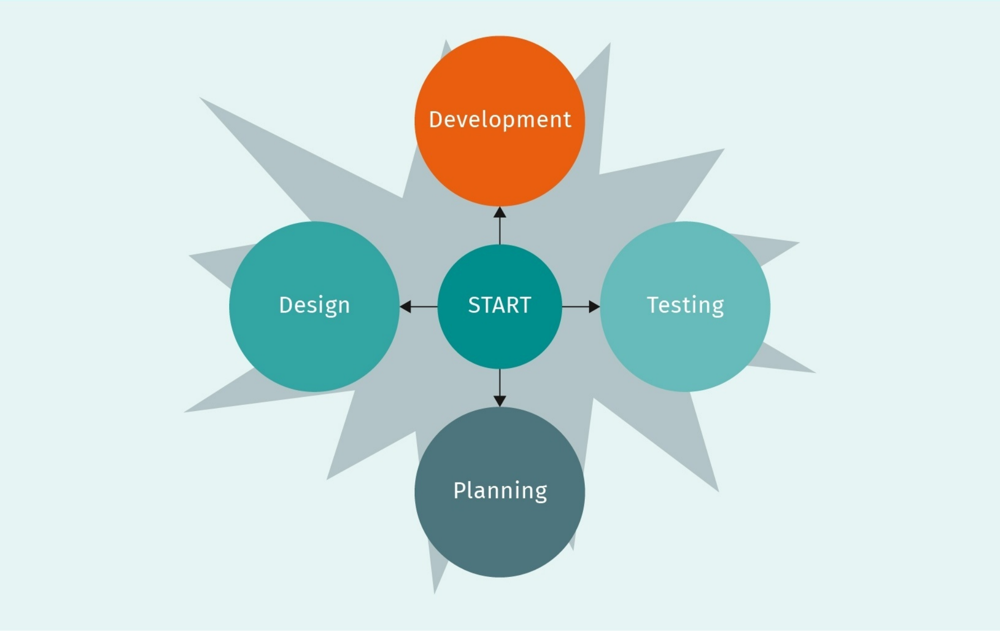
Source: Conny, 2004.

Several prototypes of the product are developed before final release in the spiral model. More than one round of testing and verification happens, as well as multiple rounds of redesigning. This model may take longer, but it is very thorough. The more recent “agile” model is based on this iterative model and has become popular.

The Big Bang model

The Big Bang model is a newer model often used by startups and other small companies. It is best used for smaller projects. The main feature of this model is that coding, design, requirements, and testing all begin simultaneously with very little planning. It involves quite a bit of risk since there is little pre-planning, but it also can lead to the quickest release of software.

Figure 79: The Big Bang Model for Software



Source: Stephen Weese, 2020.

Though there is little planning in the Big Bang model, plans can be created on the fly as development continues. Testing is done as soon as compiled code is available. It is not recommended for large projects or essential services, but can be used to brainstorm new types of software quickly.

SUMMARY

The bridge between hardware and software is the BIOS, the basic input-output system used for computers. When a computer starts up, the BIOS contains the startup instructions (firmware). It tests hardware and begins the boot up process eventually passing control to the operating system, which is a type of software.

The operating system provides three major functions: it controls hardware, provides a user interface, and runs applications. Without an operating system (OS), apps cannot be used. To store apps and data, the OS must use storage. A file system is required for operating systems to store information.

Organizations using software applications benefit from different information systems. Information systems help manage data, personnel, and products. Different types include MIS for management and EIS, which executives use to make top-level company decisions.

Applications are designed to perform specific tasks or similar groups of tasks. They are tied to a specific operating system; a Mac app cannot run on a Windows computer. There are different categories of apps including freeware, adware, and subscription applications.

Embedded systems are systems that run electronic devices that perform a very specific task. They do not relate to things like personal computers and laptops but are instead devices such as fire alarms, washing machines, and traffic light systems. They must be carefully designed to use minimal memory and to have high stability.

Creating large enterprise applications requires a software development model. For this process, companies rely on the software development life cycle. This model is broken into several phases: Analyzing Requirements, Planning, Designing, Development, Testing, and Deployment/Maintenance.

Software encompasses many different forms, from simple, tiny devices to large enterprise systems. Coding for these applications often requires careful planning and design.

UNIT 8

COMPUTER SCIENCE AS A DISCIPLINE

STUDY GOALS

On completion of this unit, you will have learned …

- the role computer science plays in the modern workforce.
- about different types of jobs related to computer science.
- the basics of artificial intelligence and data science.
- about the ethics of computer science.

8. COMPUTER SCIENCE AS A DISCIPLINE

Introduction

Computer science first emerged as an educational discipline in the 1960s. One of the first colleges to have a computer science department in the United States was Purdue University in Indiana (although Cambridge and Princeton in the UK had already been working on computer research prior to this). The department was established there in 1962 with five faculty members and a handful of students (Rice & Rosen, n.d.). Stanford started their program in 1965, and UCLA theirs in 1968.

It was not until the 1970s that widespread interest in computer science degrees emerged. The department at Purdue grew to around 100 students and new departments started up at other universities. The degree was not yet widely available as a major; many departments had some computer classes as part of a mathematics focus.

One of the most notable programmers of the 1970s (and beyond) is Warren Robinett. He graduated from Rice University in 1974 with a bachelors in Computer Applications to Language and Art, which included programming classes in FORTRAN. Not long after, in 1979, he wrote the legendary computer game *Adventure* for the Atari 2600 gaming console. Part of the reason for *Adventure*'s notoriety is that it contained the first video game "easter egg": a secret element placed by the author that was not part of the standard gameplay. At the time, Atari was not crediting the programmers for their video games. The solution for Robinett was to write his name in a secret room that players could only find by carrying around an invisible dot—a key to pass through a wall. Ingenious players discovered the invisible dot and carried it around trying to find its use—eventually discovering the room with Robinett's name (this fact is a key plot point in the Steven Spielberg film *Ready Player One*, based on the book by Ernest Cline).

Robinett took his computer science knowledge to great heights. After Atari, he co-founded The Learning Company which created educational software for children, including the well-received *Rocky's Boots*, which taught kids basic concepts of circuit design and logic. He also worked for NASA on virtual reality projects and created a VR interface for microscopes, which included haptic (touch) feedback for examining and moving around microscopic entities like DNA. He continues to work in the field designing educational software (Robinett, n.d.).

8.1 The Role and Subdisciplines of Computer Science

At first glance, it would seem that computer science is focused around programming. It is true that programming is important, but the true discipline of computer science is using computer technologies to solve complex problems. Before writing any code, computer scientists often analyze data and must come up with complex mathematical algorithms that are eventually placed into the programming. Without mathematical and analytical skills, computer scientists would find it difficult to create original code to solve new problems.

Computer scientists do not create computers; they use them as tools. Computer engineers and electronic engineers are the people that create and design computers. However, it is important that computer scientists understand some of how computers are designed, which is why most computer science (CS) majors take at least one digital electronics class.

Mathematics and logic are extremely important to the computer science major—a computer can only perform the instructions that it is given. If there is a flaw in the logic, it is not hardware that is at fault, but the programmer who gave it the instruction. Computers can help; they can detect syntax errors in the correct use of computer language, but they can't read minds and understand the outcome that is desired (at least, not yet).

In a traditional computer science program, students take classes in advanced mathematics, operating systems, algorithms, networks, databases, information technology, logic, software engineering, and even artificial intelligence. This gives future computer scientists a variety of tools for different types of programming and work in the computer field.

Computer Science-Related Disciplines

As the world becomes more information dependent (we are indeed described as existing in the "Information Age"), the need for computer experts increases. There are many different specialties (in many fields) that a computer science major can pursue.

Software engineer

A software engineer is a crucial part of a team of developers and takes part in the process of creating large-scale software applications. The engineer will be familiar with the software development life cycle (SDLC) and should be able to create clean and efficient code, work with a team, create documentation, and implement algorithms. This term is often used interchangeably with "software developer."

Web developer

A vast amount of information and services exist on the World Wide Web. Programmers that create code for websites are called web developers. They work with the markup language HTML but also are competent in **CSS**, PHP, JavaScript, and databases such as MySQL. Some developers provide all-in-one services which include graphics and design.

CSS
This stands for Cascading Style Sheets, these are used to create layout and formatting for web sites.

Systems analyst

These analysts examine existing computer systems and provide recommendations for changes or replacements. Systems analysts are focused on user experience and productivity. They usually do not write computer code directly, but can recommend what code should be written. They can work on a small scale or for a large corporation.

User interface (UI) designer/developer

Software is often thought of as existing in layers. The “front” layer of software is the user interface; the UI is how the user interacts with the application. The UI designer designs the interface and a UI developer creates the code for it. The UI then connects to the main part of the software and database, which is the “back end.”

Database administrator

Databases are complex to create and manage. They have their own specific programming code to create queries and manipulate data. A database administrator is competent in a database language like **SQL** and also knows how to run administrative tools for systems such as Oracle. Flaws or errors in database design can cause problems and must be identified and corrected quickly by database administrators.

SQL
This stands for Structured Query Language, a language created to access, query, and modify databases.

Data scientist

A data scientist is an expert in mathematics, statistics, and computer programming. A strong mathematics background is required to analyze large data sets and manipulate them to produce information through data mining. Algorithms and logic are used to create code that can quickly sort and filter data to produce results.

Software manager

After many years as a software developer, many people choose to move up to management and lead teams of developers to create new software. Project managers are the top level of these managers. Understanding the software design process is key to helping manage technical teams in the business world.

Information security analyst

A security analyst will need skills from programming, networking, and a strong understanding of security protocols and software. These analysts look for weaknesses in a computer system before it is breached. They recommend changes and help implement company security policies.

Information systems (IS) manager

A business or other type of organization that has many users and a computer system will need some type of IS manager. This person oversees the hardware and software of the organization, including purchasing and distributing software. Software choices are examined and the ones that best meet the needs of the company are chosen.

Computer hardware engineer

Although a computer engineering major would typically perform this job, there is enough overlap that many computer scientists also enter this field. It involves designing and creating hardware, including creating the low-level firmware instructions that are placed on the device. This includes things such as circuit boards, network cards, and design for embedded systems.

Video game developer

A video game developer is a specific type of programmer—their skills are focused around the knowledge and tools used for creating video games. Most video games today are created with special tools that go beyond just a programming language. Software tools like Unity and Unreal Engine are used prominently in this field.

There are other roles as well that relate to computer science such as software tester or **QA analyst**, cloud computing engineer, and network administrator, to name a few. In the future it is expected that there will be new types of computer-related jobs and skills.

QA analyst
This stands for Quality Assurance analyst, a job that involves testing products, including software, electronics, and other commercial products.

8.2 Artificial Intelligence, Data Science, and Computer Science

Two areas of notable technological growth in the twenty-first century are data mining and automated systems. It is estimated that 2.5 quintillion bytes (a quintillion is a large number in which ten is raised to the power of 18) of data are created every day (Wise, 2022). There is therefore a growing need for much of this data to be analyzed.

Repetitive tasks that have historically been performed by humans are being transferred to automated systems. Decades ago, farming equipment replaced the need for workers out in the field. Today, machines replace humans on assembly lines and even in shipping warehouses for companies such as Amazon. Other modern tasks like data entry and sorting can now be improved by automation, improving efficiency and accuracy.

Figure 80: Amazon Shipping Warehouse

Source: Domdomegg, 2019.

Data Science

As mentioned previously, it is a data scientist's job to analyze data and provide meaningful and useful information. There are many different fields where this would be applicable.

Hard sciences

"Hard science" fields such as biology, physics, and astrophysics deal with large sets of data. One example would be biomedical data science. As the world's population grows, so does the need for health care. A world with seven billion people provides a lot of data points. A researcher for biomedical data science could analyze data to predict how a medication would perform on a certain population. In physics, the LHC, or Large Hadron Collider, produces enormous amounts of data—about 50 petabytes (50,000 terabytes) per year. This data not only requires data scientists to study it; it also requires a supercomputer (InsideHPC, 2018). This type of data mining was used to identify the Higgs-Boson particle, and this discovery was awarded a Nobel Prize in 2013. NASA has one of the largest sets of data on earth, containing countless images and other readings of the universe. Data mining is key to finding meaningful data points in the vastness of the cosmos.

Business and marketing

Large businesses that have been around for decades have accumulated vast troves of data —think of companies such as Sony and Coca-Cola. They have sales data, marketing data, product performance data, and more. Data mining can answer questions such as "How can this be used to predict what products will sell?" and "What kind of product will solve this problem the best?" Besides their own data, companies can purchase data and services from other data brokers and data collection companies. Companies who do not leverage their data are sure to fall behind companies who quickly identify future needs based on past information.

Artificial Intelligence

There are many different types of machine intelligence. They fall into two broad categories: specific and general. Specific (or narrow) artificial intelligence (AI) is a system created to perform a specific task. For example, the predictive text that Google uses for searching is very good at predicting what you might want to search for, but it would not be good at finding your lost keys. General AI is still a major programming challenge. A system in this category means that it is able to examine new situations, learn, and perform a variety of broad tasks. Basically, this kind of AI functions similarly to a human being. Humans can learn and perform a variety of tasks, which is difficult for a computer. However, performing specific tasks is not a hard problem to solve; in fact, computers can easily perform many tasks better than humans can.

It is important to understand what is meant by the term "intelligence." Intelligence is more than just being able to perform a series of mathematical algorithms; intelligence implies deduction and learning. An AI is able to come to conclusions with data it has never encountered before. Since computer systems are able to perform billions of calculations per second, their power can be harnessed to simulate intelligence.

Robots and Devices

Some AI is used to control physical machines. There are many assembly lines around the world that use robots to create other machines and devices. Some are fully-automated, but many still need human supervision for problems that the AI cannot fully deal with or comprehend.

Assembly line robots can work with exact precision—the parameters of movement can be exactly programmed. More importantly, they don't get tired or distracted. Yes, they can break or wear out, and that must be planned for. Many of these robots have four or six arms that can all work at the same time, exceeding the capabilities of a human. Some of these robots also have their own "vision"—cameras that can see the work area, locate objects to be manipulated, and possibly even detect and solve problems.

Other AI on assembly lines that package food can be programmed to detect defective food products and remove it from the line. Some systems achieve better results than human food-checkers. Other systems are made to work alongside human beings for maximum efficiency. The AI systems must make an intelligent observation of the food product and determine whether it should be packaged.

Intelligent Data Systems

Other systems are not tied to physical robots, but function instead as data systems. One such system commonly used today is facial recognition. Photos or video of people can be scanned in less than a second and different people can be identified by the mathematical measuring of facial features. These systems use intelligence in factoring light, shadow, angle of the face, and other variations to accurately recognize a face. There are also intrusion detection systems that provide security for networks. In the past, a human would have to act against a network attacker by checking the alerts from the server. Now these systems can take automatic countermeasures based on information. For instance, an intelligent system might detect a DDoS? attack coming from a specific part of the world and immediately block traffic from the primary routers passing it along.

Designing Artificial Intelligence

There are different approaches to creating an AI system. One of them attempts to simulate a human brain—it is known as a neural network. It attempts to simulate the levels of neurons working together in the brain. Neural networks can be trained on data to "learn" things. Other types of AI are algorithm based, solving a specific problem by using predictive mathematics on data sets. For instance, there are AI systems that are able to diagnose patients based on their symptoms at a rate that is as good as, or better than, physicians (Longoni & Morewedge, 2019).

Programmers use languages such as C++, Java, and Python to create AI. Another language often used is LISP. An older language, LISP has features and flexibility that make it useful for artificial intelligence.

Computer science is the backbone of both data science and artificial intelligence. These fields require a solid CS background. Data scientists must not only have computer skills, but also skills in mathematics and statistics. AI designers must understand complex mathematical algorithms and data structures. Both disciplines are currently in high demand.

8.3 Ethical Aspects of Computer Science

Ethics is the study of moral concepts such as personal behavior, responsibility, good, and evil. There are many ways that ethics can apply to the field of computer science. These can be broken down into three layers of responsibility: corporate/government, developer, and user.

An example of ethics would be a person's (or an entity's) moral code: the rules that they use to determine decision making and actions. There are individual differences between ethics, although in societies there are group ethics that apply. For instance, most countries have laws against stealing, assault and murder. Most people would agree that these things are generally wrong and that to have a prosperous society they should be avoided. Being a prosperous computer programmer also requires an understanding of ethics.

Corporate (or Government) Ethics

There are many opportunities for a company to act unethically in the software world. One way is to steal intellectual property. New innovations and applications are always coming out and code is protected as a company secret. A company could steal an insider away from another entity and coerce that former employee to reveal secrets. A more direct approach would be stealing a competitor's idea and implementing it first. Often this results in a lawsuit, but the party on the ethical high ground does not always win.

Companies (or governments) could also lie about their product—claiming that it is better than is actually is, or promising features that are not delivered. Failure to meet contract requirements and deadlines is another ethical problem companies may face. Another issue that has been prominently discussed lately is the use and sale of private user information. User privacy is an important ethical code to maintain—when that is broken, it is a breach of ethics as well. Companies could also breach ethics by influencing minors to become addicted to a software, in attempts to manipulate parents to spend money—or worse, allowing minors to create accounts and incur debts they cannot pay.

To maintain proper ethics, a corporation or government must be direct about product expectations, specifications, budgets, and deadlines. They must not attempt to deceive their user base about the product or include deceptive charges. Companies can and should hire an ethics officer to help prevent lapses in ethics. This officer's duties include preventing discrimination and any kind of harm to other entities. They help to enact standards that the company will abide by for business practices.

Developer Ethics

A software developer has several ethical issues to consider as well. First, writing code is similar to writing a novel, in that you are expected to create your own original work and not copy others (look up copyrights for more information). There are some "open source" software libraries which can be borrowed from freely, but taking someone else's code without permission is not only unethical, it is illegal.

As a developer for a company or organization, you work under a certain expectation of confidentiality. Many companies will have developers sign confidentiality agreements as well. Even without one, it is standard practice to keep sensitive company information private.

A company will have best practices and standards for work performed—taking shortcuts and slacking off is considered unethical behavior; programmers are being paid for their time with a certain expectation of performance.

As an employee, you will also be expected to report coworkers who do not uphold the company's standards or who break laws or ethical codes. Covering up for another's unethical behavior is also a breach of ethics.

User Ethics

As a user of software, one of the primary temptations is to pirate (illegally copy) software. Since it is not an actual physical product, but only consists of information, it is easier to copy in many instances and lacks the visceral feel of stealing—which it certainly is. As a computer software user in the twenty-first century, there are many options to avoid piracy, including free versions and subscription plans.

A user is also bound by the terms and conditions of the license agreement (which almost no one reads). These conditions determine how the software can be used and which computers it can be used on. It is unethical to use the software in violation of the EULA (end-user license agreement).

8.4 The ACM Code of Ethics and Professional Conduct

The ACM is the Association for Computing Machinery. They are the world's largest computer society and are open to industry professionals and educators—anyone who is using computer technology. The Association also has its own code of ethics. The entire code can be found on the ACM website (ACM, 2018).

Why Create a Code of Ethics?

The ACM Code of Ethics and Professional Conduct exists to inspire computing professionals to work for the greater good of society. The specific tenets are laid out to help guide those who wish to follow that aim.

Who is it for?

The code of ethics was written to apply to all computer professionals worldwide. Regardless of membership in ACM, the principles in the code are useful and helpful to maintain a high standard of ethics. The code is used internally for all ACM members to use as a measure for ethical performance.

General Principles of the Code

Regardless of specific career or section of computing, an individual can find guidance in the general principles of the ACM code.

Contributing to society

The computer user should understand the power of their actions and strive to create outcomes that benefit others.

Avoid harm

In contrast, the computer professional should intentionally consider what actions could cause possible harm to clients, users, consumers, and others.

Honesty

Keep your word, have reasonable expectations that can be met, and do not intentionally deceive.

Respect creativity

Do not copy others' work without proper payment or credit, where it is applicable. Do not pirate software.

Honor confidentiality and privacy

Keep sensitive company information protected, and do not expose user private information or sell private user information without permission.

Professional Principles of the Code

Those who work in the industry creating software, hardware, and other computer products can hold themselves to more specific standards:

Quality work

Have a high standard for the quality of your process and your finished product.

Professional conduct

Conduct yourself professionally at all times: keep to your ethical standards. Strive to improve your skills and education.

Professional knowledge

Continually be aware of—and update—your knowledge of the company policies and procedures and follow them.

Review and criticism

Be able to accept feedback on your work and products and be able to give honest reviews when needed.

Work in your expertise

If you are not skilled in a certain area, do not “fake” it, but request help, training, or that someone else perform the task.

Enable public awareness

Help the public understand current technologies and the consequences associated with them.

Keep security in mind

Create systems that are secure. Do not release systems until thorough security testing has been completed.

Leadership Principles of the Code

Leaders in the computing world should follow these guidelines for ethical behavior:

Public good

Make sure the public good is the center of professional computing work.

Social responsibility

Clearly state social responsibilities for professionals and encourage their practice.

Personnel management

Allow for resources and management of employees to enhance the quality of life while at work.

Central policies

Encourage and support policies that embody the principles of the ACM code.

Growth

Set forth opportunities for group members to learn and grow as professionals.

Societal systems

Use high levels of scrutiny and care when working with, or creating, a system that will become integrated into society.

ACM Membership and the Code

Members of the ACM are expected to follow these principles and to encourage other members to do so as well. A member observing a breach in the code by another member is asked to consider reporting the violation to ACM, which may result in remedial action (ACM, 2018).

SUMMARY

Though many think of computer science as centered around programming, there are many other important subjects involved in the focus including mathematics, logic, algorithms, and analysis. This allows someone with a CS background to work in fields such as data science, web development, information systems, and database administration.

Data science involves the use of mathematics, statistics, and computer algorithms to extract useful information from large sets of data. It can be applied across many subject areas with large data sets including biology, medicine, physics, business, marketing, and manufacturing.

Artificial intelligence systems are computer systems that can perform actions that usually require human intelligence. Specific or “narrow” AI systems can perform one task at a very high efficiency, usually greatly surpassing human ability. General AI systems are harder to develop as they are classified by being able to learn new tasks and generally make deductions like a human being.

The ethics of computer science can be broken down into three levels: corporate (or government), developer, and user. Each of these levels involves different ethical concerns. A large corporation must make sure to deliver to promised expectations and avoid stealing company secrets from other entities. Often an ethics officer is employed to help establish and maintain ethical business practices.

It is important to maintain code originality. Like other creative works, code is under copyright. Code must be original and not taken from another programmer. The developer must also follow business practices and confidentiality agreements as stipulated by the employer.

Users must do two things to remain ethical: do not pirate software and keep to the terms of the license agreement for the software. Piracy is still a problem in the twenty-first century since it is still an easy matter to break copy protection and distribute software online.

The ACM's (Association for Computing Machinery) code of ethics helps guide industry professionals and others who work with computers. It includes personal principles such as honesty and avoiding harm, professional principles such as being aware of and following company policies, and leadership principles such as creating public awareness of technology and encouraging growth in employees.