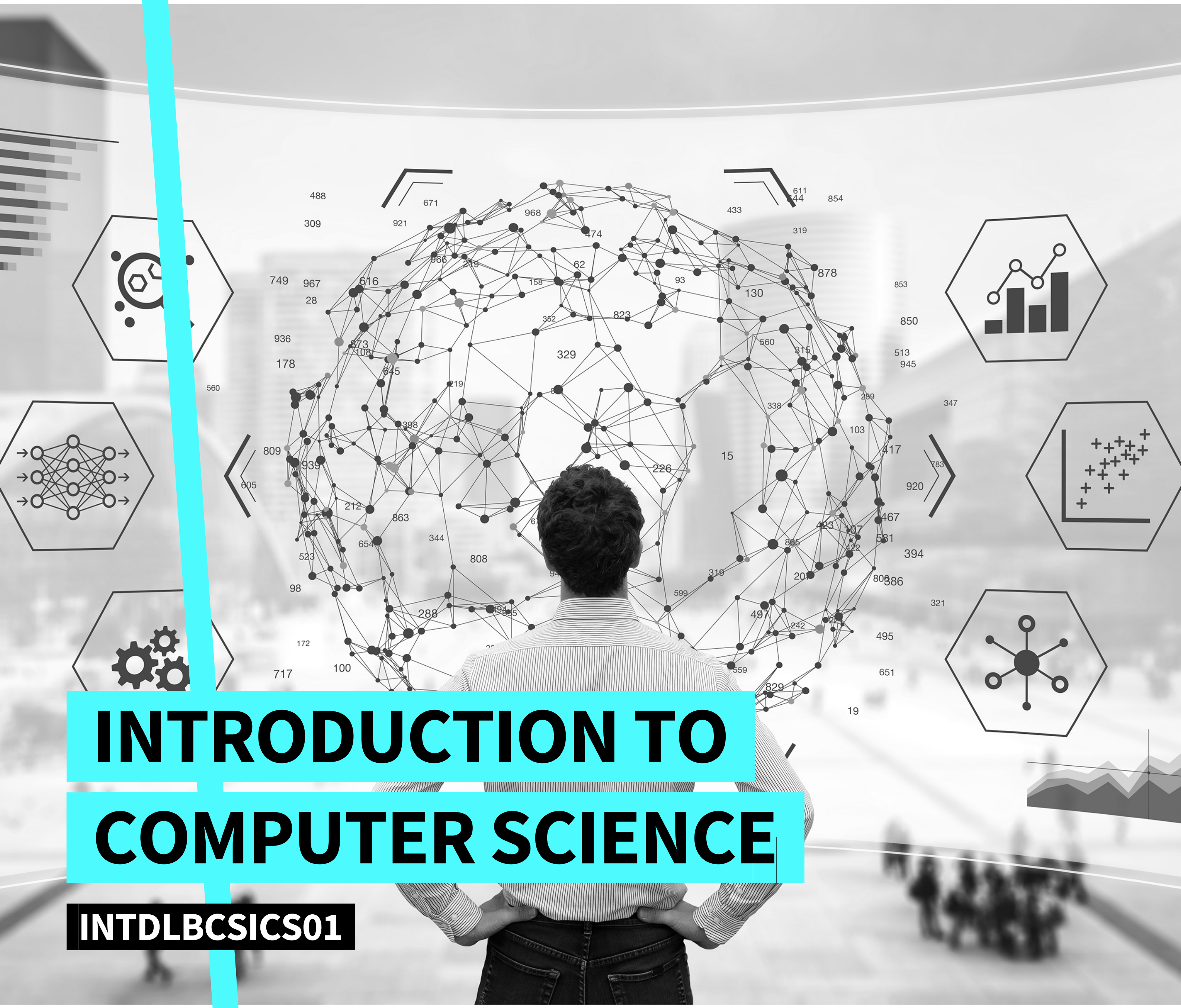
Course Book



INTRODUCTION TO COMPUTER SCIENCE

INTDLBCSICS01

INTERNATIONAL UNIVERSITY OF APPLIED SCIENCES

INTRODUCTION TO COMPUTER SCIENCE

MASTHEAD

Publisher:
IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Mailing address:
Albert-Proeller-Straße 15-19
D-86675 Buchdorf
media@iu.org
www.iu.de

INTDLBCSICS01
Version No.: 001-2023-1016

Stephen Weese

TABLE OF CONTENTS

INTRODUCTION

WELCOME

SIGNPOSTS THROUGHOUT THE COURSE BOOK

This course book contains the core content for this course. Additional learning materials can be found on the learning platform, but this course book should form the basis for your learning.

The content of this course book is divided into units, which are divided further into sections. Each section contains only one new key concept to allow you to quickly and efficiently add new learning material to your existing knowledge.

At the end of each section of the digital course book, you will find self-check questions. These questions are designed to help you check whether you have understood the concepts in each section.

For all modules with a final exam, you must complete the knowledge tests on the learning platform. You will pass the knowledge test for eachunit when you answer at least 80% of the questions correctly.

Whenyou have passed the knowledge tests for all the units, the course is considered finished and you will be able to register for the final assessment. Please ensure that you complete the evaluation prior to registering for the assessment.

Good luck!

FURTHER READING

UNIT 1

Balakrishnan, H., Terman, C., & Verghese, G. (2012). *Bits, signals, and packets: An introduction to digital communications and networks* [Course notes]. MIT Open Courseware. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsbas&AN=edsbas.F2B3E5D&site=eds-live&scope=site

Kapko, M. (2015, October 7). *History of Apple and Microsoft: 4 decades of peaks and valleys*. CIO. (Available on the Internet)

Rojas, R., & Hashagen, U. (2000). *The first computers: History and architectures*. MIT Press. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.47516&site=eds-live&scope=site

UNIT 2

Henderson, H. (2017). Data types. In *Encyclopedia of computer science and technology*. Facts On File. (Available on the Internet)

UNIT 3

Henderson, H. (2017). Data structures. In *Encyclopedia of computer science and technology*. Facts On File. (Available on the Internet)

Mehlhorn, K., Sanders, P., Dietzfelbinger, M., & Dementiev, R. (2019). *Sequential and parallel algorithms and data structures: The basic toolbox*. Springer. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edshbz&AN=edshbz.DE.605.HBZ01.036392268&site=eds-live&scope=site

Nakov, S. (2013). *Fundamentals of computer programming with C#: The Bulgarian C# programming book*. Faber. Chapter 19 (Available on the Internet)

UNIT 4

Harris, D. M., & Harris, S. L. (2007). *Digital design and computer architecture: From gates to processors*. Elsevier. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.47513&site=eds-live&scope=site

Rautenberg, W. (2010). *A concise introduction to mathematical logic* (3rd ed.). Springer. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.47515&site=eds-live&scope=site

Read, C. (1909). *Logic: Deductive and inductive*. Alexander Moring Limited. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsbas&AN=edsbas.2105AAAC&site=eds-live&scope=site

UNIT 5

Henderson, H. (2009). Neumann, John von (b. 1903—d. 1957). In *Encyclopedia of computer science and technology*. Facts On File. (Available on the Internet)

Upton, E., Duntemann, J., Roberts, R., Mamtora, T., & Everard, B. (2016). *Learning computer architecture with Raspberry Pi*. Wiley. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.47518&site=eds-live&scope=site

Viswanath, D. (2017). *Scientific programming and computer architecture*. The MIT Press. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsors&AN=edsors.e7a2504f.08b9.4c44.b9f9.5cc42096d21c&site=eds-live&scope=site

UNIT 6

Bonaventure, O. (2012). Computer Networking : Principles, Protocols and Practice. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsotl&AN=edsotl.OTLid0000352&site=eds-live&scope=site

Kurose, J. F., & Ross, K. W. (2017). Computer networking : a top-down approach (seventh edition, global edition). Pearson. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.47514&site=eds-live&scope=site

UNIT 7

Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2016). *Operating Systems: Three Easy Pieces*. (Available on the Internet)

Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson. http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.47517&site=eds-live&scope=site

Salihun, D. M. (2007). *BIOS disassembly ninjutsu uncovered* (1st ed.). Unpublished manuscript. (Available on the Internet)

UNIT 8

North, M. (2012). *Data mining for the masses*. Global Text Project. (Available on the Internet)

LEARNING OBJECTIVES

Computers were created to assist humanity in calculating. The idea was simple: computers will perform these tasks with much greater certainty and speed than humans, and a computer never becomes tired or bored. It never makes simple mathematical errors, no matter how often it executes the same equation – assuming it is properly programmed and designed.

Introduction to Computer Science begins by looking at the most basic concept of computing: data. Interpreting numerical data in the context of an abstract human concept is the goal of man-machine cooperation. Data are processed by a collaboration between hardware, firmware, and software.

Creating a computing machine involves not only assembling the parts but also programming the machine. This programming is done in a complex language created to be understood by humans and machines alike – this course introduces 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 structures, circuit designs, propositional logic, and basic computer architecture. The invention of networks revolutionized computing, and this course examines these concepts along with TCP/IP (TCP stands for transmission control protocol, IP stands for internet protocol), 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 computer science.

UNIT 1

BASIC CONCEPTS OF DATA PROCESSING

STUDY GOALS

On completion of this unit, you will be able to …

- describe the concepts of data, information, and computer messaging.
- explain the difference between hardware, firmware, and software.
- master the basics of binary and data interpretation.
- use the syntax and semantics in programming languages.
- outline 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 and Dragons (D&D), which is set in a similar fantasy world. Soon thereafter, Garriott started programming the rules of the game into a computer.

The game of D&D is a nearly perfect match for the computer. It involves ability scores, which are numbers (i.e., data) assigned to human attributes such as strength, intelligence, and dexterity. 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 specific actions, dice are rolled and the results – modified by said attributes – determine the success or failure of the action. Simple computer equations and algorithms can simulate the rolling of dice (random number generation) and calculations based on data.

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 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).

Computer programming involves taking human concepts and ideas and reducing them to numbers, logic, rules, and, finally, to output that a human can interpret. To begin our study of Computer Science, we are going to take a look at the core concept of data.

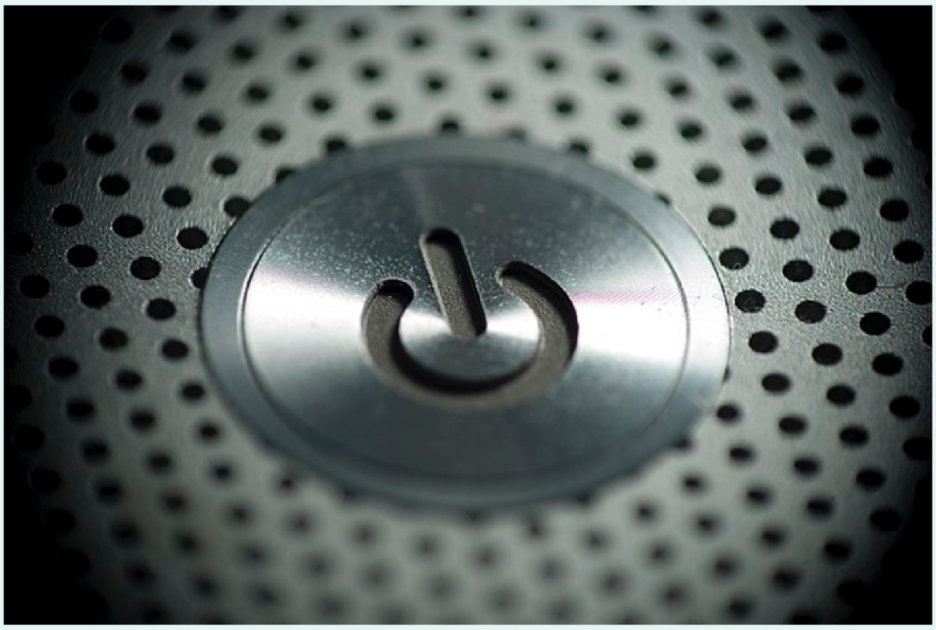
1.1 Data, Information, and Messages

Binary
A base-two numbering system consisting of only ones and zeroes.

Data consists of raw numbers and symbols arranged into meaningful information, which is then used for specific research or analysis. To a computer, data are numerical and the basic data storage system is the **binary** numbering system. While humans are accustomed to using numerous symbols to represent information, including letters of the alphabet and decimal numbers, computers must represent all types of information as ones and zeroes. If you asked a computer to remember how you like your coffee, it could store this information in the form 01011100. This format is essentially meaningless for a human, but it could easily represent sugar, cream, size, or temperature of the first drink you have in the morning. In a computer, your favorite song 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.

The reason computers use the binary number system is that, when they 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, respectively. This is still seen today in most digital devices in the labeling of 1and 0 for the power button in the on or off position.

Figure 1: Modern Power Switch



Source: Vincentg, 2007.

If we connect a series of switches, 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; this group of eight digits is called a byte. A byte can represent a letter of the alphabet, a specific color, or different application preferences. The information represented by computer code 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 have a 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 progression of zeroes and ones.

As mentioned earlier, binary digits 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", the latter has a different binary representation: 01100001. Thus, using this binary code, a computer can represent all of the letters of the English alphabet along with various punctuation symbols. Given 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 letters into binary to store the data—and then translate the binary code back into letters to show them to humans again! This introductory concept gives a glimpse of how computers work. They are constantly translating back and forth between human context and binary code.

Figure 2: Full ASCII Table

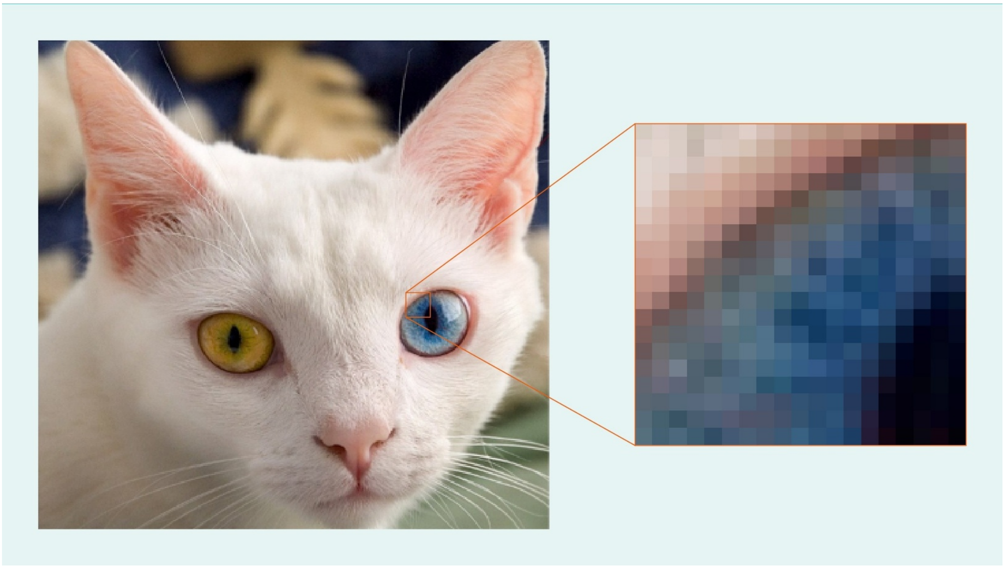
	Hexa-					Hexa-			
Decimal	decimal	Binary	Octal	Char	Decimal	decimal	Binary	Octal	Char
0	0	0	0	[NULL]	64	40	1000000	100	@
1	1	1	1	[START OF HEADING]	65	41	1000001	101	A
2	2	10	2	[START OF TEXT]	66	42	1000010	102	B
3	3	11	3	[END OF TEXT]	67	43	1000011	103	C
4	4	100	4	[END OF TRANSMISSION]	68	44	1000100	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 single square assigned a color and assembled together to form a larger image

Another form that data is an 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 separate it into discrete squares, which is what we do to this day. It is a simple solution that allows humans to recognize photographs on a computer screen.

Figure 3: Pixels Used in a Photo



Source: Kissel, 2012.

The concept behind pixel images is to slice the image into small squares and assign each square one (and only one) color. If the squares are small enough, the human eye cannot distinguish them individually, so our brain sees 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 may start at the top left for the first pixel, and then fill in the first row before jumping down to the next row of pixels. Since the order is implicit, pixels need not be numbered. The basic data contained in a pixel is its color, so each color must be assigned a binary number. Eight bits (a byte) can represent 256 unique colors. This is called 8-bit color.

Earlier computers used the video graphics array (VGA) system, a display standard introduced in 1987 that supports a resolution of 640 $\times$ 480 pixels and a palette of 256 colors. The VGA system used an 8-bit color scheme, where the first three bits represented red, the next three green, and the last two blue. This color scheme is called “RGB” for red, green, and blue. These three colors can be combined to create any color visible to the human eye. However, with only 8 bits, only 256 possible colors can be represented out of the essentially infinite number of potential colors.

Figure 4: Digital Representation of Color

Bright Yellow in 8-bit color: 11111100

Source: Stephen Weese, 2020.

If we combine the most intense red with the most intense green, we get bright yellow.

Music can also be stored digitally (which simply means “as binary digits”). In the same way that we slice an image into pixels, we can slice a sound or a song into tiny intervals of time and then string them back together. If the slices are sufficiently small, our ears cannot hear the difference with the original sound, in much the same way that the eye cannot see individual pixels.

For example, a Microsoft Word document is stored completely as ones and zeroes. The context for this data includes binary codes that dictate the formatting, such as bold and italic, and others that dictate the page margins, font color, and other settings. Finally, of course, the file contains binary codes that represent letters, numbers, and punctuation. Microsoft Word, in essence, invented its own code for storing data in a **.**docx file. When you open a Microsoft Word document, the binary code is translated into what you see on the screen so that you can read and edit it.

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

Data Messaging

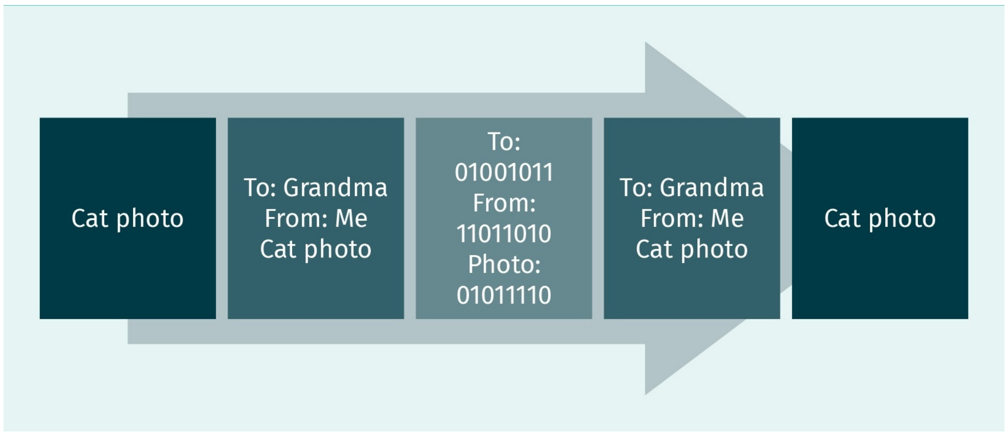
Computers use various hardware components to perform tasks and must communicate via messages coded in binary. Computer components have a special list of commands that allow them to pass information and instructions to each other; these are sent primarily through the motherboard[1] or data cables. Computers also send messages to each other through networking.

Having seen how to store letters, images, music, and documents as binary data, we now examine the process of sending these data over a network. The process of sending data is complex and involves numerous stages, including establishing and verifying a connection, agreeing on the mode of communication between computers, preparing the data to be sent, and then, finally, sending the data and error checking to ensure they have arrived properly.

For our purposes, consider sending your grandma a picture of a cat over the internet. The cat picture, of course, is just zeroes and ones. But we have to ensure they get to grandma’s computer. Below is a highly simplified description of that process.

1 A motherboard is the main circuit board in a computer that allows communication between the central processing unit (CPU), memory, and other peripherals.

Figure 5: Sending a Cat Photo



Source: Stephen Weese, 2020.

You see the photo of the cat on your screen and save it to a file. To send it to your grandma via, for example, email, you must fill in two addresses: the TO and FROM fields. Once you fill these in, everything is converted to binary: TO FROM, and all the data of the cat picture. These binary data are sent over the network to Grandma's computer, which translates them back into human-readable form so she can see that she has received a cat photo from you. Once the photo data are received, the TO and FROM data are stripped away because they are no longer needed. When Grandma opens the file, the computer translates the binary data into a viewable photo.

1.2 Software, Firmware, and Hardware

Three types of components work together to enable a computer to function: the software, firmware, and hardware. The most visible is the hardware, which consists of all the physical components of a device. For our purposes in this unit, we examine the hardware architecture of a modern personal computer. Other types of computers use a similar hardware architecture.

Hardware

A vital piece of hardware is the central processing unit (CPU), which is often referred to as the "brain" of the computer. This square computer chip is inserted into the motherboard and does all the processing, which includes calculations and sending instructions to other hardware. Computers also need to store data in short- or long-term memory, which is done in memory chips for short-term memory and in large-capacity memory chips, magnetic disks, or other hardware for long-term memory.

Computer hardware may be clarified by considering it analogous to human consciousness. We process with our brain, but we also store things in both short- and long-term memory. Short-term memory stores the things you are thinking about right now. How many things can you think of at once? Although it may seem like a lot, compare it with your long-term memory, which stores everything you know. You cannot possibly think of all of those things at once; they will not fit into your short-term memory. A computer works the same way: everything that the computer knows (i.e., data) is stored on the hard drive.[2] Whatever you are currently working on is moved into **RAM**, which is the computer's short-term memory.

RAM
This stands for random access memory, which refers to memory that can be read starting at any random point in the data.

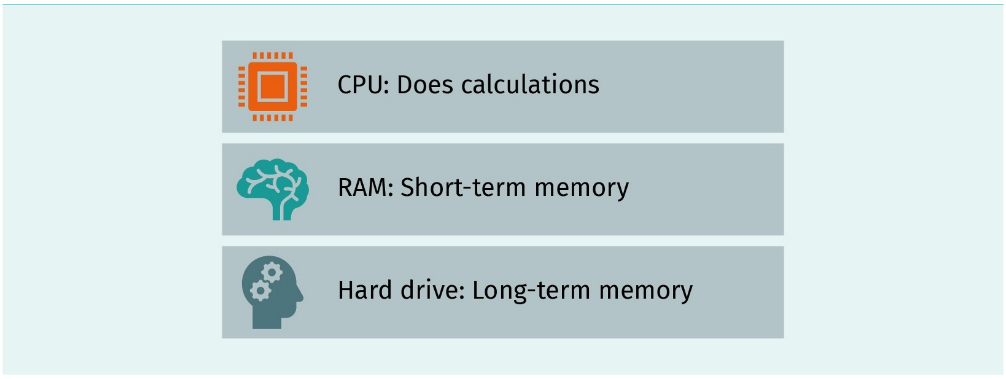
Imagine you are editing your résumé using an application such as Microsoft Word. While you are working on the document, the data in your resume and in the Microsoft Word application are stored in RAM (short-term memory). **Saving** the document transfers a copy from the RAM and into the hard drive for long-term storage. When you are finished editing, you close Microsoft Word, freeing up RAM to perform the next task. Since you have saved the file a long-term storage, it will be there when you want it again, even if you turn off your computer (note that nothing is saved in RAM if you turn off your computer). Thus, the use of RAM mirrors real life: once you finish working on 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 transferring data from short-term memory to a file stored in long-term memory is called saving.

You might wonder why RAM is needed when the hard drive works perfectly well as a storage medium. Basically, accessing RAM (i.e., reading and writing data to RAM) is a lot faster than accessing the hard drive. RAM serves as an intermediate stage between the CPU and the hard drive to quickly provide data for calculations and other uses.

Computer memory may also be compared with a library. The hard drive is the entire library, whereas the RAM is your desk. It can accommodate fewer data than the hard drive. You may want to look at several books, but only so many will fit on your desk. In addition, you can only look at a few books at a time due to the way your brain (i.e., your CPU) is structured. When you are finished with the current books, you put them back into the library (i.e., long-term storage), freeing up the desk (i.e., RAM) so you can cover it with different books.

Figure 6: Analogy: Computer to Human Thinking



Source: Stephen Weese, 2020.

2 Herein, a hard drive is either a magnetic disk storage device or a solid-state storage device (i.e., a large-capacity memory chip) that allows for long-term data storage in computers.

Firmware

Before we discuss firmware, we must briefly explain software. Software is data represented in binary. The difference between normal data and software is that the latter consists of computer instructions – 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 called 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, which means that it only preserves data when the computer is on. This is why in years past (before laptop computers with batteries became more popular), if you lost power while working on a computer project, all your work was lost unless you had saved it to the hard drive before losing power. 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 basic input-output system chip (the BIOS chip) (Tarnoff, 2007), also known as read-only memory (ROM). This means the data are static (i.e., do not change) and cannot be overwritten, except by a special process. Thus firmware is designed to be stable and not easily changed. When a computer first starts, it follows the instructions in the BIOS. These instructions are always the same, which is why they can be written into a stable chip that retains the data even without electrical power (a nonvolatile chip). When a modern computer boots up (i.e., is powered on), the instructions on the BIOS chip tell the computer, for example, to identify all of the hardware connected to the computer (hard drives, monitors, RAM, graphics card, audio devices, webcams, etc.). Once these instructions are executed, the computer copies the operating system (OS) from the hard drive or other long-term storage device into RAM. This is what you are waiting for while a computer “boots” up. Once this process is finished, you can begin using your computer.

Firmware describes computer chips that contain static instructional software. These chips are used for booting up a computer or other 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. For advanced personal computers to perform the myriad of tasks requested of them, they need RAM and a hard drive with significant storage.

Software

Software, as mentioned above, is data that tells the computer to do something – in other words, instructions (remember, all computer data are binary). There are two main types of software: operating systems and applications. Both types of software must be updated, so they are not stored on permanent ROM chips but in long-term storage that can be written to, such as hard drives.

Operating systems

The OS is the first software loaded into RAM during the boot process. An OS is essential for a standard computer to function. Examples of OSs include Windows, Linux, and MacOS. A computer's OS

1. provides an **interface.**
2. controls hardware.
3. runs applications.

Interface
The computer interface allows a human to communicate with the computer. Most modern interfaces include a mouse pointer and icons on a desktop.

An interface is required for people to interact with computers. Without the OS, the hardware of a computer would simply remain idle, doing nothing. Applications need an operating system to run and provide specialized functionality. This is why applications are designed to run on certain OSs, such as Windows or MacOS – they are designed to work with the specific set of instructions that are unique to the chosen OS.

Applications

Applications perform specific tasks. If you want to write a document, you open an editing application. If you want to play a game, listen to a song, edit photos, or browse the Internet, you must open the appropriate application stored on your hard drive or in external memory. All such specialized tasks (i.e., tasks not done by the OS) are done by applications. Applications provide specific instructions to the hardware to complete your task, which is why they are considered software.

1.3 Languages, Syntax, and Semantics

Now that we understand that computers work on a binary level, the next step is to find a way for people to communicate with computers. We have learned that the 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) use special languages that both humans and computers can understand.

There are many programming languages and all perform a similar function: they contain words and symbols that translate directly into binary commands that the computer executes. A single word in a human language (for historical reasons, essentially all computer programming languages use English words) may translate into a long series of binary commands, making it much easier for the programmer to specify what they want to do (Eck, 2009). One such programming language is Java.

Syntax

Besides using words and symbols that translate into 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 make sense in the correct order. For example, putting the words of this sentence in a random order would

Machine language
a set of binary instructions that can be executed by a CPU

ignore proper syntax and make little sense. Syntax also includes rules that determine when certain symbols can be used, which is just like English, which has rules governing where punctuation is allowed.

The Java code below is an assignment statement. It takes the variable `foodItem` and stores it in a dialog box that appears on the screen.

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

The text to be displayed is in quotes. The special code “`\n`” is the “newline” character and tells the computer to insert a new line and place the cursor at the beginning of the new line. Syntax is also 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 do not say exactly what you want according to the rules, they will not work, or they will output garbage.

Semantics

Once you are sure that you have followed all the rules, you must ensure that your meaning is correct; in other words, the semantics. If I describe how to make a peanut butter sandwich but actually want a chicken salad, then I have poor semantics. Although the code may be perfectly readable for a computer, it may do something other than what you intended. Experienced programmers learn many skills to avoid poor semantics; these skills should be learned at the outset of one’s programming career because 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. This practice is especially helpful if the programmer returns to a code after a day or more because the comments tell the programmer what the code is doing, saving the programmer the trouble of figuring it out all over again. Advanced programs consist of thousands of lines of code; without comments, programmers would have difficulty working with such codes.

Speaking computerese

In many ways, a programming language is similar to human language, it contains words and symbols that have meaning, and a specific set of rules governs how these elements may be ordered and when they can be used. Of course, following these rules does not free you from the obligation of assembling a message that makes sense and that explains the concept that you intend to convey. As with learning a new human language, learning a programming language requires starting from the basics and a willingness to devote the required time. The good news is that, once you have learned one programming language, learning other programming languages is greatly facilitated because the concepts and functionalities are similar.

Initially, the semantics of programming languages may seem pedantic and difficult to understand. However, the task is simplified if we “think like a computer”. In English, we have multiple ways of conveying the same concept – this is artistic expression. However, for a computer, such artistic expression is ambiguous and unnecessary. The more precise and exact you are with a computer, the better results you can expect. Consider the following sentence:

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

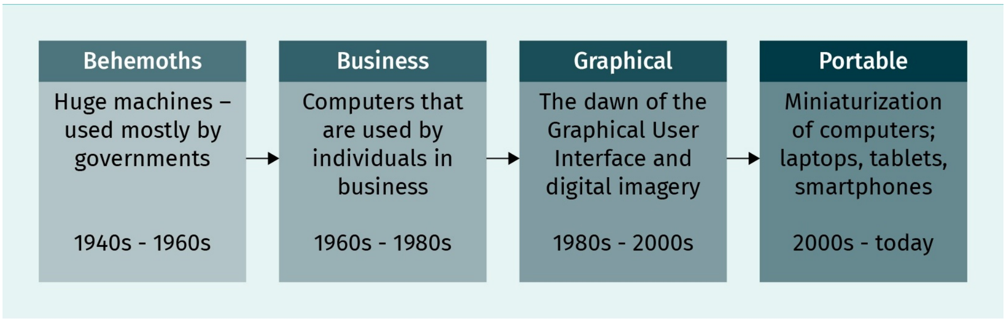
This sentence likely conjures up an image of someone sitting in their underwear eating a hot dog. However, you might also imagine that the hot dog is wearing the underwear. The sentence is grammatically correct but semantically unclear. A programming language must be used with precision to get the proper results.

1.4 Historical Overview of Computers

No machine can claim to be “the first computer.” The development of computers proceeded instead through 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 set of instructions was translated into a “machine language” that told a machine what to do. However, this machine was not computing anything, it was 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 coded text from regular- or “plain-text” messages. Breaking those codes, however, required actual computing. The 2014 film The Imitation Game tells the story of Alan Turing, who invented a machine that could 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 computers, we break their development into four eras.

Figure 7: Eras of Computing

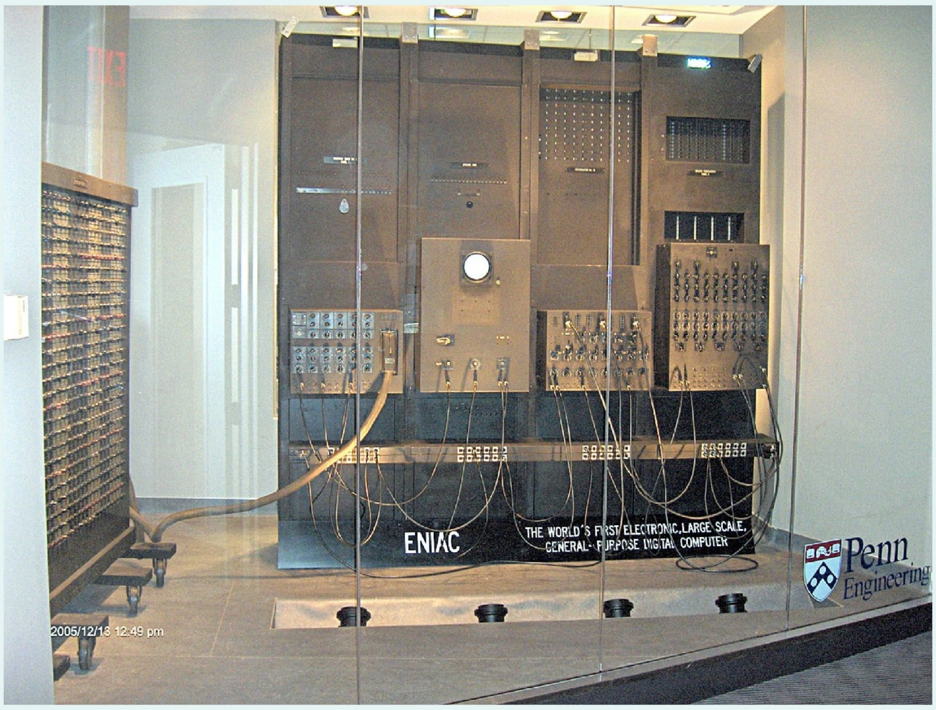

Behemoths
Huge machines – used mostly by governments
1940s - 1960s
Business
Computers that are used by individuals in business
1960s - 1980s
Graphical
The dawn of the Graphical User Interface and digital imagery
1980s - 2000s
Portable
Miniaturization of computers; laptops, tablets, smartphones
2000s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

Source: Stephen Weese, 2020.

Behemoths

The first computers were enormous. They were built from vacuum tubes and wires and occupied large rooms. The input was usually entered by flipping switches and turning dials, and the output was often given via lights or holes punched in paper. They were basically number-crunching machines used to perform large mathematical computations or work with large amounts of numerical data. The U.S. Census Bureau purchased one such computer, the UNIVAC (Universal Automatic Computer), in 1951 to help count the population of the United States (Fabry, 2016). The ENIAC (Electronic Numerical Integrator and Computer) was the first general-purpose computer; it began operation in 1946 and was programmed by a team of six operators (Schwartz, 2019).

Figure 8: ENIAC Computer 1945

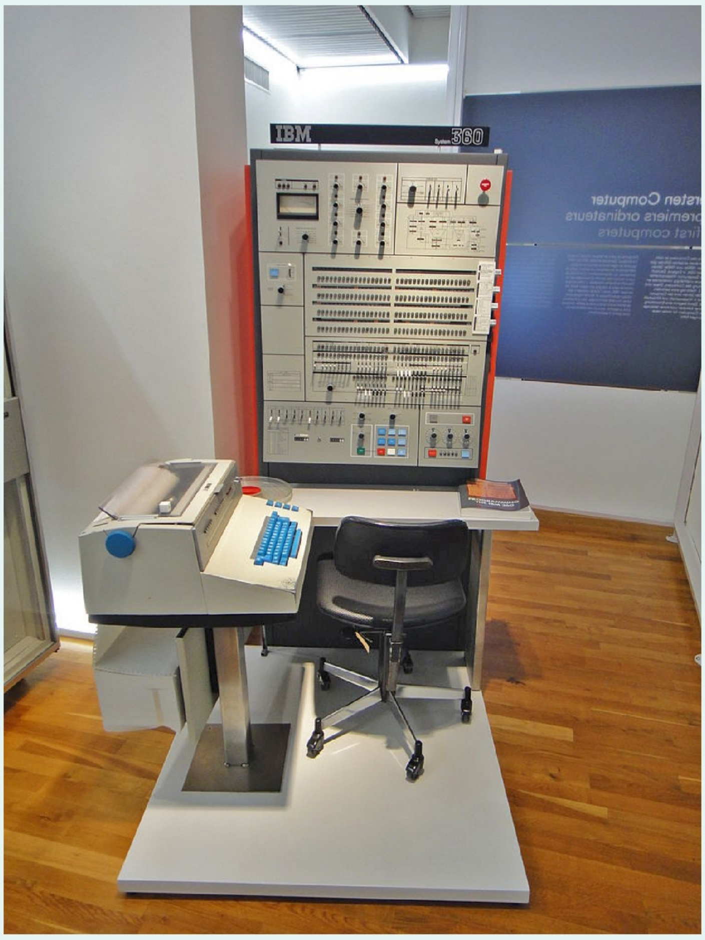

ENIAC
THE WORLD'S FIRST ELECTRONIC, LARGE SCALE, GENERAL-PURPOSE DIGITAL COMPUTER
Penn Engineering
2005/12/13 12:49 pm

Source: TexasDex, 2006.

Business

Eventually, these large computers became smaller and were produced more quickly. They ultimately became affordable for medium-to-large businesses instead of being restricted to government agencies.

Figure 9: IBM System/360 Mainframe 1964



Source: Sandstein, 2011.

The output of these machines was produced by a teletype terminal, which was a typewriter-like machine that output text. Eventually, in the 1970s, “dumb terminals” were created with **monochrome** cathode ray tubes (CRTs) for screens (i.e., like the television screens that predated current flat-panel displays). Unlike today’s personal computers, these screens with keyboards did not have their own CPU or RAM; they were connected to a mainframe **mainframe**, which did all of the calculations. Several “dumb terminals” could be connected to a mainframe, sharing the mainframe’s processing power. Universities became major purchasers of mainframes, many of which ran the Unix operating system (i.e., the predecessor of Linux). The 1980s saw the birth of the personal computer – computers small enough to take home for personal use. Early personal computers were text only – there were no graphics 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 with terminals connected for processing requests.

Graphical User Interfaces

The invention of Apple's Macintosh (or "Mac" for short) computer in the 1980s started a personal computer revolution. Although previous computers had graphics capabilities (e.g., the Commodore 64 in 1982), the Mac 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 used not only for business applications, home budgeting, and the like but also for gaming (i.e., playing computer games). Richard Garriott's Ultima II was released for the Mac in 1985. Initially, the Mac's display was only black and white, but as computers became more powerful, more colors and pixels became available. The main competitor of the Mac was the personal computer (PC), a computer originally made by IBM using an Intel CPU. Although the Mac was technically also a personal computer, the label ""PC"" became synonymous with the Intel-based IBM system (and later other similar "clones"). In the 1990s, Microsoft released Windows, a GUI OS 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 heavy **CRT** monitors giving off waves of heat. Once the much lighter flat-screen technology was refined, it enabled truly portable computers to be mass-produced. In the past, 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 laptop computers and also replaced the bulky CRT monitors used for desktop computers. As the technology developed, more pixels and colors were squeezed into the ever-shrinking screens of tablets and smartphones. Today's smartphone remains a computer – it has a CPU, a display, and long- and short-term memory.

CRT
A cathode ray tube (CRT) is a type of monitor or television that uses a large evacuated glass tube to display images.

SUMMARY
Computer data is stored as binary numbers. This base-2 numbering system uses only the digits 0 and 1. Large sequences (or "strings") of binary digits, or bits, can be used to represent all different types of data.

Translating text, photographs, music, video, and other binary data into information that is meaningful to people requires that the data be contextualized (i.e., given a context). This means that the bits must be given a meaning beyond just the values of zero and one.

After translating useful information into binary code for storage in a computer, it must be translated back into a form that people can understand. That is the job of the computer.

A basic computer consists of a brain (i.e., the CPU) and memory. Memory may be short-term, such as RAM, or long-term, such as a hard drive.

Software consists of binary instructions for a computer and may take the form of either an OS or an application. OSs provide basic functionality for communicating with a computer, including an interface, whereas applications perform specific tasks, such as creating documents or spreadsheets.

Computer languages were created to program computers (i.e., tell the computer what to do). They bridge the gap between human language and binary code. These languages allow people to give instructions to a computer without having to write directly in binary. Like natural languages, computer languages have rules, including syntax, that dictate how to use the words and symbols of the given language.

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

UNIT 2

INFORMATION REPRESENTATION

STUDY GOALS

On completion of this unit, you will be able to …

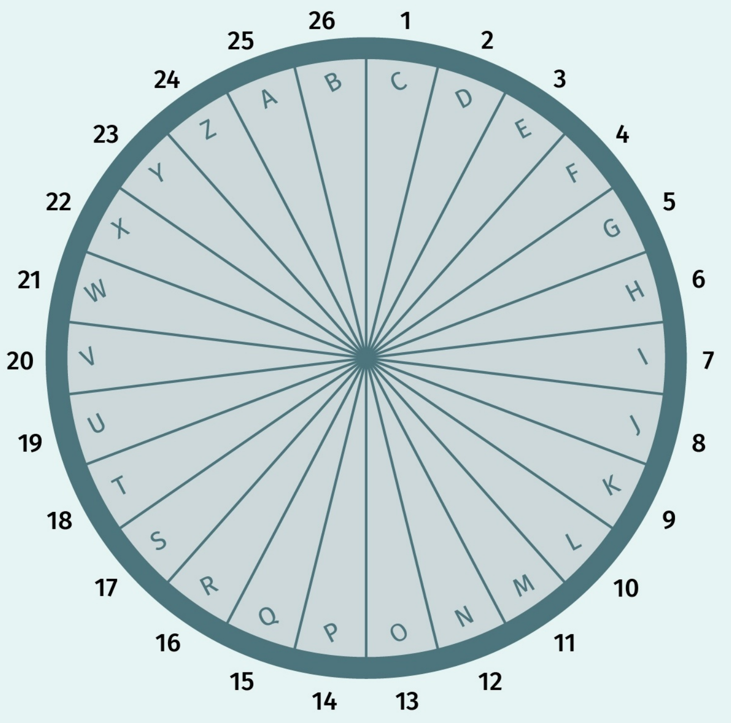
- explain the binary numbering system and how to convert decimal to binary.
- summarize data storage sizes, such as kilobyte, megabyte, and terabyte.
- describe how graphics, documents, music, and other data are represented in memory.
- recommend which data types to use in programming (e.g., char, string, int, float).
- outline how computers perform error checking on stored and network data.

2. INFORMATION REPRESENTATION

Introduction

In 1936, children from all over the United States would listen to the Little Orphan Annie radio show. They knew that, sometime during the show, they would be asked to get their decoder pin– the one they ordered by sending in a proof of purchase of a chocolate drink made by a sponsor (Ovaltine) of the show. This decoder pin had symbols arranged in two concentric rings: the inner ring was formed by letters, and the outer ring by numbers (see the figure below). The children would listen for which letter to position at the top of the inner ring, and then be given a sequence of numbers that composed the secret code for that week. The children 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 the following week. The scheme was brilliant marketing because it not only encouraged the children to ask their parents to buy more Ovaltine but also made the children likely to tune in the following week, keeping up the audience numbers.

Figure 10: Ring Cipher



Source: Stephen Weese, 2020.

The children probably did not realize that they were using something called a ring cipher, a simple device used to encode and decode messages.

The Orphan Annie decoder pin they used is a good representation of how computers deal with nondigital data. The 26 letters of the English alphabet were converted into decimal numbers. You may notice that this conversion is not symmetric – decimal numbers use ten different symbols (0–9), not nearly enough to have one symbol for each letter. Therefore, some of the letters must be converted into two-digit numbers. Since all computer data are stored digitally (as numbers), all non-numeric data must be converted to numbers, not unlike the case of 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 number system.

2.1 Number Representation: Formats

Human beings have ten fingers, which explains why we use a base-10 number system: the decimal system. The decimal number system uses ten symbols, 0 through 9, to represent all possible numbers. Instead of fingers, the first computers had switches that could be turned on or off, giving two possibilities, which led to a base-2 numbering system: the binary system. The binary number system uses two symbols, 0 and 1 to represent all possible numbers. Since humans and machines use different numbering systems, data must be translated between the two systems.

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 for a human. We recognize that each successive digit to the left of the first digit represents the next-higher power of ten. In other words, 806,412 = 800,000 + 6,000 + 400 + 10 + 2. Each digit that is added produces ten new possible numbers, which is why we each column in the figure above represents a power of ten. In binary, there are only two possibilities for each digit, which means that adding a digit produces two new possible numbers, which is why we each column in the figure below represents a power of two.

Figure 12: Binary Table

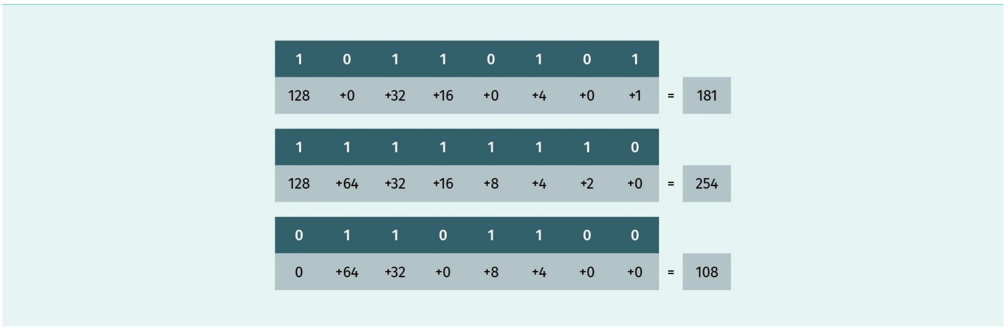
One twenty eights	Sixty fours	Thirty twos	Sixteens	Eights	Fours	Twos	Ones
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	1

Source: Stephen Weese, 2020.

The eight-digit binary number 00101101 in the bottom row of the figure above represents a numeric value to a computer. A human would have to translate this number as follows: $0 \times 128 + 0 \times 64 + 1 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 32 \quad + \quad 8 \quad + .$
$4 \quad + \quad 1 \quad = \quad 45$
There is a 1 in the 32 column, the 8 column, the 4 column, and the 1 column and zeros in the 128, 64, 16, and 2 columns. 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 = 256, meaning that eight digits in binary can represent 256 combinations (0 to 255). Just as we can keep adding digits in decimal to get larger or more precise numbers, we can add more digits in binary to get the same effect.

Byte
eight bits (digits) grouped together in a binary number, often used to indicate the size of data (e.g., megabytes, MB; gigabytes, GB)

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.

Converting decimal to binary is done by dividing the decimal number by two and recording the remainder. This procedure reveals the binary number with the digits in reverse order. For example, let us convert the number 97 in decimal and 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 value 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 represents only two possibilities, a larger grouping of bits is often used for indicating a quantity of data. A byte contains eight bits and can represent simple data such as a letter, a color, or a musical note. Different types of data can occupy different amounts of memory space, which are measured in bytes or multiples of bytes (see the figure below).

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 text document might occupy around 28 KB of storage, a photograph about 3 MB, and a movie around 4 GB. Most modern hard drives are capable of storing around 1 TB of data. Video, applications (especially video games), and operating systems (OSs) typically take up the largest amounts of memory space. Notice that the named quantities of memory in the figure above increase by a factor of 1,000 each time.

Octal

Notice that a relatively large number of digits are needed to represent numbers in binary. A good method to shorten binary is to use the octal number system, which is base 8, so each digit may take on a value 0 through 7. The value multiplying the digit in any given column increases from right to left in powers of eight (…, $8^3, 8^2, 8^1, 8^0, 8^{-1}$, …). So one digit of octal represents exactly three digits in binary (see the table below).

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.

Although we use some of the same symbols in all three systems, they mean different things. The notation "10" means ten in decimal, eight in octal, and two in binary. The joke below 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

Although octal may be useful in some situations, hexadecimal is the preferred intermediate between decimal and binary. One digit in hexadecimal (or “hex”) represents four digits in binary. Hexadecimal is base 16, which means that each digit can take on one of sixteen possible symbols. As shown in the table below, the decimal symbols 0–9 are used for the first ten hex digits, and the letter symbols A–F are used for the remaining six hex digits (i.e., the decimal values 10–15).

Table 2: Hexadecimal Numbers

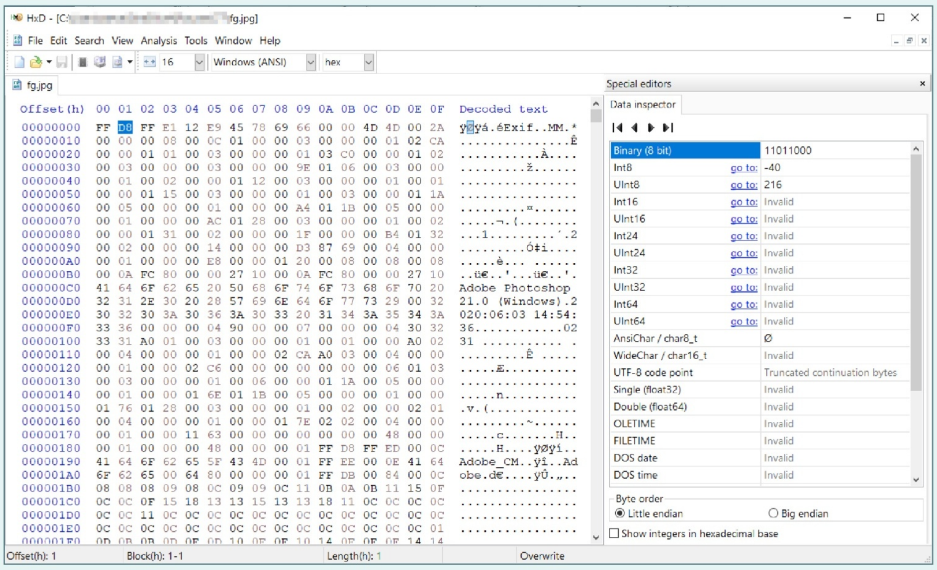
Binary	Hexadecimal	Decimal
1000	08	8
1001	09	9
1010	0A	10
1011	0B	11
1100	0C	12
1101	0D	13
1110	0E	14
1111	0F	15
10000	10	16
10001	11	17

Source: Stephen Weese, 2020.

In computer applications, hexadecimal numbers are usually written in groups of two digits, since two digits represent eight digits in binary (i.e., one byte). Hex numbers are often padded with a leading zero to represent a byte, as in the table above. Raw data are often represented in hex; it is a sort of shorthand for binary. Often, computer scientists would like to analyze the contents of a file, but viewing data in binary form is very difficult and nearly meaningless for a human. As a result, “hex editors” are often used to view a file's content in hex. Another popular option is to use **ASCII** decoders.

ASCII
a binary coding scheme where letters, numbers, and symbols are represented by a single byte of data

Figure 17: Hex Editor


HxD - [C:[redacted]fg.jpg]

File Edit Search View Analysis Tools Window Help

16 | Windows (ANSI) | hex

fg.jpg

```
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 0F 0D 10 0F 0F 10 14 0F 0F 0F 14 14
```

Special editors

Data inspector

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).

The figure above shows the raw hexadecimal data from a photograph. Note the hex number D8 is highlighted, showing a binary value of 11011000. The “decoded text” field displays the ASCII interpretation of the data.

Computing and Numbering

Given that“ computers can only “understand” binary numbers, different representations of data that are more understandable to humans have been developed. These include various codes, such as ASCII, and various numbering systems, such as octal and hexadecimal. Hexadecimal is often used as an intermediate code between computers and humans because (a) it has an exact 1:4 ratio of digits with binary, and (b) it has more symbols than binary, 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 shown how decimal is converted to binary and stored in a computer. This approach works for any datset that consists only of numbers. However, real data also consist of words, images, and even sound. To store this type of information on a computer requires having a system to convert all of these data sources to binary code (i.e., ones and zeroes).

Everything is Binary

Consider your favorite song. 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 before being displayed on a display screen. From Handel's greatest works to The Rolling Stones, from DaVinci to Dali, a computer stores everything as binary numbers. The Bible, the writings of Goethe, the sayings of Confucius – all become binary data when stored on a computer. How is this done? It all depends on context.

Data Contextualization

A relatively simple digital representation of data is to convert alphabetical letters into numbers. Looking back at our decoder ring from the 1930s, we see that it is easy to assign a specific number to a letter. But what about upper- and lowercase letters? We must create separate numbers for each letter. In addition, punctuation gets unique numbers as well. Once we decide how many letters and symbols to represent (including symbols like spaces), we can create a one-to-one mapping between symbols and numbers (of course, these numbers are all in binary). Although ASCII covers only basic English, it has been internationalized over the years in an encoding standard called Unicode.

Digital Documents

ASCII can only display English characters (with a few exceptions), whereas Unicode can represent most of the alphabets in the world, including world currency symbols and italics. Unicode uses 16 bits, so it can represent 2^{16} = 65,536 different symbols. Unicode is often used for internet communication and is at the heart of everything from email to web pages. However, although Unicode allows us to represent raw, unformatted characters and symbols in many languages, creating something like a Word document requires another layer of complexity.

In 2007, Microsoft updated its Office software suite, introducing a new way to store its data files. This new method was based on the Extensible Markup Language (XML). Creating a document, Word or otherwise, requires more information than just letters and symbols (e.g., margins, page size, fonts, text size, text color). Word documents and other documents thus use binary codes to "tag" different sections of text with various attributes, such as bold font or a specific indentation. These tags are stored in binary in a very specific order. When an application reads a Word document, the binary code for the symbols is interpreted according to the tags before being displayed on the screen.

A Word document (or any other data file) is a meaningless string of zeroes and ones without context. When you open a document in Word, the application reads the context in the file and therefore knows exactly how to translate all those ones and zeros.

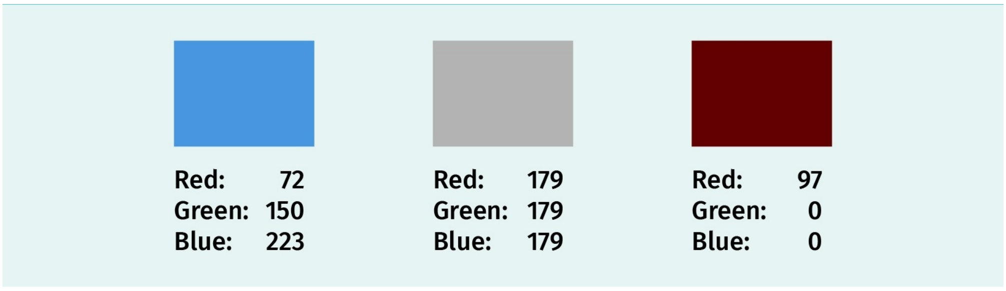
Images

Photographs and other graphics displayed on screens are broken down into pixels (i.e., very small squares, or sometimes rectangles). For instance, you are currently viewing a document that was stored as text with formatting and that has been converted to illuminate certain pixels on your screen and darken others. When a human uses a computer, several layers of conversion and translation must operate continually, carried out by the OS and the application. Once sufficiently large data storage devices were developed, computer displays were designed with very small pixels so that millions of pixels could fit on the display screen, giving a high **resolution,** which makes photographs look realistic and produces very readable text.

Resolution
The number of pixels displayed is called the resolution. This is usually measured as width by height in pixels (e.g., 1920 × 1280).

Many displays now use 24-bit color, which means that eight bits are assigned for each of red, blue, and green. Combining these colors allows the full spectrum of visual light to be displayed. To store any image as binary code, the image must first be sliced up into pixels, each of which is given a binary value showing how much red, blue, and green light intensity to display.

Figure 18: Digital Pixel Colorization



Red:	72	Red:	179	Red:	97
Green:	150	Green:	179	Green:	0
Blue:	223	Blue:	179	Blue:	0

Source: Stephen Weese, 2020.

Since eight bits (a byte) can store values from 0 to 255, each color can have that many degrees of intensity. The absence of a color would be zero, whereas the most intense color would be 255. Using 255 for all colors would display white, whereas using for all colors would display black. This requires sufficient memory to store 24 bits multiplied by the number of pixels on the screen, which means that, to store a high-density image (1920 × 1080 pixels) with 24-bit (three-byte) color, you would need

$$1920 \times 1080 \times 3 = 6{,}220{,}800 \text{ bytes}$$

, which is about 6 MB. Older computers from the 1990s had only 1 MB of memory and had no means of displaying this many pixels and colors, which is why images on such displays looked **pixelated.**

Pixelated
This term describes an image represented on a computer display with pixels sufficiently large that they become individually visible.

Audio and Music

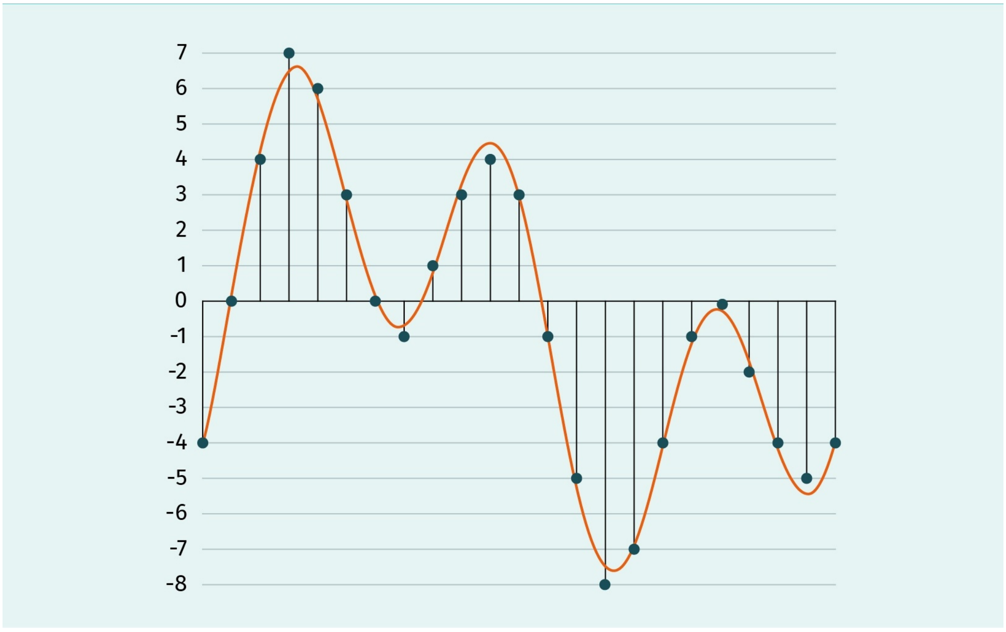
In the physical world, light and sound most often can be described as waves. Information stored as waves (e.g., an audio cassette tape) is called analog information. To store analog audio information on a computer, it must first be converted into digital data. This conversion loses some of the information because analog waves have an infinite continuum of values, whereas a digital representation of waves must simulate the analog wave by **sampling.**

Sampling
taking samples of analog waves over small time intervals.

Analog audio information is thus sliced into tiny time samples, and a digital value is assigned to each time slice, as shown in the figure below. The smaller the slices, the more accurate is the digital representation. However, the time samples cannot be infinitely short, just as we cannot have infinite pixels.

When we observe the macroscopic world with our eyes, we do not see pixels but continuous waves of light. However, when the pixels displayed on a computer screen are small enough, our eyes cannot see the individual pixels but sees the entire image. The same principle applies with audio sampling. If an audio wave is sliced into small enough samples and the samples are reassembled, the human ear does not hear the individual samples, it hears the overall audio signal.

Figure 19: Digital Audio Sampling



Source: Aquegg, 2013.

Even before digitization, the same concept was used for movies. A movie or video is actually a series of still images shown in rapid sequence so that the human eye (and brain) considers the scene as continuous motion.

Thus, whether analog audio is music, speech, or sound effects, it is sampled and stored as binary data, and the context tells the computer to convert the binary data into an audio analog sound wave through a speaker so humans can listen.

Context is Key

Computers need assistance to decipher binary data. There are two primary ways a file (i.e., a discrete grouping of data) is labeled to contextualize the data. The first way is through the file extension. The three or four letters appended to the filename (e.g., docx, jpg, pdf) tell the operating system what type of application can read the data in this file properly. The second way is that files contain headers (or file signatures) at the beginning with specific binary sequences that indicate that the data form 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 the information in binary code. The proper standard must 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, or .dmg. New file types and data contexts are constantly being created.

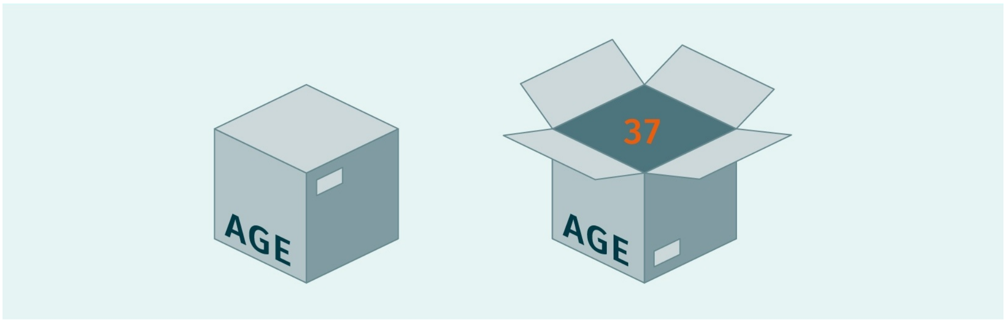
2.3 Data Types

In this unit, we have learned that files, or large sets of data, need a context to be interpreted. On a more granular level, individual units of data must also have context. For example, a data point must be defined for the computer as a number, a character, a memory location, or something else. As another example, consider a Word document, which contains numerous types of data (e.g., numerical, character, formatting, image). The data type of each separate data point must be correctly identified.

Variables

In computer programming, the coder (i.e., the person writing the program code) creates variables to store individual units of data. A variable is a container (or a “box“) for information and has three important defining factors: a unique name, a type, and a memory location. The name is how the variable will be identified in the program code, just as if you wrote a label on a box (see the figure below). The memory location tells the computer where to find this box in the RAM.

Figure 20: Variable as a Box



Source: Stephen Weese, 2020.

Imagine you create a variable called “AGE”. The computer labels the box with the name and stores the box somewhere in the RAM. However, the box is still empty. A data value must be stored in the box because a variable is a container for a data value. Imagine the value of 37 is assigned to the variable “AGE”. When the computer goes to find the box and opens it, the value 37 is discovered within. Like in real life, a box can be reused; we can put other values in it at different times. This is why the box is called a variable – the value inside it can vary. A variable differs from a constant because the latter is declared at the beginning of the program and never changes throughout the program.

The data inside the box must also have a “type”. Just as a paper bag is not made to contain liquids, variables are designed to contain specific types of data.

Integers

An integer is a number that contains no decimal component. Integers include all the natural numbers (1, 2, 3, ...), their negative versions, and zero. This definition is essential to a computer because it does not have to assign any memory to contain the decimal component. An integer would be a suitable data type for the variable “AGE” because the convention is to give a person‘s age as a number without a fraction or decimal.

In the programming language Java, the integer data type is abbreviated “int.” The Java code to declare (i.e., create) a variable “age” is

```
int age = 0;
```

Some programming languages require an initial value to be given when declaring a variable, which is referred to as “initializing.” In this case, the variable is initialized to zero. The program presumes that an actual age will be calculated later or input by the user.

Numerical data types

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

byte: integer from –128 to 127 (eight bits) int: integer from –2,147,483,648 to 2,147,483,647 (32 bits)

float (smallest value): 1.4×10^{-45} to 3.4×10^{38} (32 bits)

float (largest value): 3.4028235×10^{38} (32 bits)

double (smallest value): 4.9×10^{-324} to 1.7×10^{308} (64 bits)

double (largest value): $1.7976931348623157 \times 10^{308}$ (64 bits)

As you can see, data types are specified in bits – this is important to programmers. You do not need to use a huge box to store a pair of sunglasses, it is a waste of space. Likewise, coders should not 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 negative, some positive, and one is zero. To declare these variables in Java, you wite

```
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 appear in (or "float" to) different places in the number. This is an important point for a computer: the decimal is stored within the variable itself. A human would take the decimal placement for granted, but a computer has to be told specifically where the decimal is placed. Many programming languages are also case-sensitive, which means that the variable "age" differs from "Age," and both differ from "AGE". Good coders follow naming conventions for all of their variables.

Non-numeric data types

Most languages, including Java, use a basic data type to store one "character." This represents one symbol in a human language, so it could be a letter, number, space, or punctuation. You might think that we have already covered numbers, but remember that everything must be specified for computers. The numeric value of "2" is a separate concept from the character for "2". Humans a accustomed to accounting for the context when we see the symbol for "2", but computers require more specific instructions. Note that displaying the number 2 on the screen as a specific group of pixels is completely different than multiplying a value by 2.

In English and many other languages, words and sentences are represented by groups of characters strung together, which explains why computers have a data type called a string. A string is simply a group of characters strung together, so it is a complex data type consisting of several simple (or “primitive” data types) variables combined together. Java’s non-numeric data types include (Oracle, 2020)

- char (a character), which is a letter, number, or symbol (16 bits).
- Boolean, which can be “true or false” (1 bit).
- string, which is a sequence of characters (varies in size).
- array, which is an ordered sequence of primitive data types (varies in size).

A boolean variable (named after mathematician George Boole) takes the value TRUE or FALSE, so it has only two possible states. Thus, Boolean variables require a single 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 most computer languages have the string data type for storing language text. In Java, variables of these data types would be declared as follows:

```
char middleInitial = 'A';

String firstName = 'Joseph';

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

Note that the syntax “[]” indicates an array, so “int[]” is an array of integers. The array “`lottoNumbers`”contains **elements** that represent different guesses for lottery numbers. To refer to the second number, you would write “`lottoNumbers[1]`”, which would return the value “`22`”. Note that the first value is “`lottoNumbers[0]`”, which would return the value “`5`”

Element
a single value in an array, addressed by its natural number starting at zero for the first element

Arrays are often used for collections of data of the same type that you want to store in order. You can search through the data sequentially, pick out single array elements of interest, and even sort the elements in order in the array. Arrays may be used for many things, such as a list of options to choose from, a grouping of data, or to collect user input.

Using Variables Correctly

When creating variables, you should consider the data type and the memory requirements. Variables should be given descriptive names that follow the standard naming convention of the given programming language. Novice programmers often name their variables “x” and “y” for simple programs, but these names are nondescriptive and will be problematic in programs of any reasonable length. In programs with hundreds of variables, nondescriptive variable names will make the programming unreadable, including for the original programmer. Variables should be named in a way that describes their use in the program.

Different variable types require different amounts of memory for storage. Integer variables require less storage than floating point variables, but the latter are more precise than the former and cover a larger range.

The data types provided by a programming language must be used correctly. You cannot “add” together two char data types and expect the same result as adding two integers. The computer will interpret the two operations differently. If you have the characters (char data type) “4” and “2” and “add” them, most programming languages will output “42” instead of six. The characters are seen as symbols and are simply strung together. If the variables were numeric (such as int), then adding them would produce the numeric answer “six” rather than a character-based answer. Computers do exactly what they are told, so data types and variables must be treated with precision.

2.4 Redundancy and Error Tolerance

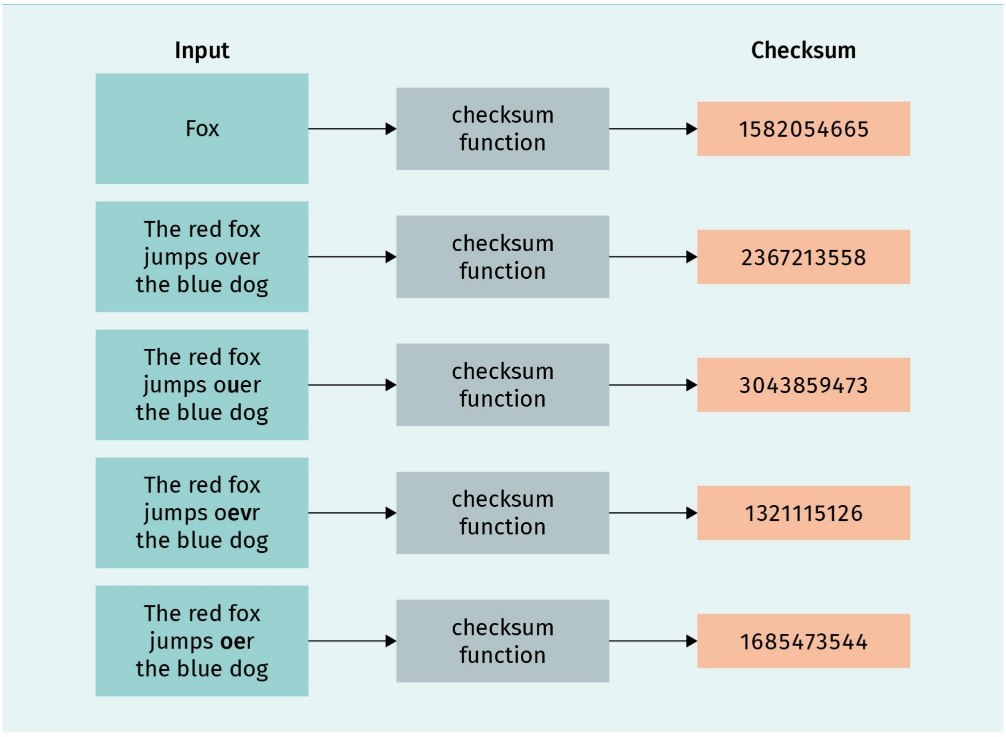
Imagine writing a 100,000-word novel. You carefully read the manuscript after your first draft and check for mistakes – but did you really find all of them? You might do a second check or hire an editor. Are you sure you corrected all of the errors this time? Ensuring perfection in such a large volume of words is difficult. Computers process words and information much faster than humans. A simple image on your screen of 800 $\times$600 pixels has 480,000 pixels. Are they all the correct color? Did some become corrupted in transmission? A computer, although very accurate in calculation, can 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, and then perhaps from Canada to Argentina. Thus, sending a digital photo of a cat from Montreal to Buenos Aires is a journey fraught with errors. Therefore, computers must have built-in precautions against errors.

Storage Error Checking

Some errors are not caused by transmission but by media corruption. Hard drives typically last about three to five years. Some hard drives undergo a gradual corruption, where parts of the physical surface become unstable and can no longer store data. An early method to protect stored files 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 the file is corrupted is to store an exact copy somewhere and compare the versions of the file. However, because some files are megabytes or even gigabytes in size, this approach would be very inefficient. Instead, since any file is one large binary number, it is run through an algorithm that is like a mathematical function – it gives a numeric solution. The checksum is the solution to this mathematical function; it is calculated and appended to the file when the file is saved. When the file is opened, the calculation is run again to see if it matches the number stored in the file. If the calculated checksum matches the file’s checksum, then the file has a very low probability of being corrupted. There is a small probability that changes to a file might give the same checksum, but the probability is small enough that this error-checking method is considered reliable. Usually, corruption in files is very minor and easy to detect with a checksum.

Figure 21: Checksum Examples



Input
Checksum
Fox
checksum function
1582054665
The red fox jumps over the blue dog
2367213558
The red fox jumps ouer the blue dog
3043859473
The red fox jumps oevr the blue dog
1321115126
The red fox jumps oer the blue dog
1685473544

Source: Helix84, 2008.

In the example in the figure above, a slight change in the text produces a significant change in the checksum. Since the checksum is 10 digits, 10,000,000,000 different checksums are possible, so the probability that two texts produce the same checksum is one in ten billion.

Transmission Data Fault Tolerance

The problem of data corruption due to data transmission has existed since before computer networks were invented. Radio transmission of secret codes by military forces faced the same problem, as did communication by telegram. Even a letter written by hand could be corrupted by water or other environmental hazards.

Parity

One of the earliest methods of error checking binary data is called parity. Communication with telephone-based **modems** was serial (i.e., one bit at a time). For every seven bits, an eighth parity bit was added for error checking. Parity checking works 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 eight bits,

Modem
A portmanteau of “modulator-demodulator”; this device translates one type of signal to another.

PREVIEW-PDF, erzeugt: 2024-11-23T13:54:54.832+02:00

the number of ones in the data had to be even or odd. If the parity did not match, then an error was assumed to have occurred, and a retransmission request was made by the receiving computer.

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 ones sum to five, five, three, five, and one. The first bit is not included in the primary data; it is used for error checking and then discarded, just like the checksum at the end of a data file. If a computer set to odd parity receives eight bits with an even number of ones, it assumes that an error occurred in transmission. Of course, if two bits flip during transmission, the byte would pass the parity test. The parity check assumes that the most common error is an error in one bit in an eight-bit sequence. In addition, this scheme does not account for the possibility that the parity bit itself is corrupted. The parity check 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 (CRC), which is similar in principle to the checksum. A block of data is processed by a mathematical algorithm, and the result is appended to the data. After the data are transmitted, the calculation is repeated to check the integrity of the data. The CRC can be applied to any length of binary data and always returns a code of the same length. An algorithm such as the CRC that returns a value of consistent length in binary digits is called a “hashing” algorithm.

TCP/IP Error Detection and Correction

TCP/IP
This stands for transmission control protocol/internet protocol, which is 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 against data corruption. The first layer is the checksum.

TCP/IP divides data to be sent over a network into segments or datagrams. These segments start with a “header” that defines the sending computer, the receiving computer, and transmission settings and provides a 16-bit checksum to verify the data. The maximum size of this initial segment is 65,536 bytes (Stevens, 2004.) When each segment arrives at the destination computer, it is checked against the checksum. If this verification fails, 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 acknowledge (ACK) each datagram received, which is done by giving each datagram a sequence number in the header. The recipient machine acknowledges each sequence number – if one of the sequence numbers is not acknowledged after a certain time, the source computer retransmits the lost datagram.

On another level, TCP/IP also detects broken routes over the Internet and re-routes the transmission to new routes. This fault tolerance is one reason why TCP/IP has been used for decades on the Internet. Part of the network can stop working without taking the entire network offline. Routers and devices that stop forwarding network traffic can be circumvented by finding alternate routes to the destination.

SUMMARY

All data on computers are stored as binary, so a method is required to translate these numbers into concepts useful to humans. One such method involves providing a context for the data, which means assigning the binary numbers a significance beyond their numeric value.

For images, the context is pixels and colors. For documents, the context is the symbols, punctuation, formatting, margins, and other settings. The context for audio data allows binary numbers to be translated into sound waves. Each context involves a pre-formed code or set of rules to give the binary data an extra layer of meaning. This is analogous to the way in which combinations of letters and symbols can create words and paragraphs with more meaning than is possible with only the original letters and symbols.

In computer programming, all variables must be assigned a specific "type," which defines how they are used and their limitations. A good programmer knows which variable types to use and how to name the variables to maximize code readability.

There are numerous standard data types. Some are numeric and serve to represent integers and decimals. Other data types can represent letters, symbols, or character strings. Simple data types can be grouped together into arrays that can be ordered and searched.

Given the possibility of data corruption during data storage or transmission, computers must check their data for errors. The error-checking systems checksum and CRC process binary data through mathematical functions that return a specific value for the given data. If the value sent matches the value calculated on-site, the data are assumed to be valid.

TCP/IP has various built-in error-prevention technologies, including a checksum in the header of its data segments. It also incorporates a system to acknowledge the reception of datagrams whereby the receiving computer confirms the receipt of data. If a datagram is not confirmed as received or is corrupt, a retransmission request is sent to the source computer.

The better we understand how digital data are treated by computers, the better we can design computer programs to create software applications.

UNIT 3

ALGORITHMS AND DATA STRUCTURES

STUDY GOALS

On completion of this unit, you will be able to …

- describe how the term algorithm is used in computer science applications.
- develop flowcharts.
- explain the details of data structures such as arrays and stacks.
- summarize how sorting algorithms work, including quick and bubble sort.
- assess the quality of algorithms.

3. ALGORITHMS AND DATA STRUCTURES

Introduction

Since the early 2000s, Google has been the main search engine on the Internet, capturing over 90% of the global market (StatCounter.com, 2022). But how do search engines work? A Google search follows a set of rules called an algorithm.

An algorithm is a set of rules or instructions for completing a task. An algorithm may apply to people as well as computers. The vast amount of data available on the Internet makes them challenging to search. Google has developed an algorithm to search through these data with excellent results.

According to Google, several algorithms work together to search the Internet. One algorithm determines the meaning: Google can substitute synonyms for words in its search, such as move and relocate, to find similar matches. Another algorithm analyzes the relevance. The keywords in the search are matched to a page – the better the match, the higher the search results. The Google search algorithms also assess the quality of the website in terms of expertise, authoritativeness, and trustworthiness (Google, n.d.).

This is a good start, but the algorithms go even further. The website is also ranked for "usability", which means how well the website displays in different browsers and how quickly the pages load. Finally, the algorithm includes personal information about the user who is searching. If the user lives in Barcelona and searches for "fútbol", Google will prioritize local teams and FC Barcelona would be a higher-ranked result than for someone in Berlin.

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

Is Google's algorithm perfect? No. New sites take a while before appearing in search results. In addition, a good match for a particular search may not appear on the first page of results. However, the algorithm is good enough that it is the top choice of most web searchers around the world.

In data processing, one is often challenged by an extremely large data space such as the Internet. Certain mathematical problems also have a vast number of solutions and can be difficult to manage. In these situations, even a computer has difficulty rapidly searching through every possibility. Thus, the Google algorithm is designed to give a "good" result quickly rather than a "perfect" result that takes much longer.

3.1 Algorithms and Flowcharts

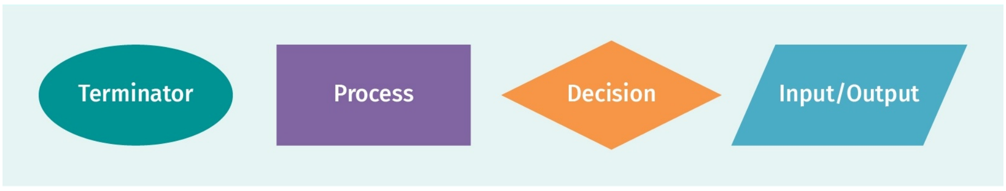
The simplest definition of an algorithm is that it is a set of instructions. This definition works for computers as well as people. Imagine you want to tell someone how to fix a broken lamp. You would probably ask the usual questions, such as “is it plugged in?” and “did you replace the light bulb?” However, to form a set of instructions, you would have to organize things better, as follows:

1. Check that the lamp is plugged in. If not, plug in the lamp. If the lamp works, you are done. If not, go to step 2.
2. Check that the lightbulb functions. If not, replace it. If so, go to step 3.
3. The lamp is broken. Buy a new lamp.

Flowcharts (Flow Diagrams)

Programmers have used flowcharts for many years to translate human instructions into computer instructions. Flowcharts make it easy to understand the structure of an algorithm and implement it in a programming language. Programmers use standard symbols to indicate different types of instructions, as shown below.

Figure 22: Flowchart Symbols

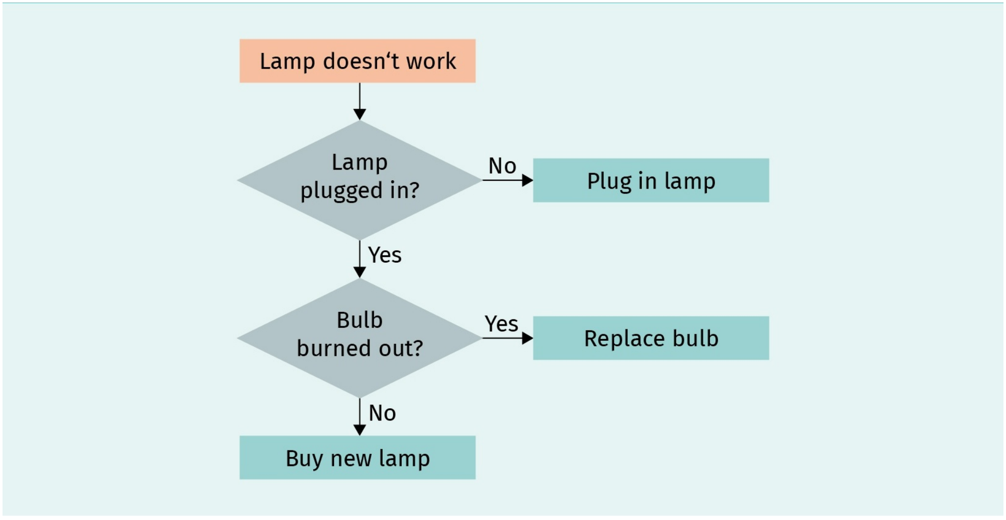


Source: Stephen Weese, 2020.

A terminator symbol (oval or rectangle) starts and ends a flowchart. This symbol represents the external nodes of flowcharts (i.e., where the flowchart connects with external processes). A rectangle represents a process, such as calculating, filtering, or sorting data. A diamond is a decision; multiple pathways stem from a decision. A parallelogram represents input or output, where data are input into to process or results are output from the process. Finally, arrows indicate the flow from the beginning to the end of the process.

Numerous other symbols appear in flowcharts, but these suffice for now. Let us return to our example about fixing a lamp and use a flowchart to illustrate the process (see the figure below).

Figure 23: Lamp Repair Flowchart

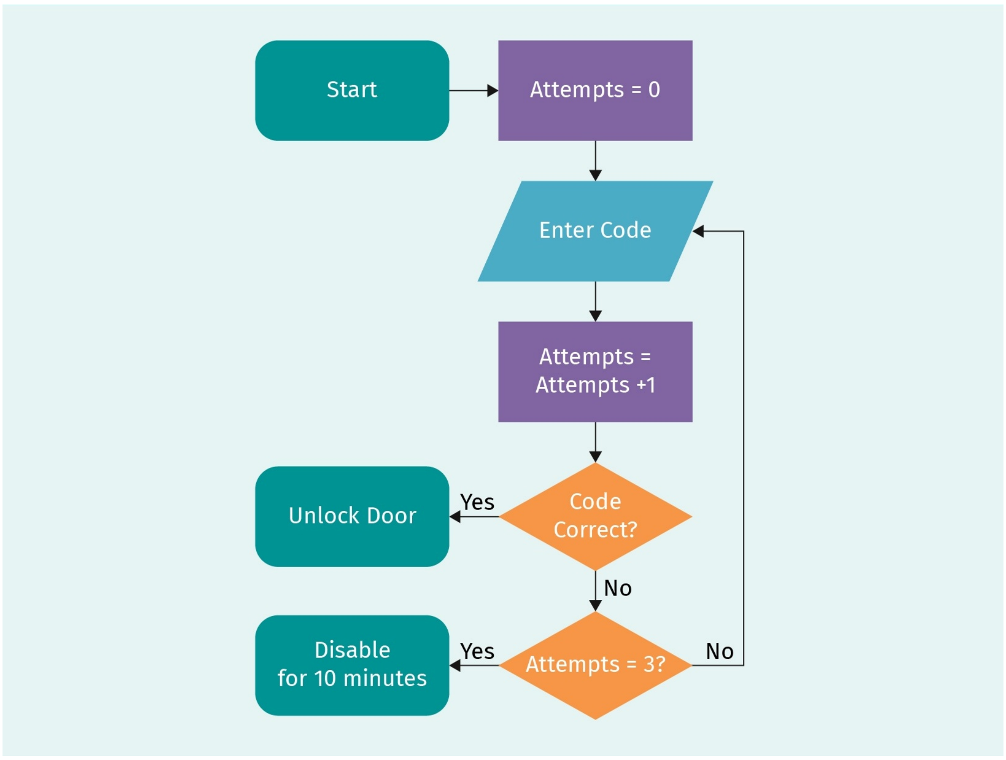


Source: Ilkant, 2006.

The process flow starts at the beginning of the process; from there we just follow the arrows along the pathway chosen. The beginning is a terminator with the starting point “Lamp doesn’t work.” We then proceed to a yes or no decision asking if the lamp is plugged in. A binary decision such as this mirrors the binary nature of computers where “yes” can be symbolized by a one and a “no” by a zero.

Consider a computer-based example of a flowchart, such as a simple key code door lock. With such a lock, the user has three attempts to enter the correct four-digit code to unlock the door. If they fail three times consecutively, the key code door lock locks for ten minutes, during which time the lock will not accept any input.

Figure 24: Door Lock Flowchart



Source: Stephen Weese, 2020.

The flowchart in the figure above illustrates the algorithm for our door lock. Initially, when the lock is activated, we go from Start to the processing step. At this point, the counter variable Attempts is set to zero. The input step then accepts a four-digit code from the user, following which Attempts is incremented by one to account for the user‘s input. Next, a decision is made: is the four-digit code correct? If so, the door is unlocked and the process terminates. If not, the process flows to the next decision. A person would ask the question, “is this the third failed attempt?” However, a computer is different: it checks whether “Attempts” = 3“.” If so, the user is allowed no more attempts and the door is locked for ten minutes. If Attempts < 3, the process returns to the step where the user can enter a code. Notice that every time the user enters a code, the counter variable Attempts is incremented by one. This flowchart represents an algorithm and can be used as a guide to create a program 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” store one unit of information, but to store them in a sequence they must be arranged in a series. Various ways exist to do this in computer memory and each has its advantages and disadvantages.

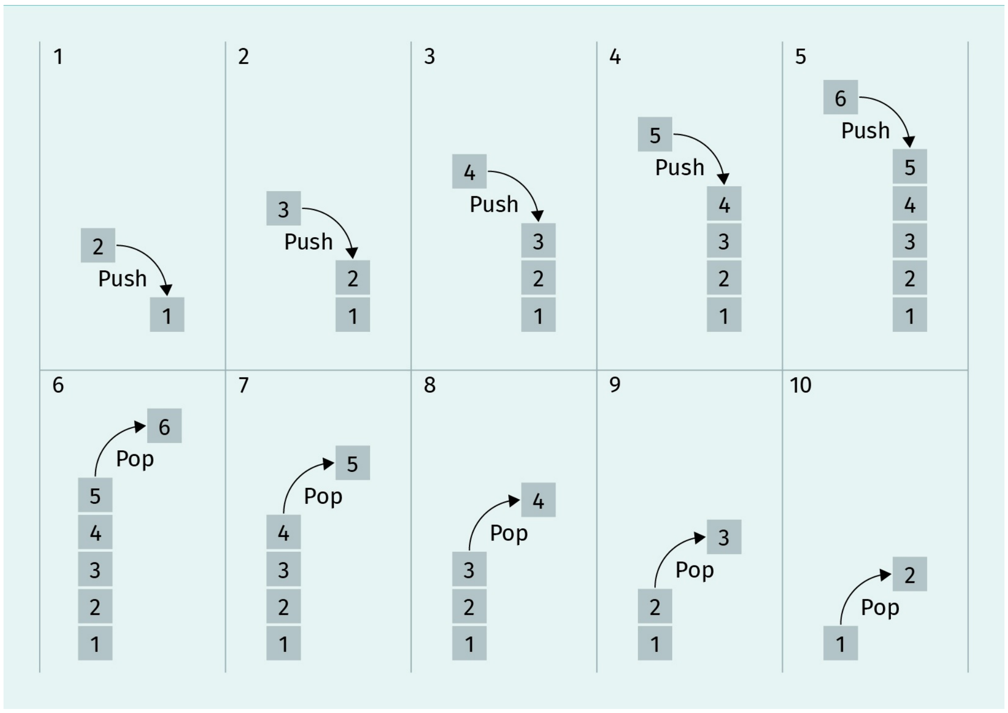
Stacks

A stack is a data structure guided by the philosophy “last in, first out” (LIFO). Imagine you are inputting data into a card game simulator program. The program uses a standard deck of 52 cards, which are randomized and placed in the stack.

The simulator only allows you to draw a card from the top of the deck. This is the last card that was placed in the stack (last in) and is therefore the first to be drawn (first out). Once the game is started, the order of the cards in the stack cannot be changed, and the next card to be played is always the card on top of the stack.

A stack in computer programming allows for two basic operations: push and pop. Pop removes the top item from the stack, and push adds a new item on the top of the stack.

Figure 25: Using a Stack Data Structure



Source: Maxtremus, 2015.

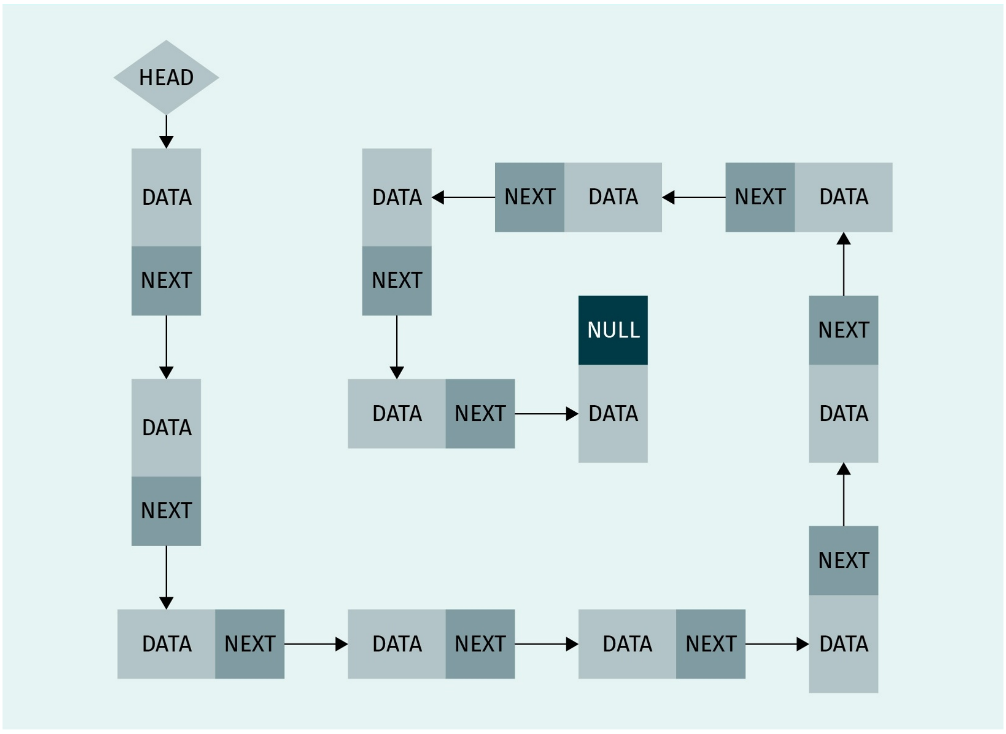
To populate the stack with data, we use the push operator to append one element at a time on the top of the stack. Once all of the data have been pushed onto the stack, they can be read by using the pop operator. Note that the pop operation returns the value of the element and removes the element from 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 requiring less space than more complicated data structures. However, it cannot be searched randomly.

“Random Access Memory” (RAM) is computer memory that can be accessed at any random memory address. Although such functionality might seem obvious, initial computer memory was only accessible in sequence, which means that, to read some particular data, you had to start reading at the very beginning of the data until you got to the particular data of interest. Imagine you read to page 77 in a book and want to continue reading it the next day. Sequential access would oblige you to start reading again from page 1 all the way to page 77 before being able to read new content. Obviously, this approach is inefficient. Random access means you can start reading on whatever page you want, in this case, page 77. You could also, if desired, skip ahead to parts you have not read yet. 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!). Sometimes, this order might be useful (such as in our deck of cards example), and sometimes, it is not. Stacks are useful because they require less memory than other data structures, so they should be used when the data access suits the application.

Linked Lists

A linked list is also a sequential data storage structure. Instead of being stored in a simple stack of data, each data element is linked to the next data element. In other words, a data element in a linked list data structure 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 programming. It represents zero, nothing, or the end of a sequence, depending on the context.

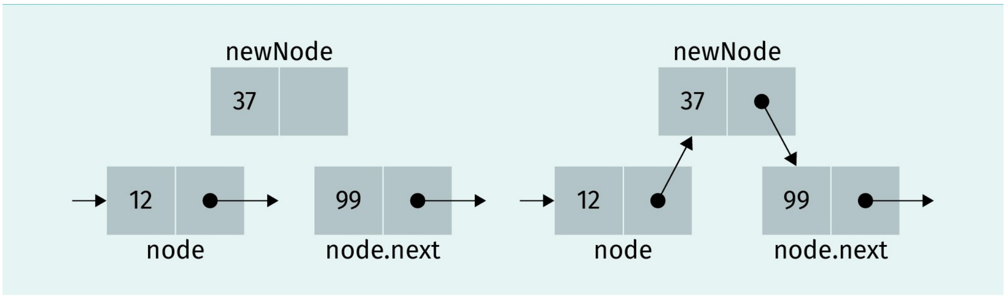
The last link in a linked list data structure does not point to the next value; instead, it is **null**, which indicates that there are no more data elements. Null is often used to end a series of data. The link in each data element is a pointer, which is a memory location (or memory address). It identifies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

In many computer implementations, a linked list is unbounded, meaning that it is not limited to a specific number of elements (other than by the physical size of the computer memory or the structure of the programming language). This is useful when you are working with a set of data of arbitrary size and/or one whose size changes continually.

One thing that linked lists offer that stacks do not is the capability to insert data into the middle of the list. Imagine a linked list as a chain; the chain may be temporarily broken and a new link inserted, making it whole again. To do this with a linked list data structure, one simply changes the pointer of data element *n* to point to the new element (this is akin to breaking the chain). To reassemble the linked list (i.e., reassemble the chain), the pointer of the new element should point to the location pointed to by pointer *n*.

Imagine you have a linked list of integers.

Figure 27: Linked List Example



Source: Pluke, 2011.

The list contains integers 83, 23, 12, 99, 52, and 94, in that order. That means, for example, that the data element containing the integer 12 (call it element 12) points to element 99. If we want to insert into the list a new node containing the integer 37, we would simply have the pointer of element 12 point to element 37, which would break the linked list, and then have the pointer of element 37 point to element 99, which would reassemble the linked list.

Linked lists work well if you need rapid access to sequential data and the ability to insert new data anywhere in the list.

Arrays

One of the most useful and common data structures is the array. Though it requires more memory than a stack, it provides random access for reading and writing data. Usually, bounded arrays are declared in a programming language, meaning that they have a predetermined size. Linked lists offer a similar functionality but are unbounded in size. However, linked lists do not allow random access to elements. Thus, while arrays are more costly in size, they offer the greatest functionality of the sequential data structures.

The data in an array are arranged in elements, just like the other sequential data types. However, each element in an array is given a specific index – a number that identifies it. The index is a natural number and starts at zero in most programming languages. This means that the first element is element zero. It may be bothersome to remember that computers start counting at zero, but when you are working with small amounts of data and fail to use zero as an index, you are basically wasting that piece of memory. Thus, to be more efficient, zero serves as the first number when counting.

Imagine storing 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.

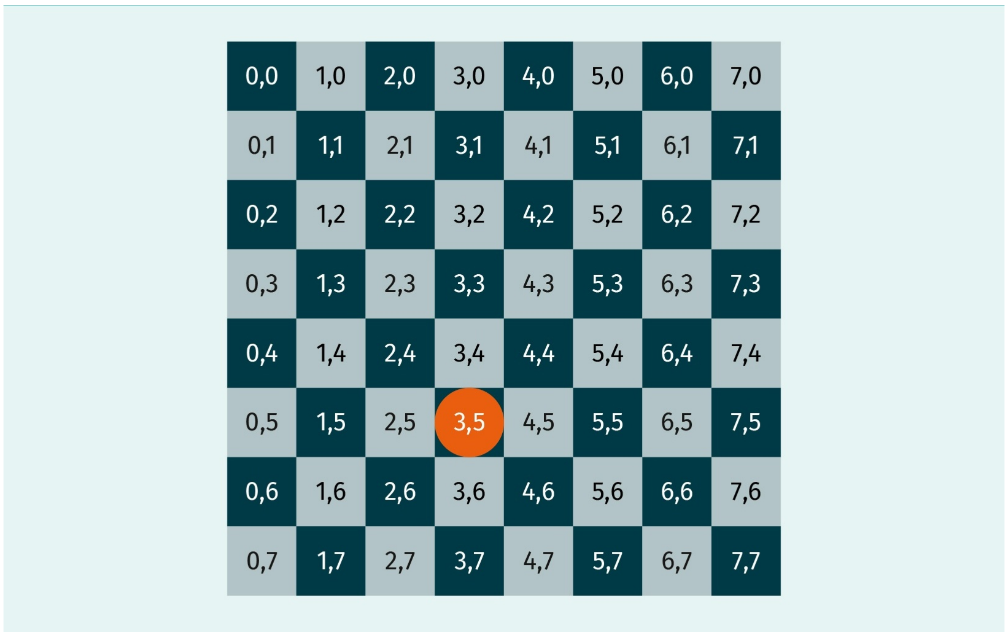
The array name is CustomerName, and the array contains four elements. The first element is element 0. The number in square brackets after the array name is called the index. The value stored in element 0 is “Sandra Whitehall.” The advantage of an array is that you can access any element directly. If you want to read or write to element 3 (i.e., the fourth element), you can do so directly without having to first read elements 0, 1, and 2.

Multidimensional Arrays

Another advantage of arrays is that they can be multidimensional. In a one-dimensional array, each data element is identified by a single index. In a multidimensional array, each data element has more than one index. The most common multidimensional array is a two-dimensional array, which has two indices. Such arrays may be thought of as tables, with the first index giving the row number and the second index giving the column number. This indexing is similar to coordinates, in which an ordered pair of numbers (x, y) give a location in a plane.

Consider the standard checkerboard shown below, which has eight rows and eight columns, giving a total of 64 squares. The squares can contain a red checker, a black checker, or no checker. Let us store these data as RED, BLACK, and NULL.

Figure 29: 2D Checkerboard Array



Source: Stephen Weese, 2020.

The top-left square is square 0,0 – remember that computers start counting at zero. The square with indices 3,5 refers to row 3, column five (or the fourth row and the sixth column if we count like a human). Let us call this array “`checkerBoard`”. If a red checker occupies square 3,5, we could write

```
checkerBoard[3,5] = “RED”
```

This array could store information for the entire checkerboard, where each element of the array would contain `RED`, `BLACK`, or `NULL` stored in it as data. We could then implement algorithms to move the checkers on the board.

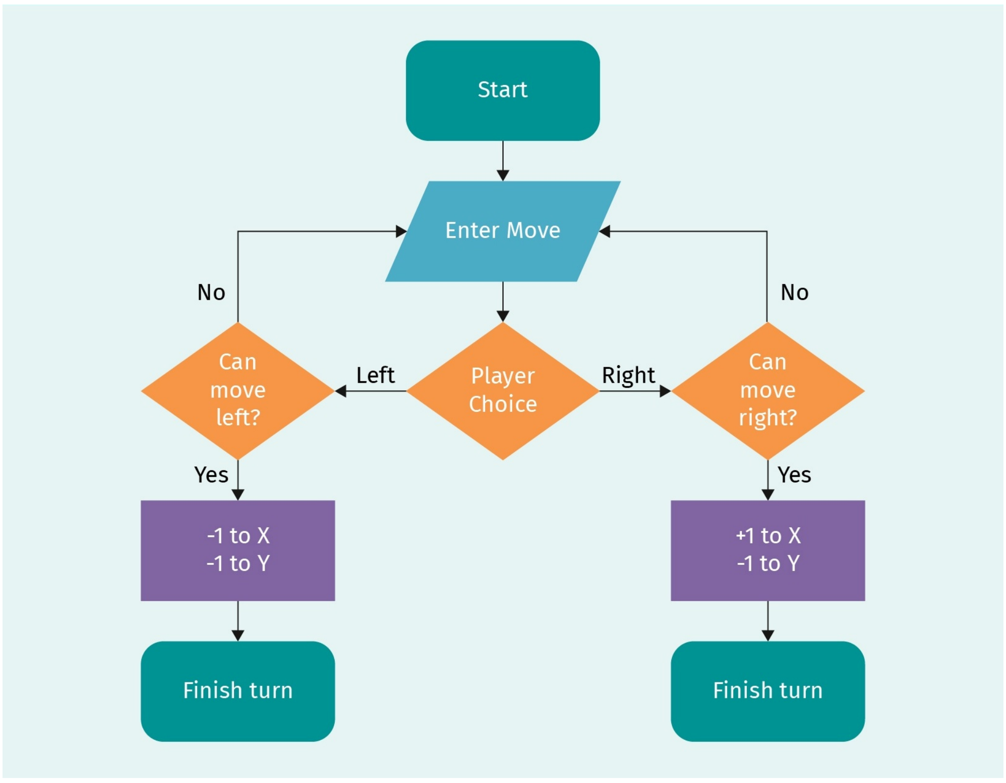
Consider two types of moves, a regular move and a “capture” where you take one or more of your opponent’s checkers. The checkers can only occupy black squares and therefore must move diagonally. If a red checker occupies 3,5, and we are facing the opponent (we are on the bottom of the board), then the valid regular moves are to move to square 4,4 or 2,4. The algorithm would look something like this:

Regular move

1. Player inputs move choice.
2. Can player move right and down?
3. Move right and down is $X \rightarrow X+1, \quad Y \rightarrow Y-1$ (where *X* and *Y* are the array indices).
4. Can player move left and down?
5. Move left and down is $X \rightarrow X-1, \quad Y \rightarrow Y-1.$

The algorithm would have to test if the requested move is valid by checking whether the checker is at the edge of the board (i.e., that the $X < 7$ and $Y > 0$ for step 2 and $X > 0$ and $Y > 0$ for step 4) and whether another checker occupies the destination square (i.e., the destination square must contain NULL). A flowchart for this algorithm appears below.

Figure 30: Checkerboard Flowchart: Regular Move



Source: Stephen Weese, 2020.

This algorithm only applies to a regular move of one piece. A more complete algorithm is needed to handle all the possible moves during a player's turn, which would include the rules for both regular and capture moves. Arrays make it easy to work in two or three dimensions, like checkers on a board.

Arrays with more dimensions

There is no need to stop after two dimensions, although every dimension added exponentially increases the size of the data. For instance, our checkerboard requires $8 \times 8 = 8^2 = 64$ elements of memory space. An analogous three-dimensional cube would require $8 \times 8 \times 8 = 8^3 = 512$ elements of memory space. As a programmer, you should never waste precious memory space 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 (a big “if).” Any seemingly small mistake that is made in code can be amplified exponentially by the power of a computer and transformed into a serious mistake. If your algorithm has a tiny flaw that makes it less efficient, that flaw will be magnified because the task is repeated millions, if not billions, of times by the computer.

The tasks of searching and sorting are analogous to everyday tasks done by people. However, whereas people search and sort for objects, other people, in their own memory, etc., computers sort data. Usually, we have a strategy when searching for something; for example, we often search among things of the same type. For example, cutlery is kept in the silverware drawer, so when searching for a fork, we often search in the silverware drawer. Our search is further facilitated if the silverware in the drawer is sorted by type (i.e., all the forks are in the same tray). The same concept applies to computers: sorting data facilitates searches.

We began this unit with a discussion of a Google search, which searches the biggest database ever made, the Internet. Fortunately, programmers are not always faced with such a daunting search space; more often, they have a simple data set held in something like a linked list or an array.

Gathering and Sorting Data

The process of inputting data into a computer depends on the type of data involved and whether it is already organized (i.e., sorted). This is similar to the situation we face in real life: sometimes information is collected in a random order, and sometimes it is already ordered. More often, the information is somewhat ordered but not perfectly ordered. Most often, data input into a computer are not sorted, so they must be sorted before they are used.

Imagine you have a box full of toys, and you want to sort them to put them in some kind of order. First, you must choose the sorting criteria. You could sort them alphabetically by name, but what if the stuffed “bear” is also called “Peter”? Do you use “b” or “p” for sorting? You could sort by toy size, but would you use toy height or total volume? Sorting criteria must be precise. Even when sorting alphabetically you must decide how to treat upper- and lowercase. Which comes first?

Unicode
an international encoding standard used for numbers, letters, and symbols

Computers cannot work with ambiguity – everything must be exact. Fortunately, all data in a computer are represented by binary numbers, which can sorted based on their value. The most common sorts are numeric or alphanumeric sorts based on one of the common computer encoding systems such as **Unicode.** Since each letter and symbol is associated with a binary value, that value can be used to sort these data. Such sorts must be ascending or descending; in other words, smallest to largest or vice versa. Once you have chosen a sorting method, you can start sorting.

Consider again sorting the toys in our box of toys by toy size. To do this, you visually scan all the toys, select the most voluminous toy, and place it first (or last) in your sorted arrangement of toys. Each time you select a toy from the unsorted toys, you must scan all remaining unsorted toys. However, the number of unsorted toys decreases each time you select the most voluminous among them to put in the sorted arrangement, so each successive search should take less time than the previous search. The search for the last of the unsorted toys requires almost no time because only one toy is left, and no visual scanning is required. In addition, some of the searches for the most voluminous toy may require little time because of a large difference in size, whereas some searches might take longer because two toys are of similar size. Sorting is thus a complex task that takes time and requires a clear set of criteria or rules. In fact, we could call a set of rules for searching an algorithm.

Searching for Data

Let us imagine a different kind of scenario. Today, we live in a highly digital world and look up phone numbers on the Internet. However, not long ago, before the Internet, many governments produced phonebooks (see figure below) containing the phone numbers of all persons and businesses in a local area.

Figure 31: Old Phonebooks

Source: Gentle (2010).

Phonebooks are a perfect example of sorting and searching. All personal (nonbusiness) phone numbers were first sorted alphabetically according to the last name (i.e., family name or surname) of the person followed by their first name (i.e., given name). Of course, the phonebook publishers could have chosen to sort directly by phone number, but most people knew the name and were searching for the phone number. Once all the names were sorted, their phone numbers and addresses were printed next to them in the phonebook.

Imagine a person (Juliette) searching for a name (Josh Rodrigues) in the phonebook. What would be her best search strategy? You might think that she should open the phonebook near the end, since “R” is relatively near the end of the Latin alphabet. However, there may be many names that start with “S” so “Rodriques” may be closer to the middle part of the phonebook. Whichever starting point Juliette chooses, she will have to decide whether to proceed forward or backward with respect to her starting point in the phonebook. If she sees names starting with the letter “S” when she opens the phonebook, she will search backward in the phonebook. However, if she sees names starting with “H,” she will search forward in the phonebook. She can also search through all the names in the phonebook one page at a time, starting at the beginning, or simply skip to random pages to see what letter she finds.

In the initial stages of the search, before finding last names that start with the letter “R,” searching through each page in sequence will take more time than jumping over a group of pages to a new page. However, once you approach the answer, searching each page might become faster. The optimal search strategy depends on the data.

Let us give Juliette the following simple search algorithm:

1. Open the phonebook roughly in the middle.
2. If the name “Rodriques” is on the middle page, stop the search.
3. If the name “Rodriques” is not on the middle page, decide whether to search forward or backward from the middle page.
4. If searching forward (backward), reduce the search to the front half (back half) of the phonebook. You will now search the chosen half of the phonebook, which is half the size of the original phonebook.
5. Go to step 1 and remember that “phonebook” now means the half phonebook chosen in step 4.

This search is called a binary search because it continually divides the data size in half.

Search Algorithms

We shall now introduce some common search rules. Some are more efficient than others, and many depend on the data itself. The data could be random, sorted, partially sorted, or in some other order.

Linear search

A linear search is fairly simple: it starts at the beginning of the data and examines each data element in order until it finds the desired data element. This would be analogous to telling Juliette to start at page one of her phonebook and read every single entry until she found Josh Rodrigues. If she was lucky, Josh would appear near the beginning of the phonebook, but that is not possible with a last name starting with “R”.

Jump search

For a jump search, we need a little more information. Imagine the phonebook is $n = 2,500$ pages long. We calculate the square root of n, which is 50 and which we use for the jump magnitude. Juliette's first jump is thus to page 50, following which she determines whether Josh is on a greater page number. If so, she continues jumping; if not, she stops jumping (she is thus less than 50 pages past Josh's page). Next, she returns to her position before her last jump and does a linear search (page by page) until she finds Josh (she will have to search fewer than 50 pages).

Binary search

The binary search was described in our first example. In a binary search, Juliette would start at page 1,250 of her 2,500-page phonebook. If Josh's page is greater than page 1,250, then Juliette would jump forward half of the remaining pages (i.e., 625 pages). This process continues until she finds Josh.

Depending on the data, any of these techniques could be faster or slower than the others. Note that the jump and binary searches require sorted data. Since a linear search simply reads every entry until it finds the correct one, it works with sorted or unsorted datasets.

Sorting Algorithms

We have seen that searching sorted data is generally faster than searching unsorted data. But how is data sorted? Several different methods are available to instruct a computer how to sort data.

If we go back to our toy box example, where we sorted toys by size (i.e., volume), we can imagine a computer doing something similar. However, our brains allow us to look at a collection of objects and pick out the largest one without focusing on each object separately – this is one of the amazing things our brain does for us. In contrast, a computer must look at each element individually – it is unable to focus on an entire array or linked list as we do with the toys in a box. The search process for a computer is akin to Juliette's search through her phonebook, where the pages are hidden and the computer can only focus on one page at a time.

Bubble sort

A simple sorting algorithm is the bubble sort, in which the largest values ““bubble”” to the end of the data list like air bubbles in a drink. The bubble sort starts by comparing the first two data elements. If the first element is larger, it swaps positions with the second element. If the first element is less than or equal to the second element, then they are already in order so they remain at their positions. The bubble sort then compares the second and third values in the same way, and so on until all the data elements have been examined. Once this is done, the entire process is repeated for all data elements except the last one, since it is now in order. Each pass sorts one more data element. Note that, although some data elements may happen to be in order, the bubble sort verifies all elements regardless.

To better understand the bubble sort, consider the following simple array as an example: (Kayli, Katie, John, Nathan, Anna). The array has five elements. The first step of the bubble sort is to compare Kayli and Katie. Since “y” comes after “t”, these two data elements are out of order, so the bubble search swaps their positions, giving (Katie, Kayli, John, Nathan, Anna).

Next, John and Kayli are compared. Kayli comes after John, so the bubble search swaps their positions, giving (Katie, John, Kayli, Nathan, Anna).

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

Finally, Nathan and Anna are compared. Nathan comes after Anna, so the bubble search swaps their positions, giving (Katie, John, Kayli, Anna, Nathan)

The last element, “Nathan,” is now in its proper place, so the bubble sort compares the remaining four elements, then the remaining three, then the last two. The number of cycles required by a regular bubble sort is the same as the number of data elements in the dataset. However, a modified bubble sort could be instructed to stop sorting if no swaps are made over a single sorting cycle, which would mean that the dataset is sorted. In each sorting cycle, $n-1$ comparisons are made, where n is the number of data elements in the dataset. Thus, our five-element array requires four cycles for the regular bubble sort.

Binary insertion sort

The binary insertion sort is based on the same method we used for a binary search – repeatedly dividing the data in half. To sort the data, it builds a new output list and populates it with data elements from the input list. The input list contains items to be sorted, and the output list contains items already sorted. Imagine you are sorting toys in a box. First you dump all the toys on the floor – that’s the input list. Next, you place the toys 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. We use numerical data this time and assume that the first five elements have already been sorted. The input list to be sorted is (66, 3, 23, 76, 27, 1, 11), and the already-sorted output list is (2, 35, 37, 44, 56). The original list thus had 12 elements. We now need to insert into the output list the first data element in the input list (i.e., 66).

We use a binary search to find the position of 66 in the output list (2, 35, 37, 44, 56). We use leftIndex, rightIndex, and midIndex to denote the positions of the leftmost, rightmost, and middle data elements in the output list. The position of 66 in the output list is found as follows:

1. Start with leftIndex = 0 and rightIndex = 4 (since there are five data elements in the output list).
2. midIndex = (leftIndex + rightIndex)/2 = (0 + 4)/2 = 2. Compare 66 with the element at position 2 (37).

3. 66 > 37, so search the right half of the output list from step 2. This means that leftIndex = 3 and rightIndex = 4.
4. midIndex = (3 + 4)/2 = 3.5. Round down to obtain midIndex = 3. Compare 66 with the element at position 3 (44).
5. 66 > 44, so search the right half of the output list from step 4. Now, leftIndex = 4 and rightIndex = 4.
6. midIndex = (4 + 4)/2 = 4. Compare 66 with the element at position 4 (56).
7. 66 > 56, so insert 66 at position 5.

The new output list is (2, 35, 37, 44, 56, 66). The insertion of the new data into the five-element output list took three comparisons. A bubble sort would take four comparisons, so the binary insertion sort algorithm should be faster.

The next step is to remove data element 2 from the input list and insert it into the output list. This is done as follows:

1. Start with leftIndex = 0 and rightIndex = 5 (since there are six data elements in the output list).
2. midIndex = (0 + 5)/2 = 2.5. Round down to obtain midIndex = 2. Compare 3 with the element at position 2 (37).
3. 3 < 37, so search the left half of the output list from step 2. This means that leftIndex = 0 and rightIndex = 1.
4. midIndex = (0 + 1)/2 = 0.5. Round down to obtain midIndex = 0. Compare 66 with the element at position 0 (2).
5. 3 > 2, so search the right half of the output list used in step 4. Now, leftIndex = 1 and rightIndex = 1.
6. midIndex = (1 + 1)/2 = 1. Compare 3 with the element at position 1 (35).
7. 3 < 35, so insert 3 at position 1.

The output list is now (2, 3, 35, 37, 44, 56, 66) and the input list is (23, 76, 27, 1, 11). This sorting process continues until all elements are added to the output list in the proper order (i.e., increasing order).

The advantage of the binary insertion sort is that it can, at the very least, eliminate comparisons to half of the data elements in the list in the first step.

Quicksort

Although the performance of search algorithms depends on the dataset, Quicksort is 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 greater than or equal to 55 (this number is the pivot and is chosen by different methods) and those less than 55. The two piles are then split around another pivot bringing the total number of piles to four. The teacher then sorts each pile separately and then puts the piles back in order from lowest-grade pile to highest-grade pile. Quicksort can be thought of as a “divide-and-conquer” type of method. In this example, the stack of papers was split into four piles; however, in reality, the data determines how many stacks are required for sorting.

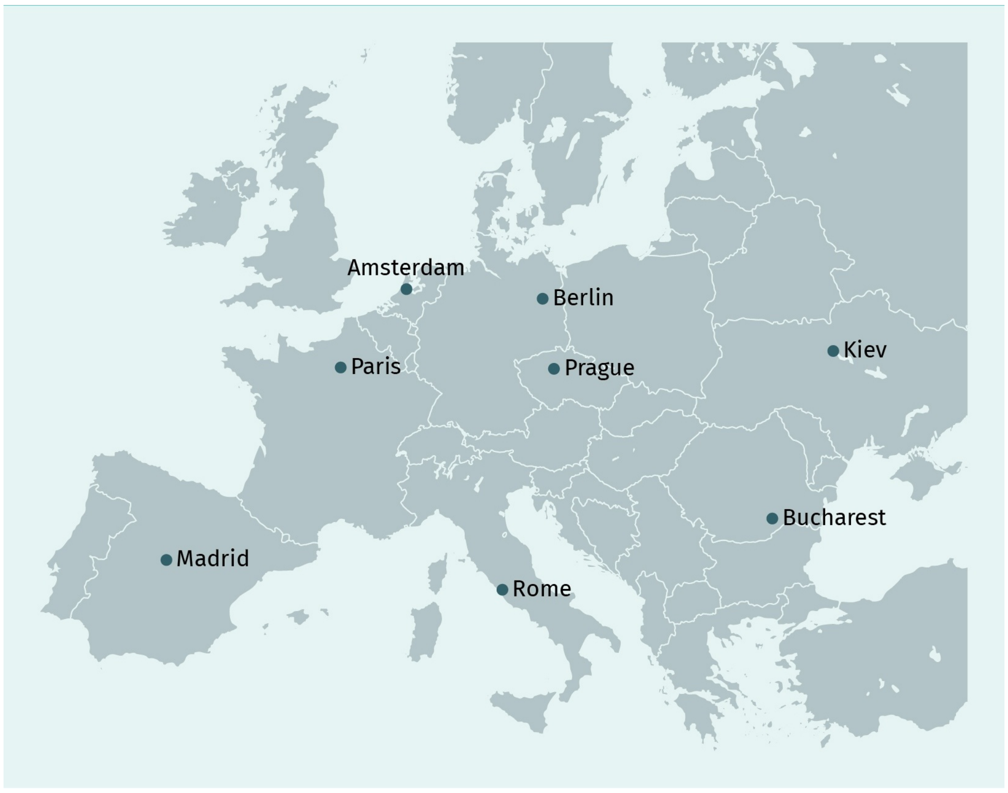
3.4 Quality of Algorithms

There is much to be said about algorithms – enough to fill an entire course. This unit provides an overview that will facilitate further study in the future. Although we have mostly examined searching and sorting algorithms, an infinite number of algorithms are possible for an infinite number of potential tasks. Even within searching and sorting, the number of types of datasets is infinite. Some datasets are small and static; others are multidimensional and dynamic. An illustration of a relatively complex dataset is the traveling salesperson problem. A book exploring the many challenges and permutations of this problem is titled *The Traveling Salesman Problem: A Computational Study* (Applegate et al., 2007).

The Traveling Salesperson Problem

Imagine a salesperson of computer parts who must travel to a series of cities all over Europe to make sales presentations. What is the route between cities is the shortest? No salesperson wants to fly more miles than necessary and waste money, fuel, and time.

Figure 32: Traveling Salesperson Map



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

Our salesperson is Alexa. She has to travel to the eight European capitals shown on the map above, so her route has to include all eight cities. If she starts in Madrid, she is left with seven choices for her next city. If she chooses Rome second, that leaves six cities for her next visit. Thus, the number of possible routes is

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

Eight factorial ($8!$) is the number of possible routes, which is 40,320 routes. This is too many routes for Alexa to analyze to find the best overall route between the eight cities. This problem involves more than just a simple search for a solution: each intercity route must be summed with the other intercity routes to obtain the total distance between the eight cities. The $40,320 \times 7$ distance calculations (between each city) may be doable for a modern computer, but what if the problem grows to 13 cities? Thirteen factorial (13!) is 6,227,020,800 routes, which is multiplied by the 12 intercity-distance calculations for each route. The data size grows quickly (factorial growth is faster than exponential growth after a certain point, depending on the dataset).

Huge data spaces

Exhaustive search
Sometimes referred to as a “brute force search,” such search algorithms look through the entire datset.

Some data spaces are so large that an **exhaustive search** of the data space costs too much time. Instead, programmers develop algorithms that offer a “good” solution as opposed to an optimal solution. With thirteen cities, our salesperson problem involves over six billion routes – with sixteen cities, the number increases to over twenty trillion routes. At this point, any algorithm that finds a good solution must involve a trade-off between time and quality.

Algorithm accuracy

Some algorithms must give the optimal solution. For instance, if we are sorting data, the algorithm cannot stop until all data are sorted with 100% accuracy. If the algorithm requires searching for a specific data point, it must be found. However, if the algorithm only needs to find a good solution to a complex problem, the ““good enough”” standard applies.

Correct algorithms

Any algorithm that is used must be mathematically correct, which is why computer science requires an in-depth study of mathematics. Numerous computing algorithms have already been developed and researching their formulas should yield appropriate solutions. As mentioned above, the effect of an incorrect algorithm can be magnified exponentially once it is put in use.

Algorithm termination

An algorithm must know when to stop its calculations. A good programmer terminates the algorithm as soon as the task is complete to avoid wasting processor time. As an example, consider again the bubble sort. A standard bubble sort passes $n-1$ times through the dataset, where n is the number of data elements. However, the data may be sorted before all the passes are completed. To determine whether the data are sorted, the programmer need only check if no data elements were swapped during the last pass. If so, the data are already in order, and no further passes are required.

Algorithm Complexity

The most (least) complex algorithm for a given task is the one that requires the most (fewest) operations to complete the task. Of course, we want to use the least complex algorithm because it will require fewer calculations and therefore less time and energy. Some algorithms have an exact pattern and always perform a predictable number of steps. Others, like quicksort, perform a variable number of steps (the number of steps is determined by the data itself).

“Big O” notation

In computer science, the “Big O” function (the “O” stands for “order of”) gives the mathematical complexity of an algorithm operating on a dataset with n elements. For instance, the bubble sort takes approximately n^2 steps to complete, so we write its Big O function as

$$O(n^2)$$

However, n^2 increases rapidly as the number n of data elements increases, so $O(n^2)$isn’t very efficient compared with other sorting algorithms (see the table below). The less complex an algorithm is, the more efficient it tends to be.

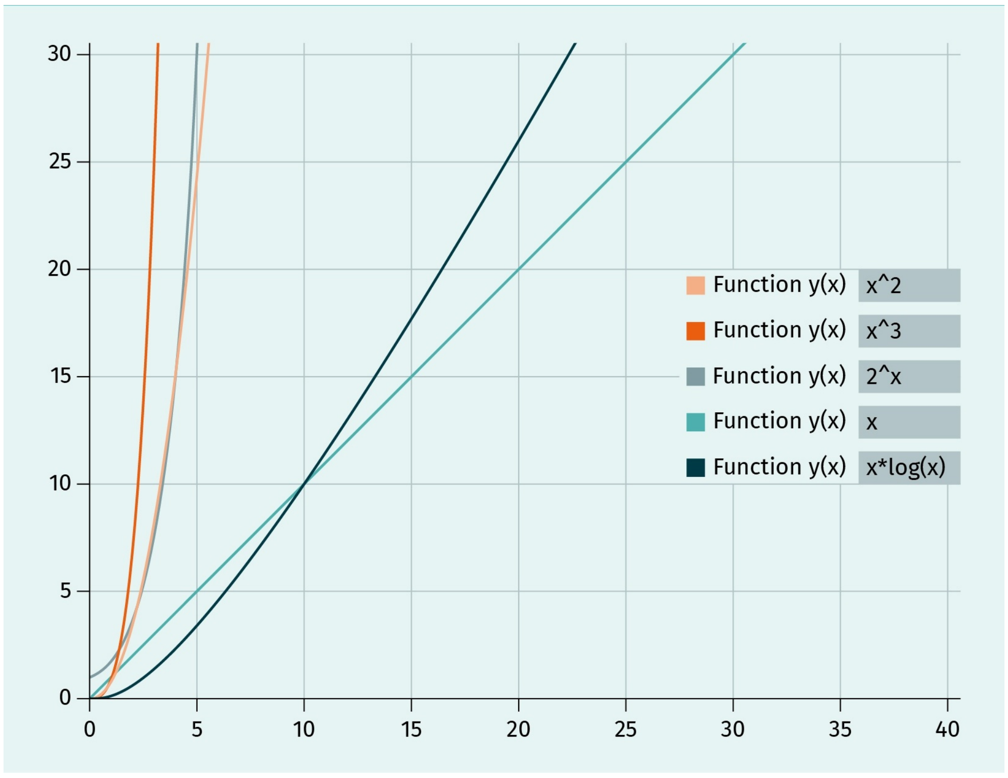
Table 4: Algorithm Complexity

Name	Complexity	Details
Constant	$O(1)$	The number of steps remains constant regardless of the data size. The number in parentheses is the exact number of steps.
Linear	$O(n)$	The number of steps is roughly the same as the number n of data elements in the dataset.
$n\log(n)$	$O(n\log(n))$	The number of steps is roughly n times the logarithm of n. This eficiency is close to linear efficiency for many datasets (see graph). Quicksort falls into this category.
Quadratic	$O(n^2)$	The number of steps is approximately the square of the number of data elements in the dataset. Bubble sort and binary insertion sort fall into this category.
Cubic	$O(n^3)$	The number of steps in the algorithm grows as the data size cubed.
Exponential	$O(n^k)$	This complexity includes any n raised to a power greater than three.

Source: Stephen Weese, 2020.

Algorithms that grow exponentially with the number of data elements are very inefficient and are only useful for problems with small datasets because they quickly become unusable as the data size grows. The same is true for an algorithm that grows cubically with the number of data elements. A better solution should be found if faced with such algorithms.

Figure 33: Complexity Graph



Source: Venkatraman, 2020.

The computation time of any algorithm with an exponent in the complexity graph above grows very quickly as the dataset grows.

SUMMARY

An algorithm is a series of steps used to perform a task and helps to bridge the divide between humans and computers. Computers are designed to perform repetitive tasks at incredible speeds.

One way to describe an algorithm is through a flowchart (or flow diagram). A flowchart identifies the decision-making path that leads to the completion of a specific task. It could be as simple as changing a light bulb or as complex as finding the shortest route between thirteen cities.

Various data structures are available to organize datasets. A stack is a simple data structure that follows the LIFO (last in, first out) convention. A linked list is a more complex structure and has the advantage of being expandable and allowing simple data insertion. Arrays are another complex structure and can be one-dimensional or multidimensional. Arrays refer to data elements in the structure by an index number.

Data stored in such data structures may need to be sorted to facilitate searching. This unit explores different sorting algorithms such as bubble sort, binary insertion sort, and quicksort. These sorting algorithms contain rules to tell a computer how to sort step by step. Once data are sorted, they are much more easily searched. An example of sorted data is a phonebook, where the data are listed alphabetically.

Algorithms must be mathematically and semantically accurate to produce the correct results. In addition, they must not perform unnecessary operations, which wastes time. Some algorithms are optimal and find the best solution for small datasets. However, with large datasets, algorithms must often find a “good” solution without searching every possible solution.

Measuring and comparing algorithm complexity is done using the “Big O” notation. If an algorithm takes approximately $n \times n$ steps (where n is the number of data elements), then it is $O(n^2)$complexity. Algorithm complexity is very important because algorithms can rapidly become inefficient as datasets grow.

UNIT 4

PROPOSITIONAL LOGIC, BOOLEAN ALGEBRA AND CIRCUIT DESIGN

STUDY GOALS

On completion of this unit, you will be able to …

- Invent the basic language and concepts of propositional logic.
- generate truth tables.
- use the conjunctive and disjunctive normal form.
- Explainstart the basic concepts of digital circuits and logic gates.

4. PROPOSITIONAL LOGIC, BOOLEAN ALGEBRA AND CIRCUIT DESIGN

Introduction

In the 1986 film Labyrinth, the protagonist Sarah is trying to get to the center of a labyrinth to save her younger brother when taken captive by the Goblin King. At one point, she is given the opportunity to pass through one of two doors. One of these doors leads to the castle at the center of the maze, and the other leads to certain death. In front of each door is a guard. She learns that one of the guards always tells the truth, and the other always lies, but she does not know which guard is which. She is allowed to ask a single question of one guard to determine which door to take.

Thus Sarah faces a binary choice: one door or the other. Given the rules of the game (one guard always lies and the other always tells the truth), Sarah applies logic to solve 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 first guard thinks for a while and finally answers "yes". Sarah then takes the other door and proceeds to the castle. How did she know which door to take?

There are only two possibilities for the guards: either the first guard always lies, or he always tells the truth. If the first guard is the liar, then he would lie about what the truthful guard would say. So the answer would be false. If the first guard tells the truth, then he would truthfully tell the false answer that the lying guard would say. So again the answer would be false.

No matter which door is correct, by phrasing the question this way, Sarah knows she will get a false answer, which is good enough for her purposes. She thus takes the second door and continues toward the castle.

In this unit, we will learn how to apply this type of logic to statements in common language and also in computer language. Computers can be instructed to make logic decisions, but it all depends on receiving the correct input from the user. If Sarah was wrong about her logic, her choice could have resulted in certain death. Fortunately for her, her logic was sound and she solved the riddle.

4.1 Propositions and Logical Conclusions

To understand how logic works, we must first define some terms and set the rules. We start with the term "statement." A statement is a phrase that claims that something "is" true or false"." A statement differs from a sentence: a sentence may or may not contain a statement. ""Come here now!"" is a command that states nothing 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, the statement could be true or false. The speaker could be intentionally lying or could 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 become “very hot?” However, if Austin claims, ““It is more than 27 °C outside,”” then we have a statement that can be evaluated for truth (Rautenberg, 2010).

A proposition is the idea behind a statement. For instance, claiming “it is more than 27 °C” is a statement, but the proposition behind it is that the environmental conditions include the fact that the outdoor temperature exceeds 27 °C at the moment. Many statements can be used for the same proposition. For example, ““Right now, it is hotter than 27 °C”” would be a different statement saying the same thing. Technically, when we evaluate logic, we are evaluating the truthfulness of propositions. The real power of propositional logic is seen when different propositions are combined and evaluated together.

And, Or, and Not

When we speak to others, we use the simple word ““not”” to indicate negation. ““It is not more than 27 °C”” 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” changes 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 cold outside and it is raining.””This is a combined proposition containing two ideas: one, that it is cold outside, 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 outside but not raining, the sentence is not true. If it is not cold outside and it is raining, the sentence is not true. If it is neither cold outside nor raining, the sentence is not true. However, if it is cold outside and it is raining, then the sentence is true.

We can also use the ““or”” connective to evaluate two propositions together. Consider the example, ““It is cold outside or it is raining.”” For this sentence to be true, one or both of the propositions must be true since the two propositions are connected 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. In mathematical form, the sentence ““It is cold outside and it is raining”” is written

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

where p is the proposition ““It is cold outside,”” and q is the proposition ““It is raining.”” The symbol $\wedge$ indicates that p is conjoined with q (a mnemonic device to remember the conjunction symbol is that it resembles the letter “A” as in ““and””).

The sentence “It is “cold outside or it is raining,”” is a disjunction. This is written

$$p \lor q$$

Finally, an implication means that, if one proposition is true, then another proposition is true. Consider, for example, the sentence, ““If it is raining outside, then I should bring my umbrella.”” The first clause is the proposition p, and the second clause is the proposition q. An implication evaluates the truthfulness of the second proposition (q) only if the first proposition (p) is true. If the first proposition is false (i.e., it is not raining outside), then the sentence implies nothing (i.e., you are free to bring or not bring your umbrella). An implication is written

$$p \Rightarrow q$$

A common mistake many people make is to confuse the terms “imply” and “infer.” An implication is something the information does; it points toward a conclusion, as shown by the notation $\Rightarrow$. Inferring is noting or observing a conclusion that has been implied. For example, the sentence “if it is raining, then the ground is wet” is an implication. There is a cause-and-effect relationship between the first clause and the second clause. The conclusion drawn based on the implication is the inference: “Since it is raining, the ground must be wet.”

Finally, in propositional logic (PL) notation, negation (or “not”)is denoted $\neg$. The negative of proposition “p” is written as follows:

$$\neg p$$

Truth Tables

Now that we have established 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 is the truth table for the compound proposition ““It is colder than yesterday or it is raining outside””? This is a disjunction, and its truth table is shown below.

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 proposition ““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 part is true, so the entire proposition is true. Although the English language might use “or” slightly differently, 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 returns “true” only if one of the two propositions is true. If both are true, then XOR returns false.

Let us consider another example for the case of implication. A father tells 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, as shown in the table below.

Table 7: Implication Truth Table

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

Source: Stephen Weese, 2020.

If the daughter scores a goal (T), she gets candy (T), so the implication is true. If she scores a goal (T) and he does not give her candy (F), then he is not keeping his word, so the implication is false. 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 the implication is 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 by using connectives to form more complex propositions. The most common connectives include "and", "or", and implications. Truth tables are show the result of a proposition for all possible true-or-false combinations of its component elements. True or false is a binary proposition; a computer represents true and false by 1 and 0, respectively.

Table 8: Logic Symbology and Terminology

Name	Symbol	Description
Conjunction (AND)	$\wedge$	This operator connects two propositions, both of which must be true for the result to be true.
Disjunction (OR)	$\vee$	This operator connects two propositions, at least one of which 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 a proposition. If the proposition is true, the result is false, and vice versa.
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. The same logic is used in digital electronics to design circuits and computer chips. To make logical propositions easier to implement in computer code, they are often written in the normal forms: conjunctive and disjunctive.

Conjunctive Normal Form

A logical conjunction is described as using the logical "AND" operator. To explain the conjunctive normal form, a few more terms must be defined. First, propositions are generalized into literals. A literal is any element (including negation) that can be evaluated as true or false. 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 p or ¬p, using any of the variables desired to represent Boolean values.

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

George Boole
A nineteenth-century mathematician and philosopher, Boole created Boolean logic.

$$(p \vee \neg q) \wedge (r \vee s)$$

There can be any number of disjunctions provided they are joined by a conjunction. Technically, a proposition or negation on its own also qualifies, so p or ¬p is also in conjunctive normal form. To evaluate the logic formula above, we can create a truth table. Since the formula contains four variables (or literals), there are 2^4 different combinations of ture-or-false inputs, so the truth table is much larger than before. The table uses ¬q instead of q, which simply means that q is already negated. To create the table, we first fill in all possible true-or-false combinations for p, ¬q, r, and s.

The proper order of operations must be followed to solve a logic formula. First, apply the negations; second, consider the expressions in the parentheses; third, resolve the conjunctions, and fourth, resolve the disjunctions. As an exercise, you may try to create the truth table for the same logic formula but without the parentheses.

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 $p \vee q$ gives true when at least one of these literals is true. Another way to state this result is that the disjunction is true provided both values are not false. If we look at the column for $p \vee \neg q$, we see that twelve of the sixteen evaluations are true. Evaluating $r \vee s$ 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), so the full statement is true only if both $p \vee \neg q$ and $r \vee s$ are true.

Boolean Decision Making

If we think of the logic formula $(\mathrm{p} \vee \neg \mathrm{q}) \wedge (\mathrm{r} \vee \mathrm{s})$ as computer code, all four of the literals are inputs. The computer inputs these values into the logic algorithm, and the result is returned as true or false. This is very important in **branching code**. The computer code decides which code to run next based on the boolean result of the logic formula. Consider the following proposition:

Branching code
computer code that causes the computer to execute a specific branch of instructions as opposed to other branches of instructions

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

TRUE: Withdraw x yen.

FALSE: Send insufficient funds message.

This proposition is represented in simple conjunctive normal form as p $\wedge$ q. Depending on the Boolean evaluation, the computer will take different actions. If both conditions are true, the money can be withdrawn; otherwise, the user gets an error message.

Disjunctive Normal Form

The other normal form, disjunctive, is where any number of conjunctions are connected by a disjunction, as in the following logic formula:

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

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

Figure 35: Disjunctive Normal Form Truth Table

p	q	¬r	¬s	p ∧ q	¬r ∧ ¬s	$(p \wedge \neg q) \vee (\neg r \wedge \neg s)$
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 as 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 represent the inputs "not r" and "not s." After these propositions are evaluated, they are disjoined. If one of the propositions is true, then the disjunction gives true as the result.

Satisfiability

Expressing a logic formula in normal form is only useful to a computer program if at least one outcome evaluates to true. To test a logic formula, it must be evaluated for each combination of inputs – once a result of "true" is obtained, the formula is proven to be satisfiable. The more literals a formula contains, the larger the number of input combinations. The number of input combinations is 2^n, where n is the number of inputs. Thus, a formula with six literals would have $2^6 = 64$ possibilities, and a formula with ten inputs would have $2^{10} = 1{,}024$ possibilities. Filling in such a large truth table is problematic. However, computer programs can be written to evaluate the formula to determine whether it is satisfiable. That means a computer would tell you whether you should use this formula in your computer code. Fortunately, online tools exist to do exactly that. Stanford University in California has an online truth-table generator that can generate relatively large truth tables, and other such services are easily found using your favorite Internet search engine. You can test them with the following formula:

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

This formula has six unique literals (not q does not need to be listed in the table since it is the implied opposite of q), so the table will have $2^6 = 64$ entries. Feel free to devise your own formulas to put into the online truth-table generator. If a logic formula returns true at least once, it is satisfiable. This formula evaluates to true for several of its input combinations.

A simple unsatisfiable formula is $p \land \neg p$. You can evaluate this logic formula by making a truth table or pasting it into the online truth-table generator. Since a conjunction returns true only when both inputs are true, this formula is not satisfiable; p and not p cannot both be true.

4.3 Digital Circuit Design

The subject of digital circuit design would normally encompass an entire class or even be a major course of study. As a programmer, it is extremely helpful to understand how processors and circuits work since these are the specific entities that implement the programs. Studying digital circuit design is also an excellent introduction to how computers "work." When a programmer can think more like a computer, their code becomes more streamlined. This section overviews the terminology and concepts involved with this discipline.

The previous discussion of logic formulas directly relates to digital circuit design. A boolean value of "true" translates to a binary 1 and "false" to a binary 0. The fact that all PL operators have an equivalent in digital circuits is one of the main reasons to learn PL: if you understand logic, you understand basic circuit design.

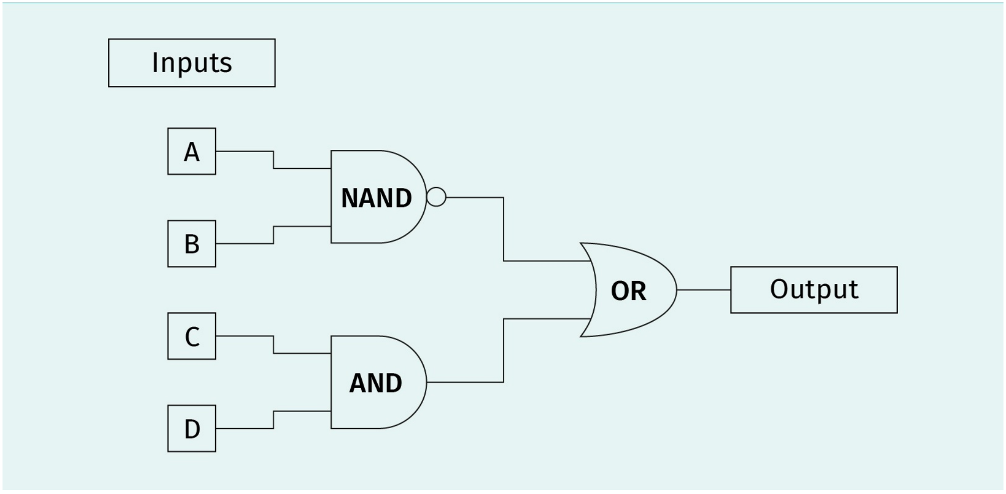
Logic Gates

Digital circuits transmit and manipulate electrical signals that indicate a 0 or 1 state. The essence of a logic circuit is that it accepts various electrical inputs representing 0 or 1 states and returns electrical signals representing 0 or 1 states. This seems rather elementary, but once thousands of logic circuits are combined, complex operations become possible. The good news is that with today's technology, we can use computers to design other computers. We can tell a design program what the inputs and desired outputs are, and it will design the most efficient logic circuit for the situation. However, to supervise this task, a programmer or designer should understand the basics of logical circuits. Even if a programmer does not deal directly with circuits, the code they write can be made much more efficient with a basic understanding of logical circuits. Simple logic gates take two binary inputs and give a single result for output. These logic gates form the building blocks of all computer logic.

AND, OR, and NOT

AND and OR perform true and false evaluations on input (remember that the input consists of the values 1 and 0 now). NOT negates a value, turning 1 into 0 or 0 into 1.

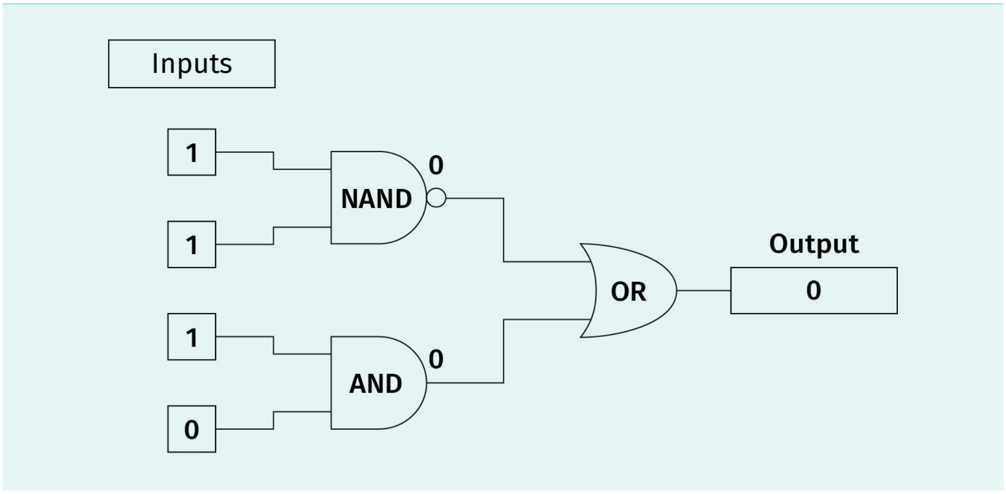
Figure 36: Simple Circuit



Source: Stephen Weese, 2020.

The logic formula for the circuit above is ¬(A ∧ B) ∨ (C ∧ D). The 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 NAND gate indicates negation. The gate for inputs C and D is a standard AND gate. After evaluating the first two gates, the outputs are sent to an OR gate, which outputs the final result. We shall now input data into this circuit to 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 is the output? NAND is just the negation of AND. 1 AND 1 gives a result of 1, and the negation of 1 gives 0. Inputting 1 and 0 into the lower AND gate also gives 0. Both results then go into the OR gate, which outputs 0. Thus, for the given input, this circuit outputs 0. By evaluating the circuit for different inputs, we can generate a truth table for this circuit.

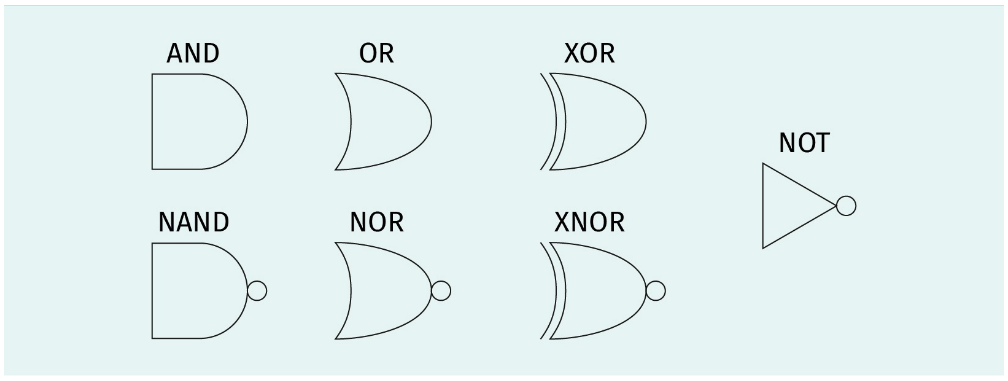
Figure 38: Truth Table for Circuit Example

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 truth table above shows that the inputs of 1, 1, 1, 0 are one of three possible input combinations that will give a result of 0 for this circuit. The other input combinations are 1, 1, 0, 1 and 1, 1, 0, 0. Now that we've seen an example circuit, let us look at the common gates that make up a circuit.

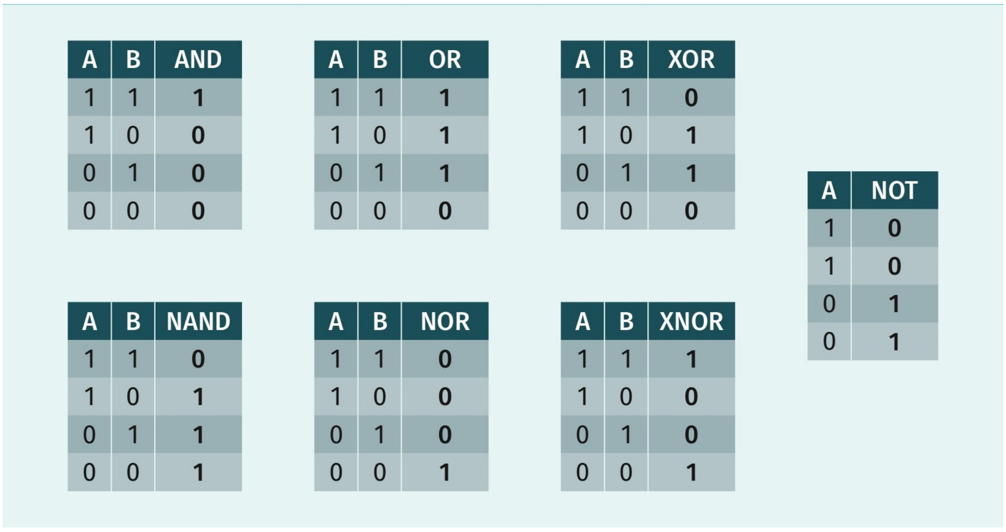
Figure 39: Common Logic Gates



Source: Stephen Weese, 2020.

We know how AND and OR gates work since they work the same way as a conjunction and a disjunction in logic formulas. The NOT is simply negation – it converts 1 to 0 and 0 to 1. The NAND gate is the negation of an AND gate, and a NOR gate is the negation of an OR gate. The XOR (exclusive or) is a new gate. Above, we mentioned that, in English, ““or”” we usually mean one or the other but not both. The XOR gate expresses this exactly – it outputs 1 only when its inputs differ. The XNOR is the negation of an XOR gate, so it outputs 1 only when its inputs are the same. The truth tables for these gates are shown below.

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**

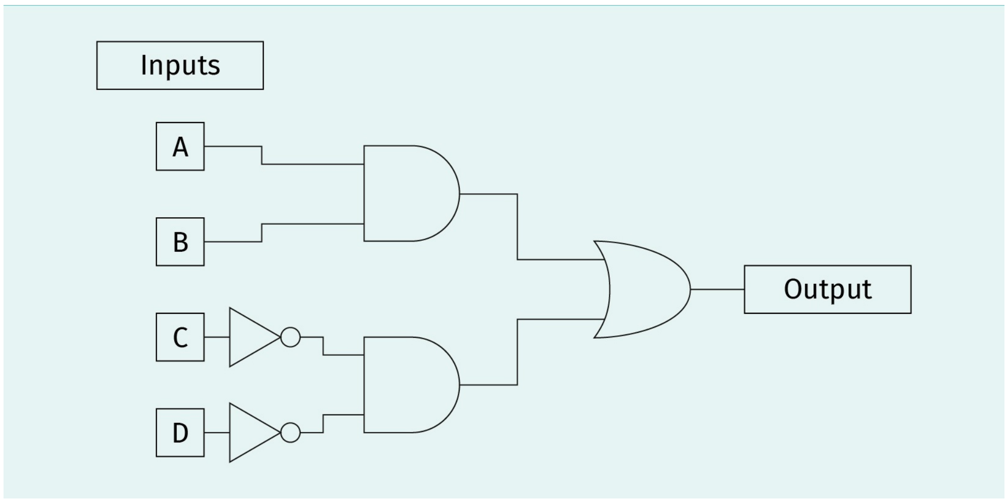
A	B	NOR
1	1	**0**
1	0	**0**
0	1	**0**
0	0	**1**

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 of propositional logic. When multiple gates are combined, more complex formulas can be represented. Let us convert the formula $(p \wedge q) \vee (\neg r \wedge \neg s)$ from the previous section into a circuit design. Converting the variables to the more common A, B, C, and D and the operators to gates gives (A AND B) OR (NOT C AND NOT D)

Figure 41: Digital Circuit Based on Formula



Source: Stephen Weese, 2020.

To obtain NOT C and NOT D, a NOT gate is placed just after inputs C and D to negate both inputs. Note that NOT C AND NOT D differs from C NAND D. The reader is encouraged to make a truth table for each to see how they differ.

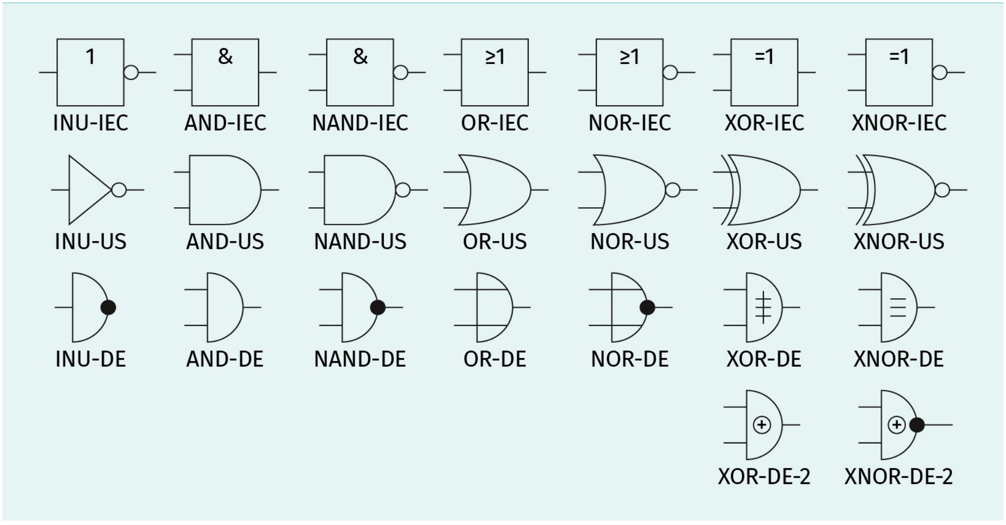
Logic, Formulas, and Design

PL gives us a solid working basis for circuit design. The concepts from PL translate into similar constructs in a digital world. For example, true becomes 1, and false becomes 0. Complex circuits can be designed by combining many gates to generate the desired output based on all possible input combinations. Numerous tools are available online to design and test digital circuits. One such tool is available from Academo.org, and you can search for others online.

International use

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 below.

Figure 42: International, German, and U.S. Logic Gates



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

SUMMARY

Propositional logic takes the concepts embodied in human-language statements and represents them in equation form. These logic equations can be used to evaluate whether claims or propositions are true or false. Propositions may be combined with conjunctive or disjunctive connectives and may also be negated. Combining several propositions with connectives gives a logic formula. Since computers can only make true and false decisions, PL is useful for computer programmers.

Logic formulas can be written in the conjunctive and disjunctive normal forms: A formula in the conjunctive normal form consists of several disjunctions joined by a conjunction. A formula in disjunctive normal form consists of several conjunctions joined by a disjunction. These normal forms help simplify logic formulas used in computer programming and circuit design.

Digital circuit design describes the individual bit level where the inputs of 1 and 0 are the equivalent of “true” and “false” in logic formulas. Logical operators are embodied by logic gates such as AND, OR, NAND, and XOR. Given binary inputs (zeros and ones), these gates give consistent outputs. Many gates combined can be used to represent complicated mathematical functions.

Truth tables list the results of a logic formula or a digital circuit. They show the exact output for all possible inputs. Digital circuit design is a vast field that employs countless engineers who design digital circuits.

UNIT 5

HARDWARE AND COMPUTER ARCHITECTURES

STUDY GOALS

On completion of this unit, you will be able to …

- describe the basic elements of computer architecture.
- explain how processors and memory work.
- illustrate how a computer processes input and output.
- report how operating systems and hardware communicate.
- summarize high-performance computing.

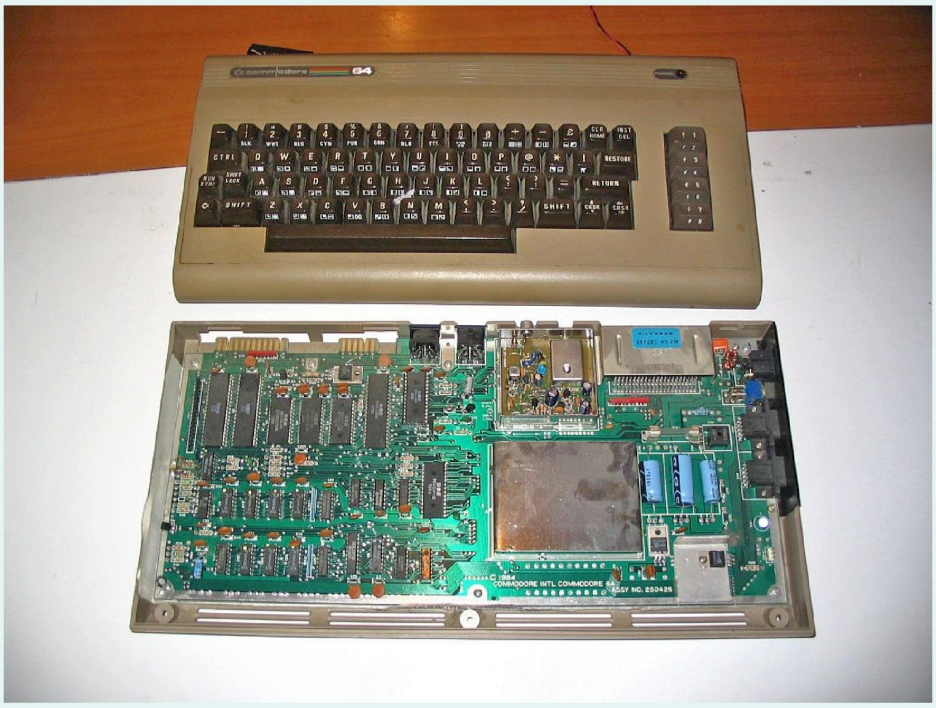
5. HARDWARE AND COMPUTER ARCHITECTURES

Introduction

In 1981, Jack Tramiel owned Commodore, a computer and calculator company. Commodore also owned the chip-making company MOS Technologies ("MOS" stands for "metal-oxide semiconductor"). It was at this time that the company embarked on an ambitious project: offering a high-powered PC for about $600.

The new computer was based on the iconic MOS 6510 chip, which MOS Technologies developed. Since Commodore owned the chip-making company, they did not need to negotiate with an independent company for the chip – they got them at cost. These chips operated at about 1 MHz clock speed and had a data width of 8 bits and an address width of 16 bits. Compared to 21st-century chips, they were very limited, but in 1981 they were more than adequate. Sixteen-bit addressing allowed the Commodore 64 to address 64 KB of RAM – which is where the name came from. This amount of RAM was unheard of at the time, especially at $600 (Commodore 64, n.d.).

Figure 43: Commodore 64



Source: Dake, 2005.

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

The Commodore 64 was built around the BASIC programming language, so some programming knowledge was required to use the Commodore 64. In the 1980s, the Commodore 64 became a gateway for many young programmers to full-time jobs in the industry. Today, many in the computer industry have never heard of the Commodore 64 and its record-setting history, yet it is considered by some to be the best computer ever built (McFadden, 2019). The Commodore 64 used the Von Neumann architecture, a computer architecture still used today.

5.1 Computer Types and their Architecture

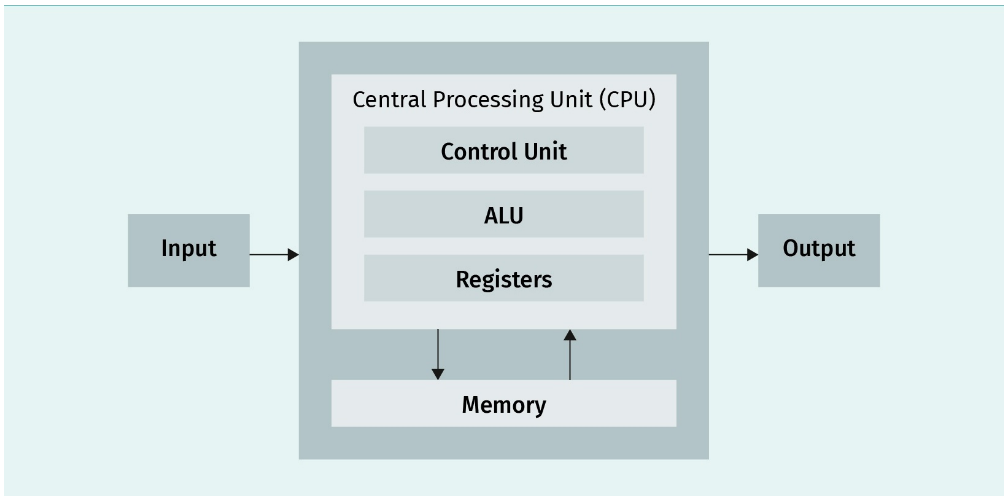
At its most basic level, a computer consists of a processor and memory. The processor performs various calculations and executes commands using the data stored in memory. The computer takes input from memory and processes it to produce a unique output.

The bits (zeros and ones) stored in memory fall into two categories: data and instructions, both of which are 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 basic concept behind Von Neumann Architecture, a basic design for computer systems that remains in use today.

Besides the processor and memory, an input-output system is required. The basic architecture does not require a specific definition of the input-output (I/O) system, but it is assumed to exist since a human must communicate with the device to use it. Suffice it to say that some system must exist to enter data (in binary, of course) and to output data.

A system based on a central processing unit (CPU) checks for the memory address of the next instruction and then reads the instruction. If the instruction requires data, the CPU gets the address of the data and reads them. Next, it processes the data and sends the result 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, performs basic arithmetic calculations on data, such as adding and subtracting, and also performs logic operations such as AND and OR. The control unit controls the operation of the ALU and communicates with the input and output devices. It interprets processor instructions and carries them out. A CPU has its own internal memory storage built into the chip as registers. These are very small in size and typically hold 8 or 32 bits. Newer processors have 64-bit registers, meaning that each CPU register can store 64 bits (or eight bytes). These registers are used for multiple purposes.

Von Neumann Registers

The Von Neumann design has the following five specific registers:

1. Program counter: contains the memory address of the next instruction to read from memory.
2. Memory address register: contains the address of the current instruction in memory or the next data transfer address.
3. Memory data register: contains the contents of the memory address pointed to by the memory address register and the data to be transferred.
4. Accumulator: contains data that have been processed or are about to be processed, including arithmetic or logic results.
5. Current instruction register: contains the current binary instruction being executed.

Data Buses

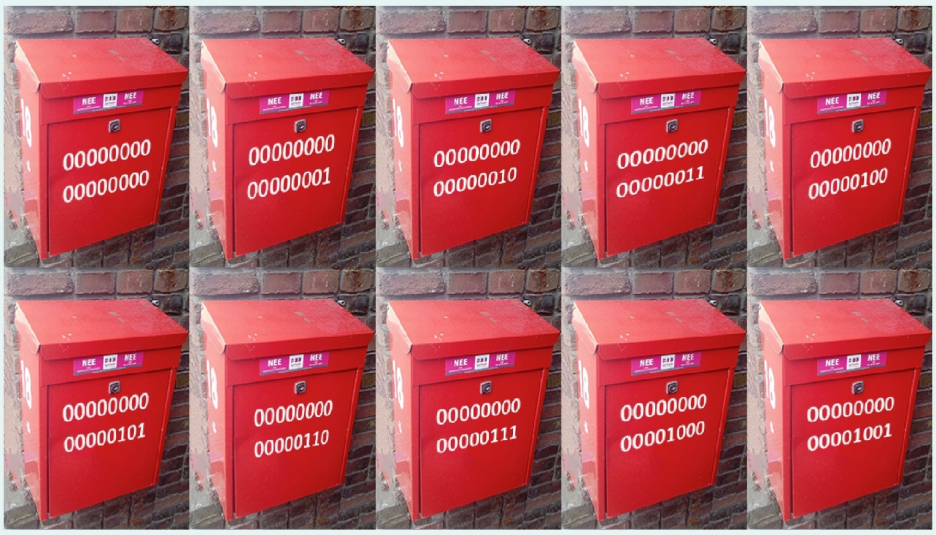
A data bus (or simply “bus”) is simply a connection used for data transfer. Data and instructions are transferred between memory and the CPU via a bus. The CPU also transfers its output to the relevant devices via a bus. The standard Von Neumann design incorporates an address bus, which transfers memory addresses; a control bus, which transfers instructions and special signals from devices (such as their status); and a data bus, which transfers data.

Since the CPU not only performs calculations but also controls devices, it needs to know the status of other devices and, in particular, whether they are ready to perform a new task. The control bus sends and receives these status updates from devices and sends instructions to them.

Memory Addressing

The memory addressing architecture assigns each byte of memory a unique binary address. You could think of each piece of binary data as residing in little mailboxes, each with a unique binary address (see image below). This approach means that the number of bits used for addressing determines the number of data bytes available. For instance, the Commodore 64 had 64 K of addressable memory. This was accomplished by using 16-bit addressing. The total number of addressable bytes is 2^{16} or 65,536 bytes. This was considered 64 K. [At the time, a kilobyte referred to 1,024 (2^{10}) or 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 mailbox). Some registers store memory addresses and use them to point to where the data are stored. This strategy keeps track of the last CPU-accessed memory byte so that the CPU can read the next byte in sequence (if reading sequentially).

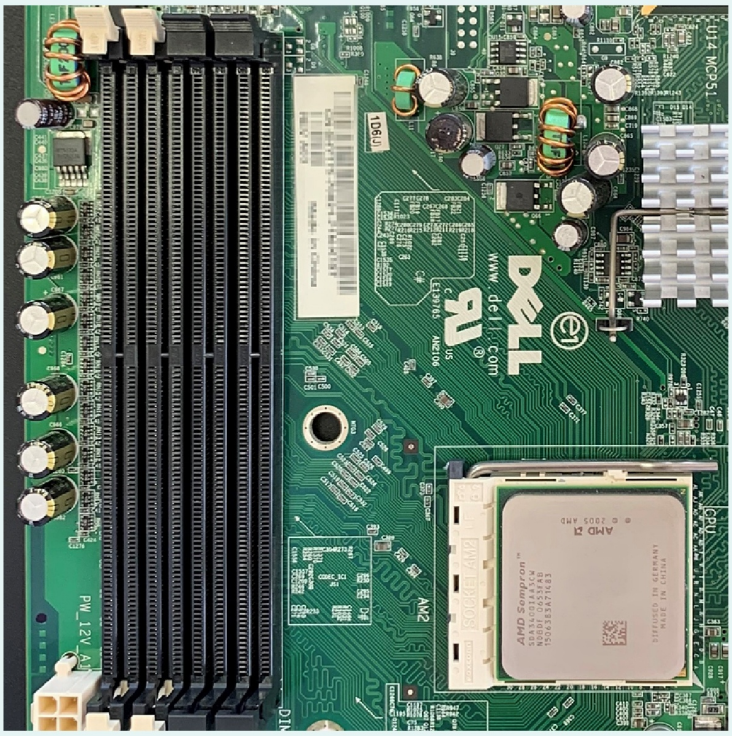
Memory Usage

In the Von Neumann architecture, memory can store both data and instructions. This lends versatility to the design because the available memory can be split between data and instructions however the programmer wishes. The computer program itself can also be written to use the memory any way it wishes. A slight disadvantage of this design is that the data and instructions use the same 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 more rapidly access the random access memory (RAM) from the CPU, a local cache of memory was added to the CPU . Since the CPU continually accesses the RAM, PC designers place the memory very close to the CPU on the motherboard. The motherboard has tiny lines etched into its surface that act like little wires to transmit binary data signals back and forth.

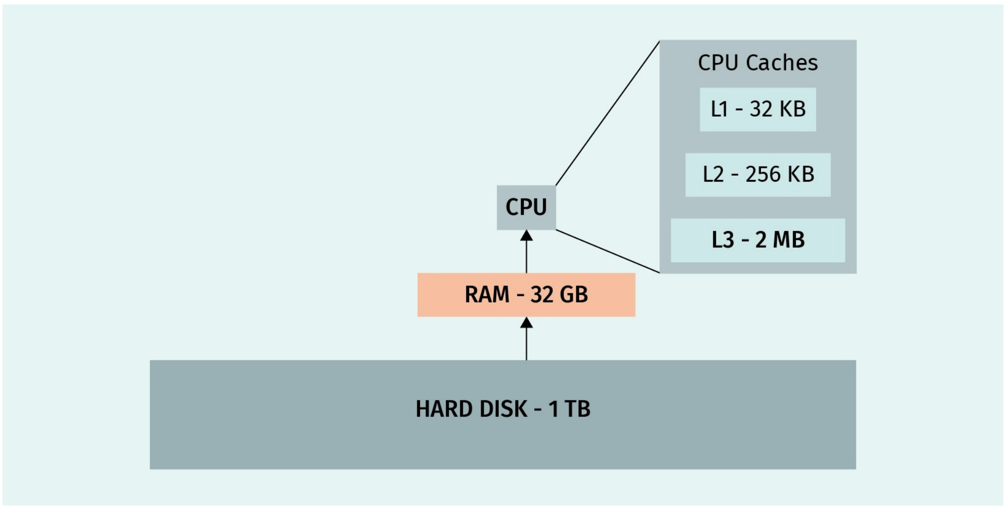
Figure 46: CPU and RAM Slots on Motherboard



Source: Project Kei, 2020.

Although the CPU is very close to the RAM on a PC motherboard, when operating at billions of cycles per second, this distance can still cause slowdowns. This is where cache memory comes in; it functions as a small storehouse of memory directly on the CPU, 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 (modern solid-state drives are faster than magnetic disk hard drives, although both are slower than RAM).

Figure 47: Computer Memory System



Source: Stephen Weese, 2020.

Imagine you have a PC with a 1 TB hard drive, 32 GB of RAM, and an Intel i7 CPU. The CPU has three levels of cache with 2 MB, 256 KB, and 32 KB of storage. The closer you get to the CPU, the smaller the memory size needed. As you can see from the diagram above, the processor has five stages of memory. The hard drive is the slowest but also the largest. It stores your entire library of applications and data. However, you do not need to access all that at once – it depends on which task you are performing at the moment. When you open an application, it copies itself into RAM. RAM is smaller in memory size but much faster than a hard drive (although solid-state drives are faster than hard disk drives, both are slower than RAM). RAM is smaller because it only stores what you are currently working on instead of storing everything. The CPU directly accesses data from the RAM and copies a smaller segment of the current instructions or data to its level-3 (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 to overcome the weakness of the Von Neumann design, where memory is accessed through a single bus, whether for data or instructions. Using algorithms, the CPU decides which data should be placed in its local caches to access them very quickly. Notice that the L1 cache in our example is only 32 KB of memory, whereas the hard disk is 1 TB, which is approximately 30,000,000 times larger. This system sifts the data down to the most significant data at any given moment and provides these data to the processor from the L1 cache. If the required data are not in the L1 cache, the CPU checks L2, L3, and RAM. The closer the needed data are, the faster they can be accessed.

Personal Computers and RAM

Currently, Windows 11 is the most popular desktop operating system in the world, claiming about 75% of the market (StatCounter.com, 2024). When a desktop computer is powered on, it first loads (i.e., copies) the operating system into RAM. Windows 11 requires at least 2 GB of RAM. Thus, if you have 32 GB of RAM, as in the last example, only 30 GB would be available for data and applications. Remember that RAM is a holding space for current programs and data that are currently being used by the CPU, and the contents of RAM can change rapidly. When you close an application, memory is freed up for a new application. RAM is **volatile** memory, unlike the hard disk, which is more permanent. If you were opening a picture of your cat in photo-editing software, both the software and the data from the cat photo would be stored in RAM while you were working on it. The memory is released when you close the application.

Volatile
memory that needs power to store data – turning off the device erases the content.

5.3 Input and Output

Data are stored in memory that extends from the hard drive to the CPU cache. When a person uses a computer, they create new data for input. In response, the computer produces output. Input begins with the user and ends with an output device.

Table 9: Input Devices

Input device	Description
Keyboard	A user enters keystrokes that are converted into electrical signals and then into binary code.
Mouse	A user enters input by moving and clicking the mouse, creating electrical signals that are converted into binary coordinates.
External Devices	Universal serial bus (USB) devices such as cameras and microphones also input electrical signals that are then converted into pixels and frames for video or digital sound waves for audio.
External Data Sources	A computer can receive new data through a network card. It can also receive data from 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 millions of 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 on paper.

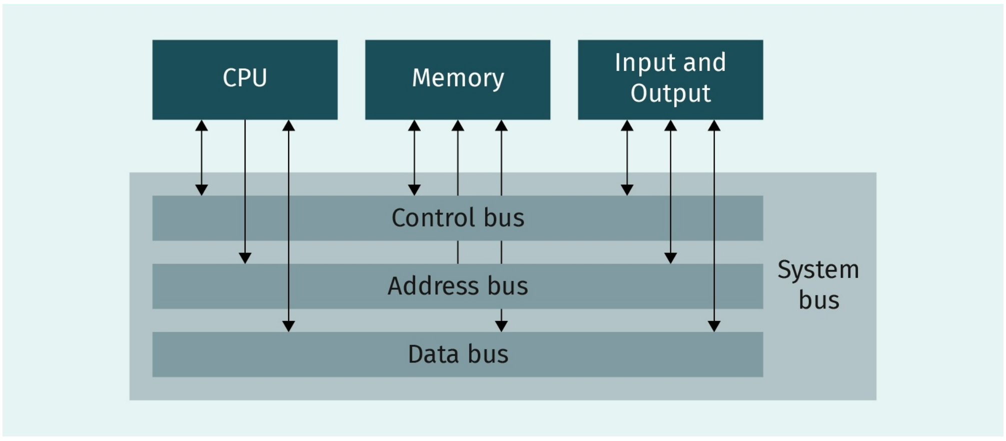
Output device	Description
Speakers	The computer converts sound to analog from digital and plays it through speakers.
Network card	A computer can send and receive binary information over a network.

Source: Stephen Weese, 2020.

The tables above list some typical input and output devices at each end of a computer system. However, an internal input-output system exists for the processor. Data from devices such as the keyboard or a microphone must be processed before being sent to the computer; this is often referred to as input-output processing. Once the data are properly arranged into bytes matching the proper **word** size, they can be sent through the data bus to the processor.

Word
the standard data size used by a processor: a 64-bit system uses 64-bit words; data are 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 items are merged into a system bus.

Input and output devices on a computer are typically assigned a binary “channel code” for identification by the CPU. When a CPU receives this code, it expects a status message to follow from that device. The CPU can also use this code to send information back to the device. This information is transmitted by the control bus. Control codes for a device can include things such as ready, busy, error, and input or output requests. 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.

At this point, we have discussed communication between hardware elements such as the CPU, RAM, monitor, and mouse. The operating system adds an extra but necessary level of complexity to a computer system. Without an operating system, the user could not communicate with the computer.

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

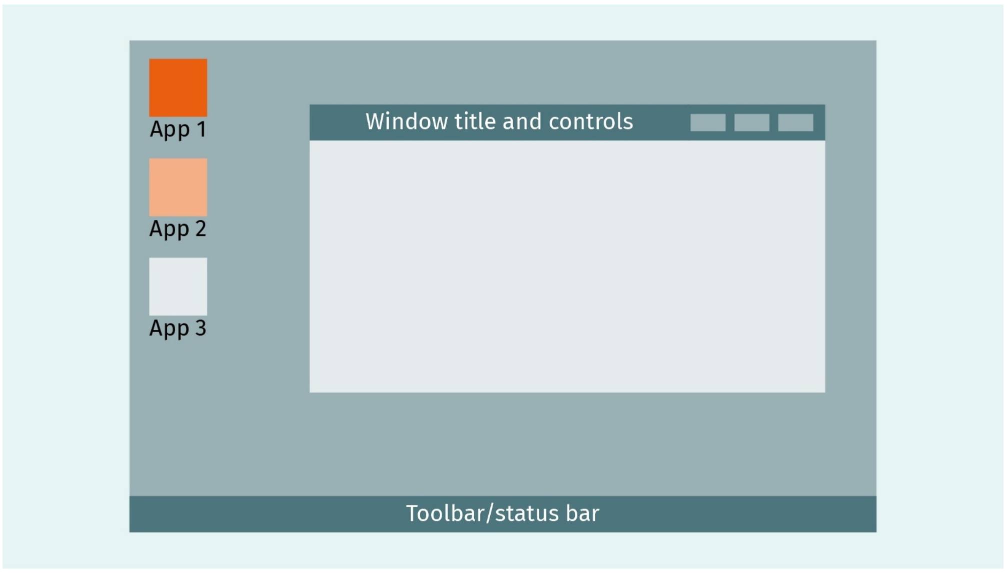
An interface combines all the input and output methods for a user to interact with a computer. The CPU controls the hardware and executes the commands received from the operating system. A user gives a command, which is relayed through the operating system to the hardware, including the CPU. The CPU then directs the hardware to perform the individual tasks at a granular level.

The Computer Interface

Command line interface
a visual computer interface that uses a series of text commands entered by using a keyboard.

The earliest computers used switches and punch cards to communicate with their users. Eventually, video screens were attached – these were large and heavy cathode-ray tube monitors. There were no graphics when these monitors first appeared, 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 (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 provide a very similar experience for the user.

Figure 49: Modern Desktop Interface



Source: Stephen Weese, 2020.

The Mac and Windows operating systems use similar concepts. Applications are run by clicking “icons” on the desktop. Double-clicking, single-clicking, and right-clicking all have similar meanings in both operating systems. Windows and MacOS also launch applications in windows, where an application occupies a rectangular area on the screen that can be resized or temporarily “minimized” (i.e., removed from the screen and represented only as an icon). Today, computer users are expected to have a mouse, a monitor, and a keyboard and know how to use icons, interpret the status bar, and manipulate and use windowed applications. Operating system programmers carefully consider the user interface for each new upgrade and rarely make significant changes to avoid alienating users who have grown accustomed to specific standards.

The operating system accepts input from the user primarily through the keyboard and mouse and displays output primarily via the monitor and speakers (e.g., alarms triggered when the user attempts a forbidden action).

Let us consider a small part of the GUI: the throbber (also known as the loading icon). When you click an application, it is copied into RAM. Without the throbber icon, the user could not know that the computer is active and most likely would repeatedly click the icon, thinking that the input is not being accepted. Thus, the seemingly insignificant throbber provides important feedback to the user that the computer is working on the task.

Although interface devices such as the keyboard and mouse remain ubiquitous, new trends are emerging. Who among us still remembers the spinning wheel on Apple IPods – the first touch surface integrated into a device? This technology was further refined until it climaxed in smartphones and tablets. The concepts behind these devices have strongly influenced other aspects of our lives. For example, toddlers practice swiping or touching gestures with children’s books, or an adult may attempt to select a vending-machine option by touching the screen despite being offered mechanical buttons on the left and right for selection.

Touch and gestures, like pointing or swiping, play a significant role in GUIs, but what about language? Previously, voice input was only used to control special applications, such as medicine or automobiles. Today, voice control is integral to the functionality of smartphones, tablets, smart speakers, and their operating systems. Previously, individual spoken commands had to be recognized by computers; today, the processing and understanding of natural language play a unique role in information technology. In addition, technologies such as facial recognition or iris recognition can now recognize faces and legally identify people. Facial expressions can even be interpreted and analyzed. We are approaching a technological point where all human senses will be used to create a complete human-computer interface and where reality and computer-generated reality intertwine.

Drivers: Talking to Devices

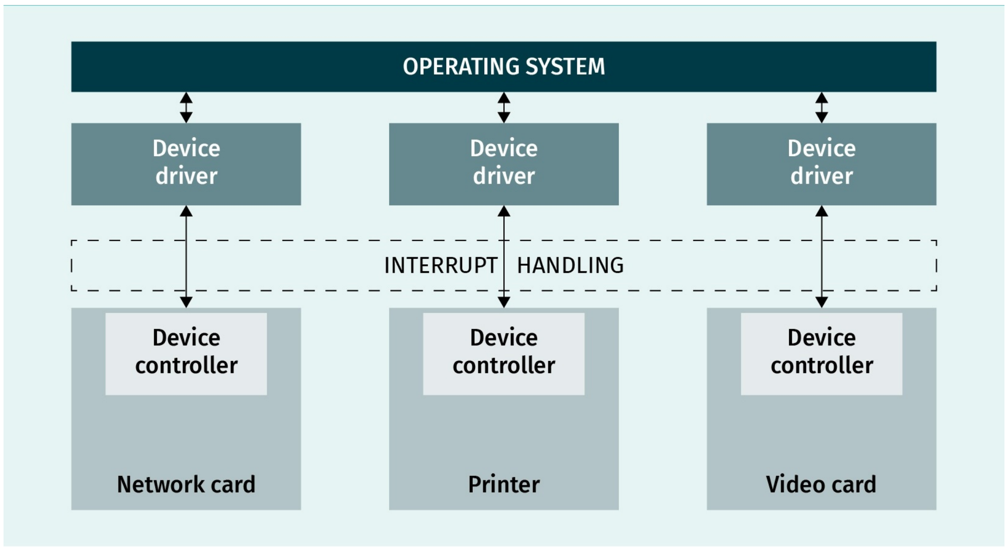
One of the jobs of the operating system is to control the hardware, which includes any device connected to the computer. Standard internal devices such as the hard disk and video card are included, as well as devices such as external webcams and printers. Each device is different and performs a different function, yet they all must communicate with the same operating system. The communication involves a special piece of software called a “driver.”

In the early days of Windows, installing a driver for a device was a complex task. The user often had to specify the **IRQ** and the I/O port address, obliging the user to ensure that no other device was using the same settings. Standard PC design allowed for 16 IRQs for devices to send IRQs to the CPU (an IRQ basically demands that the CPU immediately treats a request). Modern Windows and Mac machines configure devices automatically (known as plug and play), so the user need not deal with IRQs and port addresses.

IRQ
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 the device and the operating system. As you can imagine, a network card, a printer, and a video card all use 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 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 of the device. The device controller can also signal the CPU via an IRQ. A device driver has to be designed specifically to communicate with both the operating system and the device’s controller. Sometimes, **bugs** are discovered as new driver updates are released, so it is recommended 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 communicates with the device controller. The latter then communicates with the device.

Bugs
software flaws that range from minor inconveniences to serious problems

5.5 High-Performance Computing

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

Server
This is a dedicated computer designed to handle user requests over a network. It is often kept in a dedicated facility.

Parallel Processing

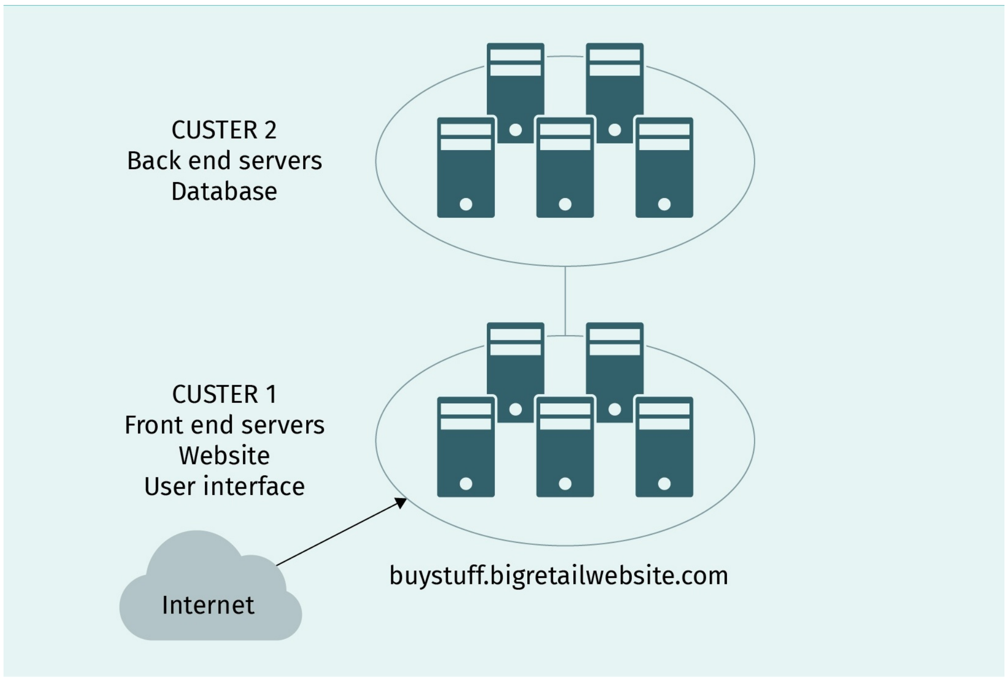
To make a supercomputer, you might imagine supersizing a regular computer. However, the technologies involved scale differently. Consider an automobile – you can make bigger and bigger cars with bigger engines, but the larger engines will weigh more, eventually cancelling any gains in horsepower and speed. A single desktop computer with an enormous RAM would spend more time managing all of that memory and eventually become useless. Instead of building a massive computer with one CPU, we use the concept of parallel processing, which involves breaking a task down into subtasks and assigning separate CPUs to complete the subtasks in parallel (i.e., at the same time). Several CPUs are linked together in parallel to accomplish this.

Server Clusters

Imagine a popular website such as Wikipedia or Amazon. A single computer could never handle all the processing involved with those websites. One method to achieve this is by using a server cluster, which is a group of identical servers (think of them as “clones” that can respond to web requests). These clones are spread worldwide and can respond to multiple requests. For instance, a user in South America could connect to a nearby clone. When a server is updated, a “master” server then copies or clones all of the servers in the cluster by using**version control** software that ensures that all servers receive the updates.

Version control
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 use multiple server clusters, splitting the load between the front end (the website users see) and the back end (the database containing all the images, descriptions, and prices). When the user enters the universal resource locator for a site, their browser connects to one of the clones in the cluster (usually selected by a **load balancing** system). For example, when a user searches for a "large black umbrella," the front-end server connects to one of the back-end cluster servers and requests a list of umbrella items that match the query. A cluster not only evenly splits up tasks (load balancing) but also provides fault tolerance: if one server stops functioning, the others remain functional and compensate.

Load balancing
a management system that distributes the processing load as evenly as possible among servers

Supercomputers

When you need more computing power than is available from a cluster, then a supercomputer is the answer. These computers are rare and cost hundreds of millions of dollars. They are similar to clusters in that they use parallel processing, where tasks are split up between processors. One such supercomputer called Fugaku is in Kobe, Japan.

Computing power for both desktops and supercomputers can be measured in "FLOPS," which stands for "floating point operations per second." A FLOP is an operation on numbers with a floating (movable) decimal point. Modern desktop computers typically perform at a respectable 100 gigaFLOPS (a gigaFLOPS is one billion FLOPS).

Fugaku is rated at approximately 442 petaFLOPS (a petaFLOPS is 10^{15} FLOPS). This supercomputer is 4,420,000 times more powerful than an average desktop. This is accomplished by assembling 158,976 high-end, water-cooled CPUs in the same room and connecting them to work in parallel. This computer has 32,000 TB of RAM (Sato, 2021).

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 types of problems. Scientists can request time on supercomputers to do research, allowing them to harness the massive power of Earth's most amazing computers.

SUMMARY
Most computers in use today are designed according to the Von Neumann architecture. This simple design involves the interaction of a CPU with data or instructions stored in memory. Input and output are delivered to the CPU via registers, which have a fixed number of bits. These registers are loaded with data or instructions read from memory.

One way to shorten memory access time is to put some memory directly on the CPU chip; this is called a cache. Different levels of cache exist: some processors have three levels labelled L3, L2, and L1. Of the three, L1 has the smallest memory capacity but the fastest access time. This

strategy minimizes memory bottlenecks because the CPU makes fewer requests of the system RAM; a small amount of the most relevant data sits in the cache.

The Von Neumann architecture also includes buses, which are connections that transfer data. Buses carry input and output related to memory addressing, command instructions, and pure informational data. The results of processing, such as an arithmetic operation, are transferred along the data bus.

One of the primary functions of an operating system is to provide a human-computer interface. Personal computers in the twenty-first century have relatively standard graphical user interfaces (GUIs) that enable users to use a mouse, point and click, open multiple applications, and resize the applications. The operating system also provides a way to control and communicate with hardware devices added to the computer, such as a printer, a video card, or an external web camera. The device driver is software that communicates between the operating system and the device.

When a PC or single mainframe computer does not have enough computing power for a job, supercomputers can provide a solution. Created by combining thousands of CPUs in parallel, supercomputers can perform calculations in the petaFLOPS range.

UNIT 6

NETWORKS AND THE INTERNET

STUDY GOALS

On completion of this unit, you will be able to …

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

6. NETWORKS AND THE INTERNET

Introduction

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

The military realized that a communications network could be partially destroyed in a conflict. Early network designs were not “fault-tolerant” – a disruption in part of the network caused the entire network to stop working (i.e., crash). Eventually, the ARPA started using TCP/IP (TCP stands for transmission control protocol, IP stands for internet protocol) for its ARPANET network. This technology was fault-tolerant: TCP/IP could direct data around damaged parts of a network to reach the undamaged parts. This network was eventually made available to American universities for research. Large universities such as Stanford and the University of California became part of the network. In the late 1980s, commercial entities saw the value of a large interconnected network (Internet), and the general public gained access to the network through a university account or Internet service providers (ISPs) such as America Online.

The World Wide Web is often confused with the Internet. However, the World Wide Web is a set of services accessed through the Internet using software called a web browser (e.g., Firefox, Chrome, Safari). Other data services on the Internet 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), and data sent over networks are no exception. Electrical signals have been created to symbolize either a one or a zero. These numbers have little meaning by themselves, but given the proper context, the receiving computer can interpret the data.

Before the advent of wireless networks, all network data was sent over wires. Originally, the layout of a network (i.e., the physical topology of the network) could be designed in several ways. Over the years, the layouts have been simplified to two dominant varieties: local area networks (LANs) and wide area networks (WANs). A LAN connects all the computers within a specific physical location, such as an office building or home. A WAN consists of LANs connected over significant distances (i.e., much larger distances than a LAN). An example of a WAN would be an office building in Sydney connected to an office from the same company in Singapore. The Internet is the largest WAN, connecting millions of LANs together through wired and wireless connections.

Network Devices

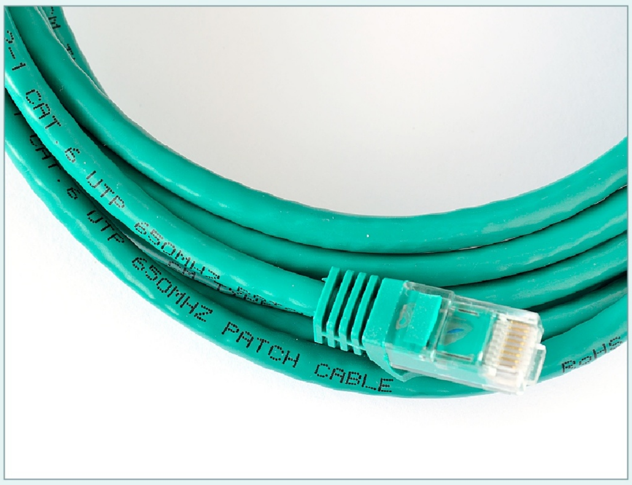
Using a wired network naturally requires cabling. Standard wiring for a LAN (inside a home or an office) consists of **UTP** cables of various categories. Data transfer speed for networks is usually measured in bits per second (bps.) Occasionally, you may see bytes per second (Bps); 1 Bps is eight times more data throughput than 1 bps. The following table lists some of the categories of UTP cables:

UTP
This stands for ““unshielded twisted pair,”” which means that the individual wires are twisted together in pairs without extra insulation.

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

UTP cables of different categories appear physically identical; only the labeling printed on the cable identifies its category.

Figure 53: Category 6 UTP Ethernet Cable



Source: Raysonho, 2015.

The connector shown on a UTP cable in the figure above is called an RJ-45 (for “registered jack” number 45) connector. Although it is similar to a standard U.S. telephone connector, it is actually larger and incompatible with standard telephone connectors, which are RJ-11. Standard telephone cables are Category 1 and, although they can be used for connection to the internet, their data transfer speed is very slow. Most **ethernet** cables are at least category 5e (i.e., category 5 “enhanced”).

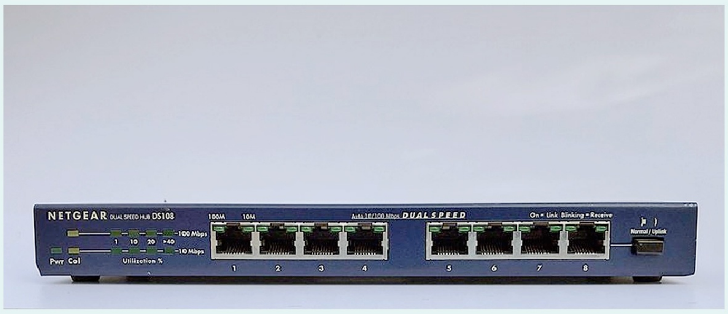
ethernet
This is a networking technology that connects devices in LANs by cables, offering greater speed and security than wireless fidelity (Wi-Fi) connections.

Uplink
This is a connection from a smaller network to a larger network. Usually, such links send information from a LAN toward the internet.

Hub

The simplest network connection device is a hub. A hub takes a signal from one cable and broadcasts it out over several cables. A hub that broadcasts the signal from its input cable over four output cables is called a “four-port” hub. A hub does no signal processing – all input is broadcast over all of the ports. Note that communication through hubs can go both ways: the ports can also send signals out through an **uplink**connection.

Figure 54: Eight-Port Hub



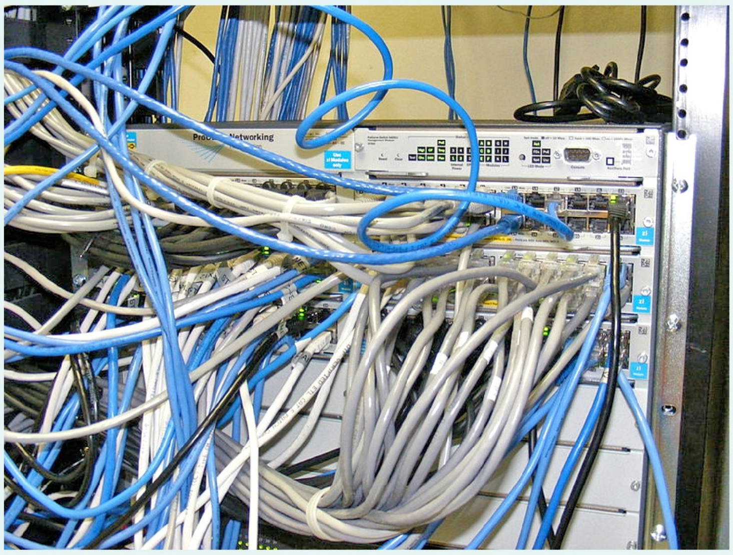
Source: Heimnetzwerke.net, 2018.

Switch

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

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

Figure 55: Switch with Cabling Connected



Source: J.smith, 2007.

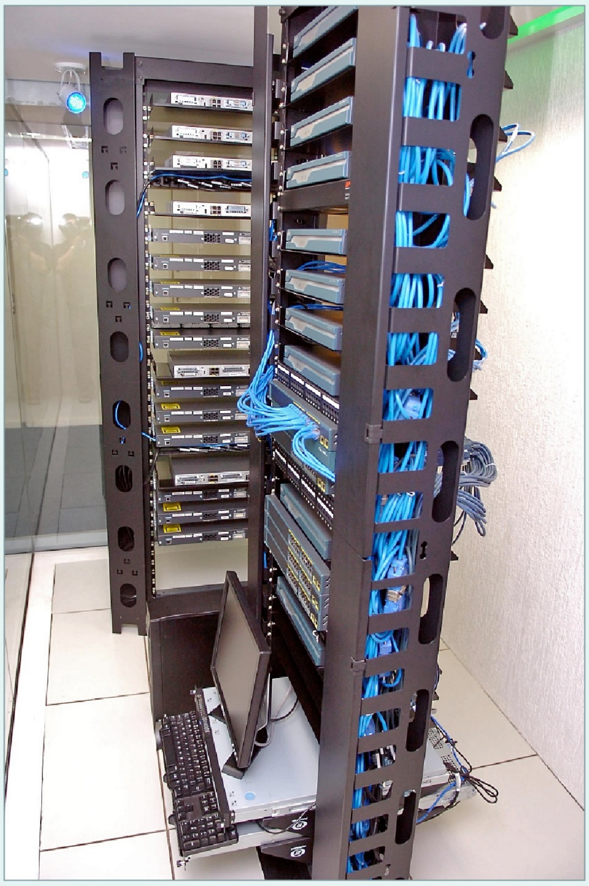
Switches are often grouped in a central location in an office building to send the data frames to different rooms and floors.

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 unwanted data from entering a LAN or WAN; 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

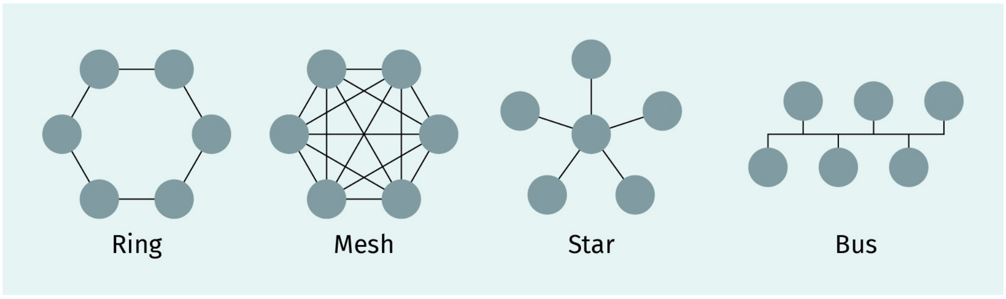


Source: de Lima, 2009.

Wired Network Topologies

A network topology is the physical layout of the connected devices. The figure below shows several examples of possible topologies, and the following table briefly describes them.

Figure 57: Topologies for Wired Networks



Source: Maksim, 2005.

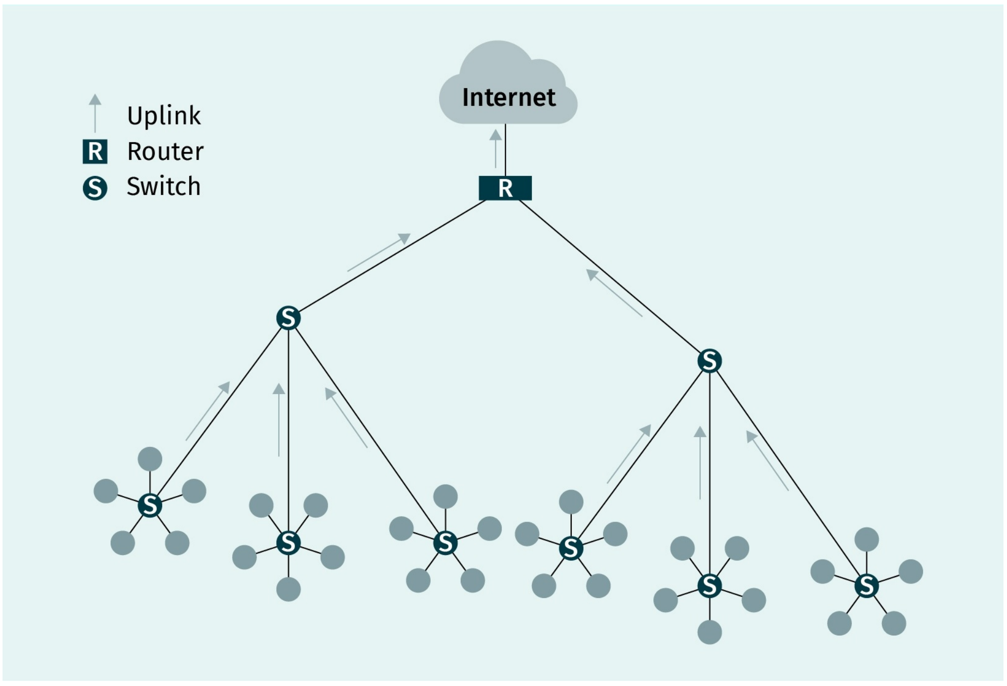
Several Possible Topologies

Topology	Explanation
Ring	A ring network consists of computers connected in a ring, with the signal traveling around the ring. With this topology, each computer must analyze all data it receives to determine if the data should be passed along. This design is not fault-tolerant because it stops working if the ring is broken, which explains why it has fallen out of use.
Mesh	Mesh networks connect every node (computer or device) to every other node. However, each device connected requires an increasing number of cables to connect with the other devices, which is extremely inefficient. The mesh topology has the advantage of being fault-tolerant because there are multiple paths to each node. The mesh topology is of interest for wireless networks, which do not use cabling. Wireless mesh networks thus benefit from fault tolerance, although at the cost of decreased security.
Star	Star networks are the most common type of network in use today (in the 2020s). Star networks also include multiple stars connected together in a bigger star network. The central node of a star network sends the data to each outer node. Star networks are somewhat fault tolerant – if the central hub or switch fails, the network ceases to function.
Bus	Bus networks require all computers to be connected to a bus. Such networks are not fault tolerant because, if the bus breaks anywhere, the entire network fails. This type of network is not commonly used, except in some network **backbones**.
Hybrid	A hybrid network is any network that connects networks of multiple topologies. An example would be multiple star networks connected via a bus network backbone.

Backbone
This is the core of a network that connects the main parts together. It can also connect multiple networks.

Although the early computer networks used several different topologies, most computer networks now use the star topology.

Figure 58: Example Office Star Network



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

In the example shown in the figure above, multiple star networks are connected via a larger star network. Switches serve as the central node for each small star network; they connect five computers (or other devices) to the uplink, which connects to a second-level switch. Each second-level switch has four connections, forming its own star. The router connects to the two second-level switches and filters data traveling to and from the internet. The router also forms the center of a star, with two connections to the second-level switches and one connection to the internet. This is a common network design for office buildings. For example, each second-level switch could handle the data from one floor of the building, and each lower-level switch could handle the data from five terminals grouped into separate rooms on the given floor.

Wired Meets Wireless

Wireless networks (Wi-Fi) became available to the public in the 1990s. These networks are used to connect devices to larger networks, which, inevitably, are networks connected by wires. Wireless devices emit electromagnetic signals whose range is in the tens of meters, so reaching devices at greater distances means signals must be translated and sent over a wired network. The table below lists the advantages and disadvantages of wireless networks and wired networks.

Table 11: Wired versus Wireless

Wired network	Wireless network
Device position is physically limited by cabling	Devices can move freely, provided they remain within range
Easier to troubleshoot	May be difficult to troubleshoot
Fastest data speeds	Medium data speeds
Very secure	Less secure; easy to intercept signals
Must purchase cabling	Only wireless access point need be purchased

Source: Stephen Weese, 2020.

Moving a device is much easier if it is not tethered by a cable. However, depending on the construction and layout of a building, wireless signals may be weak, and finding a strong signal may be difficult and nonintuitive. Throughout the history of wired and wireless networks, the former have always been faster, although wireless speeds are approaching those of wired networks. A wired network is more secure than a wireless network because it can only be accessed by plugging in a cable, whereas anyone who captures the signal of a wireless network can potentially connect to it. Finally, an advantage of wireless networks is that they tend to be easier to set up because there is no need to connect the network devices with cables.

Wireless Technology

The Institute of Electrical and Electronics Engineers (IEEE) is a professional association of engineers that provides technical standards for many aspects of electronics. In 1997, they introduced the IEEE 802.11 standard (or protocol) for wireless networks. This standard has since been updated, and the new standards are identified by additional letters added after the name of the original standard. As of 2024, over a dozen such standards are in use, several of which are listed in the table below.

Table 12: 802.11 Modern Wireless Standards

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

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

We focus now on the most prominent IEEE 802.11 protocols. The 802.11g protocol is the oldest widely used protocol and is ten times slower (in theory) than the next faster protocol. However, most networks do not achieve their maximum possible speeds. In addition, the 2.4 GHz frequency at which data are transmitted allows better transmission through physical barriers than the 5 GHz channels, even though the 5 GHz signals can carry more data. The 802.11n protocol can switch between both frequencies.

Further development of these standards is hindered because network cards and routers must be backward compatible to support all previous frequency bands. This increases the price of new devices until the backward compatibility is discontinued.

Note 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 this standard. If not, you will have to buy new network cards to take advantage of the higher speeds.

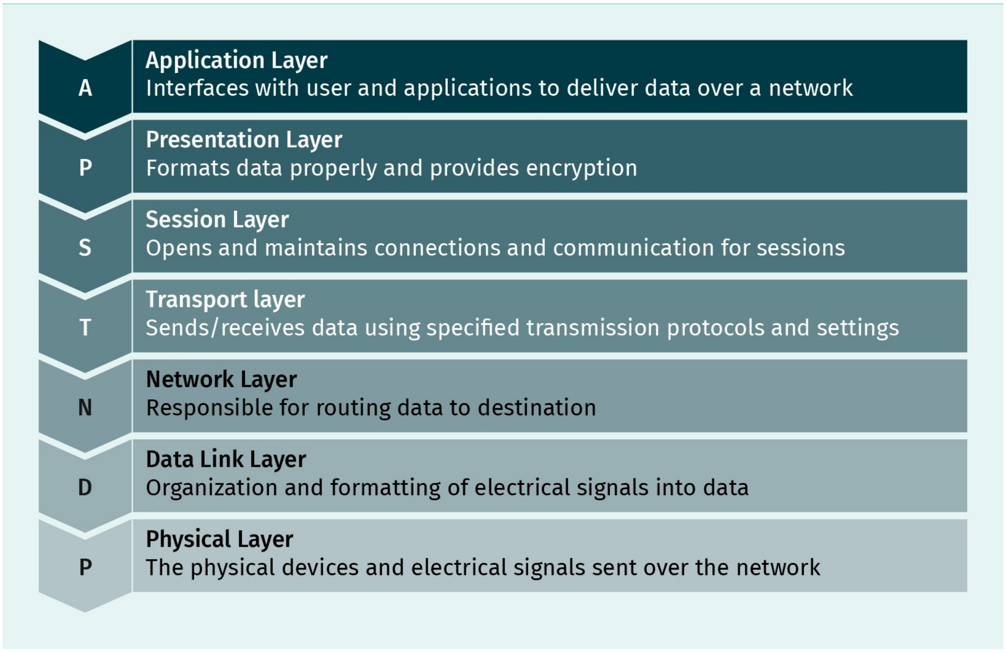
Figure 59: Wireless Access Point

Source: Pjpearce, 2011.

6.2 The OSI Model and TCP/IP

The Open Systems Interconnection (OSI) standard was created to explain the process of network communication. It is used to develop specific networking technologies and map out how they work. The main result of this effort is the OSI Model, which divides network communication into the seven layers shown in the figure below.

Figure 60: The OSI Model


A
Application Layer
Interfaces with user and applications to deliver data over a network
P
Presentation Layer
Formats data properly and provides encryption
S
Session Layer
Opens and maintains connections and communication for sessions
T
Transport layer
Sends/receives data using specified transmission protocols and settings
N
Network Layer
Responsible for routing data to destination
D
Data Link Layer
Organization and formatting of electrical signals into data
P
Physical Layer
The physical devices and electrical signals sent over the network

Source: Stephen Weese, 2020.

The application layer communicates with applications that want to use the network (usually due to a direct user request). This layer passes the data to the presentation layer, where they are formatted into the correct encoding (e.g., Unicode). This is also where the data are encrypted. Next, the session layer establishes a session with the other computer. Specifically, 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 data transmission, monitoring communications from end to end. The network layer is more specific – it handles the physical routing of the packet at each step along the way of the network. This is where IP addressing sees the most use. The data link layer provides device specifications that determine how electronic signals are sent over the network. It also provides standards for devices; this is where the hardware (MAC) address is used. Finally, the physical layer consists of the physical devices and wiring of the network, as well as the signal itself.

Port
In networking, this is a number that identifies a specific data stream or data type.

Application Layer

The application layer is the closest to the user. It provides the interface between applications and the operating system and allows applications to request access to network services. Once such a request is accepted and delivered to the operating system, it is passed to the presentation layer. Applications that request network access include internet applications such as browsers and email clients, as well as applications that require network access, such as Microsoft Word. For example, to have the network printer print a document, Microsoft Word will request network access via the application layer.

Presentation Layer

This layer prepares the data for transfer by translating the data into a format that the receiving application can understand. One of the services the presentation layer provides is ensuring that the proper encoding is used to translate the data. For instance, if an email is sent in Unicode, the presentation layer translates the symbols in the email. Encryption also occurs at this level, which involves passing plaintext or other unencrypted messages through a mathematical algorithm to encrypt them. The sending network device shares the encryption key before transferring the data to the receiving network device (this key allows the receiving device to decode the message, thereby obtaining the original message).

Session Layer

A session for network communication is analogous to organizing a meeting. A time and place must be decided, and various other requirements must be satisfied (e.g., bring your notebook). The session layer establishes, manages, and terminates communications between network devices. It determines when communication starts, the wait time for responses (i.e., acknowledgments that the message is received), and how the session will be terminated. Once communication begins, lower layers handle the details.

Transport Layer

The transport layer provides a bridge between the session layer and the network layer by ensuring reliable data transfer between two applications on different hosts. Within a session, the transport layer groups the data into chunks, or segments, to be sent out separately, ensures that the data segments flow at the proper rate, checks for errors, and reassembles data at the receiving end.

Network Layer

The network layer may be understood by considering the steps involved in mailing a package to a friend. The package has been prepared (i.e., put into a box) and given an address (e.g., an IP address) by the previous layers. The network layer then transports the package to the given address by using the network routers. At each step along the way, the address is checked, and the correct path is chosen to send the data toward its destination.

Data Link Layer

Once a data segment arrives at its destination network, it must find the specific device to which it is addressed. The data link layer works at a finer level than the network layer. It uses hardware addresses to identify network devices. In addition, in this layer, Ethernet specifications dictate how communication is implemented, which includes checking for local network errors, including collisions. Collisions occur when two network devices try to communicate simultaneously, so their signals interfere with each other in the physical media. When a collision is detected, both devices are instructed to wait a random amount of time before sending again.

Physical Layer

The physical layer includes network devices such as hubs, switches, and routers as well as the cabling that connects them. The binary signals are also included in this layer.

Thus, the OSI model allows data to travel back and forth on networks between applications. Note that the OSI model is a general model that describes how data are transmitted over networks. Specific protocols are not required to use all the concepts in the OSI model but may select the most pertinent concepts depending on the particular needs of the protocol.

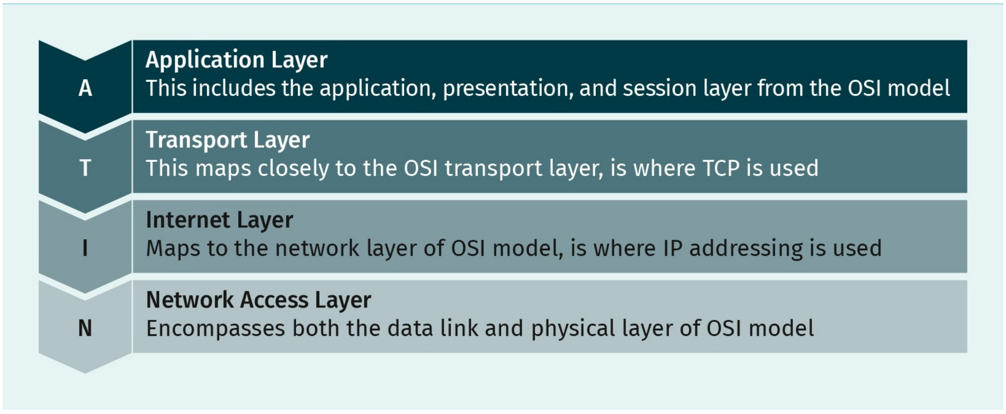
Ethernet

Ethernet is a set of standards (defined by IEEE 802.3) used for LANs and WANs. The word “Ethernet” most commonly refers to parts of a local wired network, such as Ethernet cabling (categories 5, 5e, 6, 7, and 8) and Ethernet ports, where the cables are connected. The Ethernet standards describe the physical layer (i.e., devices and signals) and the data link layer (i.e., the organization of signals).

TCP/IP and the OSI model

TCP/IP stands for transmission control protocol/internet protocol. TCP works at the transport layer of the OSI model, whereas IP works at the network layer. As shown in the figure below, TCP/IP is a four-layer model.

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, so devices must follow this protocol to access the internet. Since it also works for communication over LANs, most computers use TCP/IP as their primary and only network protocol. The TCP/IP protocol consists of the four subprotocols shown in the figure above. Network devices must have three settings configured to use TCP/IP. These settings are 32-bit binary numbers (or 128 bit with IP version 6).

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

IP address

An IP address is a unique address that identifies a network device. Each device on a given network has a unique IP address, which is divided into two identifiers (IDs): a network ID (which identifies the network segment to which the device is connected) and a host ID (which is the individual device number). The network ID is like the street name for a residential address, and the host ID is like the house or apartment number.

Subnet mask

A subnet mask is a 32-bit binary number that serves to filter IP addresses, identifying which part of the address is the network ID and which is the host ID. Routers use subnet masks to determine to which network a device belongs (using the network ID) and to route data packets to the correct device within the given network (using the host ID).

A common example of a subnet mask is 255.255.255.0. The bytes “255” are represented by the binary number 1111 1111, and the byte “0” is represented by the binary number 0000 0000. The ones (zeros) indicate which bits belong to the network (host) ID. IP addresses can be split in different places depending on the particular needs of the network.

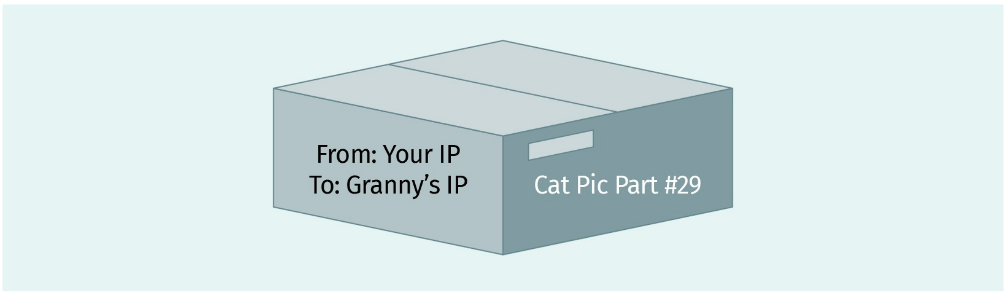
Default gateway

A gateway is a router or device that allows a computer to access devices outside its local network. A computer does not need a gateway to communicate with a device in its own network. However, it must use a gateway to communicate beyond its local network.

Packet delivery

To understand packet delivery, consider again our analogy of sending a package in the mail, but this time with more detail. Imagine you are sending a picture of your cat over Skype to your grandmother in Peru. The data are formatted and presented for transport over the internet, following which a session is established. The data constituting the picture of your cat is then divided into individual packets, with each data packet being like a separate package in our mail analogy. Each package (i.e., each data packet) is marked with the addresses of both the sender and the recipient.

Figure 62: TCP/IP Data Delivery



Source: Stephen Weese, 2020.

The internet uses your grandmother's IP address to deliver to her all the data packets constituting your cat picture. Each packet is communicated to your router (the default gateway), which then sends the packets out via your internet service provider (ISP) to the internet. The internet is designed like an extensive road system, with each router along the way deciding which path to take to get to grandmother‘s local network. Eventually, the packets arrive at your grandmother's default gateway and are communicated to her computer. The computer waits until all the packets have been received and then reassembles them back into the picture of a cat.

Fault tolerance

TCP/IP is fault tolerant. During a session, it ensures that all packets are received and verifies the destination path. If a router along the path stops functioning, TCP/IP automatically searches for (and most often finds) a different path to deliver the data. This explains why the internet has never completely failed – although parts of it may fail, the rest can continue to function without them.

6.3 Internet Structure and Services

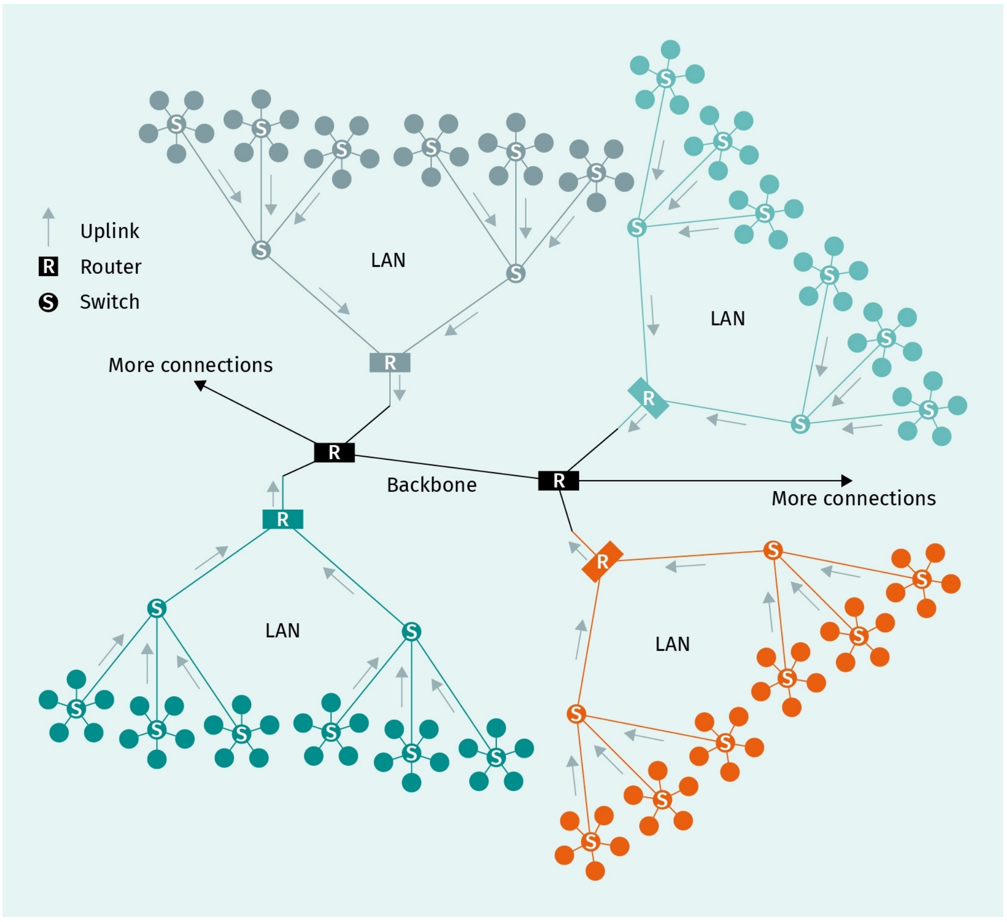
The internet is by far the largest computer network ever created. At its simplest level, it is a collection of LANs all joined together by routers and cabling all over the world. Using TCP/IP addressing, any computer on the internet can communicate with any other thanks to the built-in data-routing system.

The basic devices of the internet have already been discussed (router, hub, switch, cabling). However, once we leave the domain of the LAN, connections between devices become more complicated.

Private individuals or businesses who want a dedicated internet connection must pay an ISP for a connection to the internet. If we imagine the internet as built of buildings and roads, the cabling is the roads, the routers are the intersections, and the computers and other devices are the buildings. A neighborhood connected by roads is analogous to a

LAN; to reach the road system that connects all such neighborhoods (i.e., the internet), one must pay a company to build a connecting road. The ISP makes this connection, which allows users to connect to all other users of this vast network.

Figure 63: LANs Connected to the Internet



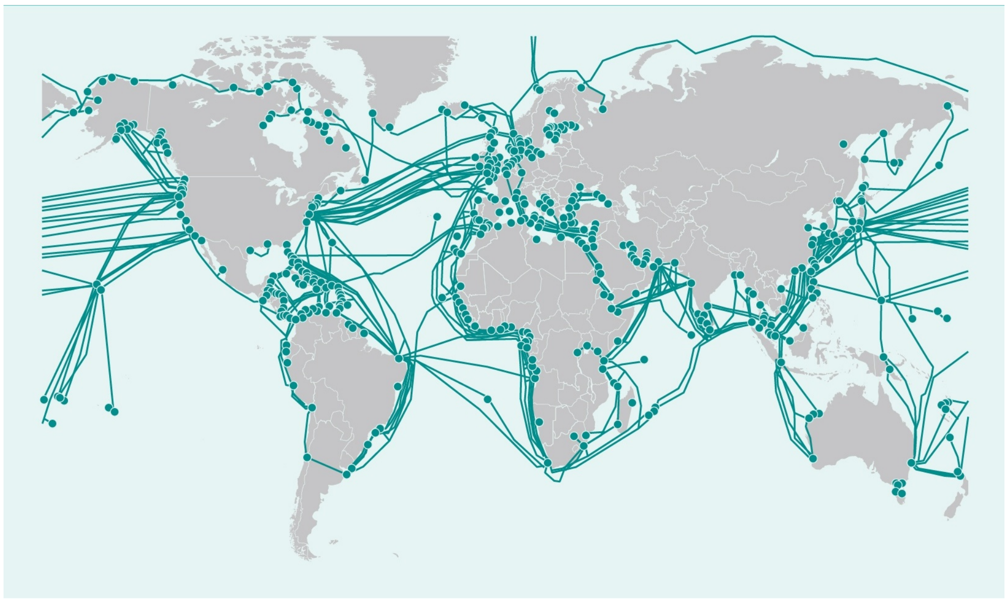
Source: Stephen Weese, 2020 based on Maksim, 2005.

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

Under the Sea

Using special equipment, thousands of kilometers of cable have been laid under the oceans to connect most of the Earth's continents. Over 99% of intercontinental data is transferred by these cables, which include copper and fiber optic (Main, 2015). Fiber optic cabling transmits light pulses that communicate binary messages over long distances. Copper cabling transmits electrical signals that are slower and degrade over large distances, so fiber optic cabling is much faster and more reliable.

Figure 64: Transoceanic Communication Cables



Source: Mahlknecht & OpenStreetMap Contributors, 2015.

Above the Sea

In addition to communication via undersea cables, communication occurs above sea level. The decisive factors for communication technologies are speed, worldwide availability, and network coverage. Communication satellites in different orbits around the Earth are essential to worldwide availability and provide maximum network coverage over sea and land. However, one shortcoming of communication satellites is latency, which is the time it takes for a signal to travel from the Earth’s surface to the satellite and back. Even if these signals travel at the speed of light, latency is noticeable when using the internet (consider the delay when opening a website). To reduce latency, satellites orbit at the lowest possible altitude, which means they orbit close to the atmosphere. Although the air is thin at these altitudes, it slows the satellites, requiring them to be equipped with additional energy sources to maintain their orbit and not descend and burn up in Earth's atmosphere.

The second issue is the numerous satellites required to cover the Earth. The U.S. company SpaceX is by far the largest satellite operator in the world, with 2453 satellites in orbit as of mid-2022 (SpaceX, 2022). Starlink is a satellite network operated by SpaceX that is intended to provide worldwide internet access in the future with short latency and internet service in previously uncovered areas (Starlink, 2022).

Finally, the Global System for Mobile Communication focuses on communication on the land masses and off-shore areas. Another organization in this category is the Next Generation Mobile Networks project, which includes mobile communication standards such as Long Term Evolution.

How the Internet Works

To this point, we have explored how internet devices are connected and examined how data are divided into packets that use IP addressing for delivery. Here, we cover other aspects of the internet that are crucial to its operation.

Hardware

Internet hardware 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 fault-tolerant TCP/IP protocol to communicate data between every node (device) on the internet. 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 serve as either “clients” or “servers.” A client is a device that uses services provided online, whereas a server is a dedicated machine that provides data to clients. Servers are typically powerful computers with a high-speed internet connection.

The World Wide Web

The World Wide Web, often called the “Web,” is a special set of services delivered to an internet browser. Basically, anything you can access using an internet 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 the Hypertext Markup Language. Eventually, the Web evolved to include video services and websites with interactive JavaScript and connections to vast databases.

Other web services

Besides ““surfing” the Web (i.e., navigating over the Web by following links in websites),” internet users can access multiple data streams and services. Examples include music streaming services, online games, and direct video chat applications. Internet applications that do not require a browser are not part of the Web.

Dynamic Host Configuration Protocol

An IP address is required to access Web services. However, users cannot simply invent any address they desire but must acquire a proper address that identifies their network. Fortunately, this is part of the package provided by ISPs. They use the dynamics host configuration protocol (DHCP) to automatically provide their clients with not only an IP address but also a subnet mask, a default gateway, and addresses for domain name system (DNS) servers. By default, computers and Wi-Fi devices search their local network for a DHCP server and request configuration. Once configured, they can begin to communicate.

Domain Name System

The DNS is an essential part of the internet. Every device on the internet requires a 32-bit IP address to transmit and receive data. If users had to memorize and type in 32 zeroes and ones to access websites, the internet would be useless. Fortunately, the DNS translates alphanumeric names to IP addresses. When you type in the name of a website or click a link to it, you access a uniform resource locator (URL), which includes the full DNS name of the computer that hosts the website.

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 contains several pieces of information, as shown in the figure above. The first part indicates the protocol used for communication. The hypertext transfer protocol (HTTP) and HTTP secure (HTTPS, a secure extension of HTTP) are both part of the TCP/IP protocol suite – they exist specifically to transfer data for displaying web pages in browsers. The secure version uses encryption to protect private data. Websites that use HTTPS are often indicated by a small padlock icon in the browser address bar.

The next part of the URL gives the name of the computer that you desire to communicate with. This is often “www,” although this is simply a convention – any alphanumeric string may be used for the computer name.

The URL next gives the domain name, for instance, “iu-fernstudium.de” or sysu.edu.cn. The domain name identifies the network to which the computer belongs. Combining the computer name with the domain name gives the hostname (e.g., “www.iu-fernstudium.de” or www.sysu.edu.cn). The DNS service uses the hostname to look up the IP address.

The DNS is like an extensive internet phone directory – you know the computer’s name, but you want to know its “phone number” (i.e., its IP address). DNS servers are placed all around the world to service IP address lookup requests. When you want to view a website, the DNS is called upon to look up the host; if the request takes more than an instant, many web browsers will display the message ““looking up host”” in the bottom status window. Once the DNS provides the IP address, your computer stores it for a specified time in the DNS cache to avoid repeatedly requesting the same address for websites you visit frequently.

Once you have connected to the desired computer, it needs to know exactly which file you wish to open. Thus, the URL gives the folder containing the file of interest after the computer hostname and then the filename itself. The file is retrieved and sent over the network to be displayed in your web browser. If no file or folder location is given in the URL, the web server provides the default file it was configured to send.

The Growing Internet

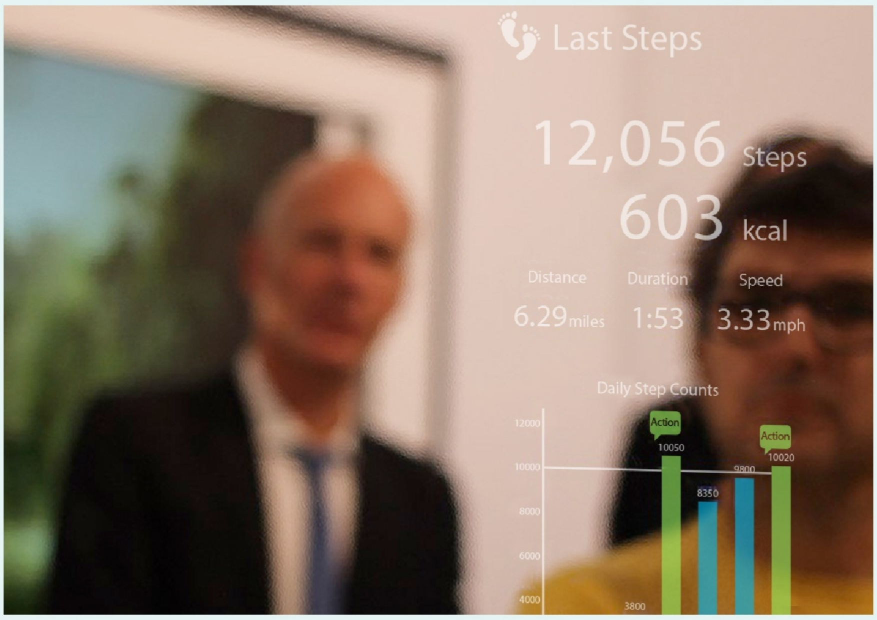
The internet is providing new services every day as they are invented. Companies now offer “cloud services” where users can store all their documents and photos online, where they can be accessed from and synchronized across multiple devices. More smart devices are constantly being added to become part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

6.4 The Internet of Things

Initially, computers were the only devices connected to the internet. As the internet grew, other types of devices (called “nonstandard devices”) were connected. These devices may have sensors that collect data and the hardware and software required to process the data. Such devices are connected not only to the internet but also to isolated networks (e.g., within buildings or personal homes). These networks of devices, despite not all being connected to the internet, form the Internet of Things (IoT).

Already, over a billion such devices are connected to the internet. They include many “smart” devices that can be controlled remotely (e.g., smart light bulbs). Regular and self-driving vehicles are also connected to the internet. Many smart devices provide feedback on their performance. A smart refrigerator can provide the user with recipes as a function of its contents, signal if it is low on eggs, report its power consumption, and give an optimal temperature setting. Wearable technology (e.g., smartwatches) and numerous medical monitoring devices are also part of the IoT. Security cameras worldwide are connected to the internet so users can access them from anywhere. Televisions can access the internet, along with wireless speakers. People can work out in front of smart mirrors and watch themselves exercise while instructions and calorie counts are displayed on the mirror. Such mirrors can also display local weather reports or stock reports.

Figure 66: Smart Mirror



Source: Pesce, 2014.

Uses for IoT Devices

Many different reasons exist for connecting a smart device to the internet. One reason is that these nonstandard devices can be programmed to function in different ways, whereas a standard device would have to be continually adjusted manually.

A good example of an IoT device is a home thermostat that can be programmed to adjust the temperature settings as a function of the day of the week and the time. A maximum and minimum temperature can be set to maintain the home temperature within these limits, with the system switching automatically between heating and cooling. In addition, the thermostat can be controlled by a smartphone application, allowing the user to change the temperature settings remotely. Such thermostats can also track energy consumption, post monthly reports to a website, and learn the user preferences to automatically create a weekly schedule.

Figure 67: Nest Thermostat

Source: Raysonho, 2014.

In addition to storing data and creating web reports, IoT devices can retrieve data from the internet. For example, a smart refrigerator can track its contents and download recipes that are made of the available food. Many smart refrigerators detect the expiration dates on food to recommend recipes that use items approaching their expiration date. Devices can even communicate with each other over the home network. For example, if the user selects a recipe suggested by the refrigerator, the latter would then send a command to the oven to preheat for cooking.

Smart devices can also customize the home experience, including setting mood lighting and playing music. Not only that, if the home network connection to the internet is sufficiently fast to handle large data streams, artificial intelligence can allow home assistant devices to converse with the user and answer questions, using the data collected to control devices such as televisions and speakers. Thus, the user no longer needs to type data into a computer or even use a computer in any way; they simply talk to their home assistant and ask for light, music, information from the web, or a specific television program.

Security and IoT

Connecting billions of devices to the internet presents security issues. One such issue is similar to a previous problem faced when Wi-Fi became popular. Internet users who purchased a Wi-Fi access point often neglected to change the default name or password, which, at the time, were the same for each manufacturer. This situation gave easy access to millions of home networks, constituting free access to the internet to anyone who cared to exploit the situation. Eventually, ISPs started preprogramming Wi-Fi devices with secure passwords, which greatly reduced this problem.

With the advent of the IoT, people who buy wireless security cameras are doing the same thing – leaving the default password in place. The result is millions of unsecured security cameras operating all over the world, so many that a new hobbyist community has emerged consisting of users who watch random security cameras for entertainment.

Although the interception of IoT-device data is a security risk, a more concerning risk is that malicious users could take over these devices. Worst-case scenarios include a self-driving vehicle commandeered to cause an accident or a device such as an oven usurped to cause a fire or other property damage. Even simple, seemingly harmless devices can be used for nefarious purposes. Since IoT devices have IP addresses, they can send requests to other devices online. Millions of such devices could thus be compromised to make a botnet – an army of internet robots controlled by a single user or program. Botnets are most often used in 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 as to be essentially unusable.

Other concerns include companies using smart devices to gather data on users. These devices could learn your shopping, eating, sleeping, and displacement habits. Parents are concerned that smart toys might spy on their children or, worse, try to influence them. As of the mid-2020s, few standards have been developed to prevent IoT devices from being used for such purposes.

IoT Standards

As of 2024, companies such as Microsoft and organizations such as the IEEE are working on creating standards to improve the operation and development of IoT devices, including addressing security issues. One such standard is the 6LoWPAN protocol, which is designed for low-power IoT devices that may be very small and run on single batteries (normal IP communication quickly depletes batteries, so 6LoWPAN implements compression along with other power-saving strategies) (Leverage, 2020).

IoT and the Future

New devices are added to the IoT every day, and new types of devices are continually being created, which means more security issues requiring new solutions. In the near future, people will likely be connected to the internet through some type of brain interface (several organizations are working on this technology). The U.S. Army is creating brain chips to help with post-traumatic stress disorder (Tucker, 2014) and, to help diagnose medical conditions, the company Neuralink is developing a brain interface that will be able to interface with networks (Alexiou, 2020). Other expected innovations include the integration of artificial intelligence to detect fraudulent activity by IoT devices (e.g., cash machines), the use of the IoT for remote learning to reduce background noise and visual distractions, and edge computing to reduce the carbon footprint of IoT devices. As more devices are added to the IoT, computer programmers will find themselves writing applications not for computers or smartphones but increasingly for “smart” IoT devices.

SUMMARY

Local area networks (LANs) use standard technologies such as Ethernet and TCP/IP, as well as standard devices such as routers, switches, and hubs. Most cabling used in LANs is now category 6 or 7 Ethernet cable, capable of transferring data at speeds of 1 Gbps or higher. Local computers and devices (i.e., network nodes) typically connect through a switch or router in a “star” topology, which allows multiple computers to connect within a LAN and with other networks using an uplink.

Wi-Fi enables devices to connect to networks without using cables but by using short-range electromagnetic signals that propagate through the air and through many materials. Communication by Wi-Fi is less secure than by cabling but allows devices to be mobile and easily connected.

Network communications are based on the OSI model, which describes the various layers that network data pass through when sent over a network. These layers are called application, presentation, session, transport, network, data link, and physical. TCP/IP is the main protocol for network connections; it uses routing and IP addressing to deliver data packets to a destination, where the packets are reassembled into the original data.

The internet is a countless and ever-changing group of interconnected networks that spans the entire world. Information is delivered by first traveling through local Ethernet networks and then through routers to ISPs connected to the internet backbone (including massive undersea cables connecting continents and satellite networks covering remote locations). Older copper cabling is being replaced with fiber optic cabling. To facilitate internet communication between users, the DHCP automatically configures computers, and the DNS assigns user-friendly host names to servers on the internet.

The IoT consists of nonstandard devices connected to the internet (e.g., smart appliances and self-driving vehicles). Billions of such devices are connected already. Protocols are still being developed for the IoT and many security issues remain, such as devices being vulnerable to commandeering for nefarious purposes. Many newly developed smart devices are added to the IoT every day, increasing the demand for programmers to write code for these devices.

UNIT 7

SOFTWARE

STUDY GOALS

On completion of this unit, you will be able to …

- describe how BIOS works.
- explain modern file systems.
- recommend application software for operating systems.
- discuss the concept of embedded systems.
- summarize the process of software development.
- delineate the functions of an operating system.

7. SOFTWARE

Introduction

Microsoft was founded by Bill Gates and Paul Allen in 1975. In the 1980s, IBM asked them to create an operating system (OS) for the IBM personal computer (PC). Microsoft 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 presented a command line on the display where the user typed in instructions for the computer. The abbreviation “DOS” stands for “disk operating system,” meaning it allows the user to access files on floppy and hard disks (floppy disks were available in the 1980s and were so named because they were made of a flexible magnetic material).

Gates sold the OS to IBM but retained the copyright, which paid off when other computer manufacturers also wanted to use the Microsoft DOS (MS-DOS). Microsoft rapidly became a major player in the operating systems market. In 1989, they released Microsoft Office, a suite (collection) of applications targeting office productivity. The brand names Word, Excel, and PowerPoint soon became the international standard for digital office files.

Microsoft released their first successful version of the Windows OS in 1992: Windows 3.1 (Windows 3.0 was released in 1990 but was not nearly as well-adopted as 3.1). Windows 3.1 was a great success and reinforced the trend whereby users were moving away from command-line OSs and toward OSs with graphical user interfaces. Further successful versions of Windows followed: 95, 98, NT, 2000, XP, 7, 8, and 10 (“9” was not used to disambiguate between 95 and 98). As of 2024, Windows operates on almost 75% of the world’s desktop and laptop computers (StatCounter.com, 2024), giving Microsoft Office and Windows the majority market share for office applications and OSs, respectively.

7.1 BIOS and Operating Systems

A computer system comprises both hardware and software. The hardware refers to the physical devices that make up the computer, and the software refers to the instructions that the hardware executes. When a computer is turned on, it must find and execute software instructions. This search would be easy if the computer always performed the same task – the hardware would simply be designed to execute the same instructions. However, a fundamental advantage of a computer is that it can perform countless tasks. Each time a computer is turned on, it can be used for something different.

When a computer is turned on (or “booted up”), it knows nothing about its previous usage; it is a blank slate and needs to know where to start. The bit input-output system (BIOS) provides a starting line for a computer each time it is turned on.

Figure 68: BIOS: The Starting Line

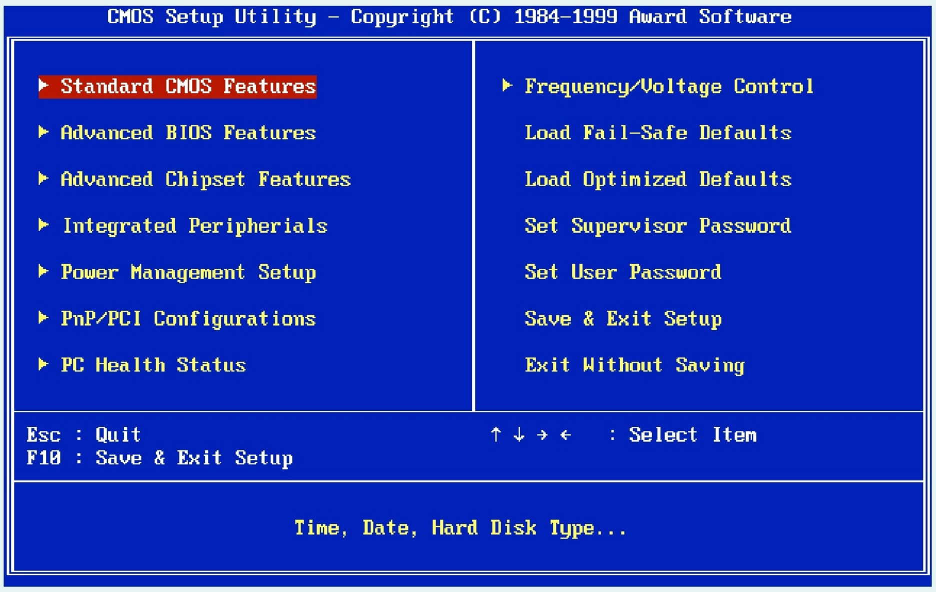
Source: Gladden, 2017.

When you turn on a computer, it is hardwired to search the BIOS chipset for instructions and execute them. This task must be done before the computer can accept any input or produce any output. Computers are shipped from the manufacturer with instructions already placed on the BIOS chipset. These instructions can be updated (or ““flashed“)” as new updates are released.

One of the first things a BIOS chipset instructs the computer to do is detect and test hardware 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. The BIOS performs a power-on self-test to check that the required hardware is present and functioning properly. Required hardware includes random access memory (RAM), the central processing unit (CPU), storage, and a display.

The BIOS is written in the “Basic” programming language, which can provide the user with simple testing and configuration functions, but it cannot run applications or display advanced graphics. Only an OS can do that. The BIOS has either already been configured with the location of the OS, or it searches the most likely locations (such as the hard disk) for an OS. Once it finds an OS, the process finishes by copying it (e.g., Windows or MacOS) into RAM. If no OS is detected, the process stops and returns an error, which may require the user to troubleshoot using the BIOS configuration screen, usually accessed by hitting a special key while the computer turns on.

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 significantly in the last three decades. They offer a basic text interface for the user to identify devices such as hard drives and provide other basic hardware configuration settings. Access to the BIOS configuration screen is often password-protected.

CMOS
This stands for complementary metal-oxide semiconductor, which is a chip that stores data when provided with electrical power, normally by a small battery.

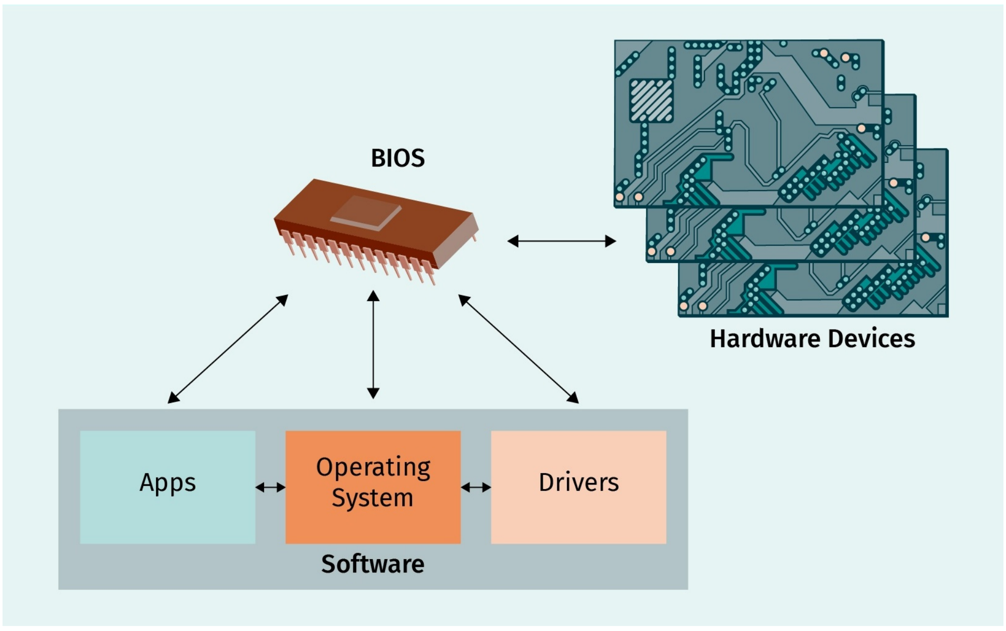
The **CMOS** is another chip used in computers. Originally, it contained all of the BIOS configuration information. A drawback of this chip is that it requires a small battery connected to the motherboard; when the battery is depleted, all the data are lost. Today's computers use flash memory, which does not need continual power to preserve the data. The CMOS still exists in personal computers today, but its main function is now to preserve the system clock (current time) for the computer. If the CMOS battery in a modern computer is depleted, the user will have to set the clock every time the computer is turned on until the battery is replaced.

Input and Output

Once the BIOS identifies the hardware on a computer, it provides software (OS and applications) access to that hardware. The BIOS chipset provides software direct access to devices, so the software need not know the hardware addresses. This arrangement is helpful because sometimes hardware addresses change, as do the actual PC components. Having the BIOS know the hardware addresses ensures a seamless transition because the software can always rely upon it to access devices.

The instructions on the BIOS chipset may seem similar to software. However, they actually fall into the class of firmware, which refers to hard-coded instructions installed on a device that generally cannot be changed (although they may be updated periodically). This strategy is logical because the BIOS must be the same every time the computer is turned on.

Figure 70: BIOS Input and Output



Source: Stephen Weese, 2020.

As indicated in the figure above, the BIOS handles hardware requests from the OS, applications, and drivers, which are all types of software. Applications use an application program interface (API) to request that the hardware execute certain tasks, such as playing sounds and displaying output. The API forwards these hardware requests to the OS, which uses drivers to translate these requests into formats that a device can understand. These requests are then passed to the BIOS, which relays them to the appropriate hardware device. Input and output thus travels through the BIOS. Application programmers must use the proper API procedures to communicate with the OS, be it Windows, Android, iOS, MacOS, Linux, or something else.

Operating System Function

We have learned that the BIOS plays a fundamental role in turning on a computer and then passes control to the “OS”. In addition, after the computer completes this startup phase, the OS continues to use the BIOS to access hardware. The DOS provided by Microsoft gave users basic access to a computer’s functions. One of the primary functions of a computer is disk access, which is crucial to an OS. In addition to disk access, an OS has to provide an interface, control the hardware, and run applications.

A crucial task for any computer is storing and retrieving data, which is done by the hard disk drive or solid-state drive. Although the OS relies on hardware drivers to translate its data-storing and -retrieving requests, the details of these requests are handled by the file system chosen by the OS. Different OSs may choose different file systems, even for the same hardware.

File systems

The MS-DOS operating system used a simple file system called file allocation tables (FAT), which required “formatting” a disk. Both floppy and hard disks were formatted with FAT (and sometimes still are today). Formatting a drive prepares it for use and also erases all 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 contains the code to launch the OS and the BIOS is given the address of this cluster to start the system. The original FAT, now called FAT16, used 16-bit addressing, so a maximum of 65,536 clusters could be created. Later, as hard drives grew in capacity, FAT16 was replaced by FAT32, which uses 32-bit addresses for over 260 million clusters.

Fragmentation
having files spread over different hard drive locations, causing slow access times

File tables identify files by file name and their starting location on the hard drive. FAT systems put files in the next available open space. Since earlier files could be deleted, this strategy created gaps of unused storage capacity called**fragmentation**. File systems and OSs work together, and every operating system has its preferred file system, as noted in the table below.

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 number of files User quotas Resizeable volumes
MacOS 10.13	APFS—Apple File System	File encryption File security Large file sizes Large number of files High access speed On-demand volume size
Linux (various types)	ext4—Fourth extended file system	File encryption File security Efficient file organization Large file size Large number of files Fragmentation protection

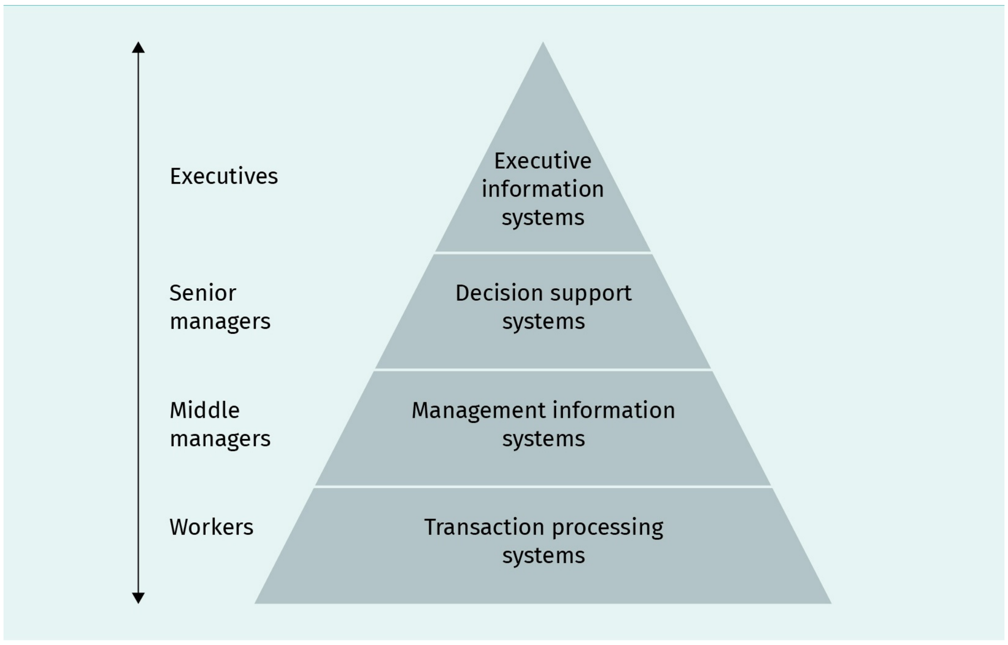
Source: Stephen Weese, 2020.

Computers using these file systems must have their primary hard disk partition or volume (a unit of space) formatted with the given file system. For example, Windows must be installed with NTFS. Sometimes, problems arise because portable external drives may not be used on different OSs. For instance, Windows cannot read a hard drive formatted with APFS.

7.2 Application Software and Information Systems

Software applications 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 serve to manage information. These applications are part of an information system, which can be divided into the levels shown in the figure below.

Figure 71: Information System Levels


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

Source: Kimble, n.d.

A traditional work environment has many levels of management, each with its own information needs. Executives are responsible for the highest-level decisions of a company and rarely interact directly with workers – usually they interact with senior or middle management. The informational needs of an executive differ from those of a senior manager, whose job it is to implement the high-level decisions made at the executive level. Managing workers is the primary job of middle managers, so they require information on the workers. The worker level requires a task management system to track which employees are working on which job as well as deadlines and progress. Information system software is thus divided into various subcategories focusing on the specific needs of an organization.

Parts of Information Systems

Information is the result of processed data; it has been organized to provide insights into the subject of interest. An information system consists of raw data and software tools that analyze the data and present it in different ways for user interpretation. Thus, software is part of the information system, as are the people using it. Of course, software needs hardware to run on, so the hardware is also part of the information system. In such a system, tasks must be judiciously assigned because, although computers can perform many tasks more efficiently than people, some tasks can only be done by humans. Searching data is an ideal job for computers but difficult for humans, especially with large quantities of data. Thus, to manage their data, businesses must decide how to use the hardware, software, and people. This organization is also part of information systems.

Management Information Systems

Management information systems (MISs) is a type of information system that facilitates 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.).

Transaction Processing Systems

Transaction processing systems (TPSs) handle operational transactions within a business. These are mostly used by workers (see bottom level of the pyramid in the figure above). TPSs manage payroll, process custom orders, track inventory, validate data, fund transfers, etc. Data from TPSs are often used in higher-level information systems (Kimble, n.d.).

Decision Support System

Decision support systems (DSSs) are primarily designed to inform decision-makers. A good DSS should present information so that it is easily understandable. DSSs 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 predicted sales. DSSs are most often used at the middle-management level and above.

Executive Information System

An executive information system (EIS) is similar to a DSS but is specifically designed to address the needs of the executive level of the information systems pyramid. EIS data and analysis focuses on the whole company (i.e., the ""big" picture)". Executives rely on such data to make informed decisions for the company. EIS systems provide overall performance analyses, compare productivity between departments, display market research data, and predict future performance. These systems are designed for executives who may not have technical skills, so a good EIS must be user-friendly.

Online Analytical Processing

Online analytical processing (OLAP) involves a high-performance data analysis system. OLAP 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, etc. These systems are good for **data mining** and creating reports. They are frequently used by middle management.

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

7.3 Applications

The two primary types of software are OSs and applications, often referred to as "apps." OSs have access to the computer hardware, whereas apps are designed to perform specific tasks. An app may be considered a tool – it is used for a given task and then put away until the next time it is needed.

Apps for Every Purpose

Many different types of apps exist. They are used not only on PCs but also on smartphones. They have many different characteristics, which we discuss below, and work in different ways.

App size

Apps may be small – sometimes just a few kB. Apps such as a to-do list or notepad do not require much RAM (around 200 kB). Some medium-sized apps, such as Zoom for video calling, require 50 to 100 MB of RAM. Other apps, such as Adobe Photoshop, perform many functions and often require over 1 GB of RAM. These apps are divided into several files that contain the program code, and only the required files are loaded into memory. For instance, an artistic filter may be useful when editing a photo. If the user requests such a filter, Adobe Photoshop loads it into RAM. Other parts of the program that are not immediately needed remain stored on the hard drive or solid-state drive until needed. Some apps, such as video games, are even larger than Adobe Photoshop and require many gigabytes of space. The popular game World of Warcraft requires over 70 GB of space on the hard drive or solid-state drive. As with other large apps, video games are split into separate files, and only those immediately needed are loaded into RAM. Some apps come in "suites" (i.e., collections), such as Microsoft Office or the Adobe Suite. These can require lots of space because they are several apps installed together.

App operating systems

Each OS has a different API to communicate with the hardware. Thus, an app written for a specific OS will only work with that OS – it must be modified to function with a different OS. This explains why some applications are available only for Macs and others only for PCs. Apps such as MS Word that are available on both Mac and PCs are actually two apps, one for each OS. The same holds for smartphones – apps must be made for either Android or iOS, but a single app cannot work on both. For example, 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 smartphones, most software companies first make apps for these OSs. Later, they may decide to make other versions for other OSs.

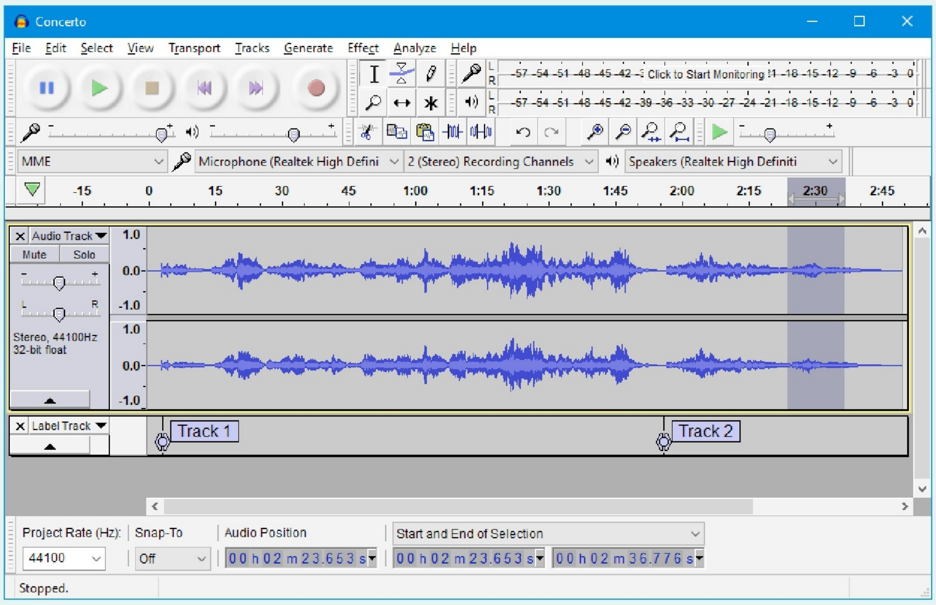
Unverified apps

Anyone with basic knowledge of programming and sufficient time can create an app. However, users should exercise caution before exposing their valued hardware to such apps. Apps found through verified websites and that are tested and known to be stable and virus-free are the best choice. Unverified apps should be avoided because they likely contain bugs that can seriously damage the user‘s hardware or be malicious apps designed to steal valuable and sensitive information (passwords, contact lists, bank account numbers, etc.).

Freeware

Some apps can be obtained without payment (although their developers may ask for donations). A good example of such an app is Audacity, a free audio editing software. Audacity is "open source," meaning that the program's source code is available and can be modified by anyone. The Audacity website also allows users to donate if desired.

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
Audio Track
Mute
Solo
Stereo, 44100Hz
32-bit float
Label Track
Track 1
Track 2
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
Stopped.

Source: Audacity Team, 2020.

Adware

Another type of app is adware, which is software sponsored by advertisements. These apps typically allow the user to use all the software’s features but at the price of occasional advertisements or banner ads. The user must pay for the software to dispense with the annoying advertisements.

Off-the-shelf apps

Before the internet became widely available, computers and apps were bought at stores (apps were sold on read-only memory supports such as optical disks). After purchasing the app, the user could use it according to the license terms on the packaging. Usually, the terms meant that the app could be installed and used personally on one or two devices. This business model remains popular today (2024), except users can now purchase and download apps directly online. With this business model, the user buys one specific version of the software and must pay again to obtain a newer version. This business model suffers greatly from piracy, whereby people circumvent the copy protection and distribute the app free of charge. This problem eventually led to the subscription model.

Subscription apps

A newer business model for apps is the subscription model, which is similar to “renting” an app. The user can install the app on one or more computers and use it as much as desired for a monthly or annual fee. The subscription model has three main benefits: First, it prevents piracy by continually verifying your subscription through an internet connection. Second, it makes software more affordable for users. Instead of paying ¥80,000 for a fixed version of the software, the user can pay ¥5,000 per month for as long as they can afford it. This price model also decreases the incentive for piracy. Third, when a new version becomes available, subscriptions are often automatically upgraded, so the user always has the most recent version of the software.

Malware

Some software contains destructive or nefarious code. This is called “malware,” which comes from the Latin word “mal,” meaning “evil.” Many websites offering “free” software actually distribute malware. Malware includes viruses that can replicate on host computers and “Trojan horses” that masquerade as a useful program but actually contain malware that may steal your data, compromise your security, or even delete data from your device.

Creating Apps

Creating complex apps is a multilayered task. Very few apps are coded by one person, and those that are tend to be small and simple. Larger apps are made by teams of programmers working together with user interface (UI) designers. Once an application team designs an app, coders begin to code it. This is where the computer language (or languages) is chosen, as well as the OS on which the app will run. The process involves coding and debugging code until an early version, called the alpha version, is produced. The alpha version is tested and is followed by the beta version. The beta version is tested extensively in a process called “beta testing.” Finally, after the app has been thoroughly tested, it is released. Even in the released version, users may find bugs or other problems, requiring the development team to modify the code once again. This process takes months or, in most cases, years.

Just as apps 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, software functions are now expected to be available to all users across all time zones (so-called 24-7 availability for all users worldwide). In addition, users now expect software functions to be available on all end devices, from the smartphone to the tablet to the home PC or Mac, which has also driven the development of cloud services.

For all these reasons, creating an app requires that it be broken down into manageable parts. The part of the app that depends on the device’s operating system is only a small part of the overall app and so may be ported to all supported devices. The part with the actual functionality in the cloud (e.g., the payment service) is executed in the secure back-end financial system and not on the user’s device, which is much less secure. However, although this measure improves the app’s security, it also requires that personal data be distributed across data centers worldwide, which has necessitated national regulations to protect data.

These requirements for developing apps have led to modifications in the processes of software design, development, and operation. Relatively recent concepts include agile development processes such as SCRUM or SAFe (scaled agile framework for lean enterprises) and the interlocking of development and operation (so-called DevOps). Due to increased security issues in cloud computing, DevOps is supplemented by security considerations, giving the DevSecOps methodology. All of these concepts are essential for the discipline of Software Engineering.

7.4 Embedded Systems

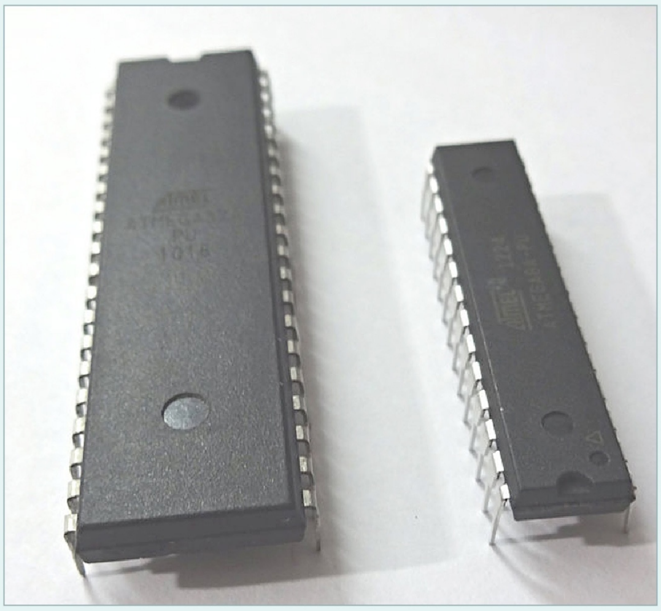
All over the world, people own multiple computing devices, such as computers, smartphones, and tablets. In addition, people own a myriad of electronic gadgets such as smart watches, headphones, televisions, kitchen appliances, digital clocks, and electric vehicles. In fact, electronic devices vastly outnumber actual computers. 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 repeatedly. Devices such as medical equipment, fire alarms, voting machines, standalone global positioning systems, washing machines, traffic lights, electronic toys – even switches and routers contain embedded systems. The simple definition of an embedded system is a computer system that is designed for one specific task. Embedded systems include a CPU, memory, and an input-output system. By using microprocessors, embedded systems 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 travel through specific pins on the chip.

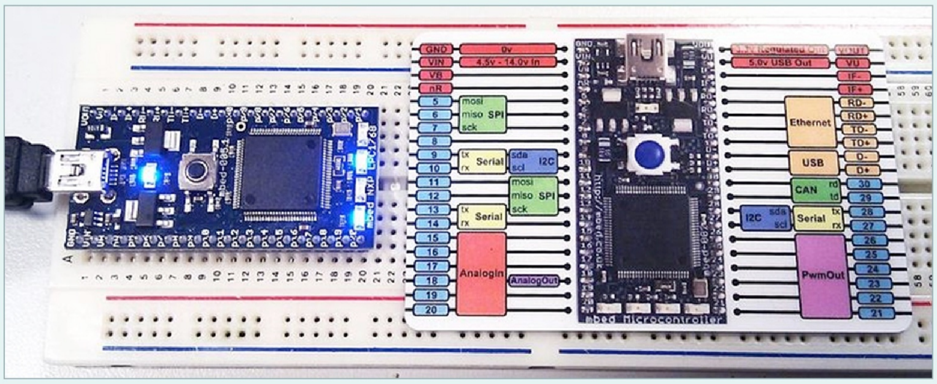
Figure 73: Microcontrollers



Source: Vahid alpha, 2013.

The chips can send data to universal serial bus (USB) ports and receive data from USB ports. The input-output pins can even connect to the Ethernet.

Figure 74: Microcontroller Pin Outs



Source: Nbauers, 2011.

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

Embedded System Design

Since an embedded system typically performs a single function over and over again, the software can be placed on a computer chip, making it firmware. However, the requirements for an embedded system differ from those for a computer app.

Unlike apps, embedded systems do not regularly upgrade their firmware, so the firmware must be extremely reliable. Thus, firmware destined for embedded systems must undergo rigorous testing before being placed on the device.

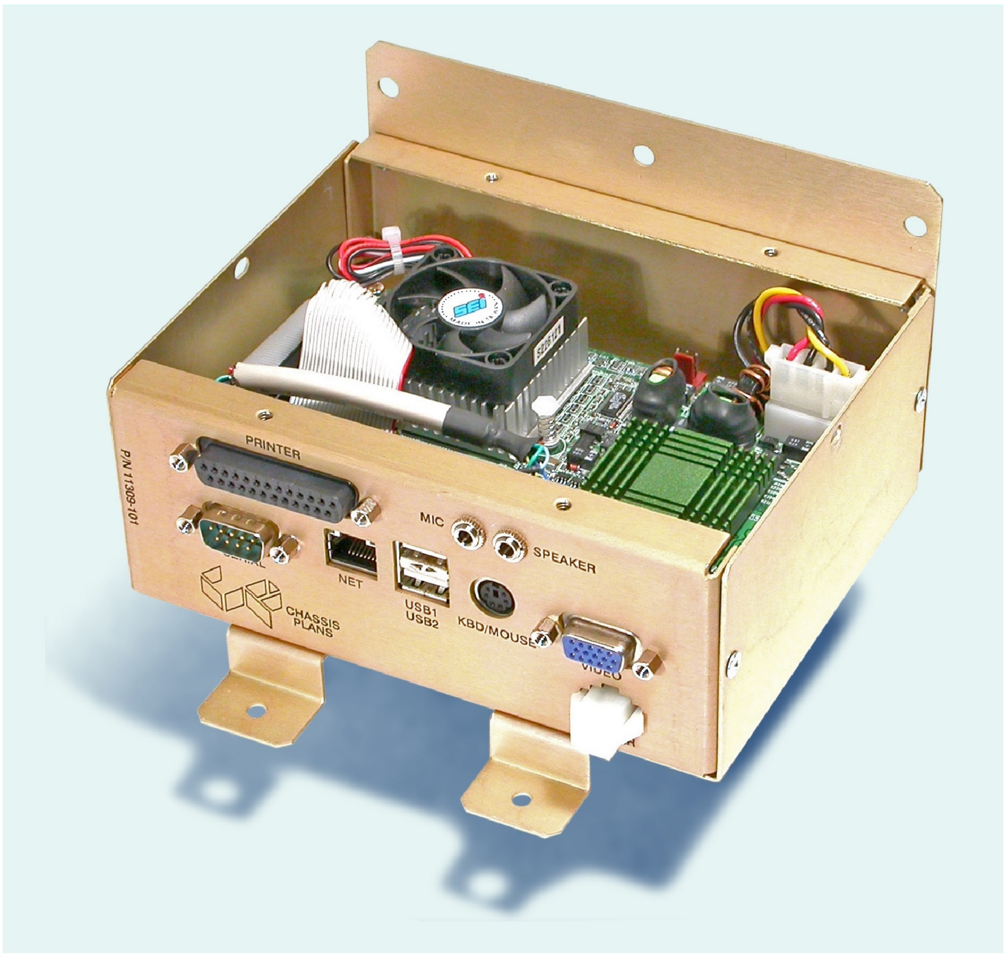
Electronic devices also have minimal RAM, so programmers must be as efficient as possible to squeeze the maximum amount of instructions into the available memory. This constraint makes programming for embedded systems very different from programming for a computer, where memory is much less of a concern (PCs have a considerable amount of RAM compared to embedded systems).

Examining Embedded Systems

An ultrasound scanning device in a medical or veterinary office is an example of a device requiring an embedded system. This device has one purpose – using ultrasound to create images of the internals of a human or animal patient. The input consists essentially of configuration choices from the user and data from the scanner. These data must be converted into the image of the patient and displayed, so the device must be designed to connect to an ultrasound scanner and a display. It must also be able to store an image and connect to a network.

Another example of a device requiring embedded systems is a voting machine (see figure below) . It must report data externally without error. Attributing a vote incorrectly or counting a vote twice are catastrophic errors for voting machines. In addition, the machine must not crash (i.e., stop working) while accepting votes. The internal memory must store the voter's choices and, when the voter has finished, send the output to a system that tallies the votes.

Figure 75: Voting Machine Internals



Source: Lippincott, 2003.

Embedded Systems and IoT

Initially, devices such as fire alarm systems or washing machines functioned independently. However, with the proliferation of the IoT, embedded systems are increasingly connecting to the internet. Connecting to the internet allows fire alarms to send alarm signals to emergency responders and collect critical data, such as when and where the alarm was triggered. Other devices, such as washing machines, might have less need to connect to the internet (perhaps for remote control), making it hard to justify the cost of additional network components. Since devices are often mass-produced for competitive markets, manufacturers continually strive to produce the best device at the least cost. Additionally, size reduction is becoming a factor. The market encourages ever smaller devices with the same functionality as larger devices. With modern IoT protocols, even small devices that consume little power can connect to a network.

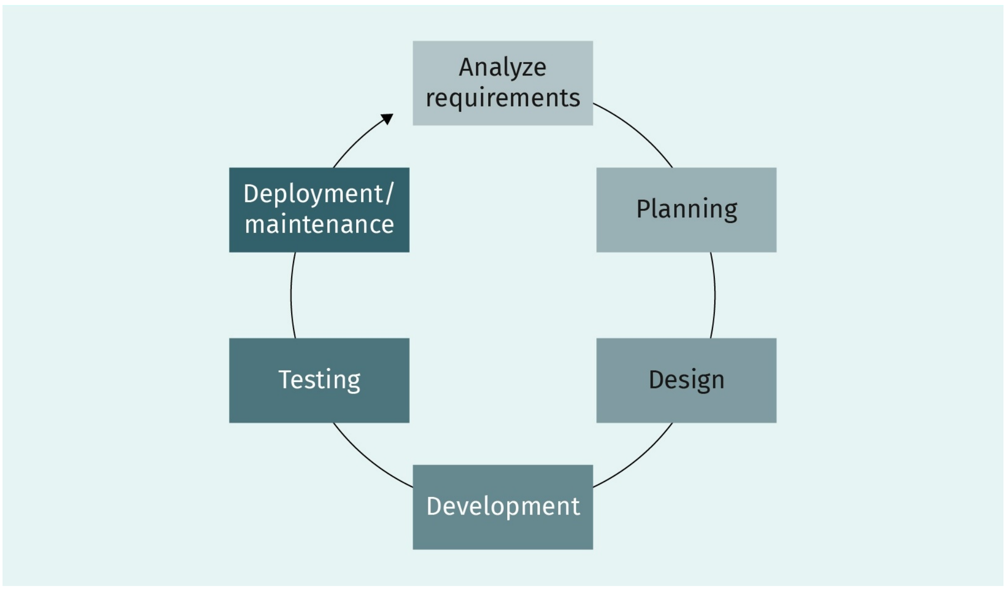
7.5 Software Development

In many ways, writing software is similar to writing a book. An author does not just sit down and start writing – many things must be decided first. For example, authors research the best title, write down the book's central themes, and create an outline. Software developers approach their task in a similar manner. Hours and hours of work are required before the first line of code is even written.

Software Development Life Cycle

Many methods exist to create software, and most follow the standard development life cycle. After the software is released, maintenance and updates are required. The updates follow a development cycle similar to that of the original software.

Figure 76: Software Development Life Cycle Concept



Source: Stephen Weese, 2020.

The software development life cycle (SDLC) concept shown schematically in the figure above incorporates the numerous aspects of software development (including technology and management) to develop software that meets a specific need.

Analyze requirements

Software development companies typically face one of two situations: they must create specific software for a client or create new software to release into the market. Both cases raise a common business question: "What problem will this software solve?" In other words, what needs are unfulfilled, either in the market or for a specific business? At this point, the company may want to poll the potential users of the software to determine their needs. In addition to user input, the company should take advice from marketing and programming experts. Once the objectives of the software are established, they are put together into a **requirements document**.

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

Planning

The planning stage of software development includes examining the project feasibility and assigning people to specific teams. Risks are examined, and strategies are developed to minimize those risks. Budgets are also created at this stage.

Design

Once a budget is approved, the relevant teams can begin working on the design, which starts by creating a design document. This phase is critical because it strongly affects the next phase, development (i.e., coding). The design must be analyzed to ensure it fits within the budget and that no major design flaws are overlooked. A faulty design could cause coders to begin working only to waste hours developing something that must be redesigned.

Development

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

Testing

Once the initial coding is complete, the software is extensively tested, which is the job of the quality assurance (QA) team. A good QA professional looks for errors in every aspect of a software application. During this stage, coders are notified of bugs and assigned to fix them. Testing is so important that tests may be coded a priori and the software developed to satisfy the tests.

Deployment and maintenance

Once the QA team has completed the testing, the software is released to the client or into the market. In most cases, despite the extensive testing of the QA team, bugs are found after deployment. Part of software maintenance is updating the software to address such bugs. If a new version is required, it starts at the beginning of the life cycle and goes through all the phases.

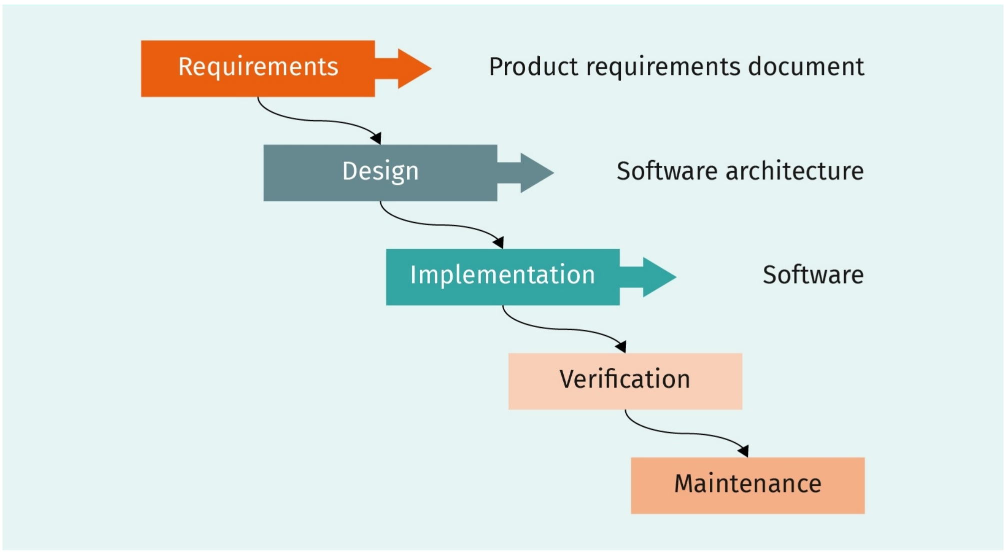
Implementations of the SDLC

The initial concept of the software life cycle has spawned numerous specific development models. We discuss below the waterfall, spiral, and big bang models.

Waterfall model

The waterfall model is shown schematically in the figure below. It is a rigid model that flows sequentially through the steps of the software life cycle. The idea behind the waterfall model is that, once a stage is completed, it will not be revisited. The steps of this model are similar to those of the SDLC.

Figure 77: Waterfall Model for Software



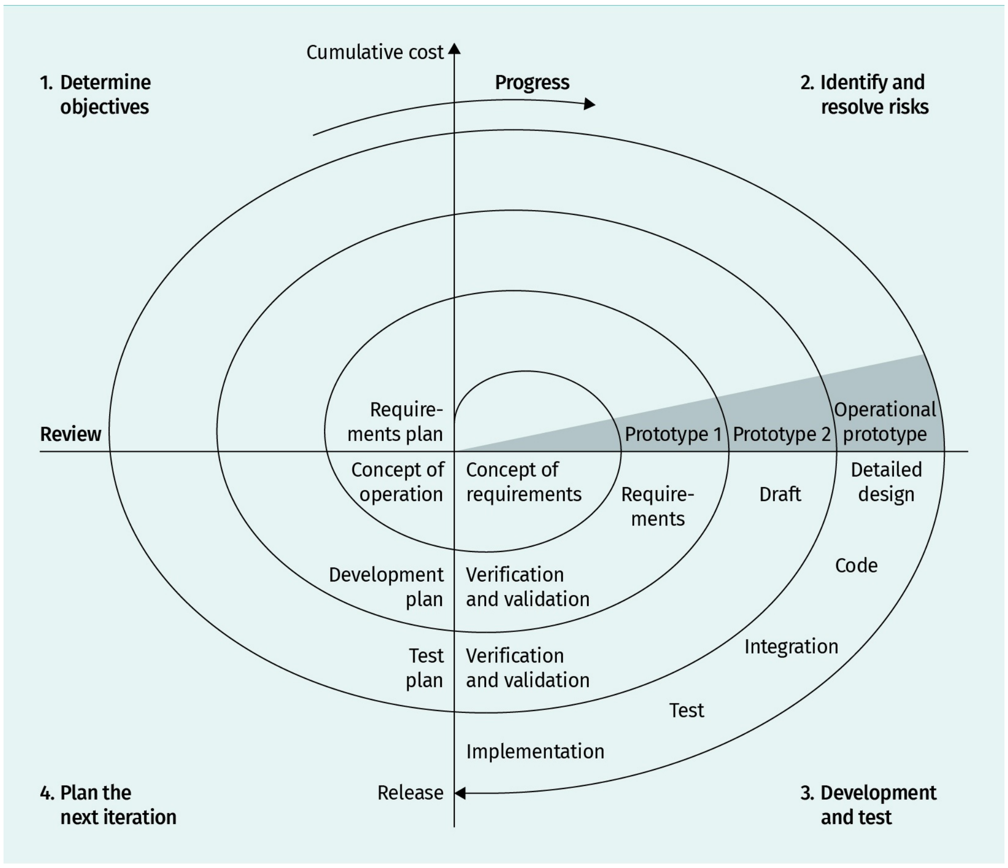
Source: Kemp & Smith, 2010.

The requirements and design phases of the waterfall model produce the requirements and design documents, respectively. The implementation phase is where the coding is done, and testing is done in the verification phase. The last phase, maintenance, involves the release or deployment of the software.

Spiral model

The spiral approach to software development (see figure below) intentionally goes through several cycles of planning, designing, development, and testing before final release. An advantage of this model is that various teams can start working earlier in the process and that some teams can work simultaneously.

Figure 78: Spiral Model for Software



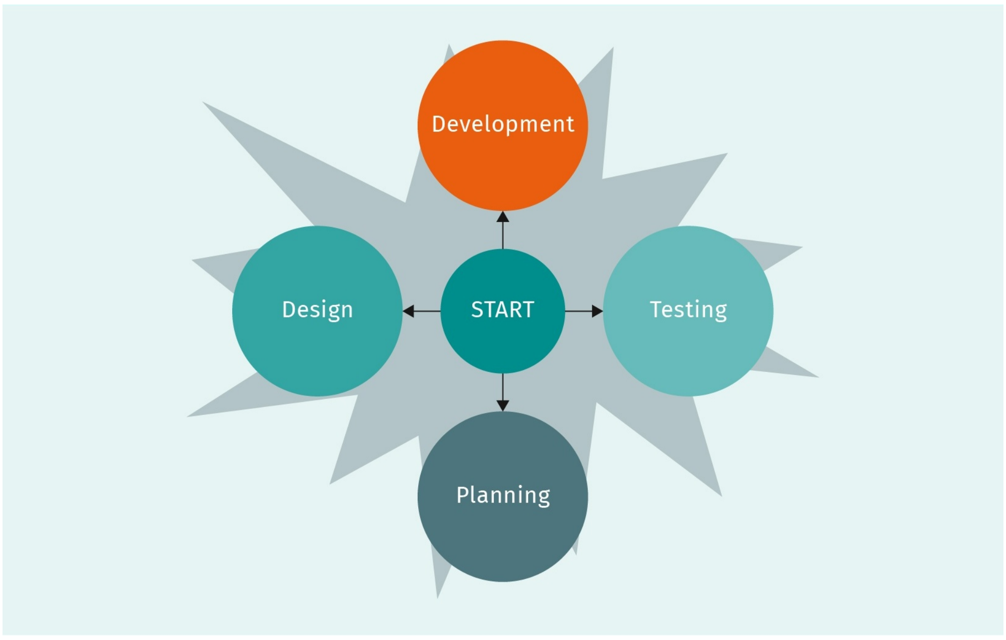
Source: Conny, 2004.

With the spiral model, several prototypes of the software are developed before the final release. More than one round of testing and verification happens, in addition to 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.

Big bang model

The big bang model, shown schematically in the figure below, is more recent than the others and is often used by startups and small companies because it is best suited for smaller projects. The main feature of this model is that coding, design, requirements, and testing all begin simultaneously with very little planning. The lack of planning makes the model risky, but it can also be the most rapid way to develop the software.

Figure 79: The Big Bang Model for Software



Source: Stephen Weese, 2020.

Although the big bang model involves little planning at the outset, plans can be created as development continues. Testing is done as soon as code is available. The big bang model is not recommended for large projects or essential services but can be used to brainstorm and quickly develop new apps.

SUMMARY

The bridge between hardware and software is the BIOS, which is the basic input-output system for computers. When a computer is turned on, it is designed to search the BIOS for the startup instructions. These instructions qualify as firmware because they are expected to reside on the BIOS chip quasi-permanently, with very infrequent updates. The startup instructions test the hardware and execute the bootup process, eventually passing control to the OS, which is software that allows the user to control the computer.

The OS has three major functions: it controls hardware, provides a user interface, and runs applications. Without an OS, apps cannot be used. The OS uses long-term memory to store apps and data, which requires it to use a file system such as FAT32 or APFS.

Organizations using software applications benefit from using information systems to manage data, personnel, and products. Different types of information systems 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 coded for use with a specific operating system, so, for example, a Mac app cannot run on a Windows computer. Different categories of apps exist, including freeware, adware, subscription apps, and malware.

Embedded systems refer to electronic devices that perform very specific tasks. They include devices such as fire alarms, washing machines, and traffic light systems. Because they use firmware, embedded systems must be carefully designed to be extremely stable. In addition, their small size (usually single chips) constricts their memory, so their firmware must be highly efficient in its memory use.

Creating large enterprise applications requires a software development model. For this process, companies rely on the software development life cycle. This model contains several phases: analyzing requirements, planning, designing, development, testing, and deployment and maintenance.

Thus, software takes many different forms, from software designed for tiny devices that repeat specific tasks to large and highly complex enterprise systems. Coding these apps requires careful planning and design.

UNIT 8

COMPUTER SCIENCE AS A DISCIPLINE

STUDY GOALS

On completion of this unit, you will be able to …

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

8. COMPUTER SCIENCE AS A DISCIPLINE

Introduction

Research into computer science began in earnest during World War II and continued after the war at Cambridge University in the U.K. and Princeton University in the U.S. The current Department of Computer Science and Technology at Cambridge University was founded in 1937 as the Mathematical Laboratory. It commissioned its first digital computer, the EDSAC, in 1949 under the direction of Professor Maurice Wilkes (Wilkes, 2024). This machine was one of the first to store both data and instructions in memory. The Cambridge group created the first programming language, Autocode, and introduced the concept of debugging. Princeton University, the home of the influential mathematician John von Neumann, contributed more to the theoretical development of computing. Von Neumann developed the basic computer architecture in use today and contributed heavily to developing the Electronic Numerical Integrator and Computer (ENIAC). The work at Princeton and Cambridge led to machine-level programming techniques and the development of assembly languages and **compilers**(von Neumann, 2024).

compilers
a computer program that converts a high-level language such as C++ or FORTRAN into machine code that can directly command a computer

Computer science first emerged as an educational discipline at universities in 1962, with the founding of the computer science department at Purdue University in Indiana (U.S.) (Rice & Rosen, n.d.). Stanford University in California (U.S.) started a computer science program in 1965, and the University of California, Los Angeles started theirs in 1968. However, it was not until the 1970s that interest in computer science became widespread. The department at Purdue grew to around 100 students, and new departments were established at universities worldwide. Computer science degrees were not yet widely available; many departments only offered courses in computer science as part of a mathematics degree.

One of the most notable programmers of the 1970s was Warren Robinett, who graduated from Rice University in 1974 with a bachelor's in Computer Applications to Language and Art, which included courses in FORTRAN programming. In 1979, Robinett wrote the legendary computer game *Adventure* for the Atari 2600 gaming console. After Atari, he co-founded The Learning Company to create educational software for children, including the well-received *Rocky's Boots*, which taught kids basic concepts of circuit design and logic (Robinett, n.d.).

8.1 The Role and Subdisciplines of Computer Science

At first glance, computer science seems to focus on programming. Although programming is important, the true discipline of computer science is using computer technologies to solve complex problems. Before writing any code, computer scientists must analyze data and create mathematical algorithms that eventually dictate the structure of the programming. Without mathematical and analytical skills, computer scientists would have difficulty creating original code to solve new problems.

Computer scientists do not create computers, they use them as tools. Computers are designed and created by computer engineers and electronic engineers. However, computer scientists must understand how computers are designed, which is why most computer science degrees offer at least one digital-electronics class.

Mathematics and logic are also extremely important to the computer scientist. A flaw in a program‘s logic is not the fault of the hardware but the programmer who wrote the instructions. Computers can help programmers by detecting syntax errors and other issues, but the logic of the program remains the responsibility of the programmer.

In a traditional computer science program, students take classes in advanced mathematics, OSs, 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 computing.

Computer-Science-Related Disciplines

As the world becomes more information-dependent, the need for computer experts increases. The field of computer science encompasses many different specialties that a computer science student can pursue. Below are listed and briefly discussed several such fields.

Software engineer

A software engineer is a crucial member of the software development team and contributes to creating large-scale software applications. The software engineer must be familiar with the software development life cycle (SDLC) and should be able to create clean and efficient code, work with a team, document her work, and implement algorithms in a programming language. The term “software engineer” is often used interchangeably with the term ““software developer.””

Web developer

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

CSS
This abbreviation stands for cascading style sheets, which serve to lay out and format web sites.

Systems analyst

Systems analysts examine existing computer systems and recommend modifications or replacements to maximize user experience and productivity. They rarely write computer code directly but can recommend which programming language should be used for a given task. They can work on small computer systems or on large systems for large corporations.

User interface designer or developer

Software is often conceptualized as being layered. The “front” layer of software is the user interface (UI), which allows the user to interact 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 called the “backend.”

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 must be competent in a database language such as **SQL** and also master administrative tools for computing systems such as Oracle. Flaws or errors in database design can cause serious problems and must be quickly identified and corrected by the database administrator.

SQL
This abbreviation stands for structured query language, which is a programming 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 to "mine” them to produce information. Algorithms and logic are used to create code to quickly sort and filter data to produce results.

Software manager

After working as software developers, many people choose to move into management and lead teams of developers to create new software. Project managers are the top level of such managers. Understanding the SDLC is essential to managing technical teams in the business world.

Information security analyst

A security analyst needs skills in programming and networking and a strong understanding of security protocols and software. These analysts look for weaknesses in computer systems before they are breached. The information security analyst recommends changes to make computer systems more secure and helps implement company security policies.

Information systems manager

A business or organization with many users and a computer system needs an information systems (IS) manager to manage the organization's hardware and software, including purchasing software and distributing it among the employees. Software choices are examined to determine what best meets the needs of the organization.

Computer hardware engineer

A computer hardware engineer designs and creates hardware, including the low-level firmware instructions that are placed on hardware. This job deals with circuit boards, network cards, and design for embedded systems, among other things. Although a computer engineering major would typically perform this job, there is enough overlap that many computer scientists also enter this field.

Video game developer

A video game developer is a programmer whose skills stem from their knowledge of video games and the tools used for creating video games. Today, most video games are created with special tools that go beyond just a programming language. Software tools such as Unity and Unreal Engine see widespread use in this field.

Other roles exists that relate to computer science, such as software tester or **QA analyst**, cloud-computing engineer, and network administrator, to name a few. Numerous new types of computer-related jobs and skills are expected to appear post-2024.

QA analyst
This term 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 has been data mining and automated systems. It is estimated that 2.5 quintillion (10^{18}) bytes of data are created every day (Wise, 2022). The demand is growing for analyzing these data because the benefits can be enormous.

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

Figure 80: Amazon Shipping Warehouse

Source: Domdomegg, 2019.

Data Science

The job of a data scientist job is to analyze data and translate them into meaningful and useful information. Many different fields require data scientists.

Sciences

"The sciences," such as biology, physics, and chemistry, deal with large datasets. One example is biomedical data science. As the world's population grows, so does the need for health care. The world population of seven billion people provides a lot of data points. A biomedical data scientist analyzes data to predict how a medication should perform on a certain population. In physics, the CERN Large Hadron Collider produces enormous amounts of data—about 50 petabytes (50,000 terabytes) per year. These data require data scientists equipped with a supercomputer to study them (InsideHPC, 2018). This type of data mining led to the identification of the Higgs particle, and this work was awarded a Nobel Prize in 2013. NASA has one of the largest datasets in the world, containing countless images and other data related to the universe. Data mining is essential for finding meaningful data points in the vastness of the cosmos.

Business and marketing

Large businesses that have existed for decades, such as Toyota and Coca-Cola, have accumulated vast troves of data. They have sales data, marketing data, product performance data, and more. Mining such data can answer questions such as "how do we predict what products will sell?" and "what type of product will perform best?" In addition to their own data, companies can purchase data and services from data brokers and data-collection companies. Companies that do not leverage their data are doomed to fall behind competitors that can quickly identify future needs based on past performance.

Artificial Intelligence

Many different types of artificial intelligence (AI) exist. Specific (or narrow) AI is a system created to perform a specific task. An example of specific AI is the predictive text that Google proposed to complete searches: although it is good at predicting searches, it is useless at finding your lost keys. General AI remains a major programming challenge. A general AI system should be able to examine new situations, learn, and perform various tasks, like a person. Humans can learn and perform innumerable tasks, which is difficult for a computer. However, performing specific tasks is easy – in fact, computers perform many specific tasks better than humans.

Intelligence in the context of AI is more than just being able to perform a series of mathematical algorithms; it implies deduction and learning. AI can reach conclusions based on data it has never encountered before. Since computer systems can perform billions of calculations per second, their power can be harnessed to simulate intelligence.

Robots and Devices

AI is sometimes used to control physical machines. Numerous assembly lines worldwide use robots to create other machines and devices. Some such processes are fully automated, but many still require human supervision to treat problems that AI cannot manage.

Assembly-line robots can work with near-exact precision and repeatability because the parameters of their movement are programmed. More importantly, robots do not tire or get distracted. Many robots have four or six arms that all work simultaneously, exceeding the capabilities of a human. Some 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.

AI on assembly lines that package food can be programmed to detect defective food products and remove them from the line. Some systems achieve better results than human food checkers. Other AI systems are designed to work alongside humans and maximize efficiency. In this context, AI systems make intelligent observations of the food product to determine whether it should be packaged.

Intelligent Data Systems

Other systems do not involve physical robots but function instead as data systems. One such system commonly used today is facial recognition. Photos or videos of people can be scanned in less than a second, and different people can be identified by measuring facial features. These systems use AI to take into account illumination, shadow, angle of the face, and other variations to accurately recognize a face. Intrusion-detection systems that provide security for networks are also available. In the past, a person was required to take action to stop a network attacker by checking the alerts from the server. Now these systems take automatic countermeasures based on information. For instance, an intelligent system might detect a distributed denial of service attack coming from a specific part of the world and immediately block traffic from the primary routers passing it along.

Designing Artificial Intelligence

Different approaches are available to create an AI system. One such approach attempts to simulate a human brain; this approach is called a neural network because 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 and solve specific problems by applying predictive mathematics to datasets. For instance, AI systems are able to diagnose patients based on their symptoms, and the resulting diagnosis is as good as, or better than, those of physicians (Longoni & Morewedge, 2019).

Programmers use languages such as C++, Java, and Python to create AI. Another language often used is LISP, which is ideal for recursive tasks.

Computer science is thus the backbone of both data science and artificial intelligence. Data scientists must master not only computer skills but also mathematics and statistics. AI designers must understand complex mathematical algorithms and data structures. Both computer scientists and AI specialists are currently in high demand in the job market.

8.3 Ethical Aspects of Computer Science

Ethics is the study of moral concepts such as personal behavior, responsibility, good, and evil and enters the field of computer science in many ways. Three layers of responsibility can be identified: corporate or government, developer, and user.

An example of ethics would be a person’s (or an entity’s) moral code, which is the set of rules used to make decisions. Although the ethics of individuals can vary, group ethics apply in society. For instance, most countries have laws against stealing, assault, and murder. Most people would agree that these things are wrong and should be condemned to have a prosperous society. Prospering as a computer programmer also requires an understanding of ethics.

Corporate (or Government) Ethics

Many opportunities exist for a software company to act unethically. For example, they could steal intellectual property. New innovations and applications are always emerging, and code is protected as a company secret. A company could hire an employee from another company and coerce that person to reveal the secrets of his previous company. A more direct approach would be to steal a competitor’s idea and implement it first. Often such actions result in lawsuits, but the party on the ethical high ground does not always win.

Companies (or governments) could also lie about their product by overstating its advantages or promising nonexistent features. Failure to meet contract requirements and deadlines is another ethical problem companies may face. Another issue is the use and sale of private user information. User privacy is an essential ethical code to maintain because it ensures the public’s trust in computer systems (e.g., medical systems, banking systems, communication systems). Companies may also breach ethics by designing software that is highly addictive to minors in attempts to manipulate parents to spend money. Worse yet is software that allows minors to create accounts and incur debts they cannot pay.

To maintain proper ethics, a corporation or government must be transparent about product expectations, specifications, budgets, and deadlines. They must not attempt to deceive their user base about the product or include deceptive charges. Companies should maintain an ethics officer to prevent lapses in ethics by establishing strict standards for the company to abide by in its business.

Developer Ethics

A software developer also has ethical issues to consider. First, writing code is similar to writing a novel in that the author is expected to create original work and not copy the work of others Copyright law protects software code from being copied without permission, used in violation of the end-user license agreement, or being stolen by former employees or competitors. Although open-source software libraries are available and may be used liberally, using someone else's code without permission is not only unethical, it is illegal and can be severely punished.

As a software developer for a company or organization, you work under a certain expectation of confidentiality. Many companies require developers to sign nondisclosure agreements. Even without such legalities, leaking sensitive company information will likely result at least in the termination of employment.

Companies also expect employees 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 software user, one of the primary temptations is to pirate (i.e., illegally copy) software. Since software is not a physical product but consists simply of instructions and data, it is relatively easy to copy, and the act of copying it illegally lacks the visceral feel of stealing. Software users in the twenty-first century have many options to choose from before resorting to piracy, including free versions of the desired software, trial versions, and subscription plans.

Software users are also bound by the terms and conditions of their license agreement, even if they have not read it. These conditions dictate 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 end-user license agreement.

8.4 The ACM Code of Ethics and Professional Conduct

The Association for Computing Machinery (ACM) is the world's largest computer society and is open to anyone who uses computer technology (e.g., industry professionals, educators, researchers, end users). The ACM has its own code of ethics, which is available on its 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 guide those who ascribe to that aim.

For Who is the Code of Ethics?

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

General Principles of the Code

Regardless of a person's specific career or computing specialty, she can find guidance in the general principles of the ACM code.

Contributing to society

Computer users should understand the power of their actions and strive to create outcomes that benefit others.

Avoid harm

Computer professionals should consider which actions may cause harm to clients, users, consumers, and others.

Honesty

Fairness should be respected in all transactions; deception is not tolerated.

Respect creativity

Do not copy the work of others without giving them proper payment and/or credit. Do not pirate software.

Honor confidentiality and privacy

Protect sensitive company information from theft, and do not expose private user information to third parties or sell private user information without permission.

Professional Principles of the Code

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

Quality work

Maintain a high standard of quality for the process used and for the finished product.

Professional conduct

Conduct yourself professionally at all times and maintain your ethical standards. Strive to improve your computing skills.

Professional knowledge

Maintain up-to-date knowledge of the company policies and procedures and follow them.

Review and criticism

Accept feedback on your work and products and 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 any eventual consequences associated with them.

Keep security in mind

Create secure systems. Do not release systems until they pass thorough security testing.

Leadership Principles of the Code

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

Public good

Ensure that the public good is the central motivator of professional computing work.

Social responsibility

Clearly state social responsibilities for professionals and encourage their practice.

Personnel management

Employee management should enhance the quality of work life.

Central policies

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

Growth

Provide opportunities for group members to learn and grow as professionals.

Societal systems

Apply high levels of scrutiny and care when working with or creating systems destined to be integrated into society.

ACM Membership and the Code

Members of the ACM are expected to follow the principles set out above and to encourage other members to do the same. Members observing other members to breach the code should report the violation to ACM (ACM, 2018).

SUMMARY

Although many think of computer science as centered around programming, the field involves many other important subjects, including mathematics, logic, algorithms, and analysis. This broadness of the field allows computer scientists to work in fields such as data science, web development, information systems, and database administration.

Data science involves using mathematics, statistics, and computer algorithms to extract useful information from large datasets. It can be applied to any subject area that uses large datasets, including biology, medicine, physics, business, marketing, and manufacturing.

AI systems are computer systems that can perform actions that usually require human intelligence. Specific AI systems can perform a single task with great efficiency (usually greatly surpassing the efficiency of humans). General AI systems are harder to develop because they must be able to learn new tasks and make deductions like a human.

Computer science ethics can be divided into three levels: corporate (or government), developer, and user. Each level involves different ethical concerns. A large corporation must deliver software that fulfills the promised functionalities and must not infringe on copywritten code produced by others. Companies often hire an ethics officer to help establish and maintain ethical business practices.

Code originality should be maintained. Like other creative works, code falls under copyright law. Code must be original and not copied from the work of others. The developer must follow their employer's business practices and abide by their nondisclosure agreements.

To remain ethical, users must not pirate software and must honor the terms of software license agreements. Piracy remains problematic in the twenty-first century because it is easy to violate copy protection and distribute software online.

The ACM's 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 following company policies and not pirating software, and leadership principles such as creating public awareness of technology and encouraging employee advancement.

BACKMATTER

LIST OF REFERENCES

ACM. (2018). *ACM code of ethics and professional conduct*. https://www.acm.org/code-of-ethics

Alexiou, G. (2020, September 8). *Could Elon Musks's Neuralink be a game-changer for people with disabilities?* Forbes. https://www.forbes.com/sites/gusalexiou/2020/09/08/could-elon-musks-neuralink-be-a-game-changer-for-people-with-disabilities/

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2007). *The traveling salesman problem: A computational study.* Princeton University Press.

Aquegg. (2013). *4-bit linear PCM*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:4-bit-linear-PCM.svg

Argonne National Library. (2007). *IBM Blue Gene P supercomputer* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:IBM_Blue_Gene_P_supercomputer.jpg

Audacity Team. (2020). *Audacity*. https://www.audacityteam.org/

Bebergal, P. (2020, August 19). The computer game that led to enlightenment. *The New Yorker*. https://www.newyorker.com/culture/culture-desk/the-computer-game-that-led-to-enlightenment

Commodore 64. (n.d.). *History-Computer.com.* https://history-computer.com/ModernComputer/Personal/Commodore.html

Compo. (2010). *Four-level-pyramid-model*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Four-Level-Pyramid-model.png

Conny. (2004). *Spiral model (Boehm)*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg

Curtis, S. (2014, February 04). Bill Gates: A history at Microsoft. *The Telegraph*. https://www.telegraph.co.uk/technology/bill-gates/10616991/Bill-Gates-a-history-at-Microsoft.html

Dake. (2005). *C64 open2* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:C64_open2.jpg

de Lima, B. (2009). *Cisco hall infnet* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Cisco_Hall_Infnet.jpg

Domdomegg. (2019). *Amazon warehouse BHX4 loading docks 2* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Amazon_warehouse_BHX4_loading_docks_2.webm

Eck, D. J. (2009). *Introduction to programming using Java*. CreateSpace.

Ehrenberg, R. (2010, October 6). *Square pixel inventor tries to smooth things out*. Wired. https://www.wired.com/2010/06/smoothing-square-pixels/

Fabry, M. (2016, March 31). *The story behind America's first commercial computer*. Time. https://time.com/4271506/census-bureau-computer-history/

Gentle. (2010). *Old phonebooks at Salton Sea* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Old_Phonebooks_at_Salton_Sea.jpg

Gladden, K. (2017). *Starting line (unsplash)* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Starting_line_(Unsplash).jpg

Google (n.d.). *Surfacing useful and relevant content*. https://newsinitiative.withgoogle.com/hownewsworks/approach/surfacing-useful-and-relevant-content/

Heimnetzwerke.net. (2018). *Netgear DS108 hub (3)* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Netgear_DS108_Hub_(3).jpg

Helix84. (2008). *Checksum*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Checksum.svg

Hörz, M. (2020). *HxD - Freeware hex editor and disk editor*. https://mh-nexus.de/en/hxd/

Ilkant. (2006). *LampFlowchart*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:LampFlowchart.png

InsideHPC. (2018, September 28). *Argonne is supercomputing big data from the Large Hadron Collider*. https://insidehpc.com/2018/09/argonne-supercomputing-big-data-hadron-collider/

J.smith. (2007). *Switch-and-nest*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Switch-and-nest.jpg

Kemp, P., & Smith, P. (2010). *Waterfall model*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Waterfall_model.svg

Kephir. (2019). *Award BIOS setup utility*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Award_BIOS_setup_utility.png

Kimble, C. (n.d.). *Information systems and strategy, session 1: Types of information system and the classic pyramid model*. Chris-Kimble.com. http://www.chris-kimble.com/Courses/index.html

Kissel, K. (2012). *June odd-eyed-cat cropped* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:June_odd-eyed-cat_cropped.jpg

Leverege. (2020, March 10). *Breaking down IoT standards and protocols*. IoTforAll. https://www.iotforall.com/glossary-iot-standards-and-protocols/

Lippincott, D. (2003). *Accupoll-embedded-computer* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Accupoll-embedded-computer.jpg

Longoni, C., & Morewedge, C. K. (2019, October 30). *AI can outperform doctors. So why don't patients trust it?* Harvard Business Review. https://hbr.org/2019/10/ai-can-outperform-doctors-so-why-dont-patients-trust-it

Mahlknecht, G., & OpenStreetMap Contributors. (2015). *Submarine cable map umap*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Submarine_cable_map_umap.png

Main, D. (2015, April 2). Undersea cables transport 99 percent of international data. *Newsweek*. https://www.newsweek.com/undersea-cables-transport-99-percent-international-communications-319072

Maksim. (2005). *NetworkTopologies*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:NetworkTopologies.svg

Maxtremus. (2015). *Lifo stack* [Photograph]. Wikimedia Commons https://commons.wikimedia.org/wiki/File:Lifo_stack.png

McFadden, C. (2019, November 14). *9 of the best selling computers of all time*. Interesting Engineering. https://interestingengineering.com/9-of-the-best-selling-computers-of-all-time

Nbauers. (2011). *Mbeb microcontroller and pin out*. Wikimedia Commons https://commons.wikimedia.org/wiki/File:Mbeb_microcontroller_and_pin-out.jpg

Paulas, R. (2017, June 14). *Why is American internet so slow?* Pacific Standard. https://psmag.com/news/why-is-american-internet-so-slow

Pesce, M. (2014). *Toshiba smart mirror* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Toshiba_Smart_Mirror_(15060938918).jpg

Pjpearce. (2011). *Wireless access point* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Wireless_access_point.jpg

Pluke. (2011). *CPT-LinkedLists-addingnode*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:CPT-LinkedLists-addingnode.svg

Project Kei. (2020). *Dell Dimension C521 motherboard* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Dell_Dimension_C521_Motherboard.jpg

Oracle. (2020). *Primitive data types*. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Rautenberg, W. (2010). *A concise introduction to mathematical logic* (3rd ed.). Springer. http://page.mi.fu-berlin.de/raut/logic3/book.htm

Raysonho. (2014). *NestLearningThermostat2* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:NestLearningThermostat2.JPG

Raysonho. (2015). *EthernetCableGreen2* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:EthernetCableGreen2.jpg

Rice, J. R., & Rosen, S. (n.d.). *History of the department*. Purdue University Department of Computer Science. https://www.cs.purdue.edu/history/

Robinett, W. (n.d.). *Biography*. WarrenRobinett.com. http://warrenrobinett.com/

San Jose. (2006). *Blank map Europe with borders*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Blank_map_Europe_with_borders.png

Sandstein. (2011). *IBM system 360-50 console* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:IBM_system_360-50_console_-_MfK_Bern.jpg

Sato, M., Kodama, Y., Tsuji, M., & Odajima, T. (2021). *Co-Design and System for the Supercomputer "Fugaku."* IEEE Micro. https://ieeexplore.ieee.org/abstract/document/9658212/authors#authors

Schwartz, O. (2019, March 24). *Untold history of AI: Invisible women programmed America's first electronic computer.* IEEE Spectrum. https://spectrum.ieee.org/untold-history-of-ai-invisible-woman-programmed-americas-first-electronic-computer

SpaceX. (2022). https://www.spacex.com/updates/index.html

StatCounter.com. (2022). *Search Engine Market Share Worldwide.* https://gs.statcounter.com/search-engine-market-share/

StatCounter.com. (2024). Desktop operating system market share worldwide. https://gs.statcounter.com/os-market-share/desktop/worldwide

Starlink. (2022). https://www.starlink.com/

Stefan506. (2005). *Logic gate index*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Logic-gate-index.png

Stevens, W. R. (2004). *TCP/IP illustrated vol. 1: The protocols*. Pearson.

Steward, J. (2022, February 15). *The ultimate list of Internet of Things statistics for 2022*. Findstack. https://findstack.com/internet-of-things-statistics/

Tarnoff, D. L. (2007). *Computer organization and design fundamentals: Examining computer hardware from the bottom to the top*. Lulu.

TexasDex. (2006). *ENIAC Penn1* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:ENIAC_Penn1.jpg

Trung Quoc Don, D. (2019). *Red mailbox at the Rolderstraat 18, Assen* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Red_mailbox_at_the_Rolderstraat_18,_Assen_(2019)_02.jpg

Tucker, P. (2014, May 29). The military is building brain chips to treat PTSD. *The Atlantic*. https://www.theatlantic.com/technology/archive/2014/05/the-military-is-building-brain-chips-to-treat-ptsd/371855/

Vahid alpha. (2013). *Microcontrollers_atmega32_atmega8* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Microcontrollers_Atmega32_Atmega8.jpg

Venkatraman, D. (2020). *Fooplot.com*. http://fooplot.com/#W3sidHlwZSI6MCwiZXEiOiJ4XjIiLCJjb2xvciI6IiMwMDAwMDAifSx7InR5cGUiOjEwMDB9XQ

Vhcomptech. (2009). *LinkedList*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:LinkedList.jpg

Vincentq. (2007). *Macbook pro power button* [Photograph]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Macbook_Pro_Power_Button_-_Macro_(5477920228).jpg

von Neumann, J. (2024). *An early history of computing at Princeton.* https://paw.princeton.edu/article/early-history-computing-princeton

Wilkes, M. (2024). *Department of Computer Science and Technology.* https://www.cst.cam.ac.uk/history

Wise, J. (2022, Juni 26). *How much data is created every day in 2022? EarthWeb.*https://earthweb.com/how-much-data-is-created-every-day/

Zimmermann, K. A. (2017, September 07). *History of computers: A brief timeline*. LiveScience. https://www.livescience.com/20718-computer-history.html

ZZT32. (2007). *ASCII table*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:ASCII-Table.svg

LIST OF TABLES AND FIGURES

IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Mailing Address
Albert-Proeller-Straße 15-19
D-86675 Buchdorf

media@iu.org
www.iu.org

Help & Contacts (FAQ)
On myCampus you can always find answers
to questions concerning your studies.