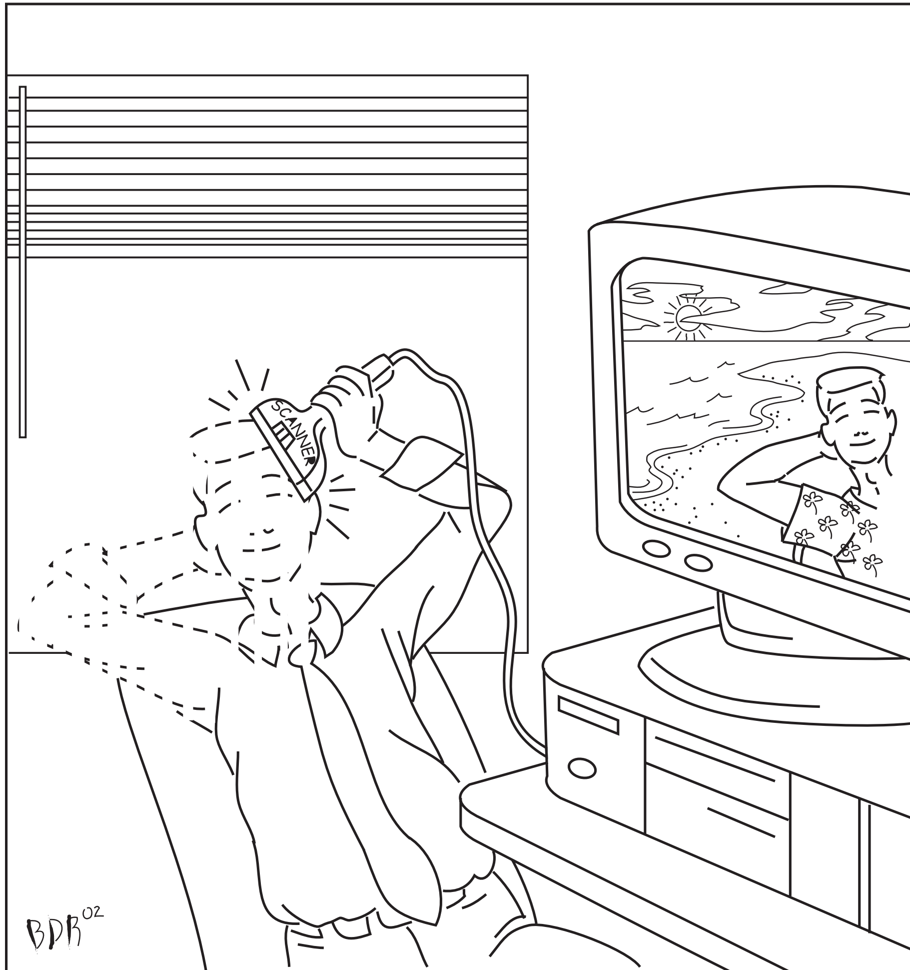
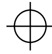
CHAPTER 4

DATA FORMATS



Thomas Sperling, adapted by Benjamin Reece



4.0 INTRODUCTION

In Chapter 3 you had a chance to explore some of the properties of the binary number system. You are already aware that within the computer the binary number system is the system of choice, both for all forms of data storage and for all internal processing of operations. As human beings, we normally don't choose to do our work in binary form. Our communications are made up of language, images, and sounds. For written communications, and for our own data storage, we most frequently use alphanumeric characters and symbols, representing English or some other language. Sometimes we communicate with a photograph, or a chart or diagram, or some other image. Images may be black and white or color; they may be still frames or moving. Sounds often represent a different, spoken, form of written language, but they may also represent other possibilities, such as music, the roar of an engine, or a purr of satisfaction. We perform calculations using numbers made up of a set of numeric characters.

In the past, most business data processing took the form of text and numbers. Today, multimedia, consisting of images and sounds in the form of video conferencing, PowerPoint presentations, VoIP telephony, Web advertising, and more is of at least equal importance. Since data within the computer is limited to binary numbers, it is almost always necessary to convert our words, numbers, images, and sounds into a different form in order to store and process them in the computer.

In this chapter, we consider what it takes to get different types of data into computer-usable form and the different ways in which the data may be represented, stored, and processed.

4.1 GENERAL CONSIDERATIONS

At some point, original data, whether character, image, sound, or some other form, must be brought initially into the computer and converted into an appropriate computer representation so that it can be processed, stored, and used within the computer system. The fundamental process is shown in Figure 4.1.

Different input devices are used for this purpose. The particular choice of input device reflects the original form of the data, and also the desired data representation within the computer. Some devices perform the conversion from external form to internal representation within the input device. At other times, the input device merely serves to transform the data into a raw binary form that the computer can manipulate. Further conversion is then performed by software within the computer.

There are varying degrees of difficulty associated with the input task. Normal keyboard input, for example, is relatively straightforward. Since there are a discrete number of keys on the keyboard, it is only necessary for the keyboard to generate a binary number code for each key, which can then be identified as a simple representation of the desired character. On the other hand, input from a device that

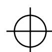
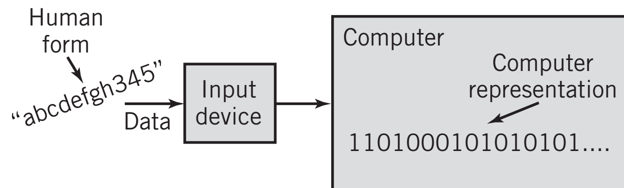


FIGURE 4.1

Data Conversion and Representation



presents a continuous range of data (i.e., analog data) presents a more formidable task, particularly if the data is continuously changing with time, which is the case with a video camera or microphone.

Adequate representation of the sound input from a microphone, for example, will require hardware designed to convert the sound into binary numbers and may require hundreds or even thousands of separate pieces of data, each representing a sample of the sound at a single instant in time. If the sound is to be processed within the computer into the form of words in a document the task becomes even more challenging, since the translation of sounds into character form is very complex and difficult, requiring sophisticated, specialized software.

The internal representation of data within the computer reflects the complexity of the input source, and also the type of processing that is to take place. There is no need to preserve all the individual points that make up a photographic image, for example, if the goal is only to extract and process the characters that are present on the page; it is only necessary to input and represent the entire set of data long enough to extract the actual data that is to be used or kept. On the other hand, if the image is to be used as a figure in an art book, it will be necessary to represent the image, with all its details, as accurately as possible. For input forms that represent a continuum of values, such as photographic images, video, and sound, the quantity of binary numbers and the number of bits in each that are required to represent the input data accurately will grow quite rapidly with increasing accuracy and resolution. In fact, some form of algorithmic data compression will often be necessary to reduce the amount of data to a manageable level, particularly if the data is to be downloaded or streamed over a low-speed transmission device, such as a telephone modem or a network with limited bandwidth.

Of course, once the input data is in computer form it can be stored for future use, or it can be moved between computers through networks or by using portable computer media such as CD-ROM, flash drives, or, perhaps, even iPods. Images and sounds can be downloaded from a website or attached to e-mail, for example. Provided that the receiving computer has the appropriate software, it can store, display, and process a downloaded image just as though the picture had been produced by an image scanner connected directly to its own input.

For storage and transmission of data, a representation different from that used for internal processing is often necessary. In addition to the actual data representing points in an image or characters being displayed, the system must store and pass along information that *describes* or *interprets the meaning* of the data. Such information is known as **metadata**. In some cases, the description is simple: to read a pure text file may require only a single piece of information that indicates the number of characters in the text or marks the end of the text. A graphic image or sound requires a much more detailed description of the data. To reproduce the image, a system must know the type of graphical image, the number of colors represented by each data point, the method used to represent each color, the number of horizontal and vertical data points, the order in which data points are stored, the relative scaling of each axis, the location of the image on the screen, and much more. For a sound, the

system must know how long a time period each sample represents, the number of bits in each sample, and even, perhaps, how the sound is to be used and coordinated with other sounds.

Individual programs can store and process data in any format that they want. The format used to process and store text in WordPerfect is different from that used by Microsoft Word, for example. The formats used by individual programs are known as **proprietary formats**. Proprietary formats are often suitable for an individual user or a group of users working on similar computer systems. As noted in Chapter 1, proprietary standards sometimes become de facto standards due to general user acceptance.

Note that it is important to distinguish between the data representation used within an individual piece of software and the data representation used for the input, output, storage, and exchange of data, however. Modern computer systems and networks interconnect many different types of computers, input and output devices, and computer programs. A Web page viewed on a Macintosh computer might contain an image scanned on a Hewlett-Packard image scanner, with HTML created on a Dell PC, and be served by an IBM mainframe, for example.

Thus, it is critical throughout this discussion that *standard* data representations exist to be used as interfaces between different programs, between a program and the I/O devices used by the program, between interconnected hardware, and between systems that share data, using network interconnections or transportable media such as CD-ROMs. These data representations must be recognized by a wide variety of hardware and software so that they can be used by users working within different computer environments.

A well-designed data representation will reflect and simplify the ways in which the data is to be processed and will encompass the widest possible user community. For example, the order of the letters in the alphabet is commonly used for the sorting and selection of alphanumeric business data. It makes sense, then, to choose a computer representation of alphabetic characters that will simplify these operations within the computer. Furthermore, the representation of alphanumeric characters will encompass as many of the world's languages as possible to aid in international communication.

FIGURE 4.2

Some Common Data Representations

Type of data	Standard(s)
Alphanumeric	Unicode, ASCII, EBCDIC
Image (bitmap)	GIF (graphical image format), TIFF (tagged image file format), PNG (portable network graphics), JPEG,
Image (object)	PostScript, SWF (Macromedia Flash), SVG
Outline graphics and fonts	PostScript, TrueType
Sound	WAV, AVI, MP3, MIDI, WMA
Page description	pdf (Adobe Portable Document Format), HTML, XML
Video	Quicktime, MPEG-2 or -4, RealVideo, WMV, DivX

There are many different standards in use for different types of data. A few of the common ones are shown in Figure 4.2. We have not included the standard representations for numerical data; those are discussed in the next chapter.

This section described the general principles that govern the input and representation of data. Next, we consider some of the most important data forms individually.

4.2 ALPHANUMERIC CHARACTER DATA

Much of the data that will be used in a computer are originally provided in human-readable form, specifically in the form of letters of the alphabet, numbers, and punctuation, whether English or some other language. The text of a word processing document, the numbers that we use as input to a calculation, the names and addresses in a database, the transaction data that constitutes a credit card purchase, the keywords, variable names, and formulas that make up a computer program, all are examples of data input that is made up of letters, numbers, and punctuation.

Most of this data is initially input to the computer through a keyboard, although alternative means, such as magnetic card stripes, document image scanning, voice-to-text translation, and bar code scanning are also used. The keyboard may be connected directly to a computer, or it may be part of a separate device, such as a video terminal, an online cash register, or even a bank ATM. The data entered as characters, number digits, and punctuation are known as **alphanumeric data**.

It is tempting to think of **numeric characters** as somehow different from other characters, since **numbers** are often processed differently from text. Also, a number may consist of more than a single digit, and you know from your programming courses that you can store and process a number in numerical form within the computer. There is no processing capability in the keyboard itself, however. Therefore, numbers must be entered into the computer just like other characters, one digit at a time. At the time of entry, the number 1234.5 consists of the alphanumeric characters "1", "2", "3", "4", ".", and "5". Any conversion to numeric form will take place within the computer itself, using software written for this purpose. For display, the number will be converted back to character form.

The conversion between character and number is also not "automatic" within the computer. There are times when we would prefer to keep the data in character form, for example, when the numbers represent a phone number or an address to be stored and processed according to text criteria. Since this choice is dependent on usage within a program, the decision is made by the programmer using rules specified within the program language being used or by a database designer specifying the data type of a particular entity. In simple languages like BASIC, for example, the programmer makes the choice by placing a $ character at the end of a variable name that is to be used to keep data in alphanumeric form. In C++ or Java, the type of variable must be declared before the variable is used. When the data variable being read is numerical, the compiler will build into the program a conversion routine that accepts numerical characters and converts them into the appropriate numerical variable value. In general, numerical characters must be converted into number form when calculations are to be performed.

Since alphanumeric data must be stored and processed within the computer in binary form, each character must be translated to a corresponding binary code representation as it enters the computer. The choice of code used is arbitrary. Since the computer does not

"recognize" letters, but only binary numbers, it does not matter to the computer what code is selected.

What *does* matter is consistency. Most data output, including numbers, also exits the computer in alphanumeric form, either through printed output or as output on a video screen. Therefore, the output device must perform the same conversion in reverse. It is obviously important that the input device and the output device recognize the same code. Although it would be theoretically possible to write a program to change the input code so that a different output code would result in the desired alphanumeric output, this is rarely done in practice. Since data is frequently shared between different computers in networks, the use of a code that is standardized among many different types of computers is highly desirable.

The data is also stored using the same alphanumeric code form. Consistent use of the same code is required to allow later retrieval of the data, as well as for operations using data entered into the computer at different times, such as during merge operations.

It also matters that the programs within the computer know something about the particular data code that was used as input so that conversion of the characters that make up numbers into the numbers themselves can be done correctly, and also so that such operations as sorting can be done. It would not make a lot of sense to pick a code in which the letters of the alphabet are scrambled, for example. By choosing a code in which the value of the binary number representing a character corresponds to the placement of the character within the alphabet, we can provide programs that sort data without even knowing what the data is, just by numerically sorting the codes that correspond to each character.

Three alphanumeric codes are in common use. The three codes are known as **Unicode**, **ASCII** (which stands for American Standard Code for Information Interchange, pronounced "as-key" with a soft "s"), and **EBCDIC** (Extended Binary Coded Decimal Interchange Code, pronounced "ebb-see-dick"). EBCDIC was developed by IBM. Its use is restricted mostly to older IBM and IBM-compatible maniframe computers and terminals. The Web makes EBCDIC particularly unsuitable for current work. Nearly everyone today uses Unicode or ASCII. Still, it will be many years before EBCDIC totally disappears from the landscape.

The translation table for ASCII code is shown in Figure 4.3. The EBCDIC code is somewhat less standardized; the punctuation symbols have changed over the years. A recent EBCDIC code table is shown in Figure 4.4. The codes for each symbol are given in hexadecimal, with the most significant digit across the top and the least significant digit down the side. Both ASCII and EBCDIC codes can be stored in a byte. For example, the ASCII value for "G" is 47_{16}. The EBCDIC value for "G" is $C7_{16}$. When comparing the two tables, note that the standard ASCII code was originally defined as a 7-bit code, so there are only 128 entries in the ASCII table. EBCDIC is defined as an 8-bit code. The additional special characters in both tables are used as process and communication control characters.

The ASCII code was originally developed as a standard by the American National Standards Institute (**ANSI**). ANSI also has defined 8-bit extensions to the original ASCII codes that provide various symbols, line shapes, and accented foreign letters for the additional 128 entries not shown in the figure. Together, the 8-bit code is known as Latin-1. Latin-1 is an ISO (International Standards Organization) standard.

Both ASCII and EBCDIC have limitations that reflect their origins. The 256 code values that are available in an 8-bit word limit the number of possible characters severely.

FIGURE 4.3

ASCII Code Table

LSD ↓ MSD →	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Both codes provide only the Latin alphabet, Arabic numerals, and standard punctuation characters that are used in English; Latin-1 ASCII also includes a small set of accents and other special characters that extend the set to major western European cultures. Older forms of EBCDIC omit certain characters, in particular, the "[" and "]" characters that are used to represent subscripts in the C and Java programming languages, the "^" character, used as a mathematical operator in a number of languages, "{" and "}", used to enclose code blocks in many languages, and the "∼" character, used for UNIX system commands and Internet and Internet URLs. These shortcomings led to the development of a new, mostly 16-bit, international standard, Unicode, which is quickly supplanting ASCII and EBCDIC for alphanumeric representation in most modern systems. Unicode supports approximately a million characters, using a combination of 8-bit, 16-bit and 32-bit words. The ASCII Latin-1 code set is a subset of Unicode, occupying the values 0–255 in the Unicode table, and therefore conversion from ASCII to Unicode is particularly simple: it is only necessary to extend the 8-bit code to 16 bits by setting the eight most significant bits to zero. Unicode to ASCII conversion is also simple, provided that the characters used are limited to the ASCII subset.

The most common form of Unicode, called UTF-16 can represent 65,536 characters directly, of which approximately forty-nine thousand are defined to represent the world's most used characters. An additional 6,400 16-bit codes are reserved permanently for private use. A more recent standard, Unicode 5.0, allows for multiple code pages; presently about one hundred thousand different characters have actually been defined. Unicode

FIGURE 4.4

An EBCDIC Code Table

	0	1	2	3	4	5	6	7
0	NUL	DLE	DS		space	&	-	
1	SOH	DC1	SOS		RSP		/	
2	STX	DC2	FS	SYN				
3	ETX	DC3	WU5	IR				
4	SEL	ENP	BYP/INP	PP				
5	HT	NL	LF	TRN				
6	RNL	BS	ETB	NBS				
7	DEL	POC	ESC	EOT				
8	GE	CAN	SA	SBS				
9	SPS	EM	SFE	IT				
A	RPT	UB5	SM/SW	RFF	¢	!	|	:
B	VT	CU1	CSP	CU3	.	$	,	#
C	FF	IFS	MFA	DC4	<	*	%	@
D	CR	IGS	ENQ	NAK	(	)	⌒⌒	'
E	SO	IRS	ACK		+	;	>	=
F	SI	IUS	BEL	SUB	¦	¬	?	"

	8	9	A	B	C	D	E	F
0				^	{	}	\	0
1	a	j	~		A	J	NSP	1
2	b	k	s		B	K	S	2
3	c	l	t		C	L	T	3
4	d	m	u		D	M	U	4
5	e	n	v		E	N	V	5
6	f	o	w		F	O	W	6
7	g	p	x		G	P	X	7
8	h	q	y		H	Q	Y	8
9	i	r	z		I	R	Z	9
A				[	5HY			
B				]				
C								
D								
E								
F								EO

FIGURE 4.5

Two-byte Unicode Assignment Table

Code range (in hexadecimal)	
0000–	0000–00FF Latin-I (ASCII)
1000–	General character alphabets: Latin, Cyrillic, Greek, Hebrew, Arabic, Thai, etc.
2000–	Symbols and dingbats: punctuation, math, technical, geometric shapes, etc.
3000–	3000–33FF Miscellaneous punctuations, symbols, and phonetics for Chinese, Japanese, and Korean
4000–	Unassigned
5000– • • •	4E00–9FFF Chinese, Japanese, Korean ideographs
A000–	Unassigned
B000– C000– D000–	AC00–D7AF Korean Hangui syllables
E000–	Space for surrogates
F000–	E000–F8FF Private use
FC00–	FC00–FFFF Various special characters

is multilingual in the most global sense. It defines codes for the characters of nearly every character-based alphabet of the world in modern use, as well as codes for a large set of ideographs for the Chinese, Japanese, and Korean languages, codes for a wide range of punctuation and symbols, codes for many obsolete and ancient languages, and various control characters. It supports composite characters and syllabic clusters. Composite characters are those made up of two or more different components, only one of which causes spacing to occur. For example, some vowels in Hebrew appear beneath a corresponding consonant. Syllabic clusters in certain languages are single characters, sometimes made up of composite components, that make up an entire syllable. The private space is intended for user-defined and software-specific characters, control characters, and symbols. Figure 4.5 shows the general code table layout for the common, 2-byte, form of Unicode.

Reflecting the pervasiveness of international communications, Unicode is gradually replacing ASCII as the alphanumeric code of choice for most systems and applications. Even IBM uses ASCII or Unicode on its smaller computers, and provides two-way Unicode-EBCDIC conversion tables for its mainframes. Unicode is the standard for use in current Windows and Linux operating systems. However, the vast amount of archival data in storage and use assures that ASCII and EBCDIC will continue to exist for some time to come.

Returning to the ASCII and EBCDIC tables, there are several interesting ideas to be gathered by looking at the tables together. First, note, not surprisingly, that the codes for particular alphanumeric characters are different in the two tables. This simply reemphasizes that, if we use an ASCII terminal for the input, the output will also be in ASCII form unless some translation took place within the computer. In other words, printing ASCII characters on an EBCDIC terminal would produce garbage.

More important, note that both ASCII and EBCDIC are designed so that the order of the letters is such that a simple numerical sort on the codes can be used within the computer to perform alphabetization, provided that the software converts mixed upper- and lowercase codes to one form or the other. The order of the codes in the representation table is known as its **collating sequence**. The collating sequence is of great importance in routine character processing, since much character processing centers on the sorting and selection of data.

Uppercase and lowercase letters, and letters and numbers, have different collating sequences in ASCII and EBCDIC. Therefore, a computer program designed to sort ASCII-generated characters will produce a different, and perhaps not desired, result when run with EBCDIC input. Particularly note that small letters *precede* capitals in EBCDIC, but the reverse is true in ASCII. The same situation arises for strings that are a mix of alphabetical characters and numbers. In ASCII the numbers collate first, in EBCDIC, last.

Both tables are divided into two classes of codes, specifically *printing* characters and *control* characters. Printing characters actually produce output on the screen or on a printer. Control characters are used to control the position of the output on the screen or paper, to cause some action to occur, such as ringing a bell or deleting a character, or to communicate status between the computer and an I/O device, such as the Control–"C" key combination, which is used on many computers to interrupt execution of a program. Except for position control characters, the control characters in the ASCII table are struck by holding down the Control key and striking a character. The code executed corresponds in table position to the position of the same alphabetic character. Thus, the code for SOH is generated by the Control-"A" key combination and SUB by the Control-"Z" key combination. Looking at the ASCII and EBCDIC tables can you determine what **control codes** are generated by the tab key? An explanation of each control character in the ASCII table is shown in Figure 4.6. Many of the names and descriptions of codes in this table reflect the use of these codes for data communications. There are also additional control codes in EBCDIC that are specific to IBM mainframes, but we won't define them here.

Unless the application program that is processing the text reformats or modifies the data in some way, textual data is normally stored as a string of characters, including alphanumeric characters, spaces, tabs, carriage returns, plus other control characters and escape sequences that are relevant to the text. Some application programs, particularly word processors, add their own special character sequences for formatting the text.

In Unicode, each UTF-16 alphanumeric character can be stored in two bytes, thus, half the number of bytes in a pure text file (one with no images) is a good approximation of the number of characters in the text. Similarly, the number of available bytes also defines the capacity of a device to store textual and numerical data. Only a small percentage of the storage space is needed to keep track of information about the various files; almost all the space is thus available for the text itself. Thus, a 1 GB flash drive will hold about sixty million characters (including spaces—note that spaces are also characters, of course!). If

FIGURE 4.6

Control Code Definitions [STAL96]

Code	Definition
NUL	(Null) No character; used to fill space
SOH	(Start of Heading) Indicates start of a header used during transmission
STX	(Start of Text) Indicates start of text during transmission
ETX	(End of Text) Similar to above
EOT	(End of Transmission)
ENQ	(Enquiry) A request for response from a remote station; the response is usually an identification
ACK	(Acknowledge) A character sent by a receiving device as an affirmative response to a query by a sender
BEL	(Bell) Rings a bell
BS	(Backspace)
HT	(Horizontal Tab)
LF	(Line Feed)
VT	(Vertical Tab)
FF	(Form Feed) Moves cursor to the starting position of the next page, form, or screen
CR	(Carriage return)
SO	(Shift Out) Shift to an alternative character set until SI is encountered
SI	(Shift In) see above
DLE	(Data Link Escape) Similar to escape, but used to change meaning of data control characters; used to permit sending of data characters with any bit combination
DC1,DC2, DC3, DC4	(Device Controls) Used for the control of devices or special terminal features
NAK	(Negative Acknowledgment) Opposite of ACK
SYN	(Synchronous) Used to synchronize a synchronous transmission system
STB	(End of Transmission Block) Indicates end of a block of transmitted data
CAN	(Cancel) Cancel previous data
EM	(End of Medium) Indicates the physical end of a medium such as tape
SUB	(Substitute) Substitute a character for one sent in error
ESC	(Escape) Provides extensions to the code by changing the meaning of a specified number of contiguous following characters
FS, GS, RS, US	(File, group, record, and united separators) Used in optional way by systems to provide separations within a data set
DEL	(Delete) Delete current character

you assume that a page has about fifty rows of sixty characters, then the flash drive can hold almost twenty thousand pages of text or numbers.

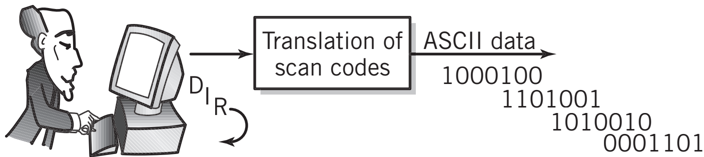
In reality, the flash drive will probably hold less because most modern word processors can combine text with graphics, page layout, font selection, and other features. And it probably has a YouTube video or two on there, as well. Graphics and video, in particular, consume a lot of disk space. Nonetheless, this book, graphics and all, fits comfortably on a single 1 GB flash drive.

Keyboard Input

Most alphanumeric data in the computer results from keyboard input, although alternative forms of data input can be used. Operation of a keyboard is quite simple and straightforward: when a key is struck on the keyboard, the circuitry in the keyboard generates a binary code, called a **scan code**. When the key is released, a different scan code is generated. There are two different scan codes for every key on the keyboard. The scan codes are converted to the appropriate Unicode, ASCII, or EBCDIC codes by software within the terminal or personal computer to which the keyboard is connected. The advantage of software conversion is

FIGURE 4.7

Keyboard Operation



that the use of the keyboard can be easily changed to correspond to different languages or keyboard layouts. The use of separate scan codes for key press and release functions allows the system to detect and process multiple key combinations, such as those used by the shift and control keys.

The keyboard operation is shown in Figure 4.7. In the figure, the user has typed the three letters "D," "I," "R," followed by the carriage return character. The computer translates the four scan codes to ASCII binary codes 1000100, 1001001, 1010010, 0001101, or their Unicode equivalents. Nonprinting characters, such as Control characters, are treated identically to Printing characters. To the computer, keyboard input is treated simply as a **stream** of text and other characters, one character after another, in the sequence typed. Note that the carriage return character is part of the stream.

The software in most computer systems echoes printable keyboard input characters directly back to the display screen, to allow the user to verify that the input has been typed correctly. Since the display circuitry and software recognizes the same character code set as the input, the characters are correctly echoed on the screen. In theory, a system could accept Unicode input from a keyboard and produce EBCDIC output to a display screen, using software to convert from one code set to the other. In practice, this is almost never done.

Alternative Sources of Alphanumeric Input

OPTICAL CHARACTER RECOGNITION Alphanumeric data may also be entered into a computer using other forms of input. One popular alternative is to scan a page of text with an image scanner and to convert the image into alphanumeric data form using **optical character recognition (OCR)** software. Early OCR software required the use of special typefaces for the scanned image and produced a lot of errors. The amount of proofreading required often nullified any advantage to using the scanner. As OCR software continues to improve, the use of scanners to read typed text directly from the page will undoubtedly increase as a source of alphanumeric data input.

A variation on OCR is also used to read specially encoded characters, such as those printed magnetically on checks. Another variation, handwriting recognition, is used to identify characters entered as input to a graphics tablet pad or the touch screen of a tablet PC, personal digital assistant, or cell phone. This technology continues to improve, but is still limited to small quantities of data, carefully printed. Attempts to extend character recognition to scanned documents and to characters written in cursory script have been largely unsuccessful to date.

BAR CODE READERS Another alternative form of data input is the bar code reader. Bar code input is practical and efficient for many business applications that require fast, accurate, and repetitive input with minimum employee training. You are probably most familiar with its use at grocery checkout counters, but many organizations use bar codes, particularly for inventory control and order filling.

FIGURE 4.8

UPC Bar Code

Bar codes represent alphanumeric data. The UPC bar code in Figure 4.8, for example, translates to the alphanumeric value 780471 108801 90000. Bar codes are read optically using a device called a wand that converts a visual scan of the code into electrical binary signals that a bar code translation module can read. The module translates the binary input into a sequence of number codes, one code per digit, that can then be input into the computer. The process is essentially similar to those already discussed. The code is usually then translated to Unicode or ASCII.

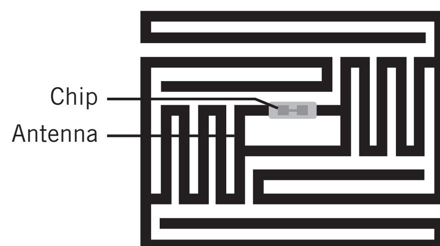
MAGNETIC STRIPE READERS Magnetic stripe readers are used to read alphanumeric data from credit cards and other similar devices. The technology used is very similar to that used for magnetic tape.

RFID INPUT **RFID** (Radio Frequency IDentification) is an inexpensive technology that can be used to store and transmit data to computers. RFID technology can be embedded in RFID tags or "smart cards" or even implanted in humans or animals. One familiar type of RFID tag is shown in Figure 4.9. An RFID tag can store anywhere from a few kilobytes to many megabytes of data. Using radio waves, RFID tags can communicate with a nearby transmitter/receiver that captures and passes the data as input to a computer for processing. Most RFID data is alphanumeric, although it is also possible with some RFID systems to provide graphical images, photographs, and even video. RFID technology is used for a wide variety of applications, including store inventory, theft prevention, library book and grocery checkout, car key verification, passport identification, cargo tracking, automobile toll and public transportation fare collection, golf ball tracking (!), animal identification, implanted human medical record storage, and much more.

VOICE INPUT It is currently possible and practical to digitize audio for use as input data. As we discuss in Section 4.4, most digitized audio data is simply stored for later output or is processed in ways that modify the sound of the data. The technology necessary to interpret audio data as voice input and to translate the data into alphanumeric form is still relatively primitive. The translation process requires the conversion of voice data into sound patterns known as **phonemes**. Each phoneme in a particular language represents one or more possible groupings of letters in that language. The groupings must then be matched and manipulated and combined to form words and sentences. Pronunciation rules, grammar rules, and a dictionary aid in the process. The understanding of sentence context is also necessary to correctly identify words such as to, too, or two. As you can see, the task is a daunting one! Progress is being made, however, and it is expected that voice input will be a major source of alphanumeric input in the foreseeable future.

FIGURE 4.9

An RFID tag used at WalMart



4.3 IMAGE DATA

Although alphanumeric data was long the traditional medium of business, improved computer technology and the growth of the Web have elevated the importance of images in the business computing environment. Photographs can be stored within the computer to provide rapid identification of employees. Drawings can be generated rapidly and accurately using tools that range from simple drawing packages to sophisticated CAD/CAM systems. Charts and graphs provide easily understood representations of business data and trends. Presentations and reports contain images and video for impact. Multimedia of all kinds is central to the success of the Web.

Images come in many different shapes, sizes, textures, colors, and shadings. Different processing requirements require different forms for image data. All these differences make it difficult to define a single universal format that can be used for images in the way that the standard alphanumeric codes are used for text. Instead, the image will be formatted according to processing, display, application, storage, communication, and user requirements.

Images used within the computer fall into two distinct categories. Different computer representations, processing techniques, and tools are used for each category:

- Images such as photographs and paintings that are characterized by continuous variations in shading, color, shape, and texture. Images within this category may be entered into the computer using an image scanner, digital camera, or video camera frame grabber. They may also be produced within the computer using a paint program. To maintain and reproduce the detail of these images, it is necessary to represent and store each individual point within the image. We will refer to such images as **bitmap images**. Sometimes, they are called **raster images** because of the way the image is displayed. (See Figure 10.16, page 324). The GIF and JPEG formats commonly used on the Web are both examples of bitmap image formats.
- Images that are made up of graphical shapes such as lines and curves that can be defined geometrically. The shapes themselves may be quite complex. Many computer experts refer to these shapes as **graphical objects**. For these images, it is sufficient to store geometrical information about each object and the relative position of each object in the image. We will refer to these images as **object images**. They are also known, somewhat incorrectly, as **vector images**, because the image is often (but not always) made up of straight-line segments called vectors. Object images are normally produced within the computer using some sort of drawing or design package. They may also result from other types of processing, for example, as data plots or graphs representing the data in a spreadsheet. More rarely, they may occur as the result of the translation by special software of scanned bitmap images that are simple enough to reduce to object form.

  Most object image formats are proprietary. However, W3C, the international consortium that oversees the Web, has defined a standard, SVG

(scalable vector graphics), based on XML Web description language tags. Macromedia Flash is also in popular use.

With only rare exceptions[1], the nature of display technology make it much more convenient and cost effective to display and print all images as bitmaps. Object images are converted to bitmap for display. Looking at an image, it can sometimes be difficult to determine whether the original form is bitmap or object. It is possible, for example, to describe subtle gradations of color within an image geometrically. The processing required to create movement in computer-animated images may dictate the use of object images, even if the objects themselves are very complex. The type of image representation is often chosen on the basis of the computer processing to be performed on the image. The movies *Shrek* and *Toy Story* are amazing examples of the possibilities of object images. (See Figure 4.14, for example.)

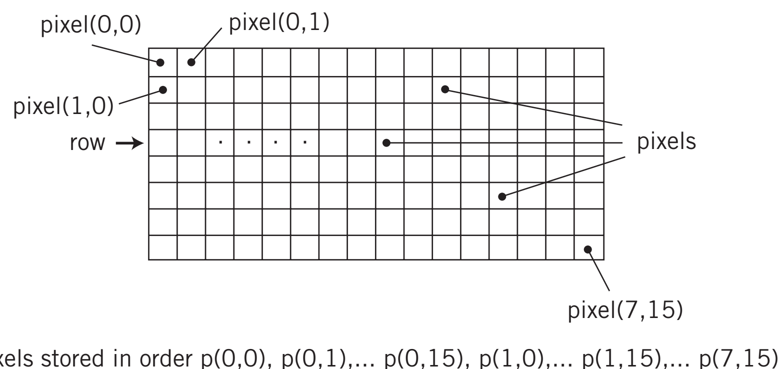
Sometimes, both types of image data occur within the same image. It is always possible to store graphical objects in a bitmap format, but it is often desirable in such cases to maintain each type of image separately. Most object image representations provide for the inclusion of bitmap images within the representation.

Bitmap Images

Most images—photographs, graphical images and the like—are described most easily using a bitmap image format. The basic principle for representing an image as a digital bitmap is simple. A rectangular image is divided into rows and columns, as shown in Figure 4.10. The junction of each row and column is a point (actually a small area) in the image known as a **pixel**, for *pi[x]*cture *el*ement. Corresponding to each pixel is a set of one or more binary numerical values that define the visual characteristics of that point.

FIGURE 4.10

A 16 × 8 Bitmap Image Format

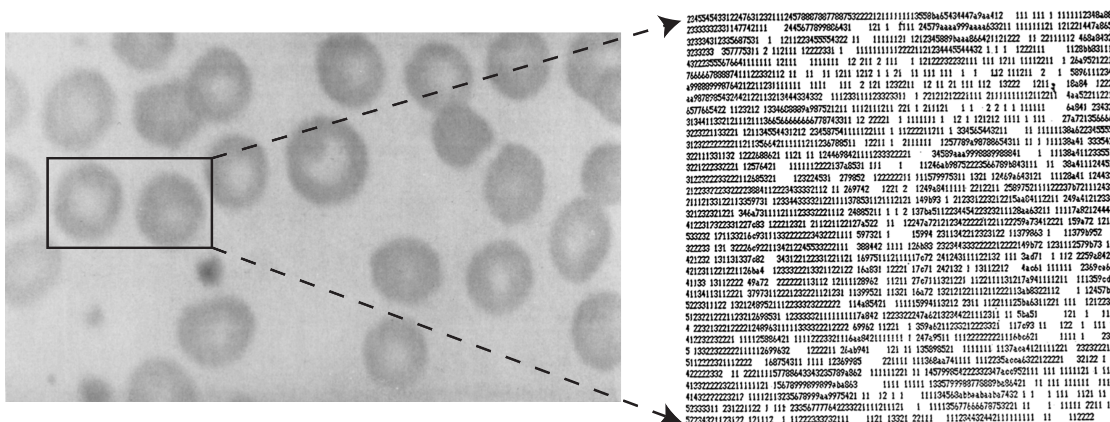


pixels stored in order p(0,0), p(0,1),... p(0,15), p(1,0),... p(1,15),... p(7,15)

[1]The exceptions are the circular scan screens used for radar display and ink plotters used for architectural and engineering drawings.

FIGURE 4.11

Image Pixel Data



Most commonly, color and color intensity are the primary characteristics of interest, but secondary characteristics such as transparency may also be present. The meaning and scales for these values are defined within the image metadata that is included with the image, along with the number of rows and columns, identification of the bitmap format in use, and other relevant information about the image.

A pixel aspect ratio may also be included so that display of the image may be adjusted if the pixel is rectangular rather than square. The specific metadata included with an image is part of the definition for a particular bitmap format.

Pixel data is normally stored from top to bottom, one row at a time, starting from pixel(0, 0), the top, leftmost pixel, to pixel($n_{row}-1$, $n_{col}-1$), representing the pixel at the bottom right corner of the image. (For quirky reasons, this image would be known as a $n_{col} \times n_{row}$ image, instead of the other way around.) Because the representation is so similar to the way in which television images are created, this layout is called a raster, and the presentation of pixels as input or output, one pixel at a time, in order, is called a raster scan. The actual pixel coordinates, pixel(row,column), do not need to be stored with their values, because the pixels are stored in order, and the number of rows and columns is known.

The actual data value representing a pixel could be as simple as one bit, for an image that is black and white (0 for black, 1 for white, for example) or quite complex. Each pixel in a high-quality color image, for example, might consist of many bytes of data: a byte for red, a byte for green, and a byte for blue, with additional bytes for other characteristics such as transparency and color correction.

As a simple example, look at the image shown in Figure 4.11. Each point in the photograph on the left is represented by a 4-bit code corresponding to one of sixteen gray levels. For this image, hexadecimal F represents black, and hexadecimal 0 represents white. The representation of the image shown on the right indicates the corresponding values for each pixel.

The storage and processing of bitmap images frequently requires a large amount of memory, and the processing of large arrays of data. A single color picture containing 768

rows of 1024 pixels each, (i.e., a 1024 × 768 image) with a separate byte to store each of three colors for each pixel, would require nearly 2.4 megabytes of storage. An alternative representation method that is useful for display purposes when the number of different colors is small reduces memory requirements by storing a code for each pixel, rather than the actual color values. The code for each pixel is translated into actual color values using a color translation table known as a **palette** that is stored as part of the image metadata. This method is discussed in Chapter 10. Data compression may also be used to reduce storage and data transmission requirements.

The image represented within the computer is really only an approximation to the original image, since the original image presents a continual range of intensity, and perhaps also of color. The faithfulness of the computer representation depends on the size of the pixels and the number of levels representing each pixel. Reducing the size of each pixel improves the **resolution**, or detail level, of the representation by increasing the number of pixels per inch used to represent a given area of the image. It also reduces the ''stepping'' effects seen on diagonal lines. Increasing the range of values available to describe each pixel increases the number of different gray levels or colors available, which improves the overall accuracy of the colors or gray tones in the image. The trade-off, of course, is in storage requirements and processing and transmission time.

Bitmap representations are particularly useful when there is a great amount of detail within an image, and for which the processing requirements are fairly simple. Typical processing on bitmap images includes storage and display, cutting and pasting of pieces of the image, and simple transformations of the image such as brightness and contrast changes, changing a dimension, or color alterations. Most bitmap image processing involves little or no direct processing of the objects illustrated within the image.

EXAMPLE

As an example of a bitmap image storage format, consider the popular **Graphics Interchange Format (GIF)**, method of storing images. GIF was first developed by CompuServe in 1987 as a proprietary format that would allow users of the online service to store and exchange bitmap images on a variety of different computing platforms. A second, more flexible, form of GIF was released in 1989. The later version, GIF89a, also allows a series of GIF images to be displayed sequentially at fixed time intervals to created ''animated GIF images.'' The GIF format is used extensively on the Web.

GIF assumes the existence of a rectangular ''screen'' upon which is located one or more rectangular images of possibly different sizes. Areas not covered with images are painted with a background color. Figure 4.12 illustrates the layout of the screen and its images. The format divides the picture information and data into a number of blocks, each of which describes different aspects of the image. The first block, called the header block, identifies the file as a GIF file and specifies the version of GIF that is being used.

Following the header block is a logical screen-descriptor block, which identifies the width and height of the screen, describes an optional color table for the images on the screen (the palette), indicates the number of bits per color available, identifies the background screen color, and specifies the pixel aspect ratio.

Each image within the screen is then stored in its own block, headed by an image-descriptor block. The image-descriptor block identifies the size and position of the image on the screen, and also allows for a palette specific to the particular image, if desired. The block also contains information that makes it possible to display individual

FIGURE 4.12

GIF Screen Layout

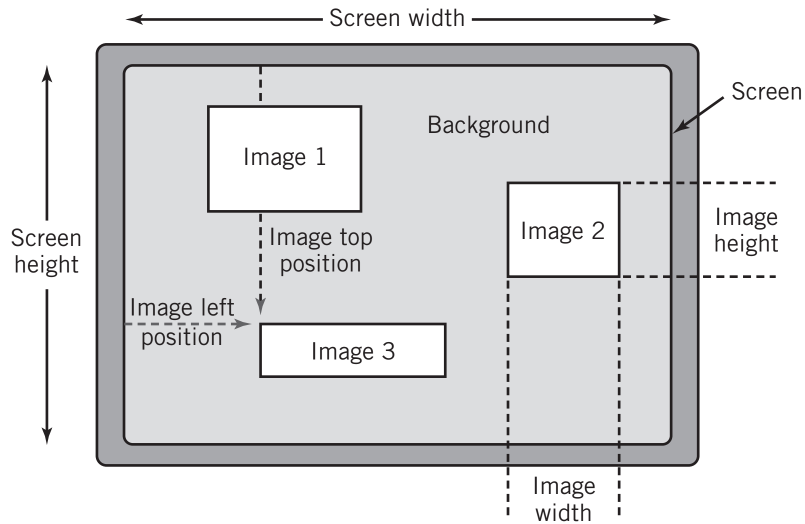
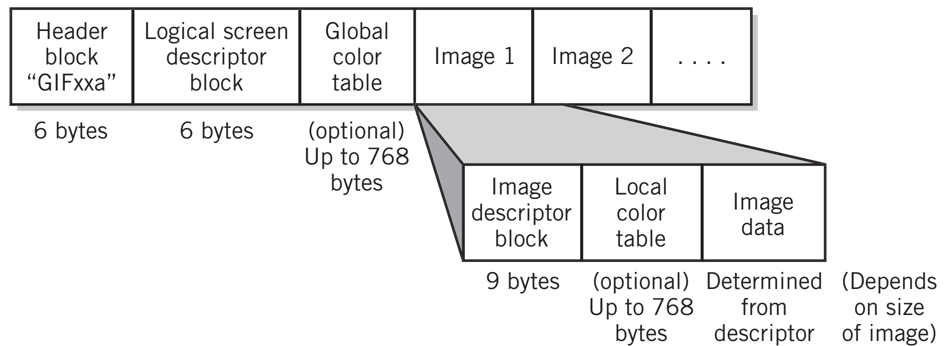


FIGURE 4.13

GIF File Format Layout



images at different resolutions. The actual pixel data for the image follows. The pixel data is compressed, using an algorithm called LZW. The basic GIF file format layout is shown in Figure 4.13.

Even though we have simplified the description, you can see that a graphical data format can be quite complex. The complexity is required to provide all the information that will allow the use of the image on a variety of different equipment.

There are a number of alternatives to the GIF format. In particular, the GIF format is limited to 256 colors, which is sometimes insufficient to display the details of a painting or

photograph, for example. A popular alternative, **JPEG format** (Joint Photographers Expert Group), addresses this concern by allowing more than sixteen million colors to be represented and displayed. JPEG employs a lossy compression algorithm to reduce the amount of data stored and transmitted, but the algorithm used reduces image resolution under certain circumstances, particularly for sharp edges and lines. This makes JPEG more suitable for the representation of highly detailed photographs and paintings, but GIF is preferable for line drawings and simple images. Other formats include TIFF, which is popular on Macintosh platforms, BMP, a Windows format, and PCX, a format originally designed for use with PC Paintbrush software. PNG is a recent format that eliminates many GIF and JPEG shortcomings. It is intended to replace GIF and JPEG for many Internet applications.

Object Images

When an image is made up of geometrically definable shapes, it can be manipulated efficiently, with great flexibility, and stored in a compact form. Although it might seem that such images are rare, this turns out not to be the case.

Object images are made up of simple elements like straight lines, curved lines (known as Bezier curves), circles and arcs of circles, ovals, and the like. Each of these elements can be defined mathematically by a small number of parameters. For example, a circle requires only three parameters, specifically, the *X* and *Y* coordinates locating the circle in the image, plus the radius of the circle. A straight line needs the *X* and *Y* coordinates of its end points, or alternatively, by its starting point, length, and direction. And so on.

Because objects are defined mathematically, they can be easily moved around, scaled, and rotated without losing their shape and identity. For example, an oval can be built from a circle simply by scaling the horizontal and vertical dimensions differently. Closed objects can be shaded and filled with patterns of color, also described mathematically. Object elements can be combined or connected together to form more complex elements, and then those elements can also be manipulated and combined. You might be surprised to learn that Shrek, the image in Figure 4.14, is an example of an object image.

FIGURE 4.14

An Object Image

Dreamworks LLC/Photofest.

Object images have many advantages over bitmap images. They require far less storage space. They can be manipulated easily, without losing their identity. Note, in contrast, that if a bitmap image is reduced in size and reenlarged, the detail of the image is permanently lost. When such a process is applied to a bitmapped straight line,

the result is "jaggies." Conversely, images such as photographs and paintings cannot be represented as object images at all and must be represented as bitmaps.

Because regular printers and display screens produce their images line by line, from the top to the bottom of the screen or paper, object images also cannot be displayed or printed directly, except on plotters. Instead, they must be converted to bitmap images for display and printing. This conversion can be performed within the computer, or may be passed on to an output device that has the capability to perform the conversion. A PostScript printer is an example of such a device. To display a line on a screen, for example, the program would calculate each of the pixels on the screen that the line passes through, and mark them for display. This is a simple calculation for a computer to perform. If the line is moved or resized, it is only necessary to perform the calculation again to display the new image.

EXAMPLE

The **PostScript page description language** is an example of a format that can be used to store, transmit, display, and print object images. A page description is a list of procedures and statements that describe each of the objects on a page. PostScript embeds page descriptions within a programming language. Thus, an image consists of a program written in the PostScript language.

The programming language is stored in ASCII or Unicode text form. Thus, PostScript files can be stored and transmitted as any other text file. An interpreter program in the computer or output device reads the PostScript language statements and uses them to create pages that can then be printed or displayed. The interpreter produces an image that is the same, regardless of the device it is displayed or printed on. Compensation for differences in device resolution and pixel shape is built into the interpreter.

PostScript provides a large library of functions that facilitate every aspect of an object-based image. There are functions that draw straight lines, Bezier curves, and arcs of a circle, functions that join simple objects into more complex ones, translate an object to a different location on the page, scale or distort an object, rotate an object, and create the mirror image of an object, and functions that fill an object with a pattern, or adjust the width and color of a line. There are methods for building and calling procedures, and IF-THEN-ELSE and loop programming structures. The list goes on and on.

A simple program that draws a pair of shaded and concentric circles within a rectangle in the middle of an 8 1/2 × 11-inch page is shown in Figure 4.15. This example shows a number of features of the language. The page is laid out as an *X*, *Y* grid, with the origin at the lower left corner. Each unit in the grid is 1/72 of an inch, which corresponds to 1 point in publishing. Each line contains a function, with a number of parameters that provide the specific details for the function. The parameters precede the function call. Text following the % symbols are comments.

The first line contains a *translate* function that moves the *X*, *Y* origin to the center of the page. The parameters for this function, 288 and 396, represent the *X* and *Y* distances moved in points. (Note that $288/72 = 4$ inches in *X* and $396/72 = 5$ inches in *Y*.) Each circle is created with an *arc* function. The parameters for the arc function are *X* origin and *Y* origin for the arc, radius, and starting and finishing angle in degrees. (0 to 360 produces a full circle.) You should be able to follow the remainder of the program on your own. Note that the statements are interpreted in sequence: the second, gray circle is layered on top of the first.

FIGURE 4.15

A PostScript Program

```
288 396 translate    % move origin to center of page
0 0 144 0 360 arc    % define 2" radius black circle
fill

0.5 setgray          % define 1" radius gray circle
0 0 72 0 360 arc
fill

0 setgray            % reset color to black
-216 -180 moveto     % start at lower left corner
0 360 rmoveto        % and define rectangle
432 0 rmoveto        % ...one line at a time
0 -360 rmoveto
closepath            % completes rectangle
stroke               % draw outline instead of fill

showpage             % produce the image
```

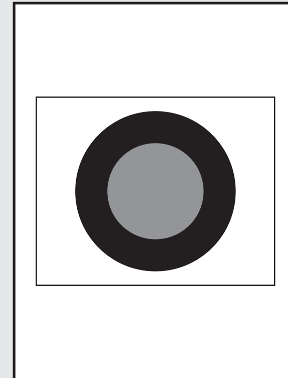


FIGURE 4.16

Another PostScript Program

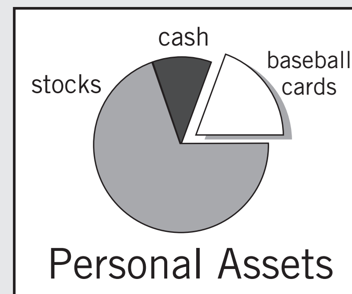
```
% procedure to draw pie slice
%arguments graylevel, start angle, finish angle
/wedge {
       0 0 moveto
       setgray
       /angle1 exch def
       /angle2 exch def
       0 0 144 angle1 angle2 arc
       0 0 lineto
       closepath } def

%set up text font for printing
/Helvetica-Bold findfont
       16 scalefont
        setfont

.4 72 108 wedge fill % 108-72 = 36 = .1 circle
.8 108 360 wedge fill % 70%
% print wedge in three parts
32 12 translate
0 0 72 wedge fill
gsave
-8 8 translate
1 0 72 wedge fill
0 setgray stroke
grestore

% add text to drawing
0 setgray
144 144 moveto
(baseball cards) show
-30 200 (cash) show
-216 108 (stocks) show
32 scalefont
(Personal Assets) show

showpage
```



Arguably, the most important feature in PostScript is the inclusion of scalable **font** support for the display of text. Font outline objects are specified in the same way as other objects. Each font contains an object for each printable character in the extended ASCII character set. PostScript includes objects for thirty-five standard fonts representing eight font families, plus two symbol fonts, and others can be added. Unicode fonts are also available. Fonts can be manipulated like other objects. Text and graphics can be intermixed in an image. The graphic display of text is considered further in the next subsection.

Figure 4.16 shows another, more complicated, example of a PostScript program. This one presents a pie chart with an expanded slice, and labels. The expanded slice includes a shadow to improve its appearance. Each slice of the pie is drawn using a procedure called *wedge*. The shadow is drawn by drawing the wedge three times, once in black, then moved a bit and drawn in white and as an outline.

PostScript is a format for storing images in object form. Nonetheless, there are occasions when it is necessary to embed a bitmap image into what is primarily an object-based image. PostScript provides this capability. It even provides the ability to crop, enlarge, shrink, translate, and rotate the embedded bitmap images, within the limits of the bitmap format, of course.

Representing Characters as Images

The representation of character-based data in a typical modern, graphically based systems presents an additional challenge. In graphically based systems it is necessary to distinguish between characters and the object image-based representations of characters, known as **glyphs**. Individual glyphs are based on a particular character in a particular font. In some cases, a glyph may also depend on neighboring characters. Should the data be represented and stored as characters or as glyphs? The answer depends on what the text is to be used for. Most text is processed and stored primarily for its content. A typical word processor, for example, stores text as character data, in Unicode format; fonts are embedded into the text file using special sequences of characters stored with the data, often in a proprietary file format supported by the particular application software. Conversion of the character data to glyphs for presentation is known as **rendering** and is performed by a **rendering engine** program. The glyphs are then converted to bitmap graphics for presentation according to the characteristics of the display device or printer. For the rare occasion where the text is actually embedded within an image, the glyphs that represent the characters may be combined, stored, and manipulated as an object image.

Video Images

Although GIF images are adequate for simple animation loops, there are a number of additional considerations for the storage, transmission, and display of true video. The most important consideration is the massive amount of data created by a video application. A video camera producing full screen 1024×768 pixel true-color images at a frame rate of thirty frames per second, for example, will generate 1024 pixels $\times$ 768 pixels $\times$ 3 bytes of color/image $\times$ 30 frames per second $=$ 70.8 megabytes of data per second! A one-minute film clip would consume 4.25 gigabytes of storage.

There are a number of possible solutions: reduce the size of the image, limit the number of colors, or reduce the frame rate. It is also possible, or, more likely, necessary to compress the video data. Each of these options has obvious drawbacks. The solution chosen also depends on the nature of the method used to make the video available to the user. One option is to present the video as a file on the system. The video file is either accessed from a removable medium, such as a DVD-ROM, or downloaded and stored on the system.

Alternatively, the video may be made available to the system in real time. The latter technique is called **streaming video**. Streaming video is normally downloaded continuously from a Web server or network server. Video conferencing is an example of a streaming video application. The requirements for streaming video are much more stringent than for locally stored video, because the amount of data that can be downloaded per unit time is limited by the capability of the network connection. Furthermore, the processor must be able to uncompress and decode the data fast enough to keep up with the incoming data stream. Generally speaking, streaming video is of lower display quality than video that is available locally.

Various mixes of these solutions are used. There are a number of proprietary formats in use, including RealPlayer from Real.com, Windows Media Format from Microsoft, and Flash Video from Macromedia. The output, although less than ideal, is adequate for many applications.

When the video data is local to the system, it is possible to generate and display high-quality video using sophisticated data compression techniques, but the processing required for generation of the compressed data is beyond the capabilities of most computer systems and users at the present time. The **MPEG-2** and **MPEG-4** formats store real-time video that produces movie quality images, with the video data compressed to 30–60 megabytes or less of data per minute, even for high definition images. Even the re-creation of the original images for display requires substantial computing power. Although high-end modern personal computer systems have adequate processing power to decode high-quality video data, many computer systems provide additional hardware support for the reading, decoding, and displaying of real-time video data from DVDs. Direct transmission of high-quality digital video data is still confined to very high-speed networks and satellite systems.

Image and Video Input

Of course the obvious input source for images and video these days is the Web. Still, those images and videos had to originate somewhere, so we'll take a brief look at the various means used to input them and convert them to the digital formats that we use. Three classes of devices provide most of the imaging capability that we use.

IMAGE SCANNING One common way to input image data is with an image scanner. Data from an image scanner takes the form of a bitmap that represents some sort of image—a graphic drawing, a photograph, magnetically inked numbers on a check, perhaps even a document of printed text. The scanner electronically moves over the image, converting the image row by row into a stream of binary numbers, each representing a pixel. Software in the computer then converts this raw data into one of the standard bitmap data formats in use, adding appropriate metadata, applying data compression when desired,

reconfiguring the data as necessary, and storing the data as a file that is then available for use.

DIGITAL CAMERAS AND VIDEO CAPTURE DEVICES Digital cameras and video cameras can be used to capture bitmap images and video. An electronic raster scan mechanism is used to collect light and digitize the data from the lens. Most modern cameras collect the data in a simple, vendor-specific so-called raw format. Because the amount of data generated for a single image in a modern camera is large (a common value is 8.1 megapixels $\times$ 3 or more bytes per pixel for example), the camera usually contains software to convert the raw data to a compressed JPEG or MPEG image for storage and transfer to a computer. Some cameras allow the direct transfer of raw images to a computer for more precise processing control.

GRAPHICAL INPUT USING POINTING DEVICES Mice, pens, and other pointing devices can be used in conjunction with drawing or painting programs to input graphical data. The input from most of these devices is a pair of binary numbers representing either *X* and *Y* coordinates on the screen or relative movements in *X* and *Y* directions. Some drawing tablets also provide a measure of the pressure applied to the drawing pen. The pointing device is an input device. The appearance of a cursor on the output screen results from a calculation within the program that detects the current set of coordinates. The program then outputs a cursor as part of the screen image bitmap at the appropriate location on the screen. Internally, the image drawn will depend on the application program being used. Paint packages provide tools that use the pointing device to create "paintings" in a bitmap image form. Drawing packages provide tools that create and manipulate objects. In this case, the result is an object image.

4.4 AUDIO DATA

Sound has become an important component in modern computer applications. Sound is used as an instructional tool, as an element of multimedia presentations, to signal events within the computer, and to enhance the enjoyment of games. Sound can be stored in digital form on CD-ROMs and other media and made available to accompany a film clip, illustrate the nuances of a symphony, or reproduce the roar of a lion. Sound can be manipulated in the computer to compose music and to create the sounds of different musical instruments, even an entire orchestra.

Sound is normally digitized from an audio source, such as a microphone or amplifier, although it is possible to purchase instrumentation that connects the computer directly to a musical keyboard and synthesizer. For most users, the sound was previously digitized and provided on a CD-ROM or downloaded from a Web site.

Since the original sound wave is analog in nature, it is necessary to convert it to digital form for use in the computer. The technique used is the same as that used for music CDs and many other types of analog waveforms. The analog waveform is sampled electronically at regular time intervals. Each time a sample is taken, the amplitude of the sample is measured by an electronic circuit that converts the analog value to a binary equivalent. The circuit that performs this function is known as an **A-to-D converter**. The largest possible sample, which represents the positive peak of the loudest possible sound, is set to the maximum positive binary number being used, and the most negative peak is set

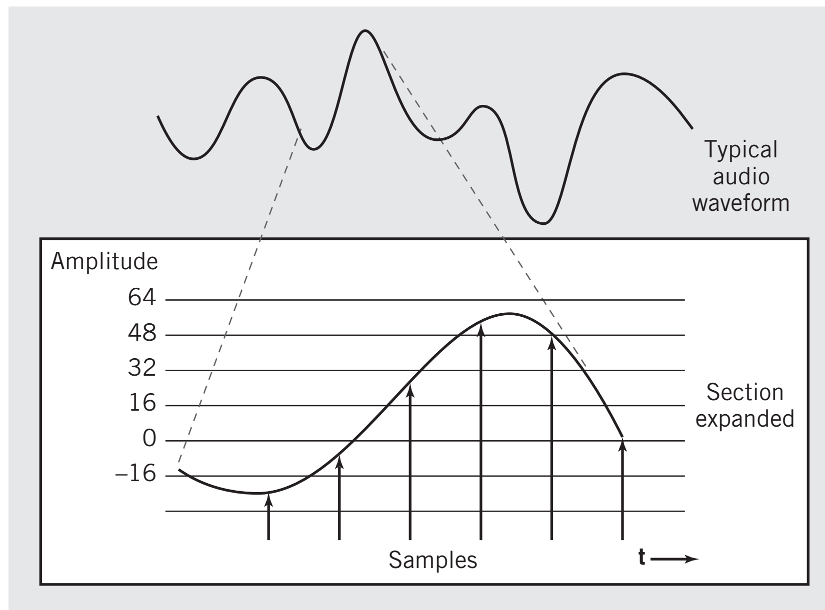
to the largest negative number. Binary 0 falls in the middle. The amplitude scale is divided uniformly between the two limits. The sampling rate is chosen to be high enough to capture every nuance in the signal being converted. For audio signals, the sampling rate is normally around 50 kilohertz, or fifty thousand times a second. The basic technique is illustrated in Figure 4.17. A typical audio signal is shown in the upper diagram. A portion of the signal is shown in expanded form below. In this diagram, the signal is allowed to fall between −64 and 64. Although we haven't discussed the representation of negative numbers yet, the consecutive values for the signal in this diagram will be the binary equivalents to −22, −7, +26, 52, 49, and 2. The A-to-D conversion method is discussed more thoroughly in Chapter 14.

Within the computer, most programs would probably treat this data as a one-dimensional array of integers. Like graphics images, however, it is necessary to maintain, store, and transmit metadata *about* the waveform, in addition to the waveform itself. To process and reproduce the waveform, a program would have to know the maximum possible amplitude, the sampling rate, and the total number of samples, at the very least. If several waveforms are stored together, the system would have to identify each individual waveform somehow and establish the relationships between the different waveforms. Are the waveforms played together, for example, or one right after another?

As you might expect, there are a number of different file formats for storing audio waveforms, each with its own features, advantages, and disadvantages. The *.MOD* format, for example, is used primarily to store samples of sound that will be manipulated and combined to produce a new sound. A .MOD file might store a sample of a piano tone.

FIGURE 4.17

Digitizing an Audio Waveform



Software could then manipulate the sample to reproduce all the different keys on the keyboard, it could alter the loudness of each tone, and it could combine them to synthesize the piano lines in a piece of music. Other instruments could be synthesized similarly. The *MIDI* format is used to coordinate the sounds and signals between a computer and connected musical instruments, particularly keyboards. MIDI software can "read" the keyboard and can also reproduce the sounds. The *.VOC* format is a general sound format that includes special features such as markers within the file that can be used to repeat (loop) a block or synchronize the different components of a multimedia presentation. Block looping can extend a sound by repeating it over and over again. The *.WAV* format is a general-purpose format used primarily to store and reproduce snippets of sound. *MP3* is a derivative of the MPEG-2 specification for the transmission and storage of music. It has gained popularity because of the large numbers of MP3-coded recordings posted on the Web and because of the availability of low-cost portable devices that can download, store, decode, and reproduce MP3 data.

Like video, audio data can also be generated and stored locally or streamed from a network or website. The data transmission and processing requirements for audio are much less stringent than those for video, however. Audio is routinely streamed from the Web. There are numerous websites broadcasting audio from radio stations and other sources, and streaming audio is also used for Internet telephony.

EXAMPLE

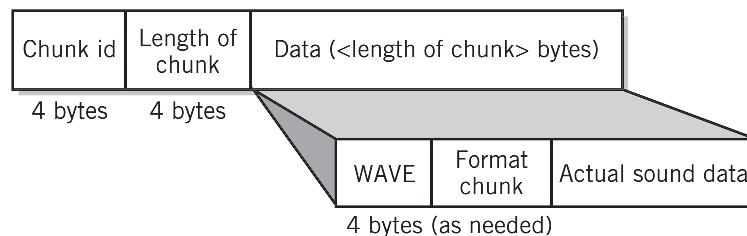
The .WAV format was designed by Microsoft as part of its multimedia specification. The format supports 8- or 16-bit sound samples, sampled at 11.025 KHz, 22.05 KHz, or 44.1 KHz in mono or stereo. The .WAV format is very simple and does not provide support for a lot of features, such as the looping of sound blocks .WAV data is not compressed.

The format consists of a general header that identifies a "chunk" of data and specifies the length of a data block within the chunk. The header is followed by the data block. The general header is used for a number of different multimedia data types.

The layout of a .WAV file is shown in Figure 4.18. The data block is itself broken into three parts. First, a 4-byte header identifies a sound file with the ASCII word "WAVE." A format chunk follows. This chunk contains such information as the method used to digitize the sound, the sampling rate in samples per second, the data transfer rate in average number of bytes per second, the number of bits per sample, and whether the sound is recorded in mono or stereo. The actual data follows.

FIGURE 4.18

.WAV Sound Format



If you have a personal computer that runs Windows and supports sound, you will probably find .WAV files in one of your Windows directories. Look for the file *tada.wav*, which holds the brief trumpet fanfare that sounds when Windows is started.

EXAMPLE

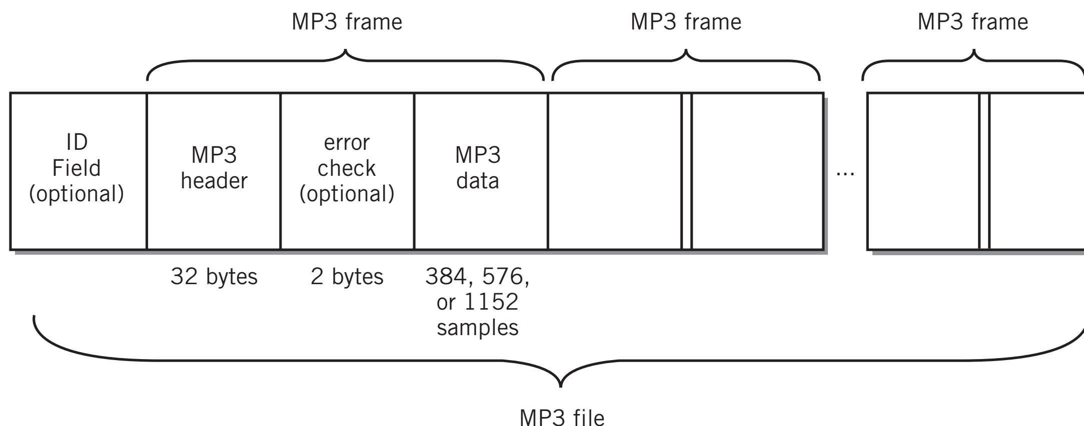
MP3 is the predominant digital audio data format for the storage and transmission of music. It is characterized by reasonable audio quality and small file size. MP3 uses a number of different tactics and options to achieve its small file sizes. These include options for different audio sampling rates, fixed or variable bit rates, and a wide range of bit rates that represent different levels of compression. The bit rate, measured in Kbits/second is, of course, directly related to the size of the file, however lower bit rates result in lower audio quality. The options chosen are made by the creator of the file during the encoding process, based on the trade-off between tolerable audio quality versus transmission rate or file size. An MP3 player must be capable of correctly decoding and playing any of the format variations specified in the MP3 standard.

The primary contributor to the small MP3 file size is the use of psychoacoustic lossy compression. The size of an MP3 file is typically about 1/10th the size of an equivalent uncompressed .WAV file. Psychoacoustic compression is based on the assumption that there are sounds that a listener cannot hear or will not notice, which can then be eliminated. As an example, a soft sound in the background is not usually noticeable against a loud foreground sound. The level of compression depends on the tolerable level of sound quality, but also on the nature of the audio being compressed. A typical MP3 file samples the audio data 44,100 times per second, which is the same as the data rate used on audio CDs, and presents the data to the listener at a rate of either 128 or 192 Kb/second.

Figure 4.19 shows the structure of an MP3 file. The file consists of an optional ID field that contains such information as song title and artist, followed by multiple data frames. Each frame has a 32-byte header that describes the frame data, followed by an optional

FIGURE 4.19

MP3 Audio Data Format



2-byte error-checking code, followed by the data itself. The header contains 2 bytes of synchronization and MP3 version data followed by the bit rate, the audio sample rate, the type of data (for example, stereo or monaural audio), copy protection, and other information. The MP3 standard requires that each frame contains 384, 576, or 1152 audio samples of data. Note that this format allows the bit rate to vary for each frame, allowing for more efficient compression, but more difficult encoding procedures.

4.5 DATA COMPRESSION

The volume of multimedia data, particularly video, but also sound and even high resolution still images, often makes it impossible or impractical to store, transmit, and manipulate the data in its normal form. Instead it is desirable or, in some cases, necessary to compress the data. This is particularly true for video clips, real-time streaming video with sound, lengthy sound clips, and images that are to be transmitted across the Internet through modem connections. (It is also true of large data and program files of any type.)

There are many different data compression algorithms, but all fall into one of two categories, **lossless** or **lossy**. A lossless algorithm compresses the data in such a way that the application of a matching inverse algorithm restores the compressed data exactly to its original form. Lossy data compression algorithms operate on the assumption that the user can accept a certain amount of data degradation as a trade-off for the savings in a critical resource such as storage requirements or data transmission time. Of course, only lossless data compression is acceptable for files where the original data must be retained, including text files, program files and numerical data files, but lossy data compression is frequently acceptable in multimedia applications. In most applications, lossy data compression ratios far exceed those possible with lossless compression.

Lossless data algorithms work by attempting to eliminate redundancies in the data. For example, suppose that you have the following string of data:

0 5 5 7 3 2 0 0 0 0 1 4 7 3 2 9 1 0 0 0 0 0 6 6 8 2 7 3 2 7 3 2 . . .

There are two simple steps you could take to reduce this string. First, you could reduce the amount of data by counting the strings of consecutive 0s, and maintaining the count instead of the string. The character is reproduced once, followed by its count:

0 1 5 5 7 3 2 0 4 1 4 7 3 2 9 1 0 5 6 6 8 2 7 3 2 7 3 2 . . .

Notice that we actually had to add a character when the 0 appeared singly in the string. Otherwise, the inverse algorithm would have assumed that the first 0 appeared five times rather than recognizing the data to be a single 0 followed by a 5.

As a second step, the algorithm attempts to identify larger sequences within the string. These can be replaced with a single, identifiable value. In the example string, the sequence "7 3 2" occurs repeatedly. Let us replace each instance of the sequence with the special character "Z":

0 1 5 5 Z 0 3 1 4 Z 9 1 0 5 6 6 8 2 Z Z . . .

Application of these two steps has reduced the sample string by more than 35 percent. A separate attachment to the data would identify the replacements that were made, so

that the original data can be restored losslessly. For the example, the attachment would indicate that 0s were replaced by a single 0 followed by their count and the sequences "7 3 2" were replaced by "Z." You might wish to restore the original string in this example for practice.

There are many variations on the methods shown in the example. You should also notice that the second step requires advance access to the entire sequence of data to identify the repetitive sequences. Thus, it is not useful with streaming data. There are other variations that are based on the known properties of the data stream that can be used, however. For example, MPEG-2 uses the knowledge that the image is repeated at a frame rate of, say, thirty times per second, and that in most instances, very little movement occurs within small parts of the image between consecutive frames. GIF images and ZIP files are compressed losslessly.

Lossy algorithms operate on the assumption that some data can be sacrificed without significant effect, based on the application and on known properties of human perception. For example, it is known that subtle color changes will not be noticeable in the area of an image where the texture is particularly vivid. Therefore, it is acceptable to simplify the color data in this circumstance. There is no attempt to recover the lost data. The amount of data reduction possible in a particular circumstance is determined experimentally. Lossy algorithms can often reduce the amount of data by a factor of 10:1 or more. JPEG and MP3 are examples of lossy algorithms.

MPEG-2 uses both variations on both forms of compression simultaneously to achieve compression ratios of 100:1 or more with very little noticeable degradation in image quality; however, the compression process itself requires tremendous amounts of computing power.

Ongoing advances in data compression technology have resulted in improved performance with existing compression techniques, as well as a number of new video formats, including new versions of Microsoft's Windows Media Video format and new formats based on the MPEG-4 standard, as well as a new JPEG bitmap format, JP-2. As of this writing, these formats have not yet achieved general acceptance, but there is much promise for greatly improved video performance.

In general, the use of data compression is a trade-off between the use of processing power and the need to reduce the amount of data for transmission and storage. In most cases, the higher the compression ratio, the greater the demand upon the computer processing resources. At some point, the incremental improvement in compression to be achieved will no longer justify the additional cost in processing or the degradation of the result.

4.6 PAGE DESCRIPTION LANGUAGES

A **page description language** is a language that describes the layout of objects on a displayed or printed page. (In this context we are using the word "object" in the more general object-oriented programming language sense, rather than as a specific reference to object images.) Page description languages incorporate various types of objects in various data formats, including, usually, text, object images, and bitmap images. The page description language provides a means to position the various items on the page. Most page description languages also provide the capability to extend the language to include

new data formats and new objects using language stubs called **plug-ins**. Most audio and video extensions fall into this category.

Some page description languages are extremely simple, with limited innate functionality. **HTML** (HyperText Markup Language), for example, provides little more than a shell. Except for text, most objects are stored in separate files, the details of layout are left mostly to the Web browser that is recreating the page, and programming language capability and other features are provided as extensions. We have already shown you many of the data formats that are used with HTML. Others, such as **PDF** (Portable Document Format) and PostScript offer the ability to recreate sophisticated pages with surprising faithfulness to the intentions of the original page designer.

PDF, for example, incorporates its own bitmap formats, object image format, and text format, all optimized for rapid page creation and presentation. It is often difficult to extract data in their original data formats from a PDF file. Interestingly, PDF does not provide programming language features. Instead, PDF is treated as a file format. The file contains objects, along with page positioning information for each object, and that's about it. It is presumed that any program execution required to preprocess the objects in the file for presentation was done prior to the creation of the file.

PostScript, on the other hand, contains a full-featured programming language that can be processed at display time. In that sense, PDF is something of a subset of PostScript, though with somewhat different goals and strengths. Many of the features of PDF are derived from postprocessed PostScript. In particular, the object image descriptions in PDF are based on the PostScript formats shown as examples earlier in this chapter.

4.7 INTERNAL COMPUTER DATA FORMAT

So now you have an idea of the various forms that data takes when it reaches the computer. Once inside the computer, however, *all* data is simply stored as binary numbers of various sizes, ranging from 1 to 8 bits, or even larger. The interpretation of these binary numbers depends upon two factors:

- The actual operations that the computer processor is capable of performing
- The data types that are supported by the programming language used to create the application program

As you will see in later chapters, computer processors provide instructions to manipulate data, for searching and sorting, for example, and to manipulate and perform basic mathematical operations on signed and unsigned integers. They also provide a means to point to data, using a stored binary value as a pointer or locator to another stored binary number. Since these pointer values are themselves stored as numbers, they can also be manipulated and used in calculations. A pointer value might represent the index in an array, for example. Most recent computers also provide instructions for the direct manipulation of floating point, or real, numbers. In other computers, floating point numbers are manipulated using software procedures.

The processor instruction set also establishes formats for each data type that it supports. If a number in the computer is supposed to be a floating point number, for

example, the instructions are designed to assume that the number is laid out in a particular format. Specific formats that are used for integer and real numbers are discussed in Chapter 5.

Thus, the raw binary numbers stored in a computer can easily be interpreted to represent data of a variety of different types and formats. C, Java, Visual Basic, and other languages all provide a programmer with the capability to identify binary data with a particular data type. Typically, there are five different simple data types:

- *Boolean*: two-valued variables or constants with values of true or false.
- *char*: the character data type. Each variable or constant holds a single alphanumeric character code representing, for example, the single strike of a key. It is also common to process groups of characters together as strings. Strings are simply arrays of individual characters. The ASC function in Visual Basic shows the actual binary number code representing a particular character. Thus, ASC("A") would show a different value on an ASCII-based system from that shown on an EBCDIC system.
- *enumerated* data types: user-defined simple data types, in which each possible value is listed in the definition, for example,

  ```
  type DayOfWeek = Mon, Tues, Wed, Thurs, Fri, Sat
  ```

- *integer*: positive or negative whole numbers. The string of characters representing a number is converted internally by a conversion routine built into the program by the compiler and stored and manipulated as a numerical value.
- *real* or *float*: numbers with a decimal portion, or numbers whose magnitude, either small or large, exceeds the capability of the computer to process and store as an integer. Again, the routine to convert a string of characters into a real number is built into the program.

In addition to the simple data types, many programming languages, including C, but not Java, support an explicit pointer variable data type. The value stored in a *pointer variable* is a memory address within the computer. Other, more complex, data types, structures, arrays, records, and other objects, for example, are made up of combinations of the simple data types.

The data types just listed correlate rather well with the instruction set capability of the processor. The integer and real types can be processed directly. The character type is translated into instructions that manipulate the data for basic character operations that are familiar to you from your programming classes. Boolean and enumerated data types are treated within the computer in a manner similar to integers. Most programming languages do not accept Boolean and enumerated data as input, but the conversion would be relatively straightforward. It would only be necessary to test the input character string against the various possibilities, and then set the value to the correct choice (see Exercise 4.10).

Other languages may support a completely different set of data types. There are even some languages that don't recognize any data types explicitly at all, but simply treat data in a way appropriate to the operation being performed.

Numerical Character to Integer Conversion

EXAMPLE

As you've already seen, the typical high-level language numerical input statement

READ(*value*)

where *value* is the name of an integer variable, requires a software conversion from the actual input, which is alphanumeric, to the numerical form specified for value. This conversion is normally provided by program code contributed by the language compiler that becomes part of your program. Some programmers choose instead to accept the input data in character form and include their own code to convert the data to numerical form. This allows more programmer control over the process; for example, the programmer might choose to provide more extensive error checking and recovery than that of the internal conversion program. (Many internal conversion programs simply crash if the user inputs an illegal character, say, a letter when a numeral is expected.)

Whether internal or programmer supplied, the conversion process is similar. Just to deepen your understanding of the conversion process, Figure 4.20 contains a simple

FIGURE 4.20

A Pseudocode Procedure that Performs String Conversion

```
//variables used
char key;
int number = 0;
boolean error, stop;
{
   stop = false;
   error = false;
   ReadAKey;
   while (NOT stop && NOT error) {
      number = 10 * number + (ASCIIVALUE(key) - 48);
      ReadAKey;
   } //end while
   if (error == true) {
      printout('Illegal Character in Input');
   else printout('input number is ' number);
   } //end if
} //end procedure

function ReadAKey(); {
   read(key);
   if (ASCIIVALUE(key) == 13 or ASCIIVALUE(key) == 32 or ASCIIVALUE(key) == 44)
      stop = true;
   else if ((key < '0' ) or (key > '9' )) error = true;
} //end function ReadAKey
```

pseudocode procedure that converts the string representing an unsigned integer into numerical form. This code contains simple error checking and assumes that the number ends with a space (ASCII 32), a comma (ASCII 44), or a carriage return (ASCII 13).

Conversion procedures for other data types are similar.

SUMMARY AND REVIEW

Alphanumeric data inputs and outputs are represented as codes, one code for each data value. Three commonly used code systems for interactive input and output are Unicode, ASCII, and EBCDIC. Within these codes, each character is represented by a binary number, usually stored 1 or 2 bytes per character.

The design and choice of a code is arbitrary; however, it is useful to have a code in which the collating sequence is consistent with search and sort operations in the language represented. Within the computer, programs must be aware of the code used to assure that data sorts, number conversions, and other types of character manipulation are handled correctly. There must also be agreement between input and output devices, so that the data is displayed correctly. If necessary, translation programs can be used to translate from one representation to another. When necessary, conversion programs within the computer convert the alphanumeric character strings into other numeric forms. Numeric data must be converted back to Unicode, ASCII, or EBCDIC form for output display, however. The most common source of alphanumeric data is the keyboard.

Data from a keyboard enters the computer in the form of a character stream, which includes nonprinting characters as well as printing characters. Image scanning with optical character recognition, voice input, and various special devices, such as bar code readers, can also be used to create alphanumeric data.

There are two different methods used for representing images in the computer. Bitmap images consist of an array of pixel values. Each pixel represents the sampling of a small area in the picture. Object images are made up of simple geometrical elements. Each element is specified by its geometric parameters, its location in the picture, and other details.

Within the constraint that object images must be constructed geometrically, they are more efficient in storage and more flexible for processing. They may be scaled, rotated, and otherwise manipulated without loss of shape or detail. Images with texture and shading, such as photographs and painting, must be stored in bitmap image form. Generally, images must be printed and displayed as bitmaps, so object images are converted to bitmap form by a page description language interpreter before printing or display. There are many different formats used for storing graphical images.

Video images are difficult to manage because of the massive amounts of data involved. Video may be stored local to the system, or may be streamed from a network or website. The quality of streamed video is limited by the capability of the network connection. Higher quality is possible with locally stored video data, but the processing requirements are demanding. Some systems provide auxiliary hardware to process video.

Audio signals are represented in the computer by a sequence of values created by digitizing the signal. The signal is sampled at regular time intervals. Each sample is then converted to an equivalent binary value that is proportional to the amplitude of the sample. Again, different formats are available for storing audio data, depending on the application.

Audio signals may be streamed or stored locally. The requirements for audio transmission and processing are far less stringent than for those of video.

For images, both still and video, as well as audio, data compression is often appropriate. Lossless data compression allows complete recovery of the original noncompressed data. Lossy data compression does not allow recovery of the original data, but is designed to be perceived as sufficient by the user.

Page description languages combine the characteristics of various specific data formats together with data indicating the position on the page to create data formats that can be used for display and printing layouts.

Internally, all data, regardless of use, are stored as binary numbers. Instructions in the computer support interpretation of these numbers as characters, integers, pointers, and in many cases, floating point numbers.

FOR FURTHER READING

The general concepts of data formats are fairly straightforward, but additional character-based exercises and practice can be found in the Schaum outline [LIPS82]. Individual codes can be found in many references. The actual characters mapped to the keyboard are directly observable using the *Character Map* accessory in Windows or the *Key Caps* desk accessory on the Macintosh. Extensive information about Unicode is available from the Unicode website at www.unicode.org.

For graphics formats, there are a number of good general books on graphics. Most of these books describe the difference between bitmap and object graphics clearly, and most also discuss some of the different graphics file formats, and the trade-offs between them. Additionally, there are more specialized books that are often useful in this area. Murray and Van Ryper [MURR96] provide a detailed catalog of graphics formats. Rimmer [RIMM93] discusses bitmapped graphics at length.

Smith [SMIT90] presents an easy approach to the PostScript language. The three Adobe books—[ADOB93], [ADOB99], and [ADOB85], often called the "green book", the "red book", and the "blue book", respectively—are detailed but clear explanations of PostScript. Adobe also offers the PDF Reference [ADOB06]. A simple introduction to PDF is the PDF Primer White Paper [PDFP05].

There are many books on various aspects of digital sound, but most are hard to read; the Web is a better resource. Similarly, new data formats of all types occur as the need arises. Because the need seems to arise continuously nowadays, your best source of current information is undoubtedly the Web.

KEY CONCEPTS AND TERMS

A-to-D converter
alphanumeric data
ANSI
ASCII
bitmap or raster image
collating sequence
control code
EBCDIC
font
glyph
Graphics Interchange Format (GIF)
graphical objects
JPEG format
lossless data compression
lossy data compression
metadata
MP3
MPEG-2, MPEG-4
numeric character versus number
object or vector image
optical character recognition (OCR)
page description language
palette
phoneme
pixel
PostScript language
plug-ins
proprietary format
resolution
RFID (radio frequency identification)
scan code
stream, character
streaming (video)
Unicode

READING REVIEW QUESTIONS

4.1 When data is input to a computer, it is nearly always manipulated and stored in some standard data format. Why is the use of data standards considered important, or indeed, crucial in this case?

4.2 Name the three standards in common use for alphanumeric characters. Which standard is designed to support all of the world's written languages? Which language is used primarily with legacy programs that execute on mainframe computers?

4.3 What is the relationship between the ASCII Latin-1 character set and its Unicode equivalent that makes conversion between the two simple?

4.4 What is a collating sequence?

4.5 Name at least four alternative devices that can be used as sources of alphanumeric character input data.

4.6 What are the major characteristics of a *bitmap* image? What are the major characteristics of an *object* or *vector* image? Which is used for displays? What types of images must be stored and manipulated as bitmap images? Why?

4.7 What is image *metadata*? Give an at least three examples of metadata that would be required for a bitmap image.

4.8 Name two advantages to the use of object images.

4.9 Explain briefly how an A-to-D converter converts audio data into binary data.

4.10 Describe briefly the most important characteristics and features of an MP3 audio file.

4.11 Explain the difference between lossless and lossy data compression. Which type normally provides a smaller file? What is "lost" in lossy audio data compression? Under what circumstances is it impossible to use lossy data compression?

4.12 What is a *page description language*? Give an example of a page description language.

4.13 Name five simple data types that are provided in most high-level programming languages.

4.14 Explain the difference between numeric characters and numbers. Under what conditions would you expect the computer to use numeric characters? When would you expect the computer to use numbers? When numeric data is entered at the keyboard, which form is used? Which form is used for calculations? Which form is used for display?

EXERCISES

4.1 **a.** Create a table that shows the ASCII and EBCDIC representations side-by-side for each of the uppercase letters, lowercase letters, and numerals.

b. Does the hexadecimal representation show you a simple method for converting individual numeric characters into their corresponding numerical values?

c. Does the hexadecimal representation suggest a simple method for changing lowercase letters into their corresponding capital letters?

d. Can you use the same methods for EBCDIC as you do for ASCII? If so, what changes would you need to make in a program to make (b) and (c) work?

4.2 **a.** What is the ASCII representation for the numeral −3.1415 in binary? in octal? in hexadecimal? in decimal?

b. What is the EBCDIC representation for the numeral +1,250.1? (Include the comma.)

4.3 What character string does the binary ASCII code

```
1010100 1101000 1101001 1110011 0100000 1101001 1110011
0100000 1000101 1000001 1010011 1011001 0100001
```

represent?

4.4 ASCII, Unicode, and EBCDIC are, of course, not the only possible codes. The Sophomites from the planet Collegium use the rather strange code shown in Figure E4.1. There are only thirteen characters in the Sophomite alphabet, and each character uses a 5-bit code. In addition, there are four numeric digits, since the Sophomites use base 4 for their arithmetic.

a. Given the following binary code, what is the message being sent by the Sophomites?

```
1100111010000011111100000010011011111111011111100000000100100
```

b. You noticed in part (a) that this code does not delimit between characters. How does one delimit this code? Suppose a bit was dropped during transmission. What happens? Suppose a single bit was altered (0 to 1 or 1 to 0). What happens?

FIGURE E4.1

Symbol	Code	Symbol	Code	Symbol	Code
≈	00001	⁘	10000	←	11111000
↗	00010	■	10011	↑	11111011
★	00100	▯	10101	→	11111101
+	01000	⋈	10110	↓	11111110
.I.	01011	x	11001		
ಲ	01101	✶	11010		
●	01110				

4.5 As an alternative alphanumeric code, consider a code where punched holes in the columns of a card represent alphanumeric codes. The punched hole represents a "1"; all other bits are "0". The Hollerith code shown in Figure E4.2 is an example of such a code. This code has been used to represent a message on the card in Figure E4.3. Each row represents a code level from 0 to 12. Levels 12 and 11, which are not labeled on the card, are the top row and next-to-top rows, respectively. Each column represents a single character, so the card can hold one eighty-column line of text. (This card, prevalent in the 1960s and 1970s as a means of data input, is the reason that text-based displays are still limited to eighty characters per line.) Translate the card in Figure E4.3.

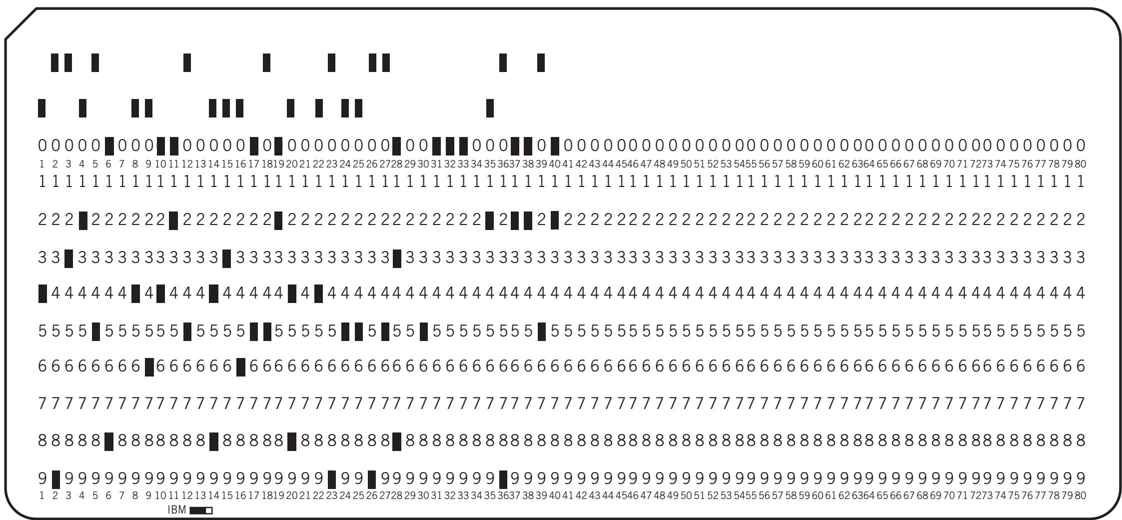
4.6 Without writing a program, predict the ORD value for your computer system for the letter "A," for the letter "B," for the letter "C." How did you know? Might the value be different on a different system? Why or why not?

4.7 Write a program in your favorite language that will convert all ASCII uppercase and lowercase letters to EBCDIC code. For an additional challenge, also convert the punctuation symbols, indicating with a failure-to-convert message, those symbols that are not represented in the EBCDIC system.

FIGURE E4.2

Character	Punched code	Character	Punched code	Character	Punched code	Character	Punched code	Character	Punched code
A	12,1	L	11,3	W	0,6	7	7	<	12,8,4
B	12,2	M	11,4	X	0,7	8	8	(	12,8,5
C	12,3	N	11,5	Y	0,8	9	9	+	12,8,6
D	12,4	O	11,6	Z	0,9	&	12	$	11,8,3
E	12,5	P	11,7	0	0	-	11	*	11,8,4
F	12,6	Q	11,8	1	1	/	0,1	)	11,8,5
G	12,7	R	11,9	2	2	#	8,3	,	0,8,3
H	12,8	S	0,2	3	3	@	8,4	%	0,8,4
I	12,9	T	0,3	4	4	‘	8,5	blank	none
J	11,1	U	0,4	5	5	=	8,6		
K	11,2	V	0,5	6	6	.	12,8,3		

FIGURE E4.3



4.8 If you have access to a debug program, load a text file into computer memory from your disk, and read the text from computer memory by translating the ASCII codes.

4.9 Suppose you have a program that reads an integer, followed by a character, using the following prompt and READ statement:

```
WRITE('Enter an integer and a character:')
READ(intval, charval);
```

When you run the program, you type in the following, in response to the prompt

```
Enter an integer and a character:
1257
z
```

When you check the value of charval, you discover that it does *not* contain "z." Why not? What would you expect to find there?

4.10 Write a program that accepts one of the seven values "MON," "TUE," "WED," "THU," "FRI," "SAT," and "SUN" as input and sets a variable named TODAY to the correct value of type DayOfWeek, and then outputs the ORD value of TODAY to the screen. (Does the ORD value give you a hint as to the internal representation of the enumerated data type?)

4.11 Write a procedure similar to procedure Convert that converts a signed integer to a character string for output.

4.12 Approximately how many pages of pure 16-bit Unicode text can a 650 MB CD-ROM hold?

4.13 Find a book or article that describes the various bitmapped graphics formats, and compare .GIF, .PNG, and .BMP.

4.14 Find a book or article that describes the various bitmapped graphics formats, and compare .GIF and .RLE.

For Exercises 4.13 and 4.14, there are several books that describe graphics formats in detail. One of these is Murray [MURR96].

4.15 Investigate several audio formats, and discuss the different features that each provides. Also discuss how the choice of features provided in different formats affects the type of processing that the format would be useful for.

4.16 If you have studied COBOL, discuss the difference between numeric characters and numbers in the context of a COBOL program. Does COBOL distinguish clearly between the two? If so, in what ways?

4.17 Provide a line-by-line explanation for the PostScript code in Figure 4.14.

4.18 Unicode is downward compatible with the Latin-1 version of 8-bit ASCII in the sense that a Unicode text file that is limited to the Latin-1 character set will be read correctly on a system that does not support Unicode, provided that an end delimiter is used, rather than a character count as the measure of the length of the message. Why is this so? (Hint: Consider the role of the ASCII NUL character.)

4.19 Use the Web as a resource to investigate MPEG-2 [or MPEG-4]. Explain the data compression algorithm used by MPEG-2 [or MPEG-4].

4.20 The MP3 audio format is described as "almost CD quality." What characteristic of MP3 makes this description accurate?

4.21 Use the Web as a resource to study the PDF format.

- **a.** Describe how PDF provides output that is consistent across different types of devices, including printers and monitors of various resolutions.
- **b.** Describe the format for storing, laying out, and managing the objects on a page. Explain the advantages to the use of this format over other formats, such as that used by HTML.
- **c.** Explain how PDF manages the many different type fonts that might be found in a document.
- **d.** How does PDF manage bitmap images? Object images?
- **e.** Explain how PDF differs from PostScript.
- **f.** Describe at least three major limitations that PDF places on the end-user of a PDF document.

4.22 Using the Web as a resource, create a table that compares the features and capabilities of .PNG, .GIF, and .JPEG.