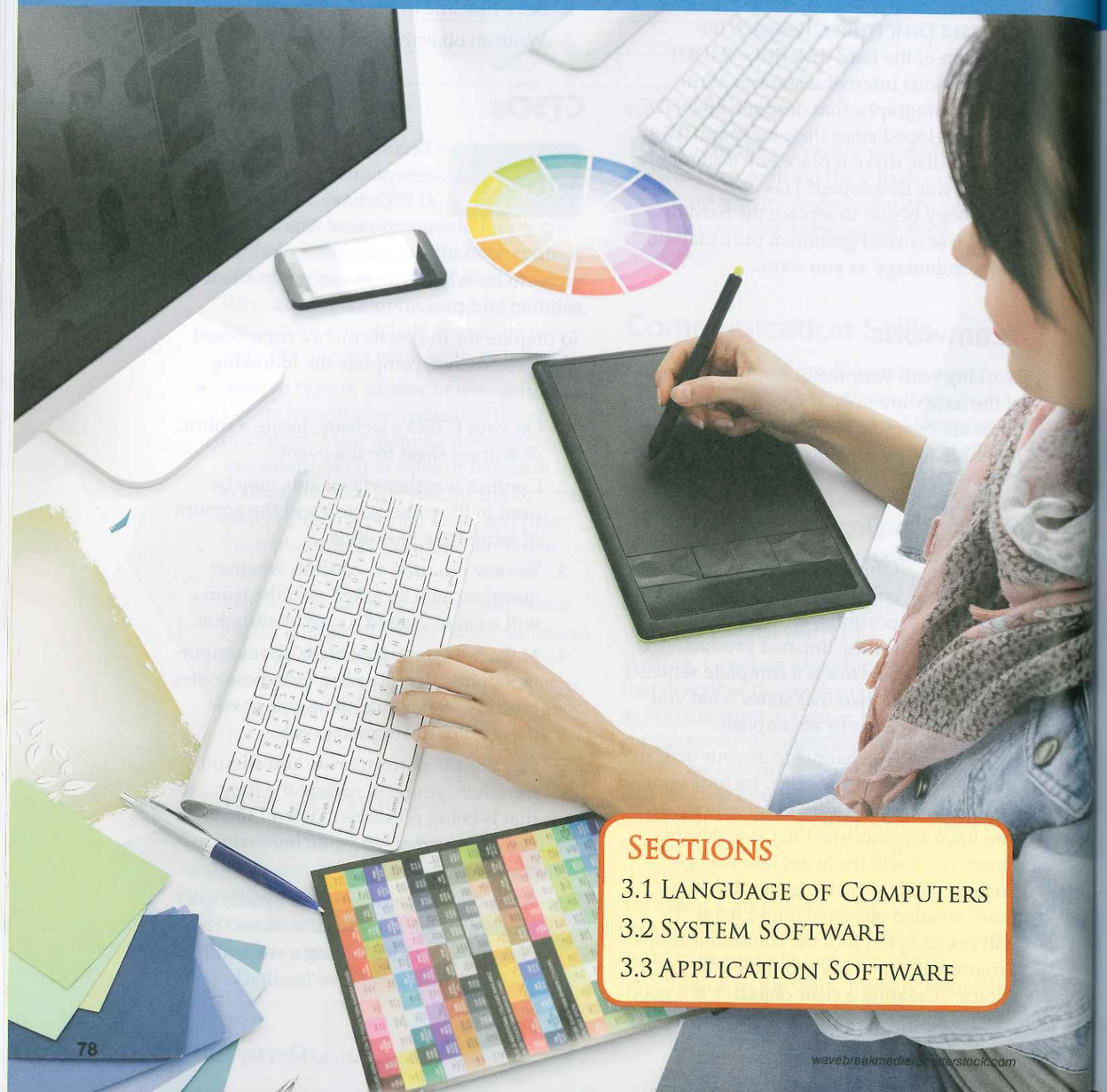
3

Software



Sections

3.1 Language of Computers
3.2 System Software
3.3 Application Software

wavebreakmedia/Shutterstock.com

Check Your IT IQ

Before you begin this chapter, see what you already know about information technology by scanning the QR code to take the chapter pretest. If you do not have a smartphone, visit www.g-wlearning.com.

In the previous chapter, you learned about computer hardware. Software is what makes the hardware work. Without software, a computer waits for instructions. Although the user cannot see, hear, or feel it, the software contains all of the instructions to operate the hardware. A set of instructions that tells the computer what to do is called a software program. Programs tell a computer to do specific jobs. These jobs include, among others, creating a document, scanning a photo, editing a video, and connecting to the Internet.

There are two basic types of software: system software and application software. System software works to help the CPU find programs, assign memory, run the devices, and provide utility programs. Application software is the software that performs the user's work. Applications include Microsoft Office, video games, phone apps, and web browsers. Users have a wide choice of software. Program selection is limited by what type of hardware is connected to the computer. Other choices depend on what tasks the user wants to perform. This chapter investigates many types of software, as well as discussing the language of computers.

IC3 Certification Objectives

Computing Fundamentals
Domain 1.0
Operating system
- **Objective 1.1** What is an OS and what does it do?
- **Objective 1.2** Manage computer files and folders

Domain 3.0
Computer software and concepts
- **Objective 3.1** Software management
- **Objective 3.2** Licensing
- **Objective 3.3** Software usage
- **Objective 3.4** Software tools

Domain 4.0
Troubleshooting
- **Objective 4.1** Software
- **Objective 4.3** Devices and peripherals

Key Applications
Domain 1.0
Common application features
- **Objective 1.3** Navigating

Living Online
Domain 4.0
Digital Citizenship
- **Objective 4.2** Legal and Responsible Use of Computers

College and Career Readiness

Reading Prep. Before you begin reading this chapter, consider how the author developed and presented the information. How does it provide the foundation for the next chapter?

SECTION 3.1

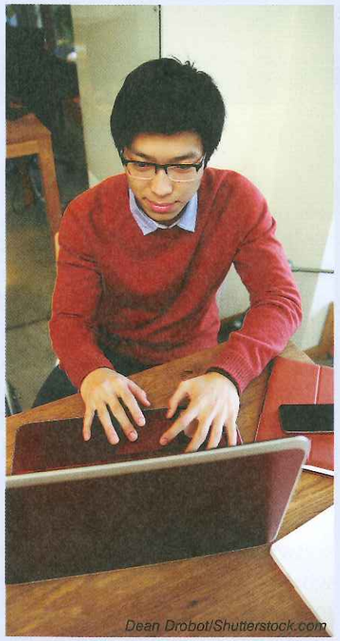
Language of Computers

Essential Question

How is a strong understanding of math important to computer programmers?

At the core of computers is the notion of two states: on-off, true-false, or yes-no. This two-state relationship is the key to how all information is stored and processed in the computer and has not changed since the concept of computers was developed. In electronic computers, which were first developed in the mid-20th century, the two states can be represented by 1s and 0s, giving rise to the use of the binary number system. A code was developed to represent alphabetic and special characters using patterns of binary numbers.

Knowledge of numbering systems, especially the numbering systems used by computers, is important to fully understanding how computers function. Information for use by the computer is encoded using these numbering systems. This section will describe how a language of binary numbers is used to store programs and data.



Dean Drobot/Shutterstock.com

Terms

American Standard Code for Information Interchange (ASCII)
assembly language
bit
byte
bytecode
code
compiler
computer algorithm
data type
encoding
high-level programming language
interpreter
low-level programming language
machine language
object-oriented languages
procedural languages
programs
unicode

Learning Goals

- Compare mechanical and electronic computers.
- Explain various number systems.
- Discuss computer programming languages.

Electronic Computers

The difference between mechanical and electronic computers is that the processor in an electronic computer has no moving parts. However, in both types of computers, the communication inside the computer is conducted through a series of on-off signals. In an electronic computer, each piece of information is kept in the form of a tiny amount of electricity. The electricity is either on (5 volts) or off (0 volts). These on and off signals are represented by the digits 1 and 0. The binary digit or **bit** is the basic building block for communication in an electronic computer. A bit can be only one of two values, which in an electronic computer is 1 or 0. Eight bits equal one **byte**.

All information in a digital computer is represented by the digits 0 and 1. Computer programs are developed in various programming languages and then decoded, or compiled, into 1s and 0s. All text, sounds, videos, and images are composed of 1s and 0s. Even numbers are ultimately combinations of 1s and 0s.

FYI

Electronic computers date from the 1940s when the Electronic Numerical Integrator and Calculator (ENIAC) was in development at the University of Pennsylvania.

Number Systems

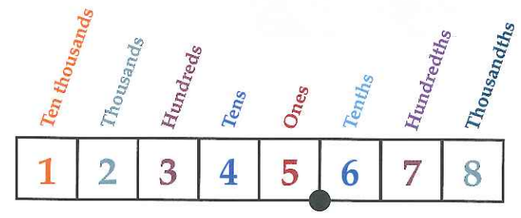
The most widely used number or counting system in the world is the decimal system. It is based on ten digits, 0–9, and was invented by Indian mathematicians between the first and fourth centuries. Arab mathematicians adopted the system in the ninth century, and the digits became known as Arabic numerals or Hindu-Arabic numerals. European societies adopted the system in the tenth century.

However, the decimal system is not the only counting or number system. Binary and hexadecimal number systems are used in certain applications, such as computer technology. Other number systems also exist.

Decimal Numbers

The decimal number system is a *positional* system. The position that a digit holds in a number indicates its value, as shown in Figure 3-1. In the number 2,587, the 5 indicates 500, but in the number 2,857 it indicates 50. Ten is the basis for the decimal system, which is why it is called a base-10 system. This means counting and arithmetic are done in groups of ten. It also means that the individual digits of a decimal number are represented by a power of 10. Recall from math class that the number 2,587 as an expanded number is:

$$
\begin{aligned}
2587 & \\
2000 &= 2 \text{ thousands} &&= 2 \text{ times } 10^3 \\
500 &= 5 \text{ hundreds} &&= 5 \text{ times } 10^2 \\
80 &= 8 \text{ tens} &&= 8 \text{ times } 10^1 \\
7 &= 7 \text{ ones} &&= 7 \text{ times } 10^0
\end{aligned}
$$



Goodheart-Willcox Publisher

Figure 3-1. In the decimal system, the position of a digit determines its value.

Binary Numbers

It became clear early in the development of computers that trying to represent ten different digits could make the computer too slow and bulky. The developers decided to use binary numbers to represent everything.

The binary number system is also a positional system. It is a base-2 system, so each individual digit of a binary number is represented by a power of 2. This is a sample binary number: 1111. As an expanded number, it is:

1111
1000 = 1 eights = 1 times 2^3
100 = 1 fours = 1 times 2^2
10 = 1 twos = 1 times 2^1
1 = 1 ones = 1 times 2^0

The binary number 1111 is the same value as 15 in the decimal system. Verify this by adding 1 eight + 1 four + 1 two + 1 one. The sum is 15.

The decimal number 2,587 converted to binary is 1010 0001 1011. This binary number can be expanded as:

1010 0001 1011
1000 0000 0000 = 1 two thousand forty-eights = 1 times 2^{11}
000 0000 0000 = 0 one thousand twenty-fours = 0 times 2^{10}
10 0000 0000 = 1 five hundred twelves = 1 times 2^9
0 0000 0000 = 0 two hundred fifty-sixes = 0 times 2^8
0000 0000 = 0 one hundred twenty-eights = 0 times 2^7
000 0000 = 0 sixty-fours = 0 times 2^6
00 0000 = 0 thirty-twos = 0 times 2^5
1 0000 = 1 sixteens = 1 times 2^4
1000 = 1 eights = 1 times 2^3
000 = 0 fours = 0 times 2^2
10 = 1 twos = 1 times 2^1
1 = 1 ones = 1 times 2^0

To verify the conversion, add 2048 + 512 + 16 + 8 + 2 + 1. The sum is 2,587, which is the original decimal number.

FYI

The binary number equivalent of a decimal number usually has more digits, but the computer arithmetic is much faster using binary numbers than decimal numbers.

Hexadecimal Numbers

Some of the numbers needed for output diagnostic procedures, memory management, and identification of network adapters and other hardware are very large. Large binary numbers can be easily misread by people because there are so many digits. For example, the binary number 1010 0001 1011 could be difficult for people to quickly interpret. Hexadecimal notation came to the rescue as a way to balance the ability for a human to read the number with a number system that could be easily read by a computer.

Hexadecimal notation, or hex, is a base-16 number system. When counting in base-16, new symbols are required to represent digits higher than 9. The 16 digits in the hexadecimal number system are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

FYI

Another number system used in programming, although less frequently than binary and hex, is octal. The octal number system, or oct, is a base-8 number system that uses the digits 0 through 7.

In hexadecimal notation, binary digits are grouped into units of four, each represented by another symbol. Four binary digits hold 16 different numbers, 0 to 15 or 0000 to 1111. Therefore, it is easy to write each group of four binary digits as one hex digit, which means fewer digits are needed to represent the number. This is why hex was adopted in computing.

Figure 3-2 compares initial values of the decimal, binary, and hexadecimal number systems. The binary number 1010 0001 1011 can be converted into hexadecimal using this table. Because each hex digit represents four binary digits, first break the binary number into groups of four digits starting from the right. Next, look up the corresponding hex digit for each group.

1010 0001 1011
1010 = A
0001 = 1
1011 = B

The hex conversion of 1010 0001 1011 is A1B. To avoid confusion with other number systems, it is a convention to write hex numbers with the 0x prefix. Therefore, the hex representation of the binary number 1010 0001 1011 is 0xA1B.

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	1 0000	10

Goodheart-Willcox Publisher

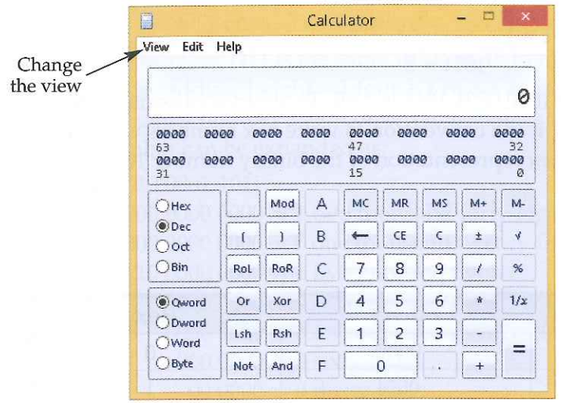
Figure 3-2. A comparison of values in the decimal, binary, and hexadecimal number systems.

HANDS-ON EXAMPLE 3.1.1

CONVERTING BETWEEN NUMBER SYSTEMS

The Windows operating system has a calculator accessory. When the calculator is used in the programmer view, changing between number systems is as easy as clicking radio buttons.

1. In Windows 8, launch the desktop calculator by clicking the **Apps** button and then **Calculator** in **Windows Accessories** group. Do not use the calculator app located in the **Apps** group. It has limited features. In Windows 7, launch the calculator by clicking the **Start** button followed by **Accessories** and then **Calculator**.
2. Click the **View** pull-down menu, and click **Programmer** in the menu. The display should look like the one shown.



3. Click the **Dec** radio button. This specifies the entry will be in decimal units.
4. Enter the number 365 using the keyboard or by clicking the number buttons.
5. Click the **Hex** radio button. The decimal number is converted to the hexadecimal equivalent and displayed, which is 16D.
6. Click the **Bin** radio button. The hexadecimal number is converted to the binary equivalent and displayed, which is 1 0110 1101 (without spaces).
7. Click the **Dec** radio button. The binary number is converted to the decimal equivalent, which is the original number entered.

Code

It is important for an IT professional to be able to understand and use number systems. Number systems represent more than numbers in the computer. They are used to convert media instructions, make programming codes, identify memory locations, identify peripherals, and many other tasks. Besides using binary numbers for calculations, computers use them to represent data and run computer programs.

Encoding Instructions

The process of converting human-readable data and computer programs into a computer-readable format is called **encoding**. The result of this process is called **code**. IT professionals often call computer programs *code*, referring to the end result rather than the program they are writing.

Different CPUs have different ways of interpreting code. This makes it impossible for one program to work on all computers. An example of code that tells a certain CPU to add two numbers is:

```
EB GB 00
```

Fortunately, most information technology professionals do not need to worry about this. There are many layers between what the CPU understands and what computer programmers write. Cross-platform tools allow a programmer to write a program once and then encode it for different CPUs.

Encoding Data

Programmers must know how computers translate data that the user may enter. Some agreements have been made by industry professionals to make sharing data among different computers easier than sharing programs. These agreements are called *standards*. The **American Standard Code for Information Interchange (ASCII)** is a standard for representing text that most computers support. The acronym ASCII is usually pronounced *askee*.

ASCII is a system for encoding the characters in the English alphabet and text for the digits 0–9. It also includes basic punctuation symbols and a blank space. There are 32 nonprinting characters used to control processes such as a hard return for the end of a paragraph.

Figure 3-3 shows an ASCII code chart for several text characters, also called symbols or *glyphs*. Notice that each ASCII character is defined by an eight-place binary number. This means eight bits are required to represent each ASCII character. Since eight bits equal one byte, each ASCII character uses up one byte of storage.

A curiosity that the ASCII table points out is the number 1 and the text character 1 are represented two different ways in a computer although they look the same when displayed on the monitor. The byte that represents the number 1 is the binary pattern 0000 0001 or hex 01. However, the byte that represents the text character 1 is the binary pattern 0011 0001 or hex 31. This is a source of concern for computer programmers. Text-character digits must be converted to binary numbers so the input can be used in a calculation.

Note that the table shows the binary, decimal, and hexadecimal code for each symbol. To store the characters in the word *Book*, these hex codes are used:

B	42
o	6F
o	6F
k	6B

STEM

Mathematics

The order of operations is a set of rules stating which operations in an equation are performed first. This order is often stated using the acronym *PEMDAS*. PEMDAS stands for parentheses, exponents, multiplication and division, and addition and subtraction. This means anything inside parentheses is computed first. Exponents are computed next. Then, any multiplication and division operations are computed. Finally, any addition and subtraction operations are computed to find the answer to the problem.

Binary	Dec	Hex	Symbol	Binary	Dec	Hex	Symbol	Binary	Dec	Hex	Symbol
0010 0000	32	20	(space)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

Goodheart-Willcox Publisher

Figure 3-3. ASCII characters (symbols) and their binary, decimal, and hexadecimal codes.

That means the word *Book* is encoded in hex as 0x426F6F6B. No calculations are performed on these hex patterns. They are used only to represent the letters.

Because of a need to represent more than 256 characters, the Unicode system was developed. **Unicode** is a system for encoding text characters in which two bytes are assigned to each character. This allows for a much greater number of characters, as well as support for other features. ASCII is incorporated into the Unicode scheme. Unicode is gaining wide acceptance.

HANDS-ON EXAMPLE 3.1.2

ENCODING TEXT STRINGS

Encode the text string *Hello World!* using hex notation. Use the ASCII chart shown in Figure 3-3 to locate the correct symbols.

1. Locate the codes for the characters in the word *Hello*.
2. Locate the code for a space.
3. Locate the codes for the characters in *World*.
4. Locate the code for the exclamation point.
5. Write the codes in order from left to right to form the code for the entire text string. Use the 0x prefix.

Computer Programming Languages

Programs are the sets of instructions that carry out the tasks of the user. Writing computer code using 1s and 0s was the original way to program a computer. Today, computer programmers use programming languages to write instructions that appear similar to English. There are two basic levels of programming languages: high and low.

High-Level Languages

Most software programmers use high-level languages. A **high-level programming language** contains instructions that are far removed from the instructions the computer CPU uses. High-level languages contain a limited set of near-to-English words, so they are easy for a person to read. Instructions may be commands such as **PRINT**, **IF**, **NEXT**, and **GET**. Programming in a high-level language can also be like algebra. For example, to add 42 to another number that is stored in the variable named JerseyNumber, write JerseyNumber + 42. Examples of high-level programming languages are Visual Basic, C++, Java, and COBOL.

The programming software contains a **compiler** that converts the programmer's code into code the CPU can understand. The compiled code is then linked with operating system–specific code to make the

executable program. Examples of languages that are compiled and linked are C, C++, Fortran, and COBOL.

Some languages are not compiled and linked, rather they are *interpreted*. In an **interpreter**, the instructions are converted to code the CPU can understand as the program is executing. This has two basic effects. First, the execution is slower than linked code because time must be taken to do the conversion for each instruction every time the program is run. Second, interpreted programs provide an opportunity to rapidly modify the code. Examples of interpreted languages are Smalltalk, Python, and some versions of BASIC.

A happy medium between a fully compiled language and an interpreted language is the bytecode. **Bytecode** is a set of instructions composed of compact numeric codes, constants, and references that can be efficiently processed by an interpreter. At run time, a software program called the *runtime engine* or the *virtual machine* interprets the bytecode for the target CPU. The advantage of bytecode is that high-level programs can be written for a wide range of CPU instruction sets and the interpretation does not take as long as fully interpreted code. Java and C# are two languages that partially compile their code so that they are more portable between operating systems and CPUs. In the case of Java, this step is performed every time the application is run. In C#, on the first execution of the program, the final linked code for the current computer is created and the resulting program is stored on the computer. Further running of the program is from the fully executable program and does not require interpreting.

Procedural languages are computer programming languages in which instructions are gathered into collections called procedures or functions. Each function contains a list of instructions for the computer to execute step-by-step. Examples of procedural languages are BASIC, C, Perl, and HTML.

Object-oriented languages contain data structures, called objects, and actions that can be performed on those structures, called methods. In addition to what can be accomplished using a procedural language, programmers are able to create programming objects to protect data related to the object. Examples of object-oriented languages are C++, C#, Java, and ActionScript.

Career Skills

Computer Support Specialist

Computer Support Specialists provide a wide range of help for technology users. Often referred to as help desk technicians, these IT workers help users understand how to operate a wide range of software and hardware technologies. They also manage a user's access to the network. Their goal is to support the users of the network.

Low-Level Languages

Once a program is written in one of the high-level languages, it must be converted into a digital format that the computer CPU can process. A **low-level programming language** is one that is very close to the instruction set used by the CPU. **Machine language,** or *machine code*, is a low-level language composed of the 0s and 1s the computer CPU uses. It is the only language that a CPU can directly understand. **Assembly language**, or *assembly*, is very close to machine language, but the CPU cannot directly understand it. An *assembler* is used to convert the program into machine code.

Programs in low-level languages are difficult for a person to read and write. For example, to store the number 42 into a register so that it can be used in a calculation, the command in assembly is MOV AL 2A. The benefit of assembly is that the code is written for a specific machine. This can make the execution of the program faster. The disadvantage of assembly programs is that they are generally not transportable to other CPUs.

Data Types Used in Computer Programming

In arithmetic, different types of numbers are used, such as counting numbers, fractions, and decimals. In a similar manner, computer languages distinguish among a variety of types of data, or data types. In computer programming, a **data type** is the description of values or information that can be accepted. Because data types are so different from each other, they are all stored in different ways.

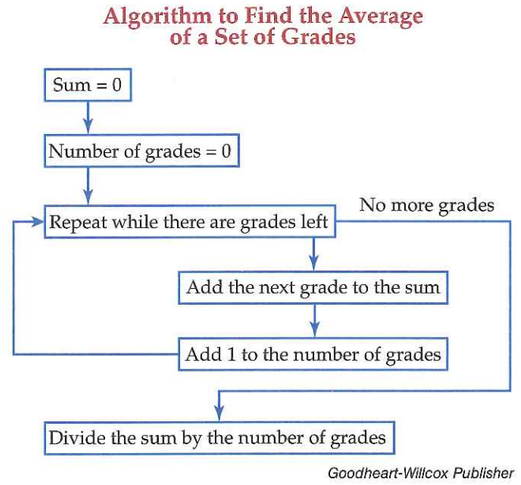
- Integers are the positive and negative counting numbers and zero.
- Floating point numbers are decimal numbers.
- Boolean types only hold values for true or false.
- Characters are single letters, digits, or other symbols.
- Strings hold alphabetic and numeric data, along with special symbols.
- Date types hold time and date information.

Programmers can also create their own data types. For example, if a programmer wants to use fractions, he or she could create a custom data type to represent the numerators and denominators of fractions.

Algorithms

A **computer algorithm** is a series of steps used to perform an action. An algorithm is not a programming language or programming code. Rather, it is a map of what needs to be done, and programmers write the code to activate the algorithm. At a basic level, this is the **IF**... **THEN** statement used in computer programming. For example, **IF** the user presses the [Ctrl][Z] key combination, **THEN** undo the last command.

A linear algorithm is processed once and the solution is output. An iterative algorithm is repeated, or looped, until a condition, called the terminating condition, is met. An algorithm may have both linear and iterative parts. For example, calculating an average of grades includes both types of algorithm. The program loops to keep adding grades until all are included (iterative) in the sum and then divides the sum by the number of grades (linear). A flowchart can be used to map out the steps in an algorithm, as shown in Figure 3-4.



Goodheart-Willcox Publisher

Figure 3-4. A flowchart can be used to map out the steps in an algorithm.

3.1 Section Review

Check Your Understanding

1. Convert this hex code into its binary equivalent: 0xF6A9
2. Convert this binary code into its hex equivalent: 1101101010011
3. What is the name of a program that converts higher-level programming language into machine language?
4. What is the difference between a compiler and an interpreter?
5. What is the basic diffference between a linear algorithm and an interative algorithm?

IC3 Certification Practice

The following question is a sample of the type of questions presented on the IC3 exam.

1. A memory address is stated as 0x14DE. What is the binary equivalent?

Build Your Vocabulary

As you progress through this course, develop a personal IT glossary. This will help you build your vocabulary and prepare you for a career. Write a definition for each of the following terms and add it to your IT glossary.

American Standard Code for Information Interchange (ASCII)
assembly language
bit
byte
bytecode
code
compiler
computer algorithm
data type
encoding
high-level programming language
interpreter
low-level programming language
machine language
object-oriented languages
procedural languages
programs
unicode

Section 3.2

System Software


lightwavemedia/Shutterstock.com

Essential Question

Which operating system is the best to use?

There are similar tasks common to most applications, such as saving data, retrieving documents, and printing. This is why operating systems were created. The operating system, or OS for short, sits between the hardware and the applications to handle all common tasks in one way. For a visual metaphor, think of a road as the hardware. The cars driving on the road are the applications. The drivers are the users, and the crossing guard is the operating system. It is the operating system that keeps all of the applications running smoothly and connected with the CPU and the peripherals.

System software contains the operating system, utilities, device drivers, and programs. Utilities help with housekeeping tasks. Device drivers provide instructions for how the operating system is to use peripheral devices. Programs are the applications that users run to complete the desired tasks.

Learning Goals

- Explain operating systems.
- Identify system utility programs.
- Describe device drivers.
- Discuss programs.

Terms

accessibility options
desktop theme
device driver
hibernation
language packs
platform
power down
power options
power states
sleep
system software
user account
utility programs